An Empirical Approach to Reinforcement Learning for Micro Aerial Vehicles

Junell, Jaime

DOI
10.4233/uuid:32765560-5fde-4c86-a778-decdc3eb5294

Publication date
2018

Document Version
Final published version

Citation (APA)
Junell, J. (2018). *An Empirical Approach to Reinforcement Learning for Micro Aerial Vehicles*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:32765560-5fde-4c86-a778-decdc3eb5294

AN EMPIRICAL APPROACH

To Reinforcement Learning For Micro Aerial Vehicles

JAIME JUNELL

An Empirical Approach to Reinforcement Learning for Micro Aerial Vehicles

AN EMPIRICAL APPROACH TO REINFORCEMENT LEARNING FOR MICRO AERIAL VEHICLES

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties
in het openbaar te verdedigen op maandag 10 december 2018 om 10:00 uur

door

Jaime Lin JUNELL

Master of Science in Mechanical Engineering,
Oregon State University, USA

geboren te Clackamas, Oregon, USA

Dit proefschrift is goedgekeurd door de promotoren.

Samenstelling promotiecommissie bestaat uit:

Rector Magnificus,	voorzitter
Prof. dr. ir. M. Mulder	Technische Universiteit Delft, promotor
Dr. Q.P. Chu	Technische Universiteit Delft, promotor

Onafhankelijke leden:

Prof. dr. A. Zolghadri	Université de Bordeaux
Prof. dr. ir. R. Vingerhoeds	Institut Supérieur de l'Aéronautique et de l'Espace, Toulouse
Prof. dr. E.K.A. Gill	Technische Universiteit Delft
Dr. ir. G.H.N. Looye	Deutsches Zentrum für Luft und Raumfahrt (DLR)
Dr. G.C.H.E de Croon	Technische Universiteit Delft
Prof. dr. ir. D.A. Abbink	Technische Universiteit Delft, reservelid

Dr. ir. E. van Kampen heeft in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

Keywords: reinforcement learning, micro aerial vehicles, quadrotor, policy-iteration, hierarchical RL, state abstraction, transfer learning
Printed by: Ipskamp Printing
Front & Back: design by Jessica Louie

ISBN 978-94-6186-965-4

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

To make a prairie it takes a clover and one bee,–
One clover, and a bee,
And revery.
The revery alone will do
If bees are few.

Emily Dickinson

CONTENTS

SUMMARY

The use of Micro Aerial Vehicles (MAVs) in practical applications, to solve real-world problems, is growing in demand as the technology becomes more widely known and accessible. Proposed applications already span a wide berth of fields like military, search and rescue, ecology, artificial pollinators, and more.

As compared to larger Unmanned Aerial Systems (UAS), MAVs are specifically desirable for applications which take advantage of their small size or light weight – whether that means being discreet, having insect-like maneuverability, operating in small spaces, or being more inherently safe with respect to injury towards people. In some cases, MAVs work under conditions where *autonomy* is needed.

The small size of MAVs and the desire for autonomy combine to create a demanding set of challenges for the guidance, navigation, and control (GNC) of these systems. Limitations of on-board sensors, difficulties in modeling their complex and often time-varying dynamics, and limited on-board computational resources, are just a few examples of the challenges facing MAV autonomy.

One approach to address these GNC challenges is reinforcement learning (RL) – a subset of machine learning and artificial intelligence. Reinforcement learning is based on the concept that humans and animals learn via rewards and/or penalties obtained through interaction with the world. When an action results in a positive experience, an individual will learn from that experience and will be more likely to take the same action in the future. Likewise, if a negative experience is had, the person will be less likely to take that action in the future.

Reinforcement learning is a promising framework for autonomous MAV tasks, which call for intelligent decision making in previously unexplored or changing environments. This is due to the ability of RL methods to be model-free and adaptive to changing circumstances in the system or environment. However, many disadvantages in reinforcement learning also exist, making it an active field in research to mitigate the known disadvantages while still maintaining the benefits of the algorithmic approach. This observation leads to the primary research question:

> *How can reinforcement learning contribute towards the goal of autonomous flight for micro aerial vehicles?*

This thesis addresses the challenges associated with reinforcement learning as it pertains to autonomous MAVs. An empirical approach is adopted by first *identifying* the problems which hold back RL from being successfully applied to MAVs; second, *designating* existing RL approaches which address those challenges; and lastly *designing and conducting experiments* on a quadrotor guidance or control task in simulation and within the TU Delft Cyber Zoo flight arena. This approach adds to the RL research com-

munity by focusing on the real-life application of RL approaches which were previously rarely seen outside of a simulated environment.

This thesis uses several different RL approaches to address the identified issues; with each chapter focusing on one approach that addresses one or more of the issues. The identified issues which are addressed in this thesis are:

- Slow learning due to *tabula rasa* learning,
- Curse of dimensionality,
- MAV limited resources, and
- MAV complex dynamics.

The designated reinforcement learning approaches are:

- "Classic" temporal difference reinforcement learning from *tabula rasa*,
- Hierarchical reinforcement learning over *options*,
- Hierarchical reinforcement learning with state abstraction,
- Self-tuning gains through policy gradient RL, and
- Transfer learning.

The first approach sets the stage with a look into the classic temporal difference reinforcement learning (TDRL) method from *tabula rasa*, applied to an MAV quadrotor, and using a camera sensor to detect reward states. The "honeybee task" is introduced as the main thematic problem – which is revisited in different variants throughout the thesis. This task is a sequential, multi-dimensional, optimization problem, formalized as a Markov decision process (MDP) which can be approached with or without a priori knowledge and can be scaled to represent different sized state spaces. Value function TDRL is shown to converge to an optimal solution for the honeybee task. The flight test gives a proof of concept for the use of vision-based rewards. Furthermore, "slow learning" is identified as one of the main limiting factors for RL – especially as it applies to MAVs whose flight time limitations are even more limited than ground-based applications or larger fixed-wing UAS, due to weight restriction on power sources.

The next approaches are based on hierarchical reinforcement learning (HRL) over *options* with Q-learning and explored in simulation. The methods are used to solve an obstacle-rich maze task where rewards are only collected when the end goal is reached. Because the reward only comes at the end, learning can be especially delayed in large scale problems since there are no organizational methods for random exploration with "flat" Q-learning. Using HRL incorporates temporal abstraction via extended actions (*options*). The result is that, even before learning, the agent requires about five times less timesteps to find the goal in the first epoch (1 epoch = 1 trip to goal). Finding the goal faster allows for the Q-function to be learned quicker; however, the better performance early on in the training of the agent comes at the cost of performance related to the convergence to the optimum. The HRL method converges more often to a suboptimal solution, while the flat Q-learning method converges, on average, more quickly to the

optimum. Therefore, this known trade-off should be considered when designing an HRL approach.

In certain MAV tasks, the vehicle will not have an "absolute" GPS-based location state. Cameras are considered one of the most information-rich, cheap, and light on-board sensors available for MAVs. Computer vision techniques are quickly becoming more advanced in order to interpret camera information in useful ways. Based on this, state abstraction is explored in the context of the HRL maze task. The state is represented "relative" to the MAV perspective and is therefore limited to the field of view of the vehicle. Since the maze is obstacle-rich, a state can be determined via the distance to obstacles at several angles within the field of view. This state can theoretically be achieved with a camera and computer vision techniques or with a sonar. This kind of state abstraction leads to several physical locations which have the same state vector – we call this state ambiguity. Though the state is now ambiguous, the state space size is effectively capped at the number of relative states, no matter how large the physical environment becomes. State abstraction is therefore one way to address the curse of dimensionality. The results with the relative state representation show that the temporal abstraction is vital for successful learning within the maze problem. Flat Q-learning with the relative state representation learns poorly within the large maze tasks. HRL with *options* uses extended actions to overcome the state ambiguity, and the combination of state *and* temporal abstraction performs better than temporal abstraction alone.

Many MAVs, such as flapping-wing MAVs, have complex flight dynamics which are not fully understood and are therefore difficult to model. This reality leads to models which are expensive, inaccurate, or both. Using a reinforcement learning approach can add *adaptability* to the model and by so doing, embrace the inaccurate model as a "good enough" starting point. From there, adjustments can be made via knowledge acquired through interactions with the environment. Using policy gradient reinforcement learning, the gains of an inaccurate model controller are tuned to optimize the performance metric. Gradient-based methods in simulation guide the direction of policy improvement so that there can be relatively few trials on the actual MAV. The gains of an F-16 are tuned using a high-fidelity model simulation as the "true model" and a less accurate model as the "inaccurate model". Results show that the method is effective; however, certain control tasks require careful attention to prevent instability. The PID gains of a quadrotor take-off task are tuned using a simple Newtonian model as the "inaccurate model" and the actual quadrotor as a "true model". On average, only three trial take-offs are needed to find the local optimum. However, the limitation is that the gradient-based method will only find local optima, so the starting policy is an important factor.

One of the greatest advantages of reinforcement learning over other methods is the ability to learn through interaction with the environment *without* any need for prior knowledge. Therefore, *tabula rasa* (blank slate) learning is at once both an attractive feature and also one of the main causes of slow learning speeds, since the agent must train with random actions until each state or state/action has been sampled a sufficient number of times. Transfer learning is an obvious solution *if* prior knowledge is available. For MAVs learning via RL, the most costly time is the time spent in-flight exploring the environment (most likely performing actions of unknown efficacy or safety). Cutting down on the in-flight exploration phase of learning can therefore be an attractive approach if

prior knowledge can be obtained in a more favorable scenario, with enough accuracy to act as a starting point. The last experiment in this thesis is the accumulation of all the techniques, with the addition of and focus on transfer learning. The honeybee task is expanded in state space and made non-Markov through the addition of a hidden state. The optimization guidance problem is implemented on a quadrotor in the Cyber Zoo flight arena. The Q-function for a set of HRL *options* is learned via simulation with controlled inaccuracies of the hidden state – affecting the state transitions. The results show that prior knowledge from a "source domain", even with inaccuracies in the state transitions, can be beneficial in certain metrics and under certain conditions when transferred to the "target domain". The prior knowledge improves the initial performance in the first several iterations of training within the target domain; however in certain cases, the "bad habits" learned in the source domain can prove to take longer to unlearn than to learn *tabula rasa*, when it comes to finding the optimal behavior.

Based on the obtained results, several recommendations can be made in terms of the direction for reinforcement learning research, and the autonomous flight of MAVs in general. This thesis has addressed a broad range of reinforcement learning techniques for realistic problems which fall outside the theoretical convergence guarantees of the RL framework. Many empirical studies do the same. A theoretical framework which includes more practical applications within its influence could give more focus to the RL research community. Further, studies into a more systematic approach for parameter tuning, reward shaping, and state abstraction could make reinforcement learning more accessible and successful.

While there has been much research into autonomy for MAV flight, less has been attempted in actual flight tests. Applications meant to be in the real-world must as often as possible, be conducted in the real world. There is no other way to know and learn from the true challenges that arise outside the artificial comfort of simulation.

Lastly, the possible MAV applications and the resulting societal impacts must be discussed by ethically responsible scientists and engineers. With so many areas of application for MAVs, the repercussions must be addressed and carefully considered. It is in the best interest of humanity and the earth in general to ensure that MAVs – or any technology – are designed and operated in a safe and ethical way; promoting privacy, antipollution, and conservation of nature.

SAMENVATTING

De vraag naar microvliegtuigen (micro-air-vehicles, of MAVs) voor het oplossen van praktische problemen in de echte wereld, is sterk groeiende. Dit aangezien de technologie steeds bekender en toegankelijker wordt. De voorgestelde toepassingen omvatten reeds vele domeinen, waaronder defensie en veiligheid, met haar zoek- en reddingsmissies, en de landbouw, zoals bijvoorbeeld het gebruik van MAVs als kunstmatige bestuivers. In vergelijking tot de grotere onbemande luchtvaartsystemen (unmanned aerial systems, of UAS), zijn MAVs specifiek wenselijk voor toepassingen waarbij een zeer klein formaat en/of zeer licht gewicht belangrijk is. Voorbeelden hiervan zijn de behoefte aan een grote, insect-achtige manoeuvreerbaarheid, het vliegen in kleine ruimtes en het veiliger opereren rondom mensen. Vaak worden MAVs in situaties waar een hoog niveau van *autonomie* vereist is gebruikt.

De wens naar hoge autonomie, in combinatie met het kleine formaat van MAVs, leidt tot grote uitdagingen betreft de begeleiding, navigatie en besturing (guidance, navigation and control, of GNC) van deze systemen. Voorbeelden van deze uitdagingen zijn de beperkingen van zowel de sensoren als de rekenkracht aan boord en moeilijkheden bij het modelleren van de complexe en vaak tijds-variërende dynamica.

Een methode om deze GNC uitdagingen aan te pakken staat bekend als de *reinforcement learning*, RL (conditionering), een onderdeel van *machine learning* (machinaal leren) en *artificial intelligence* (kunstmatige intelligentie). Reinforcement learning is gebaseerd op het concept dat mensen en dieren leren van beloningen en straffen die worden verkregen door interactie met de wereld. Wanneer een actie resulteert in een positieve ervaring, dan zal een entiteit (of 'agent') leren van die ervaring en dezelfde actie eerder nogmaals ondernemen in de toekomst. Evenzo zal een agent bij een negatieve ervaring minder geneigd zijn om de ondernomen actie in de toekomst te herhalen.

Reinforcement learning is een kansrijke methode voor autonome MAV-taken die vragen om intelligente besluitvorming en adaptieve capaciteiten. De RL-methoden veelal "model-vrij" zijn en zich kunnen aanpassen aan (onverwachte) veranderen situaties. Klassieke RL technieken hebben echter ook een aantal nadelen. Op dit moment wordt dan ook veel onderzoek gedaan naar het vinden van oplossingen voor deze nadelen, en dit zonder de voordelen van de algorithmen te verliezen. Deze observatie leidt tot de centrale onderzoeksvraag van dit proefschrift:

> *Hoe kan reinforcement learning bijdragen aan het doel van autonoom vliegen met microvliegtuigen?*

Dit proefschrift behandelt de uitdagingen die verband houden met RL toegepast op autonome MAVs. Een *empirische* benadering wordt gevolgd door eerst de specifieke problemen vast te stellen die op dit moment verhinderen dat RL met succes wordt toegepast op MAVs. Vervolgens worden bestaande RL-benaderingen die deze uitdagin-

gen aanpakken uitgelicht, waarna experimenten worden ontworpen en uitgevoerd op een quadrotor platform die veelvoorkomende GNC-taken uitvoert, zowel in simulatie als in de 'Cyber Zoo' vliegarena van de TU Delft. Deze benadering draagt bij aan de RL-onderzoekgemeenschap door zich te richten op *echte-wereld toepassingen* van RL-methoden, die voorheen zelden buiten een gesimuleerde omgeving zijn gezien. Dit proefschrift gebruikt een aantal verschillende RL-methoden om de geïdentificeerde problemen aan te pakken. Hierbij richt elk hoofdstuk van dit proefschrift zich op één methode die één of meer van de problemen aanpakt.

De volgende vier geïdentificeerde problemen worden onderzocht:

- Langzaam leren door *tabula rasa learning*(het leren met 'schone lei' beginnen),
- De "Curse of dimensionality"(de vloek van het snel groeiende aantal dimensies),
- De bovengenoemde beperkte middelen aan boord van MAVs, en
- De complexe dynamica van MAVs.

De vijf aangewezen RL-methoden die worden toegepast zijn:

- 'Klassieke temporal difference reinforcement learning' vanaf *tabula rasa*,
- Hiërarchisch RL over *options*,
- Hiërarchisch RL met 'state abstraction',
- 'Self-tuning gains' door 'policy gradient' RL, en
- 'Transfer learning'.

Deze veelal Engelse begrippen worden hieronder kort besproken.

De eerste benadering omvat de klassieke 'temporal difference reinforcement learning' (TDRL) methode vanaf *tabula rasa*. Deze methode wordt toegepast op een MAV-quadrotor en maakt gebruik van een camerasensor om beloningstoestanden te detecteren. De "honingbijtaak" wordt geïntroduceerd als het belangrijkste thematische probleem; variaties op deze taak worden door dit volledige proefschrift heen gebruikt. De honingbijtaak is een sequentieel en multidimensionaal optimalisatieprobleem, geformaliseerd als een *Markov decision proccess* (MDP), wat kan worden benaderd met of zonder voorkennis en kan worden geschaald om toestandsruimtes ('state spaces') van verschillende grootte te vertegenwoordigen. Er wordt eerst aangetoond dat de 'value function' TDRL convergeert naar een optimale oplossing voor de honingbijtaak. De uitgevoerde vluchttest verschaft een 'proof of concept' voor het gebruik van op het zicht gebasseerde beloningen. Bovendien wordt het langzame leren geïdentificeerd als één van de belangrijkste beperkende factoren voor RL, vooral wanneer toegepast op MAVs met rotors of flappende vleugels, omdat hier de vliegtijden vanwege de gewichtsbeperking op stroombronnen veel beperkter zijn dan bij niet-vliegende toepassingen of grotere vaste-vleugel ('fixed wing') UAS systemen.

De daaropvolgend toegepaste methoden zijn allen gebaseerd op hiërarchisch reinforcement learning (HRL) over *options* met 'Q-learning' en worden vooral verkend in computersimulaties. De methoden worden gebruikt om een hindernisrijke doolhoftaak

op te lossen, waarbij beloningen alleen worden verkregen bij het bereiken van het einddoel. Het uitreiken van de beloning *aan het einde* kan het leren vertragen, in het bijzonder in grootschalige problemen, omdat er geen organisatorische methoden beschikbaar zijn voor willekeurige verkenningen met "platte" Q-learning. Het gebruik van HRL bevat 'temporal abstraction' (abstractie in de tijd) door middel van uitgebreide acties (ook wel de '*options*' genoemd). Het resultaat is dat de agent, zelfs voordat enig leereffect heeft plaatsgevonden, ongeveer vijf keer *minder* tijdstappen nodig heeft om het doel in het eerste tijdvak te vinden (waarbij één tijdvak staat voor één reis naar het doel). Door het doel sneller te vinden kan de 'Q-function' sneller worden geleerd; echter, het betere resultaat in het begin van de training van de agent gaat ten koste van een suboptimale convergentie naar het optimum. De HRL methode convergeert vaker naar een suboptimale oplossing, terwijl de platte Q-learning methode, gemiddeld, sneller tot een optimum komt. Daarom moet deze (overigens welbekende) afweging worden gemaakt bij het ontwerpen van een HRL-methode.

Bij bepaalde MAV taken heeft het voertuig geen "absolute" GPS-gebaseerde locatiekennis. Camera's worden beschouwd als één van de meest informatierijke, goedkope en lichte sensoren die beschikbaar zijn voor aan boord een MAV. Computer vision technieken om visuele informatie op nuttige manieren te interpreteren worden tegenwoordig snel meer en meer geavanceerd. Op basis hiervan wordt 'state abstraction' (abstractie in de toestandsruimte) verkend in de context van de HRL doolhof-taak. De toestand wordt vertegenwoordigd vanuit het "relatieve" (camera-)perspectief van de MAV en is daarom beperkt tot het gezichtsveld van het voertuig. Omdat het doolhof obstakelrijk is, kan een MAV-toestand worden bepaald vanuit de afstand tot obstakels op verschillende plekken binnen het gezichtsveld. Deze toestand is theoretisch bereikbaar met een camera in combinatie met computer vision technieken, of met een sonar. Dit soort toestandsabstractie leidt tot de situatie waarbij verschillende fysieke locaties exact dezelfde toestandsvector kunnen hebben: we noemen deze toestand ambiguïteit ('state ambiguity'). Alhoewel de toestand ambigu is, wordt de grootte van de toestandsruimte effectief beperkt tot het aantal relatieve toestanden, ongeacht hoe groot de fysieke omgeving wordt. Toestandsabstractie is daarom een manier om de 'curse of dimensionality' aan te pakken. Resultaten met de relatieve toestand laten zien dat tijdsabstractie cruciaal is om succesvol te leren binnen het doolhof probleem. Platte Q-learning met de relatieve toestand leert beperkt in de grote doolhof taken. HRL met *options* maakt gebruik van uitgebreidere acties om de toestandsambiguïteit te overbruggen en we concluderen dan ook dat de *combinatie* van toestands- en tijdabstractie beter presteert dan tijdabstractie alleen.

Veel MAVs, zoals MAVs met flappende vleugels, hebben een complexe vliegdynamica die niet volledig wordt begrepen en die daarom moeilijk te modelleren is. Dit leidt tot modellen die duur zijn, onnauwkeurig, of beide. Het gebruik van RL kan *aanpassingsvermogen* aan het model toevoegen en daarmee het onnauwkeurige model gebruiken als een "goed genoeg" beginpunt. Vanaf dat punt kunnen aanpassingen aangebracht worden met behulp van kennis die is opgedaan door interactie met de omgeving. Door het gebruik van 'Policy gradient reinforcement learning' (een gradiënt-gebaseerde methode) worden de instellingen ('gains') van een in het begin relatief onnauwkeurig regelsysteem afgestemd om een prestatiemetriek te optimaliseren. Door gradiënt-gebaseerde metho-

den toe te passen op de modelsimulatie wordt de policy-verbetering gericht gestuurd, zodat er relatief weinig testen nodig zijn met de werkelijke MAV. Als voorbeeld worden de instellingen van een F-16 regelsysteem afgestemd met behulp van een waarheidsgetrouwe modelsimulatie als het "echte model" en een minder accuraat model als het "onnauwkeurige model". Resultaten tonen aan dat de methode effectief is, al vereisen bepaalde taken zorgvuldige aandacht om, mogelijk plotseling optredende, instabiliteiten te voorkomen. De PID-instellingen van een quadrotor opstijg-taak worden vervolgens afgestemd met behulp van een eenvoudig Newtoniaans model als het "onnauwkeurige" model en de *werkelijke* quadrotor als het "echte model . Het blijkt dat er gemiddeld slechts drie proeftesten nodig zijn om het lokale optimum te vinden. Een beperking van de gradiënt-gebaseerde methode is echter wel dat deze ook tot lokale optima zal kunnen convergeren, hetgeen de initiële instelling een belangrijke factor maakt.

Eén van de grootste voordelen van RL ten opzichte van andere methoden is het vermogen om te leren door interactie met de omgeving zonder dat er voorkennis nodig is. Dit leren vanaf *tabula rasa* (met een schone lei beginnen) is een aantrekkelijk kenmerk, maar het is ook één van de hoofdoorzaken van de soms zeer langzame leersnelheden, omdat de agent met willekeurige acties moet trainen totdat elke toestand of toestand/actie een voldoende aantal keren is geprobeerd. 'Transfer learning' is een evidente oplossing als er voorkennis beschikbaar is. Voor MAVs is de vluchttijd waarin acties met onbekende doeltreffendheid of veiligheid uitgevoerd worden erg duur. Het verkorten van de verkenningsfase tijdens de vlucht kan daarom een aantrekkelijke aanpak zijn als voorkennis kan worden verkregen in een minder kostbaar scenario, met voldoende nauwkeurigheid om als vertrekpunt te fungeren. Het laatste experiment in dit proefschrift integreert alle technieken, met toevoeging van – en focus op – transfer learning. De toestandsruimte van de honingbijtaak wordt uitgebreid en wordt bovendien 'niet-Markov' door de toevoeging van een verborgen toestand ('hidden state'). Het optimalisatie-begeleidingsprobleem wordt geïmplementeerd op een quadrotor in de Cyber Zoo vluchtarena. De Q-function voor een set van HRL *options* wordt geleerd in simulaties met gecontroleerde onnauwkeurigheden van de verborgen toestand – die van invloed zijn op de toestandstransities ('state transitions'). De resultaten tonen aan dat voorkennis van een 'source domain' (brondomein) voordelig kan zijn voor de overdracht naar het 'target domain' (doeldomein), zelfs met onnauwkeurigheden in de toestandstransities. De voorkennis verbetert de initiële prestaties tijdens de eerste iteraties van het trainen in het target domain. Desalniettemin kan het in bepaalde gevallen ook langer duren in vergelijking met tabula rasa, omdat in deze gevallen de "slechte gewoonten" aangeleerd in het brondomein moeten worden afgeleerd in het doeldomein, althans wanneer het gaat om het vinden van het *optimale* gedrag.

Op basis van de gevonden resultaten kunnen verschillende aanbevelingen worden gedaan wat betreft de richting van toekomstig RL-onderzoek en autonoom vliegende MAVs in het algemeen. Dit proefschrift heeft een breed scala aan RL-technieken behandeld voor realistische problemen, die buiten de theoretische convergentie-garanties van het RL-kader vallen. Een breder theoretisch kader dat meer praktische toepassingen ondervangt kan de RL-onderzoekgemeenschap een specifiekere focus geven. Verder zouden studies naar een meer systematische benadering voor parameterafstemming, beloningsvormgeving en toestands-abstractie, de RL toegankelijker en succesvoller kunnen

maken. Hoewel er veel onderzoek is gedaan naar de autonomie van MAV vluchten, is er tot op heden minder bereikt in echte vliegtesten. Toepassingen die bedoeld zijn voor gebruik in de echte wereld moeten zo vaak mogelijk in de echte wereld worden uitgevoerd. Er is namelijk geen andere manier om begrip te vergaren van de echte uitdagingen die zich buiten het kunstmatige comfort van de simulatie voordoen.

Ten slotte moeten de mogelijke MAV-toepassingen, en de daaruit volgende maatschappelijke effecten, worden besproken door ethisch verantwoorde wetenschappers en ingenieurs. Het scala aan mogelijke toekomstige toepassingen voor MAVs is groot en dus moeten de negatieve gevolgen worden aangepakt en zorgvuldig worden overwogen. Het is in het belang van de mensheid en de aarde in het algemeen om ervoor te zorgen dat MAVs – of welke technologie dan ook – op een veilige en ethische manier worden ontworpen en uitgebaat, zodat deze de privacy bevorderen, vervuiling tegengaan, en de natuur in stand houden.

1

INTRODUCTION

AUTONOMOUS flight for Micro Aerial Vehicles (MAVs) is appealing for a number of reasons. Small flying vehicles have the ability to survey indoor and outdoor areas, patrol streets, film aerial movie shots, inspect bridges, deliver mail, or get to areas not accessible by humans or land-based vehicles.

Autonomy for any type of Unmanned Aerial System (UAS) has the advantage of fulfilling tasks without the need of constant human monitoring and/or interference. These are often tasks which humans do not want to do, where there are communication limitations, or where computer decision making is more optimal than human input. In cases where human interaction is not possible, such as indoors or during space missions, it is necessary for the vehicle to make intelligent decisions autonomously, especially when the environment is unknown or changing over time.

The small size of MAVs [1] has the benefit of fitting into smaller areas such as indoor environments or cluttered outdoor areas. Furthermore, smaller vehicles are stealthier and can be used for missions where low visibility is required. Several MAVs have also been developed which focus on maneuverability that cannot be obtained with fixed wing platforms. Flapping wing MAVs such as those in the DelFly series [2] (Figure 1.1), for example, aim to mimic the incredible agility and maneuverability of insects in flight [33]. Quadrotors, likewise, are highly maneuverable, strong in stationary hover flight, and affordable on the commercial market.

An MAV in real-life applications will encounter unforeseen and unpredictable situations which calls for fast and intelligent decision making. When a human is not available as the decision maker, a reliable, adaptable, and autonomous method must be in place for the mission to succeed.

[1] Within the scope of this thesis, MAVs are defined as < 1kg in weight
[2] www.delfly.nl

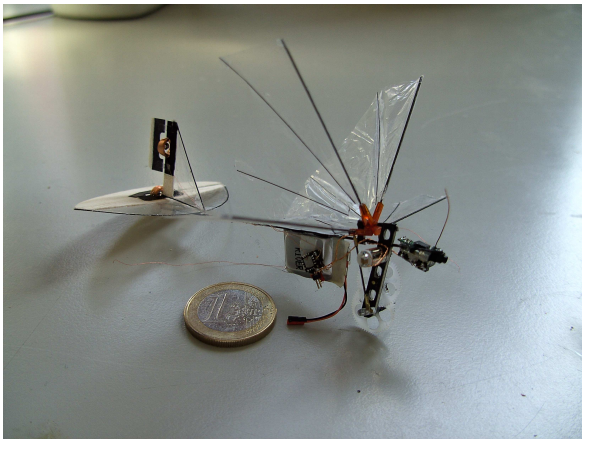


Figure 1.1: The DelFly Micro weighs only 3 grams and is equipped with a camera (photo source: delfly.nl)

1.1. MOTIVATION FOR AUTOMATION AND AUTONOMY OF MAVS

Automation of robotics has become one of the greatest human time-saving accomplishments in the last half-century. In the aerospace and aviation industry, automation is deeply ingrained in almost every sector from aircraft control to air traffic management. The vision for the MAV branch within the industry follows suit and goes even further into aspirations for *autonomy*.

Discussions on the difference between automation and autonomy in robotics easily become confusing due to the nearness of the two definitions and the inconsistent or interchangeable use of the two terms [20, 22, 76, 97, 99]. Terminology related to autonomous systems will emerge regularly within this thesis; Therefore, for the sake of clarity, the following definitions are distinguished as:

Automated or automatic systems use limited-to-no human interaction to perform a task. Automation can incorporate limited logic or feedback to react accordingly to a number of predetermined circumstances, but cannot make decisions in unanticipated scenarios (hence, the warning messages on a number of appliances which calls for human-operated troubleshooting).

Autonomous systems are a distinct though overlapping concept related to automated systems. The confusion comes because autonomy, in its traditional definition, implies independence from some entity; and by that definition, automation increases the level of autonomy for systems by removing reliance on humans. As defined in this thesis, the concept of *autonomy* can incorporate forms of automation, but an *autonomous system* must additionally allow for decision making in circumstances that the designer has not explicitly accounted for – or be able to find solutions that the designer has not predetermined or preprogrammed. Autonomy can therefore be accomplished by automation with aspects of *learning* or *adaptive* features to address unanticipated circumstances in a more intelligent way.

Motivation to limit or remove human decision making from a system comes from a number of accompanying benefits. Automation in industrial and manufacturing processes flourished because machines could more efficiently, safely, and cheaply accomplish the monotonous tasks of assembly line workers [17, 108]. The same argument can be applied to many other industries, including the aviation industry which incorporates automation in the form of autopilots onboard all major commercial aircraft [77, 90].

For MAVs, there are many tasks that only require an automation level of autonomy, while other tasks will call for truly autonomous systems with learning or adaptive features. For example, quadrotors or other MAVs can assist with the current and projected high demand for pollination in agriculture, by acting as "artificial pollinators" for plants and crops [26, 115]. An automated task could be to fly over predetermined locations in an indoor greenhouse, using the down-wash of the rotors to blow on self-pollinating plants. A task which requires autonomy beyond automation would be to find the locations of the plants in an uncharted area and cross-pollinate by visiting many plants one after another: as bees do.

The uses for autonomous MAVs are numerous and expanding everyday as the technology becomes more accessible and widely known. Achieving full autonomy for MAVs in the real world is an interdisciplinary challenge involving not only engineering fields like vehicle design, control, and human factors; but also expertise in certification or societal impact, for example. Each in her own field must do her part and there is little doubt that one of the most important fields enabling the technology for autonomous flight is the field of Guidance, Navigation, and Control.

1.2. AUTONOMY VIA GUIDANCE, NAVIGATION, AND CONTROL

At the heart of autonomous flight is the field of Guidance, Navigation, and Control (GNC). In other words, GNC systems provide the functions which give an aerial vehicle its level of autonomy from humans. As a whole, GNC systems can be described as the process that occurs between sensing and actuation of a system; however, attributing domain to each of the three branches can be inconsistent, even within the same discipline.

For the purpose of this thesis, each component is defined below and visually illustrated as a system in Figure 1.2.

Navigation Acquisition, extraction and inference of sensor information to determine information about the vehicle's state and its environment. The resulting state information will be largely dependent on the sensors available, which can include: cameras, GPS, accelerometers, barometers, sonars, and many other possibilities. MAVs are usually restricted to light-weight sensors.

Guidance Using the state information from the navigation system, guidance determines the planning, behavior laws, or decision making, to achieve some goal in a (preferably) optimal manner. This can include short term goals like avoiding an obstacle and longer term goals like reaching a goal state.

Control Execution of the commands from the guidance system. This includes manipulating the inputs to the vehicle's actuators so that the desired maneuver is performed efficiently and safely.

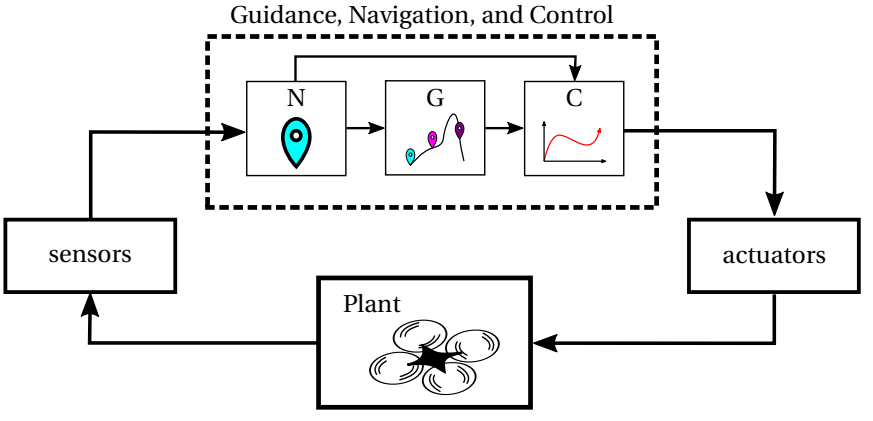


Figure 1.2: GNC concept diagram

Since GNC is a broad field, the focus in this thesis is on guidance and control, especially within decision-making and adaptability for MAV platforms. Therefore, the state is assumed to be known in simulation and only robust methods of navigation are used in the real-life test flights.

1.2.1. CHALLENGES IN MAV GUIDANCE AND CONTROL

The challenge in autonomous guidance and control for MAVs comes from inherent properties of small light-weight aircraft, as well as from the types of tasks which are desired of them. Compared to their larger counterparts, small vehicles are more affected by environmental disturbances, such as wind gusts. Likewise, they are also impacted by changes in the system brought on by damage to the vehicle or changes over time in properties, such as battery life or material loss of elasticity. Certain MAVs, like the Delfly, are built in small numbers. The manufacturing process therefore results in small differences in every vehicle. These differences are a modeling challenge since no two vehicles are alike. In order to fly, smaller vehicles are inherently limited in weight and therefore light-weight sensors and other hardware are desirable [33]. Compared to bigger or heavier versions, these sensors often have less accuracy and more uncertainty in the information obtained. Additionally, MAVs are highly desirable for complex tasks which require agile movement and/or navigation through unknown environments [72, 136].

Several taxonomies within research literature describe the current challenges for autonomous flight and roadmaps attempt to find gaps in knowledge and set an action plan for future direction of studies. Broad examples include the United States Department of Defense reports and roadmaps to help guide future military plans for the technology of interest: technologies such as military UAS [95–97]. The American Institute of Aeronautics and Astronautics (AIAA) holds workshops and publishes roadmaps for certain fields of technology in aerospace research, including one for Intelligent Systems in 2016 with one of the key focuses on autonomy, broadly on a systems level, and within GNC efforts [3, 4]. Overviews which are more recent and specific to MAVs include a sur-

vey by Kumar et al. which discusses the challenges of MAVs (≤ 1 kg) and the pros and cons with respect to these challenges for different vehicles such as fixed-wing, quadrotors, ducted-fan, and flapping wing aircraft [72]. The dissertation of S. Tijmons, gives a general overview of state-of-the-art MAVs and a complete taxonomy of the advances in control automation for flapping-wing UAVs in recent years [136]. The study focuses on control for platforms which are particularly difficult to model.

We specifically look at the challenges of autonomy on two levels: guidance and control. The sensitivity of MAVs to disturbances and changes to the system predicates the desire for control which is adaptive and robust. Sensor limitations establish that decisions will have to be made using limited or possibly inaccurate resources. A desire for agile movement of the MAV means that the dynamics can be complex and difficult to model in mathematical terms. Finally, tasks in unknown environments will require intelligent ways to efficiently explore and fulfill objectives.

GUIDANCE

Addressing autonomous guidance through unknown environments is novel since commercial and recreational uses of MAVs usually still require a human pilot to make decisions, even if remotely piloted [25]. Figure 1.3 shows examples of commercially available vehicles with various degrees of autonomy in guidance. The small toy drone from Cheerson company (*left*) only flies under control of a human pilot and therefore has no autonomy. The Roomba® vacuum cleaner (*right*) is almost fully autonomous once activated – even returning to the charging station on its own [68]. However, it is a land-based robot which is not in the scope of this thesis but still acts as an exemplary example for autonomy. The image of the Parrot® Bebop [3] *(middle)*, illustrates an off-the-shelf recreational quadrotor MAV. It is the closest example to the class of drone of interest in the scope of this thesis. Hobbyists remotely pilot the quadrotor using GPS and a smartphone or tablet as the controller, but it also has some automated features for taking off, landing, and navigation to a home location. The predecessor to the Parrot® Bebop is the Parrot® AR-Drone 2 which is the platform of choice for the research detailed in this thesis. The Parrot quadrotors are convenient as a research platform due to its ease in being overwritten with customized onboard GNC, allowing for greater automation and autonomy.

Moving from a piloted outdoor scenario to an autonomous, GPS-denied, unknown environment is not a trivial matter. Just as with a human pilot (but without the cognitive advantage), exploration of the previously unknown (unmapped) environment would be necessary. Mapping an environment as the vehicle explores its surroundings is possible, such as with the case of SLAM (Simultaneous Localization and Mapping) [92]. This allows for acquisition of previously unknown knowledge. However, sensors needed for this approach which estimate distances to walls and obstacles, are either heavy or inaccurate. Laser-ranging systems are the most accurate but also the heaviest. Camera based systems are lightest but require idealized conditions to get accurate readings. Furthermore, the computational cost of SLAM becomes substantial in large spaces since it aims to map the entire space with high uncertainty [9, 55, 140]. Once the map is created and the goal state is known, it is simple for the agent to find its way again. Less popular

[3] http://www.parrot.com/products/bebop-drone/

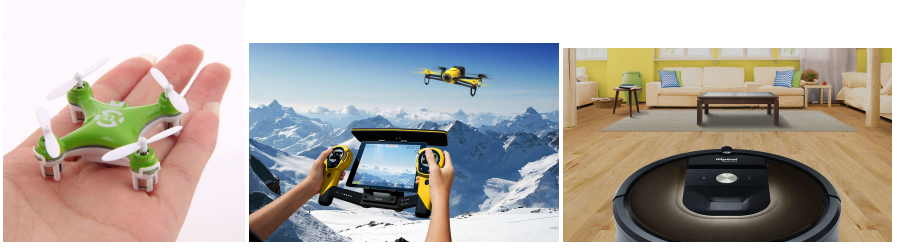


Figure 1.3: Commercially available vehicles with different degrees of autonomy. (*left*) the Cheerson CX drone is an inexpensive toy quadrotor only able to fly with human piloting (photo source: www.cheersonhobby.com) (*middle*) Off-the-shelf quadrotor for recreational use: Parrot® bebop remotely piloted with a tablet with some automated features (photo source: www.parrot.com), (*right*) Roomba autonomous vacuum cleaner, once activated can vacuum within a home and return autonomously to the recharging station (source: www.irobot.com)

than SLAM, others have approached the challenge by developing sophisticated sensor systems to enable mapping with only cameras [58] or other light weight sensors [93].

CONTROL

To address complexity in dynamics and system changeability on a control level, modeling of the system dynamics with adaptable parameters has been shown to be useful. Advanced control methods such as Model Reference Adaptive Control (MRAC) [63], Incremental Nonlinear dynamic Inversion (INDI) [117, 120], Backstepping [141], and dynamic inversion with neural networks [65], are all promising model-based methods which can handle non-linear systems and which have adaptive functionality to account for model inaccuracies. However, these methods still require a model of the system with varying degrees of model accuracy. Where model accuracy can be sacrificed for sensor-based feed-back, as in INDI, problems can arise with sensor delay [117, 120]. Therefore, no perfect control method currently exists for complex, non-linear systems.

One solution which addresses these issues is reinforcement learning (RL). This type of machine learning algorithm models itself after the way humans learn: by interacting with its environment and learning the desirable behavior using feedback it receives from the environment and a reward or penalty structure. Reinforcement learning methods allow for model-free learning, which means that it can learn without any *a priori* knowledge of the system. This approach is applicable to a large span of problems because it needs minimal to no information, is adaptable, and can learn complex behavior from a simple reward structure.

1.3. REINFORCEMENT LEARNING

Reinforcement learning is a machine learning algorithm which draws its inspiration from the way humans or animals learn [130]. *Learning* can be defined as the ability of an agent to improve upon its performance via experience within its environment. In the context of this dissertation, the learning aspect from reinforcement learning is what separates an automated robot from an autonomous one.

Learning to walk: an analogy

A baby is not born with a mathematical formula describing the mechanism of walking, nor is she immediately able to walk. She is, however, in possession of an innate will to go places: An internal reward structure which causes humans and animals to take steps in favor of survival. Through practice, the child will learn the muscle activation sequence necessary to walk. She will fall many times, and with every fall will learn from it. The process of learning to walk on one level of muscle control can be described in engineering terms as such: A child finds herself in a *state* where the left leg is forward with the weight on back leg. What is the correct *action* to take? We know from experience that the child should shift the weight to the forward leg, and then move the right leg to the forward position. From that initial state, the child who has never tried to walk before will adopt a *policy* to try many actions at *random*. Actions which end in falling will receive the *penalty* of pain from the fall. Actions which result in movement toward a desired destination will result in a *reward*. This reward structure will cause the child who finds herself again in that same state, more likely to try the rewarding action and less likely to try the penalizing action. Her policy has changed to take advantage of her new experience. Eventually, after enough successes and failures, she will have encountered all the important states and know what actions need to be taken from there. She will have learned to walk.

The principle of reinforcement learning in a simple form is visualized in Figure 1.4, where the agent, represented by an MAV, chooses an action which then interacts with the environment. From that interaction, the agent will collect some sort of feedback in the form of a reward or penalty and perceive its new state. This iterative processes gives the agent a basis to learn rewarding behavior. The mathematical approach is further discussed in Chapter 2.

Reinforcement learning is a promising tool for making intelligent decisions toward autonomous flight. Like the child in the walking analogy, Micro Aerial Vehicles (MAVs) are not always created with inherently stable and controlled flight and the mathematical model can be very difficult and expensive to obtain [6, 24]. Furthermore, the tasks desired of MAVs are often complex and cannot be solved by traditional linear controllers or rule-based decision makers. Model-free reinforcement learning approaches, like a child learning to walk, only need to gain experience to improve upon the behavior of an agent. Therefore, this learning approach is well suited for tasks within uncertain environments, decision making with sequential solutions, and where adaptation to changing conditions is required. MAVs would specifically benefit from this approach since they are often used in tasks within unknown environments where learning from experience without a human supervisor will be necessary.

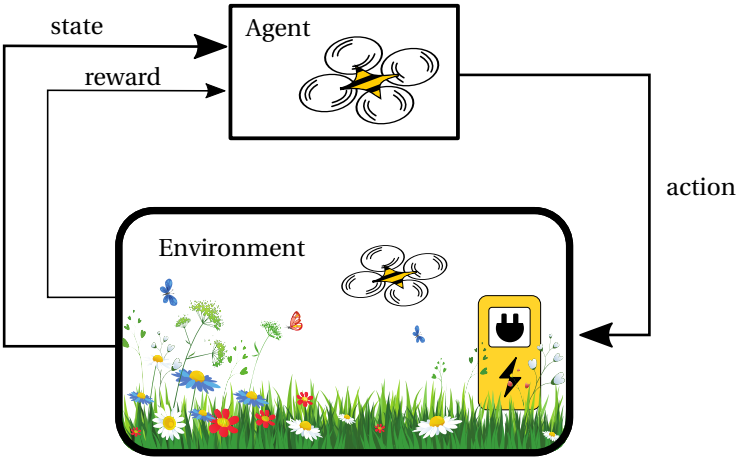


Figure 1.4: Visualization of the reinforcement learning (RL) algorithmic principle where the agent interacts with the environment to learn behavior which will return the greatest reward. In this representation, the environment represents a recurring theme in this thesis where a quadrotor agent is used to imitate a honeybee visiting flowers. The recharging station illustrates the unavoidable need for a quadrotor to recharge its battery in order to be fully autonomous in this artificial pollinator type scenario.

1.4. CHALLENGES IN REINFORCEMENT LEARNING

There are still many limitations of RL which have created the need for continued research on the topic. The search for improvements in scalability (curse of dimensionality) [13, 125], safety [15, 80, 81], partial observability [98, 135, 146], the continuous domain [38, 52, 114], and convergence guarantees [16], continue within the machine learning community.

In this thesis, the challenge of slow learning speed is specifically addressed since that is one of greatest limiting factors for MAVs. To improve the speed of learning, two limiting factors are addressed:

Slow learning speed due to *tabula rasa* learning: *Tabula rasa* learning (learning from scratch) is time consuming since nothing is known a priori and much exploration is needed to train the agent. Most tasks for real robotic systems have time-sensitive aspects which constrain this sort of large time commitment.

The curse of dimensionality: Real-life systems function in the continuous domain where there are an unlimited number of states and actions. Discretization of the state/action space for reinforcement learning purposes leads to large value function matrices which can either surpass memory limits or become intractable.

Several approaches have been investigated to address these open-ended problems. To battle the curse of dimensionality, approximations for a continuous domain have been made using function approximators such as radial basis functions and neural networks [23, 38, 52, 114]. In discrete cases, the state/action space can be reduced by decreasing the number of state inputs through state abstraction [27, 36], or by making sub-

tasks through task decompositions within a hierarchical structure [13, 37]. To speed up learning, parameter tuning and reward shaping have been used for improved convergence performance[53, 91]. Furthermore, although one of the most touted benefits of RL is its ability to be model-free, a model for simulation *can* be used to support faster learning by an iterative process [2] or to provide a better starting point through transfered knowledge [75, 133].

Great progress has been made toward remedying some of these limitations within RL, however it should be noted that it is most often only in simulation, where an idealized world does not capture the true unpredictability of real-life. General mistrust of non-conventional control and a lack of validation methods have kept industry from certifying RL and other learning methods [42, 64].

The goal of this dissertation is to provide empirical proof-in-practice for reinforcement learning on real-life flying platforms and to identify further challenges.

1.4.1. MICRO AERIAL VEHICLE RELATED CHALLENGES FOR RL

The aforementioned efforts in the reinforcement learning research community all aim to contribute toward one end-goal: Reinforcement learning in real-world applications. A considerable section of that effort is toward robotics in general, including applications like Robocup soccer [110, 126] or obstacle avoidance [86]. There are substantially fewer works specifically toward flight of MAVs, and even fewer which go past simulation into real-life flight tests. Since MAVs have their own set of characteristics, it is worth exploring the challenges of the reinforcement learning method within the context of MAV flight.

The small size of an MAV is its defining and most advantageous feature, but is also the cause of many unique challenges which other UAS do not encounter [72, 136]. The reinforcement learning method will also be subject to these restrictions in real-life applications.

Most research in reinforcement learning subscribes to the standard practice to first simulate; and for one reason or another, [4] it often ends there. However, there are a number of notable studies where machine learning has been applied to MAVs.

Figure 1.5 demonstrates just some of the MAV platforms which have recently been used for research in learning methods. The quadrotor [121] and the ducted-fan [65] in Figure 1.5(c) and Figure 1.5(d), respectively, each use a form of Artificial Neural Networks as an adaptive feature for a model-based controller. There are only a few examples of reinforcement learning used on an MAV platform. The helicopter in Figure 1.5(a) was used with reinforcement learning to learn aerial acrobatic maneuvers [1]. The quadrotor in Figure 1.5(b) was created custom at Stanford as a testbed and was used to improve upon classical linear stabilization control with RL as compared to Integral Sliding Mode [143]. Another study (photo not available) used a quadrotor to learn stable hover without a model [18].

In this thesis, the reinforcement learning approach takes MAV limitations and task-specific conditions into consideration and tries to mitigate some of these challenges, specifically focusing on:

[4] Some reasons a researcher may only simulate an RL application include: necessary resources are not available, its not applicable to physical systems, or waiting for advancement in the technology of other disciplines.

(a) (b)

(c) (d)

Figure 1.5: MAV research platforms used with learning and adaptive approaches. (a) Helicopter for aerial acrobatics in [1]. (b) Stanford STARMAC platform using RL for hover stabilization in [143]. (c) Quadrotor learning obstacle avoidance in forest with deep neural networks [121]. (d) Ducted-fan with adaptive control using dynamic inversion and neural networks [65].

Limited resources: Weight restrictions limit many aspects of MAV hardware, including: sensors, computational resources, battery size, and actuators.

Complex dynamics and time-varying properties: The small size of the vehicle results in faster dynamics which can be easily influenced by external disturbances. Furthermore, properties of the vehicle might change over time.

Designers of micro aerial vehicles are challenged to keep the vehicle as light as possible but still have the necessary hardware specifications to fulfill its purpose. Relative to other kinds of robots, the MAV is limited in its sensing capability by the number and quality of the sensors that the vehicle can carry. The setup of each experiment in this thesis takes into consideration the limitations of MAVs by working toward state representations which would be available via the sensors of most MAV platforms. Cameras can be placed on some of the lightest MAVs – such as the Delfly – and they are information rich [34]. The burgeoning field of computer vision has some promising solutions for extraction of state information, but still faces many challenges [34, 84]. Additionally, GPS is often available on quadrotor style MAVs. For this reason, GPS-style position tracking is also used within this thesis as a viable resource.

The fast dynamics and time-varying properties of the MAV make it challenging to model these vehicles. Small vehicles are, in general, more susceptible to environmental disturbances, such as wind gusts. Time-varying properties of the system create the need for adaptability. Such properties include: the blades of the quadrotor wearing out over

several hours of use, or the delicate wing of the Delfly stretching to result in different aerodynamic properties. Detailed modeling of each individual vehicle can be used to account for these intricacies [6, 24], however every model will have inaccuracies. *Adaptability* built into the system would be an invaluable asset to combat model inaccuracies and time-varying properties.

It is the intention of this thesis to show that reinforcement learning can work within – and even make improvements to – the guidance and control of the aircraft under these limitations.

1.5. RESEARCH QUESTIONS

With the big picture aim of fully autonomous intelligent MAVs, and a recognition that reinforcement learning can play an important role, the primary research question of this thesis is formulated:

Primary Research Question

How can reinforcement learning contribute towards the goal of autonomous flight for micro aerial vehicles?

There are a wide variety of ways in which reinforcement learning can be used to improve the autonomy of a vehicle. This thesis focuses on autonomy via RL learned intelligent decision making. Specifying further, the primary question can be decomposed into sub-questions which address challenges from two sources: 1) the inherent limitations of MAVs, and 2) the inherent limitations of reinforcement learning as applied to real-life systems.

The research sub-questions can then be formulated as:

Research sub-questions

What RL methods are available to overcome the following *practical* challenges associated with reinforcement learning in real-life flight of micro aerial vehicles?

Q1. Slow learning due to *tabula rasa* learning (Ch. 3, 4, 5, & 6)

Q2. Curse of dimensionality (Ch. 3 & 4)

Q3. MAV limited resources (Ch. 2, 4, & 6)

Q4. MAV complex dynamics (Ch. 5)

Each of the sub-questions is addressed in one or more chapters in this thesis, as denoted above. There is considerate overlap in several chapters because, by design, each method or task usually addresses more than one of the challenges.

As previously discussed, there is a full field for adaptive non-linear controllers for lower level control which has had great successes in recent years. Reinforcement learning can also contribute to this field. One way is by addressing the uncertainty in the dynamics of complex systems by using a simple model and RL policy improvement to

improve the ease of gain tuning for a PID controller.

The greatest way that RL can help autonomy is, arguably, in decision making for guidance tasks in unknown environments where other deterministic methods lack the adaptability, and humans lack the ability to be calculative. Therefore, RL will have most impact to improve autonomous flight by learning to find near-optimal guidance solutions in previously unknown environments while using only the sensors available on the vehicle.

1.6. RESEARCH APPROACH AND CONTRIBUTIONS

Reinforcement learning techniques could expand the capabilities of MAVs in making intelligent decisions in tasks which call for autonomy. However, the shortcomings of reinforcement learning (RL) have limited its usefulness in real-life applications. This thesis aims to progress RL techniques to real-life micro aerial vehicles.

This thesis tests some of the RL approaches which are meant for real-life application, and will reveal that the promise shown by simulated RL can be transitioned to the real world. The result of this thesis is a series of experiments which substantiates the applicability of the RL approach, as well as a guide of recommendations for what sort of tasks RL is well-suited for, and where it is currently not well-suited.

The contributions resulting from this series of experiments will now be laid out with respect to the four research sub-questions stated above.

SLOW LEARNING DUE TO TABULA RASA LEARNING

Learning speed, in the sense which we want to improve it, is the time *in-flight* it takes to learn a near-optimal solution for the task. Since MAVs have a limited battery life, learning speed is an important aspect for RL. To speed up learning from *tabula rasa* (blank slate), we have taken the approach of hierarchical reinforcement learning (HRL). Temporal abstraction in the form of extended actions (*options* [129]) are able to bypass some of the time involved with taking random actions to explore the state space. Chapters 3 and 4 show that the use of HRL *options*, drastically speeds up learning early on in Q-learning training, but at the cost of optimality later on.

The speed of learning can also be improved by NOT starting from *tabula rasa*. In Chapter 5, an inaccurate model is used to give a starting point for an RL policy, and then iteratively use the model to find the direction to change the policy. This cuts down on the number of in-flight trials needed for the task. In Chapter 6, *tabula rasa* learning is circumvented by solving for an initial state/action value function in a simulated "source" domain and then implemented in the real-world "target" domain. Depending on the commonality between the source and target domain, the agent can have a jumpstart in the greedy performance and an improved learning speed.

CURSE OF DIMENSIONALITY (SCALABILITY)

As the state space increases, the problem can quickly become intractable for discrete methods. Rather than turning to the continuous domain, hierarchical approaches (as in Chapters 3 and 4) can make large state spaces manageable with temporal abstraction. The benefits of HRL are shown for large state spaces by comparing the HRL method

against a conventional flat Q-learning RL approach in three different sized mazes. The benefits are greater as the maze size increases.

Furthermore, scalability is also addressed in Chapter 4 with state abstraction. If one state representation scales up quickly with the problem, this representation can be replaced by a new state representation which has a smaller state space. However, this new smaller representation will not provide the same level of state accuracy or specificity. Ambiguity in the state is introduced and can be a problem for flat reinforcement learning. Chapter 4 shows that HRL can reduce the disadvantages of ambiguity, even in cases where flat RL is unable to learn.

MAV LIMITED RESOURCES

In Chapters 2 and 6, vision-based rewards are used in order to demonstrate the functionality of reinforcement learning with a light-weight sensor such as a camera with a color filter. However, the position state knowledge is still used to determine location. From there, resources are further limited in Chapter 4 by completing a maze guidance task in a GPS-denied environment where only a relative vision-based state representation can be known. Hierarchical reinforcement learning is used to leverage the ability of extended actions (*options*) to overcome the ambiguity introduced by the relative state in a large maze.

COMPLEX DYNAMICS OF MAVS

Complex dynamics of an MAV can be difficult and expensive to model accurately; however, a “bad" or inaccurate model, is relatively easy to come by. The approach of Chapter 5 is to use reinforcement learning policy improvement to iteratively calculate a bias for the inaccurate model by using feedback from interactions with the real world, which in turn iteratively improves the performance of the task by tuning the gains of a PID controller along a gradient until it converges to the local optimum. The tuning is guided by experience with the environment which means the self-tuning gains will be specifically adapted to this vehicle. This is an especially attractive capability in the case of MAVs with small manufacturing differences. If a relatively accurate model and optimal gains can be found for one vehicle, this same controller can be used for a vehicle with slight manufacturing flaws and the differences can be accounted for using the reinforcement learning policy improvement method.

1.7. SCOPE AND LIMITATIONS

The subject of reinforcement learning application on MAVs is cross-disciplinary and exists within, and in association with, broad fields such as: GNC, artificial intelligence, machine learning, computer vision, and others. In order to focus on the questions stated in Section 1.5, the scope of this thesis is reasonably restricted.

1.7.1. THEORETICAL NOVELTY

This thesis focuses on empirical *application* of RL methods to real-life flying platforms and those associated challenges. Algorithmic novelty is not part of the contribution of this thesis. The researchers who contributed and published these methods are cited in

the text and in the bibliography. However, the reinforcement learning approaches cited are adjusted for the tasks which they must execute and this work does contribute to the understanding of the existing RL algorithms in respect to MAV tasks.

1.7.2. CONVERGENCE GUARANTEES

Reinforcement learning can guarantee learning convergence to an optimal policy under certain conditions [37, 130]. However, as soon as these conditions are no longer met (which often happens in practical situations), convergence can no longer be guaranteed. The goal of the research in this thesis is from a practical standpoint and therefore takes an engineering approach as compared to a mathematical one. Learning convergence is an important aspect of RL, and therefore steps are taken to converge to the optimal solution, such as slowly ramping up the epsilon in the ϵ-greedy policy. Furthermore, convergence is discussed in the results section of each chapter – just not from the standpoint of mathematical guarantees.

1.7.3. SAFETY IN REINFORCEMENT LEARNING

The challenges for reinforcement learning with MAVs addressed in this thesis are denoted in Section 1.4.1. A prominent challenge for RL which is not a focus in this thesis is *safety*. The random actions often used for exploration can lead to dangerous behavior for some real-life tasks. Others have researched safe exploration techniques to mitigate the risks of "blind" exploration required when an environment is unknown a priori [48, 81].

Though safety is not in the expressed focus of thise thesis it is also not fully ignored. In guidance, hitting walls is not allowed as the agent is hard-coded to stay in the current state and receives a penalty for Chapters 2, 3, 4, and 6. In control of a system, safety can be addressed in terms of stability. In Chapter 5, instability is discussed as a possible disadvantage of reinforcement learning use with controllers and is explored in the application of PID tuning using policy-gradient reinforcement learning around regions of sharp instability transitions.

1.7.4. NAVIGATION

Guidance, Navigation, and Control are often lumped together as they work closely together. By the definitions given in Section 1.2, the scope of this thesis focuses on guidance and control; though, navigation may be implied in a camera-based state representation such as in Chapter 4. In that chapter, we assume that the navigation state based on GPS or camera-based vision is working reliably. In real-life applications, this will of course not always be the case. Computer vision is quickly improving and is one of the technological catalysts for the growing MAV research field. Because there is already a large effort towards navigation enhancing sensors like pinpoint global positioning and computer vision techniques, the scope of this thesis has focused on having a reliable navigation source and focuses its efforts on guidance and control.

1.8. OUTLINE OF THESIS

The outline of this thesis is visualized in Figure 1.6. In Chapter 2, a reinforcement learning task is performed using a quadrotor platform and camera-based rewards. By obtain-

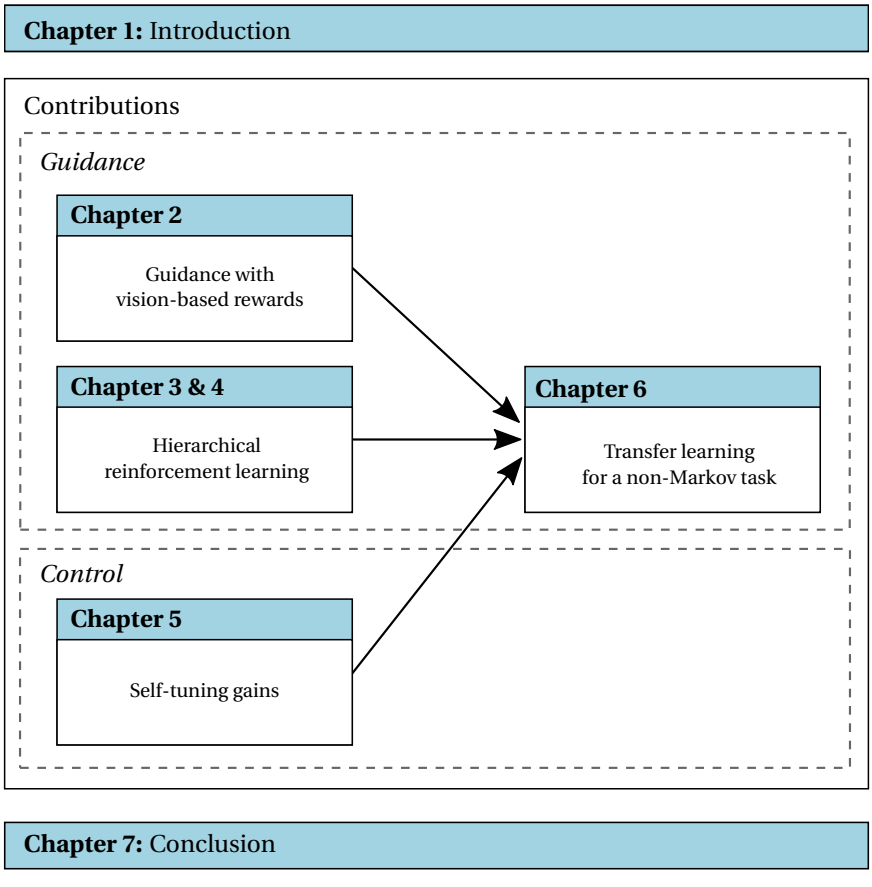


Figure 1.6: Visual outline of thesis.

ing reinforcement through its own sensors, the quadrotor can autonomously learn an optimal guidance law to complete its task. This chapter gives an introduction to one of the most fundamental forms of reinforcement learning which is still used as a foundation for many branches of RL research. Likewise, this chapter functions as a foundational starting point for the rest of the thesis.

In Chapters 3 and 4, hierarchical reinforcement learning (HRL) is demonstrated as a promising approach for decision making in large state spaces; and therefore a potential method to combat the curse of dimensionality and slow learning. Chapter 3 begins with the introduction of HRL and temporal abstraction to improve learning speed early in the training stages. Chapter 4 then expands on the utility of hierarchical methods by exploring the use of state abstraction. The state representation used is that of a completely isolated MAV in a GPS-denied environment. Therefore better representing the situation where intelligent decision making is most vital for MAV applications.

In Chapter 5, locally optimal gains of a PID controller are learned using reinforcement learning with a simple model of the dynamics to speed up learning. Using a PID

controller is desirable since it is currently the standard in industry. Using RL to tune the gains gives the low level controller better adaptability to changing conditions. By using a simple model to aid in learning, a vehicle with complex dynamics can still be efficiently and autonomously controlled with a well-tuned PID controller.

In Chapter 6, the cumulative efforts of the previous chapters are compiled onto a quadrotor platform to perform a "honeybee" guidance task in a larger and more complex environment than what is seen in the initial example in Chapter 2. Vision-based rewards from Chapter 2 are extended upon, and hierarchical methods with *options* from Chapters 3 and 4 are further explored in a different type of task. Gains tuned in Chapter 5 for the vertical loop control are used during flight test (but do not influence the results since it is a guidance task in the horizontal plane). The chapter builds upon those methods further with the addition of transfer learning and a brief analysis into the effect of hidden states. The addition of transfer learning introduces the use of a "source" domain in which to find a good starting point for the state/action value function, which can have a jumpstart and learning speed benefit over *tabula rasa* learning in the real-world "target" domain.

2

HONEYBEE TASK: TEMPORAL DIFFERENCE REINFORCEMENT LEARNING WITH VISION-BASED REWARDS

Achieving autonomous flight of Unmanned Aerial Vehicles (UAVs) within unknown or uncertain environments involves many guidance and control challenges. This chapter demonstrates the utility of reinforcement learning for UAV high-level guidance decision making with an example of a honeybee task where the bee gathers nectar from flowers to bring back to the hive. The task is first performed in simulation and is then extended to a real-life quadrotor platform where learning occurs online and rewards are recognized via the camera and vision-based identification. Through exploration of the environment and a well selected reward structure, it is shown that an optimal policy can be found within a previously unknown environment. Results demonstrate the important considerations of reinforcement learning such as: reward structure design, policy selection, and parameter selection. The basic algorithmic principles and the resources used for the real-life test set a foundation for the work in the rest of this thesis.

Parts of this chapter have been published in:

J. Junell, E. van Kampen, C. C. de Visser, and Q. P. Chu. Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, Kissimmee, Florida, USA, 2015. [67]

2.1. INTRODUCTION

Autonomous guidance and control of Unmanned Aerial Vehicles(UAVs) and Micro Aerial Vehicles (MAVs) is challenging for many reasons. Dynamics of the system can be difficult to model, the vehicles are often susceptible to unforeseen disturbances, and there are on-board sensor and actuator limitations. One method to address these problems is to create learning or adaptive controllers. Reinforcement learning (RL), for example, enables an agent to learn the right actions to take with little or no a priori knowledge of its dynamics or environment and can adapt to changing conditions [130]. For these reasons, RL has become a promising tool for improving autonomous flight in many different types of UAVs and MAVs. This chapter will focus on learning an optimal mission path using Reinforcement Learning for a Parrot® AR-drone 2 quadrotor. Furthermore, it will demonstrate how this technology is becoming more accessible by using an inexpensive platform and an open source autopilot.

When trying to achieve autonomous UAV flight, there are many levels of control to consider. In lower level control, the vehicle must be able to adjust its control surfaces or thrusters for rate or angle control. In the case of a quadrotor, it is simply to adjust the power to each individual engine for the desired attitude and accelerations. There have been many advances in lower level MAV control in recent years, both with model-based controllers, learning control and a mixture of the two [1, 18, 113, 143]. Reinforcement learning has been implemented on a real quadrotor platform and compared to non-linear control methods for accurate hover [18] and improved vertical control [143] during non-linear disturbances. Other types of learning have been used to fine tune acrobatic maneuvers of quadrotors [104] within a well equipped flight arena [59] that provides information from a camera system. Expanding to other types of MAVs, a small helicopter was successfully controlled with RL in more difficult maneuvers by using a trained human pilot during the learning phase [1]. Finally, a quadrotor has used RL to hover and track a defined trajectory [113].

For higher level control and guidance, an autonomous vehicle must make intelligent decisions in order to achieve its objective. High level control using reinforcement learning for real life ground robots has been explored to learn good behaviors of an E-Pet [61]. Finding an optimized path is of utmost importance when the vehicle is limited in battery life or mission time. Vehicles can also operate in an unknown environment where there is not enough information available to calculate the ideal path to their objective. This is a case where reinforcement learning can be effectively implemented.

Quadrotors are relatively straight-forward to model for non-aggressive maneuvers and therefore can have fairly accurate lower level control with a PID controller in this flight regime. Reinforcement learning can be used effectively in the high level decision making to result in an optimized path for the UAV despite limited knowledge of the environment. The goals of this chapter are to 1) Introduce the basics of reinforcement learning, 2) demonstrate an application of a real-life quadrotor as a reinforcement learning agent, and 3) create a foundation for further research with greater autonomy and capability to adapt to unforeseen environmental and vehicle changes.

The experiment presented is set up to mimic just one of the many possible real-world application that could benefit from a learning or adaptive controller. The quadrotor task is analogized to a honeybee task of collecting nectar from flowers and bringing it to the

hive. The reinforcement learning aspect of the agent is driven by interaction with it's environment in the form of GPS obtained locations state and vision-based reward detection. It is also easy to think of many other tasks where such a learning agent would be beneficial; for example, a reconnaissance task in a disaster scenario where MAVs are taking pictures of collapsed or damaged buildings and bringing the images back to headquarters. In any case, reinforcement learning can be used to learn desirable behavior.

This chapter presents a simulation of the honeybee RL decision process in an unknown gridworld and then applies it to a real-life quadrotor in the TU Delft Cyber Zoo [1]. The Cyber Zoo is a space outfitted with OptiTrack [2] optical tracking system, where ground robots and aerial vehicles can be employed in experiments using one or more agents. Lower level control of the quadrotor is supplied by Paparazzi [21, 109], an open source autonomous control software that provides a flexible autopilot system for fixed wing and multi-copter vehicles.

A background of Temporal Difference reinforcement learning is outlined in Section 2.2. The simulation setup and results will first be presented in Section 2.3 before the application setup and results in Section 2.4. An overview of all the results and conclusions is discussed in Section 2.5.

2.2. REINFORCEMENT LEARNING PRELIMINARIES

2.2.1. MARKOV DECISION PROCESSES (MDPS)

The basic framework of reinforcement learning [130] consists of a learning agent and its interaction with a discrete-time, finite, Markov decision process (MDP). This means that the problem setup must satisfy the Markov property stating that the policy is only dependent on the current state and not on a history of states. The MDP is a 5-tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$. Let $\mathcal{S}$ and $\mathcal{A}$ be defined as finite sets of states and actions, respectively. $\mathcal{P}^a_{ss'}$ is the state transition probability from state $s \in \mathcal{S}$ to s' given action $a \in \mathcal{A}$,

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.1}$$

and $\mathcal{R}$ is the reward function describing under what conditions the agent will be awarded or penalized. The policy, π, is a mapping between states and actions. Finally, $\gamma \in [0,1]$ defines the discount-rate parameter. The expected one-step reward is described as:

$$\mathcal{R}^a_s = E\{r_{t+1} | s_t = s, a_t = a\} \tag{2.2}$$

The value function, $V^\pi(s)$, represents the value of being in state s given that policy π is being followed. To solve the MDP, the agent must gain information about $V^\pi(s)$ for each state using interaction with the environment in order to then deduce an optimal policy from which to determine desirable actions. The value represents how "good" it is to be in a particular state and is calculated by the discounted expected returns from being in that state. The following equation defines $V^\pi(s)$ mathematically as the sum of all future expected rewards.

[1] TU Delft Robotics Institute. http://robotics.tudelft.nl
[2] Optitrack™. http://www.naturalpoint.com

$$V^{\pi}(s) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, \pi\} \tag{2.3}$$

A similar value can be calculated for each *state-action pair*, which indicates how good it is to take an action, a, given the agent is in state, s. This is called the Q function, $Q^{\pi}(s,a)$ and will come into play in the next chapter.

$$Q^{\pi}(s,a) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_t = a, \pi\} \tag{2.4}$$

The V or Q functions will be initialized and then updated as the agent explores the environment. The update is characterized by the iterative Bellman equation:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{s} + \gamma V^{\pi}(s')] \tag{2.5}$$

and results in an update rule:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{2.6}$$

until V^{π} converges to its optimum, $V^{\pi *}$.

The policy used during exploration of the state space, π, will determine to what values the value function or state-action function converges to. Given that the value function is converged to $V^{\pi *}$ then the optimal policy is a greedy policy which always selects the action with the highest resulting value or the highest state-action value.

2.2.2. ACTION POLICY

In reinforcement learning, the policy (π) decides what actions the agent will take. Introduced below are the policies implemented in this thesis.

Random Every action has an equal probability of being selected.

Greedy The "best" action will be chosen every time; where the best is defined by the action which leads to the highest valued $V(s')$, or the state-action pair which has the highest value $Q(s,a)$.

ϵ-greedy Takes the greedy action, ϵ percent of the time. Otherwise, a random action is taken (where the greedy action can still possibly be selected at random). We can calculate the probability that the greedy action, P^{a}_{greedy}, will be selected (Eq. (2.7)), or any other action, P^{a}_{other}, will be taken (Eq. (2.8)), where N_a is the number of possible actions.

$$P^{a}_{greedy} = \epsilon + \frac{1-\epsilon}{N_a} \tag{2.7}$$

$$P^{a}_{other} = \frac{1-\epsilon}{N_a} \tag{2.8}$$

After an infinite number of iterations, the V-function of the policy, V^{π}, will converge to V^{π^*}. The difference between different policies is due to the action probabilities associated to the policy. Note that $\epsilon = 0$ is just a random policy and $\epsilon = 1$ is a greedy policy. Since a higher valued action will be chosen more often with higher ϵ, the boundaries will be randomly hit less, the rewards will be found more frequently, and consequently the values in the fully converged V^{π} will be higher.

A full analysis of how the policy effects value function convergence can be found in the Results section.

2.2.3. TEMPORAL DIFFERENCE REINFORCEMENT LEARNING

In Temporal Difference reinforcement learning, a Markov property is assumed. That is, the agent doesn't need to know any of its past actions to make its next action. the agent, therefore only needs minimal information to inform its decision. Those pieces of information being: The agents current state, the Value Function, and the transition probability.

The value function, $V^{\pi}(s)$, represents the value that is assigned to each state, s. The honeybee task in this chapter, consists of 216 states and so the value function is embodied by 216 associated values. With each move the agent makes, it experiences part of the environment and can adjust the value function to learn what states are more desirable to visit.

The value function is updated by the Bellman Equation, Eq. (2.9). Where E_{π} is the expected reward in a state given a policy, π. Eq. (2.10) shows the update rule for temporal difference RL used in this chapter. Design of the reward function, r, and tuning of parameters such as the step size, α, and the discount factor, γ, are topics of great importance to reinforcement learning and have their own fields of research which are not the focus of this thesis. Further detail of the reward function and parameter design for this task are explained in Section 2.3.

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{2.9}$$

$$V(s) \leftarrow V(s) + \alpha(s)(r + \gamma V(s') - V(s)) \tag{2.10}$$

Each state that is visited will be updated based on the rewards that it receives at that state, and the value of the next state it visits. This recursive property makes it possible to propagate a reward from one state to the states that surround it, and therefore make a transparent path of increasingly higher values to the reward states. The visual appeal of this property can be seen in the simulation results section, embodied in Figure 2.4.

2.3. SIMULATION

This section will describe how a reinforcement learning problem can be set up to imitate a honeybee collecting nectar for the hive.

There are many real world applications that this kind of reinforcement learning concept could be used for. This experiment is set up to demonstrate a honeybee with a mission to search for and collect nectar from the flowers (reward states). For each flower

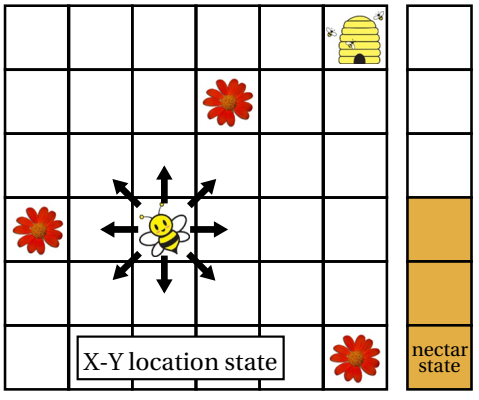


Figure 2.1: Flower reconnaissance RL concept: This figure demonstrates a discrete 6x6x6 grid world where the agent's location state is described by its x-y location in the grid and its nectar state is represented by the bar on the right. The bee changes its nectar state by visiting flowers or the hive and can change location state by moving about the grid to adjacent squares(ie. 8 possible actions).

it visits, some nectar is collected and carried with the bee. The bee can carry a limited amount of nectar, so as the bee becomes weighed down, it must return to the hive to drop off the nectar. Then it starts over collecting nectar.

2.3.1. PROBLEM SETUP

Using a reinforcement learning decision process, a honeybee agent is simulated within a grid world where 3 flowers and a hive exist. The bee gets rewards for visiting flowers until its figurative nectar state is full. The more nectar the agent has collected, the greater the reward will be for visiting the hive. The bee is set in a grid world that satisfies the Markov property.

The bee's physical world consists of a 6×6 discrete grid with the hive located at [x y] = [6 6] and the flowers located at [1 3; 4 5; 6 1]. The environment is visualized in Figure 2.1. The boundaries of the grid world are not accessible.

STATE SPACE

The state of the agent is defined by 2 state values: Location state and nectar state.

The nectar state is an integer between 1 - 6 that represents how much nectar the bee is carrying on his body. This state is visually represented with the nectar state bar along side the x-y grid in Figure 2.1. The nectar state initially starts at 1 meaning the bee has collected no nectar. With each flower visited, the nectar state value increases by 1 until the bee is at maximum nectar capacity at nectar state = 6.

The current setup leads to a $6 \times 6 \times 6$ three-dimensional (or a 36×6 two-dimensional), matrix of states with a total of 216 states. The grid size and the nectar value maximum were arbitrarily chosen and could be adjusted as desired. However, as total number of states increases, so does the computational expense.

ACTION SPACE

The agent can move to any adjacent square, including diagonal, for a total of 8 actions. Figure 2.1 shows the grid world set up as well as the possible actions. If the agent hits a boundary limit with the action, the agent will incur a penalty and stay in the same location state as the previous state.

REWARD FUNCTION

The reward states are the flowers and the hive. The agent gets penalized with a negative reward value for hitting the boundaries.

Regarding the reward values, a balance should be struck between the rewards for hitting the boundary, visiting a flower, and visiting the hive. Initial values were selected from an experienced guess and then tuned by trial and error for better performance. A normal movement produces 0 reward.

Going out of bounds gives a '−1' penalty. The flower's reward is a constant of 8 at all states until nectar state 6.

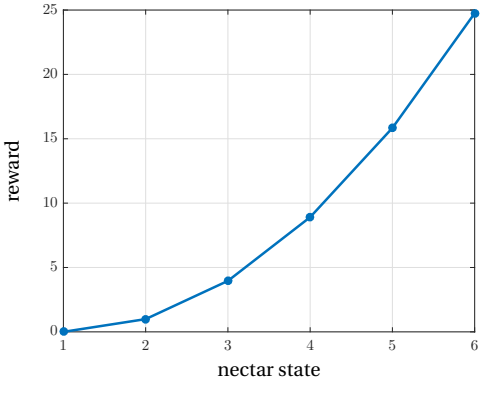


Figure 2.2: The reward at the hive location state is a second order function of the nectar state

Below is a description of the special states, their associated rewards, and the consequent automatic actions:

Hive The agent receives a reward defined by the function in Figure 2.2, the nectar state is returned to 1, and the agent is randomly relocated on the grid. Note: When the bee is full of nectar, there is no point to visit more flowers so there is no reward. Therefore, visiting the hive should be overwhelmingly more rewarding at nectar state 6 than any of the previous nectar states. Furthermore, the hive should progressively be more rewarding as the bee has more nectar to bring back.

Flower If the agent is in nectar state 1-5, it receives a reward of 8, increases the nectar state +1, and is randomly relocated on the grid. Random relocation is necessary, otherwise the agent would continually go to the same flower if in a greedy policy. If the nectar is full, then the reward is 0 (zero) and the next action is chosen by the current policy.

Out of bounds The agent is penalized by −1 and stays in the same state.

PARAMETER SELECTION

The discount factor, γ, is set to 0.9 to allow for quick propagation of the reward values through the states. The greater the value, the more future rewards will effect the current states value function update. This value was not varied for the simulation.

The step size parameter, $\alpha(s)$, in Eq. (2.10), is used to create an average value at each state. It decreases for a state each time it is visited so that $\alpha(s) = 1/N_s$, where N_s is the number of times that state has been visited.

POLICY SELECTION

When the agent is first introduced to the environment, it will have no knowledge of the surroundings so it must first explore. Once it has identified the locations of some of the reward states it can begin to exploit that knowledge. A scheduled ϵ-greedy policy allows for more exploration (low ϵ values) early in the learning stages and more exploitation (higher ϵ values) as the agent has gained more experience.

Figure 2.3 shows how ϵ is scheduled based on the change of the V-function (ΔV is calculated as the summed difference of all V-values after 100 iterations). The ϵ is stepped up after it meets the condition $\Delta V \leq 1e^{-3}$. At the switch in policy, the learning step size $\alpha(s) = 1/N_s$ is reset by reseting N_s to 1, since it is a new policy and will therefore converge to a new V^{π^*}. For example, an $\epsilon = 0.8$ policy starts converging anew, but with an initial V-function that resulted from an $\epsilon = 0.6$ policy. The final policy is stopped when an error condition of $\Delta V \leq 5e^{-4}$ is satisfied. This condition was chosen as a balance between accuracy and evaluation time.

The scheduled ϵ-greedy policy is used for the practical reason that although a fully greedy policy is the optimal policy, during the learning phase it prevents certain states from being explored, which often results in poor convergence. The scheduled approach is a good fit for the honeybee example in that it allows for more exploration in the beginning in order to find the flowers. As time goes by, it will want to exploit the locations where flowers are known to have nectar, and so a more greedy policy will result in more nectar retrieval which will be reflected in a higher valued V^{π^*}.

2.3.2. SIMULATION RESULTS

The simulation results show that V^{π} for a greedy policy is nearly converged to the optimal V^{π^*} as it satisfies the $\Delta V \leq 5e^{-4}$ condition. In Figure 2.4, the final value function and greedy-policy map is plotted in the left and right columns, respectively. In the rows of the figure, the grid world is shown for nectar states 1(top), 5(middle), and 6(bottom). In nectar state 1, the bee will get no reward for going to the hive so its greedy action is to go toward the nearest flower. In nectar state 6, the nectar is full and therefore the agent will only get a reward to go to the hive. Every nectar state in between shows a blend of the two. Memory state 5 is important in that it should still get nectar at the flowers, but if the agent is nearby the hive it is also a good action to go there.

Using the policy map with the arrows(right column), it is easy to see from any location where the agent will go given a greedy-policy. Just follow the arrows. In some cases, it seems like an indirect path is taken. However, it must be considered that each square in

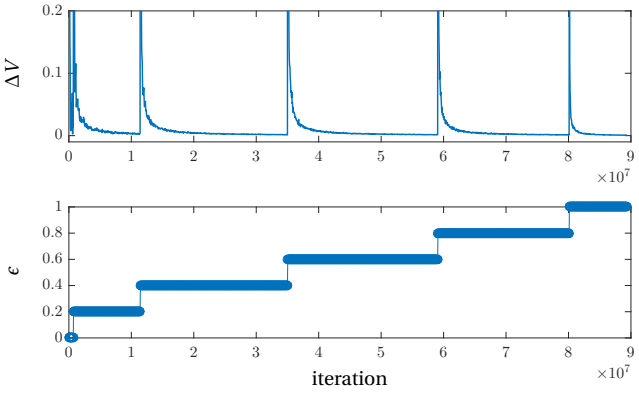


Figure 2.3: This scheduled ϵ-greedy policy increases ϵ as errors converge. The α value within the update equation (Eq. (2.10))is then reset, resulting in temporarily larger learning step sizes and therefore higher ΔV (errors).

the grid is considered one step. To move diagonally is indeed physically more distance, but the problem setup is not considering this as a longer path because the algorithm only considers number of states to the reward. In reality, there is more than one optimal path. The arrows only show one greedy direction, but there are often scenarios where adjacent states have a very similar value. If the value function were fully converged the values of the two or three contending states would be exactly the same. However, full convergence requires iterations $t \to \infty$, and will therefore always have some error. It is worth while to note that full value function convergence is not necessary to have the optimized policy. An optimal policy-map can result in much fewer iterations than is necessary to converge to V^{π}, and this will be seen in the flight test results in Section 2.4.4

2.4. FLIGHT TESTS

This section explains the setup and results of the flight tests with a real quadrotor performing the reinforcement learning task described in Section 2.3. First, the resources used in the experiment will briefly be explained. Then the setup of the experiment will detail how all these components interact together to recreate a real life version of the simulation. Lastly, the results from the flight tests will be presented and briefly discussed.

2.4.1. RESOURCES

Formerly, to work with a quadrotor platform, researchers often used in-house quadrotor designs which were often time intensive to model and tune [59, 107]. Since designing and building a quadrotor is not the main research intention for this thesis, it is very welcome to have a commercial quadrotor available which makes this research not only more convenient for this project but also more accessible to other researchers in the field. This benefit is the same for the other resources discussed; such as open source software. The uses and benefits of these resources in the context of this research will

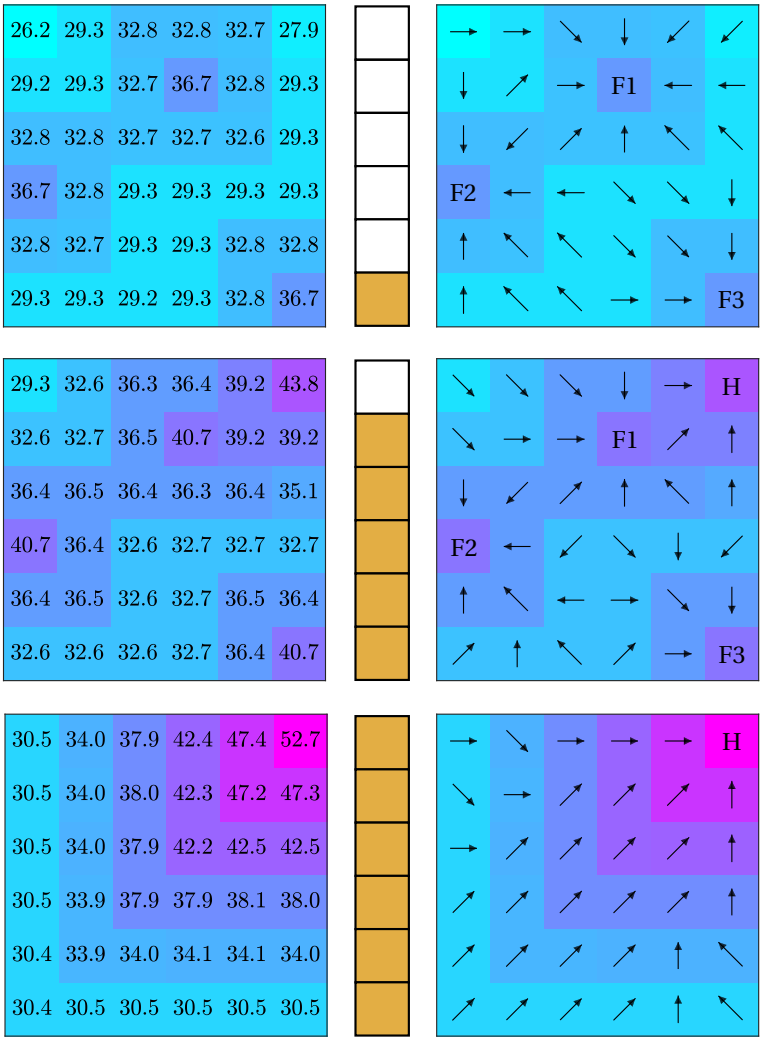


Figure 2.4: V^{π} and policy map at nectar states 1,5, and 6 for a scheduled policy that converges to a near optimal V^{*}.

now be discussed[3].

AR.DRONE 2.0 QUADROTOR

A quadrotor is an unmanned aerial vehicle consisting of 4 engines that spin independently of each other. Adjusting the thrust produced by each engine gives the necessary

[3]Note that the nature of these resources are that they are ever changing. They have changed over the duration of this dissertation project and will continue to evolve. See websites provided for the most current information.

control to perform general motion as well as other aggressive maneuvers [104]. The Parrot® AR.Drone 2.0 [4] as shown in Figure 2.5, is a commercially available quadrotor intended for recreational use. The MAV weighs about 750grams and features forward and bottom facing cameras, GPS, WiFi, and precision control. The AR.Drone 2.0 is also promising for research and educational purposes [70] as it is relatively inexpensive, and robust to damage. Most importantly for this project is that its built-in software can be overwritten.

Figure 2.5: The AR.drone 2.0 is a relatively inexpensive quadrotor MAV that can be purchased for recreational uses or for research. Picture credits by Parrot.com

TU DELFT CYBER ZOO

The TU Delft Cyber Zoo [5] was officially opened in March 2014 as a research and test laboratory for ground robots and aerial vehicles. Its space consists of a floor area that is 10×10 meters and extends 7 meters high. The space is contained inside an aluminum truss frame which holds nets to ensure the safety of people and surrounding equipment. Also supported on the truss structure are 24 cameras. These cameras are part of the *Optitrack* [6]*: Motive Tracker* optical tracking system. The quadrotor "wears" a special hull with reflective balls or stickers so that the cameras can track it. Within the Cyber Zoo boundaries, up to 32 rigid bodies can be tracked with high precision accuracy. The system can also be used as a simulated GPS. Therefore, as long as an experiment is not using any feedback from the Cyber Zoo, everything done indoors can be translated to an outdoor environment.

This system is not the first of its kind being used for quadrotors [78]. ETH Zurich has shown with their aerobatic quadrocopters how powerful a designated space with tracking can be.

This flight test will use the tracking system only for simulated GPS and for evaluating performance. Further capabilities including the high precision information from the cameras for feedback are available but not currently used. Therefore, this experiment is easy to move to an outdoor environment as long as GPS is available. The complete system as it is used is illustrated in Figure 2.6. The computer running the OptiTrack software has a wired connection to the laptop running Paparazzi and from there can broadcast the GPS signal via WiFi to the quadrotor.

[4] Parrot AR.Drone 2.0 http://ardrone2.parrot.com/
[5] TU Delft Robotics Institute. http://robotics.tudelft.nl
[6] Optitrack™. http://www.naturalpoint.com

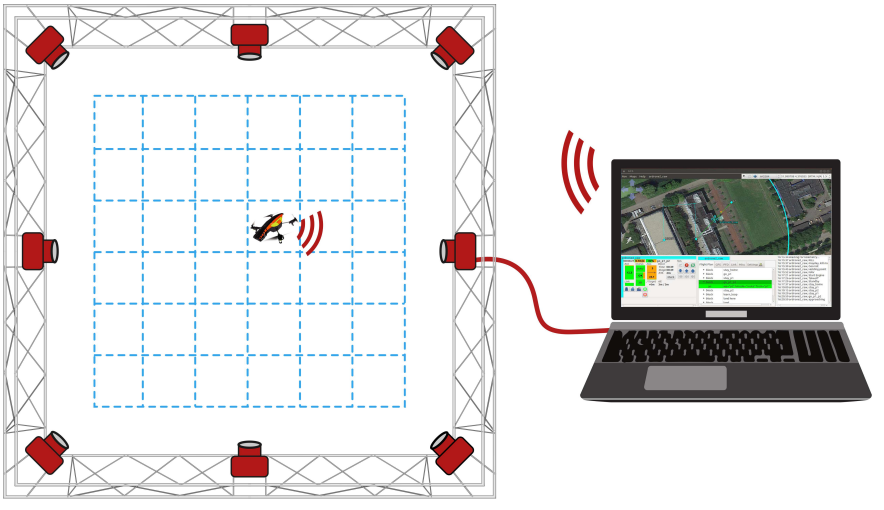


Figure 2.6: Cyber Zoo experimental setup: 24 cameras track the quadrotor. From this system, the quadrotor receives a simulated GPS signal. That position is transmitted via WiFi between a laptop and the quadrotor. Based on the location state and the nectar state, a waypoint command will be sent to the quadrotor for it to execute.

PAPARAZZI OPEN SOURCE AUTOPILOT

Paparazzi [7] is a fully open source free autopilot for fixed wing and multi-copter UAVs [109]. A version of the software can be acquired freely at GitHub [8].

The quadrotor capabilities in Paparazzi include a model of a standard quadrotor that is close enough to the AR.Drone to function as a good inner loop controller given good tuning and standard non-aggressive maneuvers. Simulation of a flight plan in an intuitive GUI is available in addition to use as a Ground Control Station (GCS) for real flights. An example of the Paparazzi GCS interface can be seen in Figure 2.7(*left*).

In the indoor setting, Paparazzi connects via WiFi to the quadrotor to give commands and via a wired connection to the Cyber Zoo computer to receive the position of the vehicle. In an outdoor situation, GPS would be picked up by the AR.Drone and used there directly.

Paparazzi is a modular platform and therefore has made it easy to add functionality by means of custom modules. Initializations, periodic and event functions can be added without effecting the main autopilot software. This makes customization very flexible and safe.

The high level position control of the quadrotor is controlled by the flight plan. Waypoints and commands involving these waypoints are written into the flight plan. The autopilot software and the flight plan is compiled and uploaded onto the quadrotor computer. Once uploaded, the names and numbers of the waypoints and types of commands cannot be changed; however, the location of the waypoints can be modified at any time. Therefore, it is important to plan the method for which the waypoints will be used. In

[7] Paparazzi Free Autopilot. http://wiki.paparazziuav.org
[8] Paparazzi GitHub account. https://github.com/paparazzi/paparazzi

this experiment, the method of use is shown in Figure 2.7(*right*). A new module was created to move the waypoints based on reinforcement learning logic.

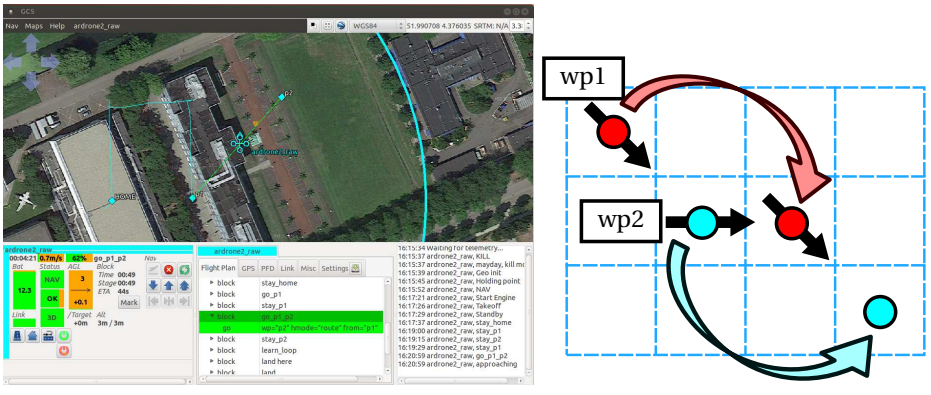


Figure 2.7: ***(left)*** Paparazzi GCS (Ground Control Station) is the main interface for the user. In the GUI it shows a 2D map with a flight path (cyan line). The quadrotor status is shown in strips box (bottom-left) along with some buttons for common commands. Commands such as "go p1" or "stay p2", can be seen in the Flight Plan tab in the Notbook box (bottom-middle). Commands that have been made are printed in the console box (bottom-right).
(right) The quadrotor is commanded by paparazzi to go to a waypoint. Waypoints cannot be added online but they can be moved. Therefore, after each action decision, the waypoint will be moved to the desired state. Commands alternate between going to waypoint 1 and waypoint 2 from the other.

2.4.2. VISION-BASED REWARD DETECTION

As part of the autonomy-focused effort, the rewards of the reinforcement learner are only granted to the agent when detected by the vision-based system.

Figure 2.9 demonstrates the vision of the agent. In this case, the flowers are represented by red paper on the floor. The bottom-facing camera of the AR drone scans the floor using a color filter module in Paparazzi to detect only colors within a range on the YUV space. The color filter algorithm works with the YUV images coming from the camera. The filter checks, pixel by pixel, if the three channels (Y represents the brightness and U+V the color) lie in the intervals defining the color range. Figure 2.8 demonstrates an example range interval of the color to detect. The actual values for the range were calibrated based on the paper color within the lighting of the Cyber Zoo. When the number of pixels of this particular color surpasses some threshold, a reward state is considered to be detected.

Figure 2.9 shows how all other colors are made gray scale, while the color within the specified YUV range, is highlighted. This particular image is the view from the agent's camera over a "flower". Since a pixel threshold was used, it was necessary to calibrate the threshold beforehand based on the height of flight and the size of the paper. Since computer vision techniques are not the main focus of this thesis, it was deemed sufficient to have a simple and reliable vision-based method for detecting the reward states.

One thing that is necessary to monitor with vision-based detection is the possibility of false negative and false positives. Computer vision is never fully reliable in that it may

see a reward where there is none (false positive), or fail to detect enough red pixels when it is actually in the flower state (false negative). This can happen for a number of reasons; for example if the quadrotor's pitch or roll makes the camera tilt anywhere other than straight down, if the filtered color shows up somewhere other than at the flowers, or if it flies too fast past the flower state and fails to detect it. After some testing and calibration, it turns out that the color filter method is very robust, resulting in no false positives and no false negatives in a 1000 iteration run.

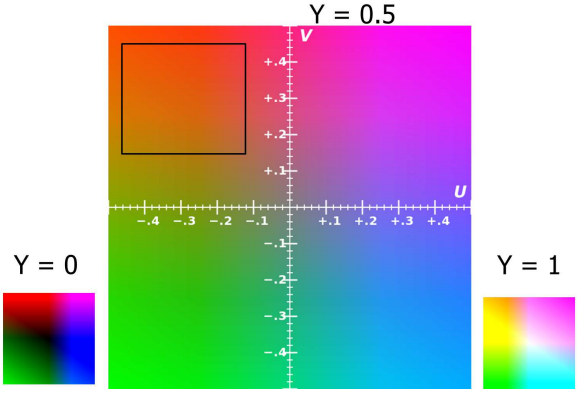


Figure 2.8: The color filter module in paparazzi isolates a range of colors in the YUV color space [137].

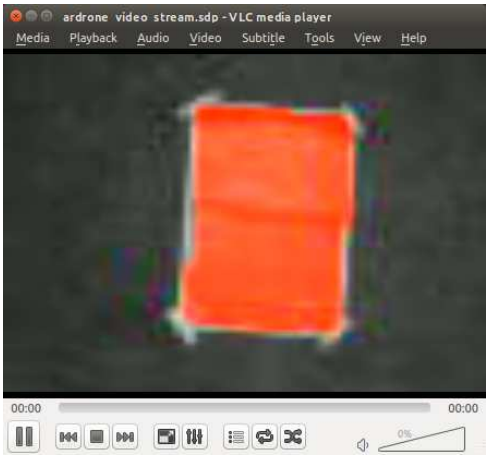


Figure 2.9: View of bottom-facing camera over a "flower", represented by red colored paper.

2.4.3. SYSTEM SETUP

The aforementioned components now come together into a comprehensive system setup. The controller consists of the Paparazzi autopilot with a reinforcement learning module. Once the software is uploaded to the quadrotor computer, the user can send commands

from the GCS flight plan via WiFi to the quadrotor. By calling the RL module function from the flight plan, the RL algorithm in the module chooses the next state based on the current position. The next state is translated to a waypoint position for Paparazzi to understand. If the quadrotor is currently at waypoint 1 (wp1), then wp2 will move to the next position. This concept of moving waypoints and then going to it is shown in Figure 2.7(*right*). Paparazzi's inner loop controller, which is already uploaded onboard, will translate the waypoint command to specific low level control commands.

Input and output to the quadrotor computer is possible by use of a usb flash drive that must be plugged into the AR-drone.

2.4.4. RESULTS

Two flight tests will now be explained and the results presented. Due to time and battery limitations, the quadrotor is not able to perform millions of iterations like is possible in simulation. The lifespan of a battery lasts about 200 iterations, or 10 minutes. However, an external wired power source is also available for longer runs.

The flight test is split into two separate tests that demonstrate the important reinforcement learning concepts in a smaller number of iterations.

TEST 1: EXPLORATION AND LEARNING

The first flight test demonstrates the exploration phase of reinforcement learning. To better visualize the results, this test was limited to a 6×6 grid of 36 states instead of the normal $6 \times 6 \times 6$ grid of 216 states. The nectar state was kept at 1, where there is no reward for visiting the hive.

The agent successfully moved to its randomly initialized spot on the grid within the Cyber Zoo and continued to randomly choose actions and execute them.

Figure 2.10(left column) shows the resulting value function after 100, 200, and 400 iterations from one flight. The right column shows the greedy-policy map and the red "x's" are states where the greedy policy is not the optimal policy.

The result from this test flight as seen in the Value function shows that after 100 iterations, several states, including one of the flowers, had not yet been updated. By 200 iterations, each of the flowers had been visited at least once, yielding a reward at that state. The reward propagates to some of the neighboring states. The boundary was hit several times as can be seen by the negative values in border states. With more iterations, more of the states have been visited and the value function better approximated.

Looking at the greedy policy map in Figure 2.10, we see that even though the value function is not even close to being converged,the optimal policy is being converged on. After 100 iterations, there are 8 states which have no information in neighboring states in which to guide a decision, and 6 states where the greedy policy is non-optimal. By iteration 200, there are only 2 states which would make a non-optimal greedy action; and by iteration 400, one of the optimal greedy policies has been found. This means, that if a greedy policy were implemented at this point, the agent would always find a direct path to the nearest reward spot.

In Figure 2.11, the value function convergence for Test 1 is shown over a run of 1000 iterations. On the Y-axis, ΔV is calculated as the summed difference of all V-values after every 10 iterations. The variability of the value function encompassed in ΔV decreases

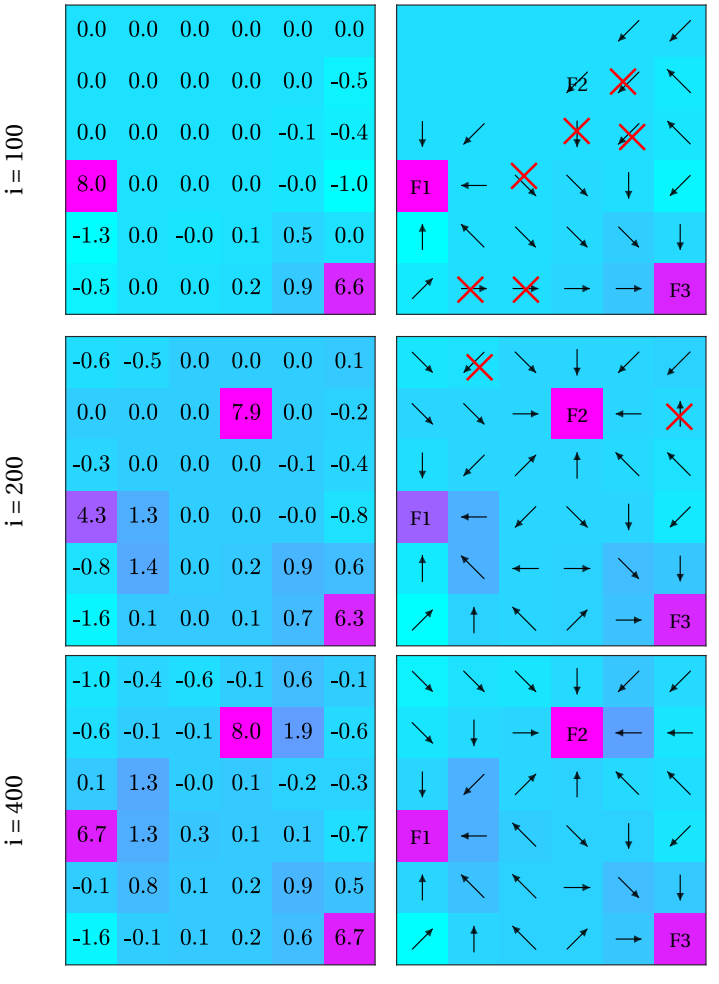


Figure 2.10: The V^{π} (left column) and policy map(right column) at nectar state 1 for a after training with random actions after 100, 200, and 400 iterations in actual flight.

after more experience, as expected.

TEST 2: EXECUTING A GREEDY POLICY FROM AN IMPORTED VALUE FUNCTION

The second flight test demonstrates the ability to upload an existing value function and follow a non-random policy determined by that value function. In this case, the quadrotor loaded a nearly-converged value function from $5e7$ iterations of ϵ-greedy simulation. Test flight 2 follows a fully greedy policy. The result is shown in Figure 2.12. There is nothing surprising with these results. It follows the greedy path as desired.

Figure 2.12 shows the grid world at each of the nectar states. In nectar state 1, it is randomly relocated in state [1 1] and it takes the most direct path to the nearest reward

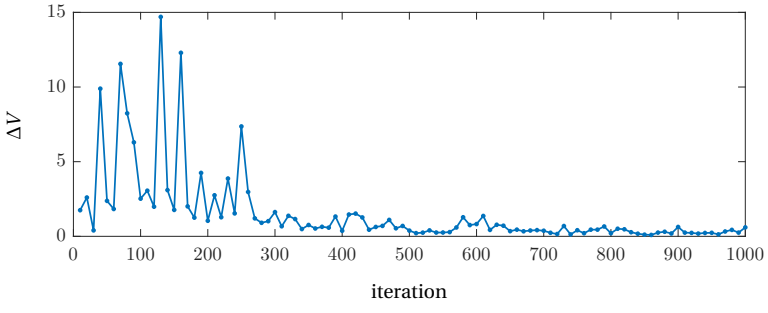


Figure 2.11: Convergence of V-function with random policy (one test run for 1000 iterations).

state. Each time it reaches a reward state it randomly relocates onto the grid in the next nectar state. The random relocation spot is shown by a solid red circle. The path it takes is shown with black arrows, and the final reward spot it navigates to is a dashed circle. Following the circles and arrows shows us the expected greedy behavior.

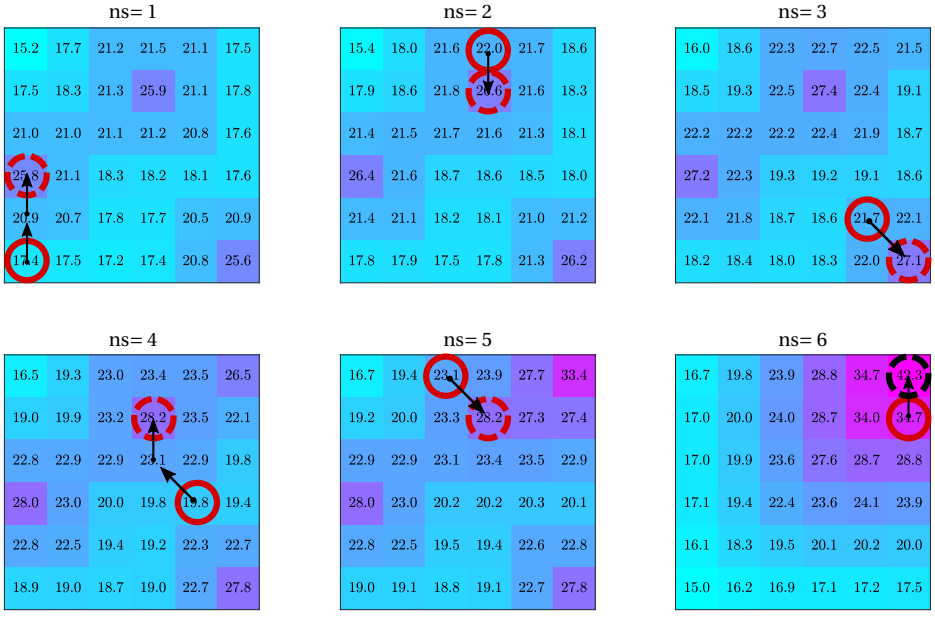


Figure 2.12: The imported value function at each nectar state and the actions chosen by the agent with a greedy policy. The solid red circle shows the location in that nectar state that the agent was randomly generated in. The arrows show the path the greedy policy the agent followed, and the dashed red circle is the reward sport it found before beginning at a random location in the next nectar state.

2.5. CONCLUSIONS

This chapter presents the first step toward greater autonomy in guidance decision making for a quadrotor task in an unknown environment. By learning from interactions with the environment, the reinforcement learning approach works towards more adaptive and robust decision making for autonomous MAVs.

A simulated honeybee task of collecting nectar and bringing it to the hive, allowed an agent to learn a near optimal value function using a scheduled ϵ-greedy policy. After convergence of the simulation, the agent chooses an optimal mission path through the grid world. Once the simulation was successful, a real-life quadrotor experiment was conducted using vision for reward state detection. The results show that the quadrotor can learn and implement a policy as well as the simulation but is limited in the number of iterations it can perform. However, since the optimal greedy policy is discovered long before the value function fully converges, it is therefore not necessary to converge the value function over millions of iterations to such a small error. This can make learning in-flight more feasible.

The flight tests show a hybrid approach where learning a value function is initiated in flight, converged over many iterations in simulation, and then is used again in a flight test. This is one approach to solve the conflict between limited battery life and the need for many iterations. The test flights also demonstrated the successful use of vision-based rewards. Vision therefore shows promise as a reliable tool to increase the autonomy of a real-life agent.

Not only does this chapter present a successful implementation of a reinforcement learning task, it also is a demonstration of an inexpensive and accessible means of research by using a commercially available quadrotor and an open source autopilot.

As the resources used for flight tests and the algorithmic approach in this chapter lay the foundation for the rest of this thesis, the limitations in state space size and the need for extensive training also act as a guide. Success within this limited example lends itself to more opportunities of autonomy such as further vision-based detection and identification of different types of rewards(Chapter 6) for more complex scenarios, or vision-based (relative) state representations(Chapter 4) which can be useful in expanding to larger, more realistic domains.

3

HIERARCHICAL REINFORCEMENT LEARNING FOR GUIDANCE IN LARGE FULLY OBSERVABLE STATE SPACES

In the previous chapter, a guidance task inspired by a honeybee was presented. The task was solved with temporal difference reinforcement learning (TDRL). The optimal solution was found, however, there are some limitations with this approach: 1) Full state knowledge is necessary, and 2) for the real-life experiment, learning until convergence is not possible because time and battery life does not allow for the many timesteps required. We desire to work within even larger environments where the curse of dimensionality makes the reinforcement learning approach prohibitive.

In this chapter, a hierarchical reinforcement learning (HRL) approach is used to address these issues. The approach is tested within a large, obstacle-rich gridworld maze. It is shown that learning can be sped up with a hierarchical approach by decomposing the problem into subtasks.

3.1. INTRODUCTION

Reinforcement learning is a machine learning approach which is promising for finding optimal decision strategies with little or no a priori knowledge of the environment. Using only interaction with the environment and a reward structure, an agent can learn desirable behavior. In Q-learning, this is done by assigning values to state-action pairs and continuously updating the values as they are visited. However, as the state space of the environment increases, so does the number of state-action pairs which must be visited in order to be updated to find a desirable policy. Therefore, scaling up can quickly make the problem intractable for a reinforcement learning solution since the time it would take to visit all state-action pairs would be too great for most real-world applications. The ultimate goal of this thesis is to use reinforcement learning techniques in real-world guidance applications, and therefore solutions to the problem of scalability must be explored.

The “curse of dimensionality", as it is commonly known in the reinforcement learning field, is a well researched problem with no one-size-fits-all solution. One approach for high-dimensional problems is to move to the continuous domain, which represents the real world more accurately. With this approach, a function approximation method such as radial basis functions or neural networks is used to approximate either the value-function and/or the policy [23, 38, 52]. However, research in the discrete domain remains popular because, for example, convergence guarantees proven in the discrete reinforcement learning do not always extend to the continuous counterpart, especially with nonlinear approximation functions. While keeping within the well-known and easy-to-implement discrete case, there are also several ways to battle the curse of dimensionality.

In this chapter, a hierarchical reinforcement learning (HRL) approach is used to address the reinforcement learning limitation associated with exploring large state spaces. The approach is tested within the large state space of the Parr’s maze: a 86×86 obstacle-rich gridworld developed as a benchmark to test problems related to large discrete environments. Learning can be sped up with a hierarchical approach by decomposing the problem into subtasks. That is, primitive action decisions (such as “go right" or “go left") are not made every timestep, but higher level decisions (such as “move forward until a wall is reached" or “turn 180°") are made with less frequency. The method *options* is one of the branches of HRL and was first developed by Sutton et al.[129].

The term *flat* Q-learning is used throughout this chapter and the following chapter to describe the traditional discrete non-hierarchical Q-learning with primitive actions.

The goals of this chapter are threefold:

1. To observe and quantify the benefit of the HRL options approach against the flat Q-learning approach in different sized environments; including a benchmark problem which has not been used before for this specific methodology.

2. To compare how different user-defined optionsets can affect the benefit of the HRL options approach and therefore make conclusions about how to design effective optionsets in the first place.

3. To be used as a stepping block for the state-abstraction HRL chapter, Chapter 4.

In the next section, the necessary background into hierarchical reinforcement learning is introduced. In Section 3.3, the setup for the experiments will be described, including the simulation environment and the chosen parameters. In Section 3.4 the results will be presented and observations made. Finally, Section 3.5 will conclude the chapter with a discussion of the results and a look into the next chapter.

3.2. BACKGROUND

A number of works lay a foundation for the experiments performed in this chapter. It is assumed that the reader is familiar with Markov Decision Processes (MDPs) and Temporal Difference reinforcement learning from Chapter 2.2 or from another source (eg. [130]). The remaining prerequisites will now be presented; beginning with an introduction to Q-learning and semi-Markov Decision Processes.

3.2.1. Q-LEARNING: STATE-ACTION VALUE

In Chapter 2, temporal difference reinforcement learning was introduced using V^π and its corresponding update laws to find a solution for a honeybee task. MDPs can also be extended from a state defined function such as $V(s)$ where values for each state is learned, to a state-action defined function $Q(s,a)$, where values for each state-action pairs are learned. The Q-function is beneficial because it does not require access to the one-step action models ($\mathcal{R}_s^a$ and $\mathcal{P}_{ss'}^a$) to follow a greedy policy as a V-function policy does [130].

There are 2 popular approaches for updating state-action value functions Q^π; namely, SARSA(Eq. (3.1)) and Q-learning(Eq. (3.2)).

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right] \tag{3.1}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_a Q(s',a) - Q(s,a) \right] \tag{3.2}$$

SARSA is an *on-policy* update method as it uses the actual states and actions taken at timesteps t and $t+1$ for the update of $Q(s,a)$. Q-learning is off-policy since it uses the max argument over all possible actions from state s for the update. In this chapter we use Q-learning and extend the ideas of "flat" Q-learning to a hierarchical approach described later in this section.

3.2.2. SEMI-MDPS

The semi Markov Decision Process (semi-MDP) maintains much of the same structure as an MDP, however it expands to a variable timestep. In this way, semi-MDPs work as a framework to practice reinforcement learning with *temporal abstraction*, allowing the agent to take temporally extended series of actions [13, 37, 100, 106, 129].

Let, τ denote the (positive) variable time it takes to take action a from state s.

While an MDP requires the Markov property "historylessness", a semi-MDP can incorporate actions which take $\tau > 1$ number of timesteps and therefore within that action does not satisfy the Markov property. The state transition probability can be stated as $\mathcal{P}(s',\tau|s,a)$ and the expect reward as $\mathcal{R}(s',\tau|s,a)$ [37]. The value function from the flat case as in Eq. (3.2) can therefore be rewritten as:

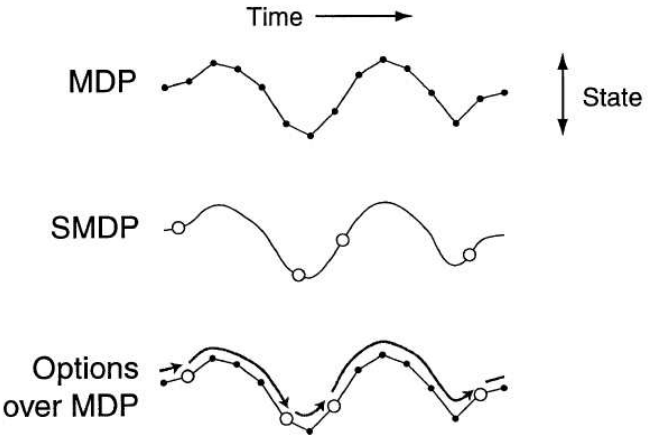


Figure 3.1: Visualization of MDP vs semi-MDP. An MDP chooses a primitive action at every iteration, while an semi-MDP can choose an option which executes several primitive actions before needing to make a new choice. Source: Sutton et. al[129].

$$V^{\mu}(s) = \sum_{o} \mu(s,o) \left[\mathcal{R}_s^o + \sum_{s',\tau} \mathcal{P}_{ss'}^o V^{\mu}(s') \right] \tag{3.3}$$

where μ denotes the policy and o is a temporally extended action (*option*) where $o \in \mathcal{O}$.

For state-action pairings, the expected value is:

$$Q^{\mu}(s,o) = \mathcal{R}_s^o + \sum_{s',\tau} \mathcal{P}_{ss'}^o \sum_{o} \mu(s',o') Q^{\mu}(s',o') \tag{3.4}$$

The benefit of semi-MDPs is its ability for *temporal abstraction*; meaning that it models temporally-extended courses of action [129]. This is a requirement for many of the various HRL approaches. Figure 3.1 (source: [129]), visualizes the concept of the semi-MDP (SMDP) and specifically *options*. The same actions are ultimately being executed, however each decision made in the MDP results in one action where one timestep is taken before a new decision must be made. Each decision made in the semi-MDP accounts for several primitive actions over an extended course of time.

3.2.3. APPROACHES TO HIERARCHICAL RL

The heart of hierarchical RL is in the semi-MDP structure and accompanying concept of temporal abstraction. Though several researchers have independently developed various formalizations using the HRL concept, the most popular ones all use temporal abstraction (see Section 3.2.2). The differences of these approaches lie in the hierarchical structure of the building blocks and the movement between them.

There are three main approaches which have gained attention in the HRL research community as described by Barto et. al [13]: MAXQ, Hierarchies of Abstract Machines(HAMs)

and *options*. In this research it was ultimately decided to follow the *options* structure. Therefore brief descriptions of MAXQ and HAMs will now be given along with a more in-depth description of *options* and the literature which has extended on that framework.

It should be noted for clarity that the following overviews of the methods will use the terminology native to the method. On a basic conceptual level, the “machines" of HAMs, the “subtasks" of MAXQ, and the “options" of *options*, are all the same building blocks of the hierarchy and in many ways fulfill the same general function with slightly differing functionality.

HIERARCHIES OF ABSTRACT MACHINES

Hierarchies of Abstract Machines(HAMs) [101], was first introduced by Parr [100] and Parr and Russell [101]. Their study resulted in “significant speedup in decision-making and learning" for a maze environment.

The HAMs approach is a two layer hierarchy with the core MDP on the top layer and on the second layer a collection of stochastic finite-state machines, denoted $\{H_i,\ H_j,\ H_k, ...\}$. Each abstract machine “can be viewed as a constraint on policies". That is, when operating within a machine, only a subset of all the possible actions are allowed. Constraining the policy-space in a meaningful way is made intuitive by the “HAM language". Abstract Machines can then represent any level of behavior using the HAM language, such as, “only move south and east" or “follow the wall". By using this language associated with the machine, a user can see the end solution and verbally explain the optimal behavior for solving the core MDP, $\mathcal{M}$.

A machine is defined by $H_i = \langle M_i,\ I_i,\ \delta_i \rangle$, where M_i is the MDP framework of the machine, I_i is a stochastic function which sets the initial state of M_i, $I_i(s) = S_i$, and δ_i is the stochastic transition function. The machine can be in one of 4 possible states: action, call, choice, or stop. As paraphrased from Parr and Russell [101]: *Action* states, execute an action within the environment; *call* states execute another HAM (ie. H_j) as a subroutine; *choice* states nondeterministically select a next machine state; and *stop* states halt execution of the machine and return control to the previous call state.

While HAMs *can* function the same as *options*, the benefit of the formalization of HAMs over *options* is additional functionality. Each HAM is its own separately defined MDP and can also call another HAM as a subroutine, making a lateral move in the hierarchy, instead of always having to move up in the hierarchy to a parent decision maker. Therefore, HAMs is more customizable than options. However, this added functionality also adds further complexity which translates to further complexity in design and implementation. Furthermore, since HAMs keep track of the state within the HAMs MDP as well as the core MDP, the state-action space is larger. Q-learning updates will update the function $Q([s,m],a)$, where a is the action taken, s is the state of the core MDP, and m is the state of the current machine [13].

MAXQ VALUE FUNCTION DECOMPOSITION

MAXQ Value Function Decomposition [37], or simply MAXQ, is an HRL approach using hierarchical decomposition of tasks. According creator T. Dietterich, MAXQ is an approach “ based on decomposing the target MDP into a hierarchy of smaller MDPs and decomposing the value function of the target MDP into an additive combination of the value functions of the smaller MDPs."

As opposed to *options* or HAMs, this approach is true task decomposition instead of just reducing the problem to one semi-MDP. MAXQ can be designed with as many layers of hierarchy as the designer wants. The overall value function for a policy consists of a collection of value functions of the decomposed subtasks.

The MAXQ subtask is defined as $M_i = \langle T_i,\ A_i,\ R_i \rangle$, where T_i is the termination predicate which determines when the subgoal has been reached and the subtask can be exited; A_i is the set of actions for the subtask, where the actions can be primitive actions or executing a child subtask; and $R_i(s')$ is the pseudo-reward function, a user defined reward structure local to the subtask.

The advantages seen for MAXQ is that both state and temporal abstractions can be realized within the subtasks which can lead to much faster learning by effectively spending less time in the state space which is unimportant to the task at hand. Furthermore, when subpolicies for the subtasks are learned, that knowledge can be easily transfered to a different core task which uses the same subtask. However, this method is also a source of suboptimality called recursive optimality; where the optimal solution for the core MDP results in suboptimal solutions in the children subtasks. Furthermore, this approach is structurally complex in that the design is multi-layered and meaningful subtasks with corresponding subgoals must be selected.

OPTIONS

Hierarchical reinforcement learning with options was introduced by Sutton et. al [129]. The hierarchy consists of 2 layers where the top layer is the core MDP and the second layer consists of the *options*. The core MDP is defined as in Section 2.2.1, with the tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$. The options method follows the same concept as flat reinforcement learning except that instead of selecting single timestep actions, an option is chosen which takes a variable amount of time. The formalization of options is closer to normal reinforcement learning than either HAMs or MAXQ and is therefore easier to implement.

Each option is defined by $o_i = \langle I_i,\ \pi_i,\ \beta_i \rangle$, where I_i is the input set $I \subseteq \mathcal{S}$ from which states the option is allowed to be initiated; π is the policy of the options; and β is the termination predicate which determines under what conditions the option terminates. An example of an option in simple language could be:

I_i: Option o_i may be initiated from any state within the NorthEast quadrant of the maze

π_i: Go forward 3 steps

β_i: Terminate when the 3 steps are finished or a wall is hit

The whole option structure can also be simplified to a flat scenario where there exists an option to represents each single step *primitive action* in the core MDP action set, $\mathcal{A}$, and each can be initiated in any state within $\mathcal{S}$, and is always terminated after 1 timestep.

Extending the formalization of flat reinforcement learning to options is simple. Where the Markov policy over actions was formerly denoted, π in order to update the value or Q function $Q^{\pi}(s,a)$, the semi-Markov policy over options, denoted μ, is used instead. Following from the flat Q-learning update and the new formalization from the semi-MDP bellman equation in Eq. (3.2), the Q-learning over options update is:

$$Q(s,o) \leftarrow Q(s,o) + \alpha \left[r + \gamma \max_{o \in \mathcal{O}_{s'}} Q(s',o) - Q(s,o) \right] \tag{3.5}$$

Many works have furthered the concept of options, including a look into automatic identification of subgoals or optionsets [71, 118, 124, 131, 138]. Applications of options are most notably the robo soccer keepaway task studied by Stone and Sutton [125, 126].

Options was selected as the base method for this chapter's work because it is the closest to the basic principles of flat reinforcement learning and also has the largest mass of successfully implemented follow-up research. With the end goal of implementation on a quadrotor task, simplicity and transparency of the method is the safest route.

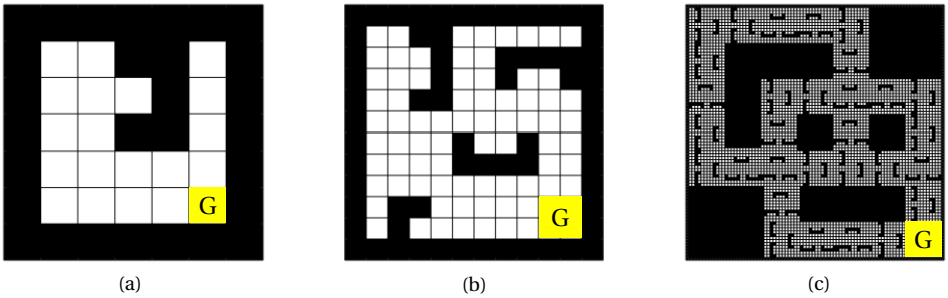
3.3. EXPERIMENTAL SETUP

The experiments in this chapter make comparisons between multiple state-space sizes, optionsets, and a few training policy schemes. These configurations will now be explained along with selected parameters and other details of the experimental setup.

3.3.1. MAZE ENVIRONMENTS

Three different mazes were used to gather results. The "Parr's maze" was chosen because it has previously been used as a benchmark for HRL problems [100] and is an interesting and challengingly large environment. The small and medium mazes were created for this study for many reasons. The smaller mazes are useful for visualizing the results which help to understand how the methods work. Furthermore, with 3 different sized mazes, patterns in results can be observed with regards to the state-space size effect on learning rate and convergence rate. Finally, smaller mazes were used in order to analyze the methods faster, since the Parr's maze is large and requires a larger computational toll.

In Figure 3.2, the mazes are shown. The grid states which the agent can move through freely are white, and the walls and obstacles are black. The goal states are in yellow labeled with a "G" for goal. The small and medium sized mazes were created for this study and therefore were given a similar structure as the Parr's maze in order to be a stepping-block for the larger sized state-space. The obstacles are similar in shape to the Parr's maze, and the location and proportion of the goal states are similar.



(a) (b) (c)

Figure 3.2: test mazes: small, medium, and Parr's (large) mazes

Table 3.1: maze state space and state/action pairing sizes

	Maze sizes		
	Small	Medium	Parr's (Large)
# valid grid states	19	75	3690
# states (total)	76	300	14760
# state/action pairings: flat (3 actions)	228	900	44280
# state/option pairings: 4 options	304	1200	59040

3.3.2. STATE REPRESENTATION AND AGENT MOVEMENT

The agent which moves through the maze, occupies one grid state which is defined by its $[x, y]$ coordinate. Within that grid state, it can have one of four headings: North, East, South, or West (NESW). Therefore, the full state of the agent consists of the grid state and the heading state. The agent cannot be inside a wall or obstacle, therefore, the number of valid states is not merely the height × width of the maze. The state-space size for each maze is shown in Table 3.1. Valid grid states is the number of white grid states which the agent is allowed to enter. The total number of states is equal to '# grid states ×4 headings'.

In flat Q-learning, the agent can choose from one of three primitive actions: 1) turn left, 2) turn right, or 3) go forward one step. Q-learning updates a table for state/action pairs. Therefore, the number of state/action pairs is also shown in Table 3.1. With HRL options, there can be any number of options in the optionset. As an example, an optionset with 4 options is shown in the table. Note that the # of state/action pairings or the # of state/option pairings is the size of the Q-function to be stored and updated. The table gives some intuition into how quickly the state-space increases when the maze gets bigger or when additional state information is added or if more actions are available. It becomes more computationally taxing to visit each state/action pairing enough times to converge to a reliable Q-function.

3.3.3. OPTIONSET CONFIGURATIONS

In Table 3.2, the configurations of the different optionsets are described. The flat Q-learning configuration consists of 3 possible actions. These are the primitive actions: turn left, turn right, and go forward 1 step. The action always takes 1 discrete timestep, t, and then terminates.

The optionset 1 series, includes the primitive actions and then adds another option to go forward for $\tau = 2$ timesteps (optionset 1a), $\tau = 3$ timesteps (optionset 1b) and for optionset 1c, the last option will direct the agent forward until a wall or obstacle is 2 or less gridblocks in front of it. This option takes a variable time duration.

The optionset 2 series uses a different approach where the primitive actions are not part of the optionset. Instead, forward movement is encouraged by forcing a step forward after each turn. In order for there still to be a chance to find an optimal path to the goal, there is also an option for simply going forward 1 step and turning 180°. Optionsets 2b and 2b follow the same logic as 1b and 1c, in that there are options for going forward for a temporally extended time.

While optionsets 1a, 1b, 2a, and 2b all terminate after a pre-defined number of timesteps,

Table 3.2: Configurations of flat Q-learning and HRL Optionsets.

flat	Option sets					
(primitive)	1a	1b	1c	2a	2b	2c
turn left	left	left	left	[left + fwd]	[left + fwd]	[left + fwd]
turn right	right	right	right	[right + fwd]	[right + fwd]	[right + fwd]
fwd×1	fwd×1	fwd×1	fwd×1	fwd×1	fwd×1	fwd×1
-	fwd×2	fwd×3	fwd until*	180^o turn	180^o turn	180^o turn
-	-	-	-	-	fwd×3	fwd until*

τ, the *termination predicate* for optionset 1c and 2c, relies on extra information about the environment. It assumes that a sensor is available to see how close in front an obstacle is.

It is important to note that a fully converged flat Q-table, can always find the optimal solution since it consists of only the primitive actions. It is, however, possible that an optionset could be designed such that the optimal solution is impossible or difficult to find. The optionset configurations tested in this study were selected such that the optimal solution is a viable solution.

3.3.4. TRAINING AND EVALUATION

A training epoch starts with the placement of the agent in a random state in the maze. The epoch ends when the agent has reached the goal and updated the Q-function. The training run follows a random or ϵ-greedy policy, where the ϵ is either kept constant throughout all the training epochs, as in Figure 3.3(a) or the ϵ is scheduled as in Figure 3.3(b) to allow for sufficient random action exploration in the beginning and then ramp up to take more greedy action later in the training. All mazes, despite the size were trained for 200 epochs. The small and medium mazes were trained using the ramp, since the early epochs of random exploration were enough to visit most of the states. However, further random exploration is needed for the larger Parr's maze, and therefore a constant ϵ is used. A short study for the flat case is performed to find the best training policy.

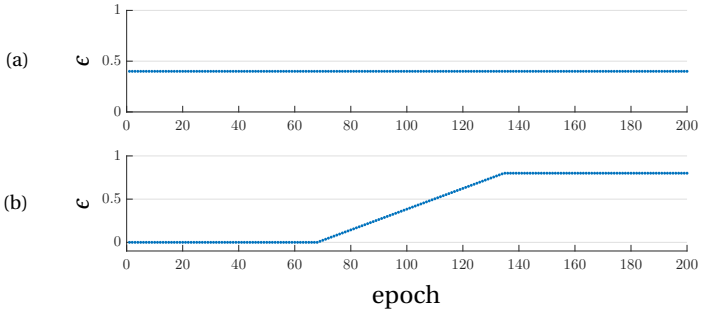


Figure 3.3: ϵ-greedy training schemes. (a) constant ϵ for training, (b) scheduled ϵ for training.

Table 3.3: Reward structure

reward type	small/med maze	Parr's maze
timestep	−1	−1
hit wall	−5	−10
goal	10	100

Every 5 epochs an **evaluation run** is performed and recorded. The evaluation run starts each time at the furthest (North-West)corner of the maze, facing south. The run uses a fully greedy policy and the most recent Q-function. The Q-function is updated temporarily within the evaluation run to avoid endless loops. The maximum number of timesteps allowed is 200, 400, and 100000 for the small, medium, and Parr's maze, respectively.

The whole training and evaluation is a stochastic process and consequently differs widely by run. Therefore, the whole event is run 100 times for statistical significance. A study into into the number of statistical samples can be found in Appendix A.

3.3.5. PARAMETER SELECTION

The parameters for Q-learning updates were selected from experience. The stepsize, α is held constant at 0.9. The discount rate, γ is also 0.9. The reward structure will now be described.

REWARD STRUCTURE

The reward structure was set in place to drive the agent toward the goal and to penalize hitting the wall to discourage this sort of damaging behavior. A penalty of -1 each timestep is implemented to encourage use of the fastest route to the goal. The reward structure was designed based on experience and did not require fine-tuning. The small and medium sized mazes required different values than the Parr's maze. Since the Parr's maze is so much larger, it needs a larger reward to more quickly propagate out to the far-away states. With a larger positive reward comes the possibility that a wall hit will, by random chance, land on a high-valued state, and therefore the wall hit penalty was also increased, to reinforce the discouraging effect for wall hits. See Table 3.3 for a summary.

Within the option, forward movement is not given a timestep penalty in the case that it is a $\tau > 1$ option. However, every option which does not reach the goal is penalized by at least -1. This rule supports the selection of options which takes multiple steps forward and therefore promotes movement about the gridworld, especially during the exploration phase when the environment is still very unknown.

3.4. RESULTS

This section first shows the results from the small and medium sized mazes which were used as a proof of concept before graduating to the Parr's maze where substantial computational time is needed to perform the statistical analysis. After the analysis is displayed and analyzed for the small and medium maze environments, the Parr's maze re-

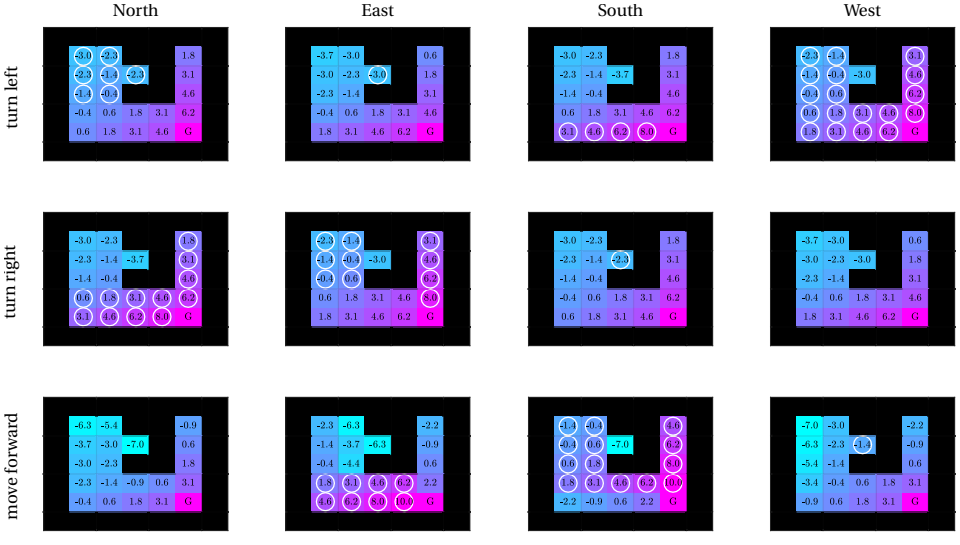


Figure 3.4: Converged Q-table for the small maze after 200 epochs - flat Q-learning using 3 primitive actions.

sults are demonstrated and compared to the smaller mazes. A comparison between state space size and HRL options performance against the flat Q-learning case is then made.

Table 3.4: optimal number of steps to goal during evaluation run

Small	Medium	Parr's (Large)
9	18	162

3.4.1. SMALL AND MEDIUM SIZED MAZES

The utility of the small and medium sized mazes was explained in Section 3.3. Since easily comprehended visualizations and extensive analysis are more accessible with a smaller state space, the results for the small and medium results will be compared and discussed first.

For 200 epochs, each of the mazes is explored by the agent. Figures 3.4 and 3.5 are examples of the end result for the flat Q-learning configuration and the Optionset 2a configuration, respectively. At the end of the 200 epochs in the small maze, the Q-table is converged and is demonstrated showing the state-action Q-values in the location on the maze, and the heading described as North, East, South, or West on the top of the column. The action is made known at the left of the row. With this visualization, the Q-value of every state-action pair can be seen to an accuracy of one decimal point. The action with the highest value in that state is circled and would therefore be chosen in a greedy policy scenario. In each case demonstrated in the figures, the Q-values have converged to an optimal solution. The solution can be said to be optimal if the agent with a fully greedy policy would arrive to the goal in minimum number of steps. Some of the states have

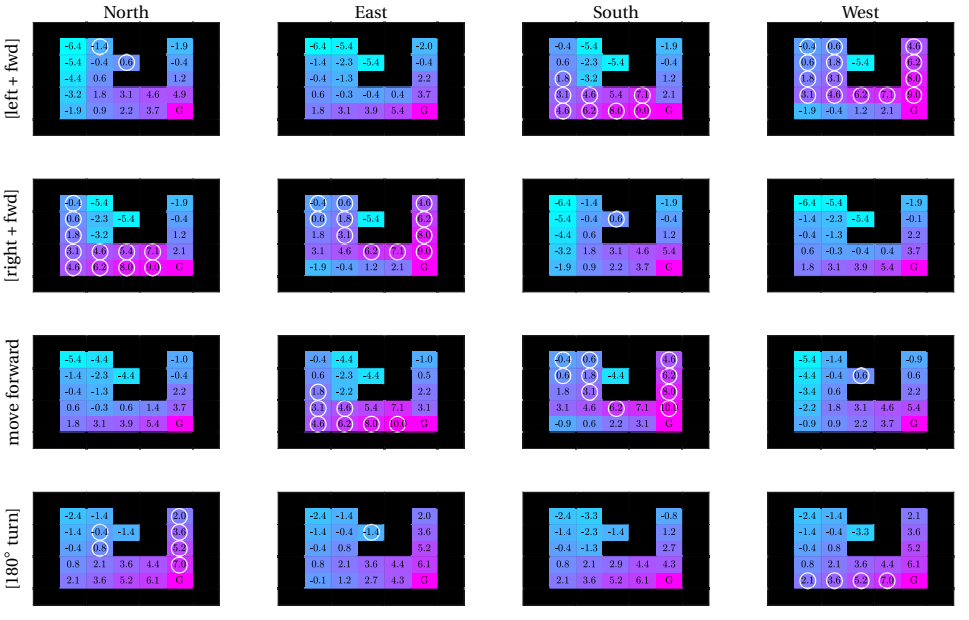


Figure 3.5: Converged Q-table for the small maze after 200 epochs - HRL with *options* using option set 2a.

more than one optimal actions and therefore the Q-values are very similar, but only one of the actions is circled. Besides the values printed in the grid square, the color of the square also denotes the value with warmer purple colors are higher value and the cooler blue color is lower values. The same colorbar can also be seen in Figure 3.6.

The full Q-table is difficult to visualize with larger state spaces; therefore, a more compact representation is shown in Figure 3.6. This result from a flat Q-learning run shows the “Value" of that location state averaged over all the heading states and possible actions; therefore, signifying how generally “good" it is to be in that location over all headings(NESW) and actions(based on policy, π). An arrow is displayed if, given that location and that heading state, the greedy action would choose to move forward. In the results of the small and medium mazes, we can see in the figures that the optimal forward-moving policy has been found. With this visualization of Q, full optimality cannot be concluded since there is no way of showing, for example, that an optimal left turn is taken instead of turning right 3 times. Therefore, an evaluation method is needed to demonstrate Q-function effectiveness and compare the quality of the learned Q-tables from different configurations.

Given enough time and enough random exploration, the flat Q-learning will always find the optimal solution[cite]. Logically,that can also be said for HRL options if all the primitive actions are included in the optionset. To see if there is a benefit in convergence speed to the optimal path to the goal, an evaluation run was performed every 5 epochs to track the performance. An evaluation run counts how many iterations it takes to reach the goal from the corner furthest from the goal (top, left square facing south). The results from the evaluation runs are shown in Figure 3.7 for the small maze, and

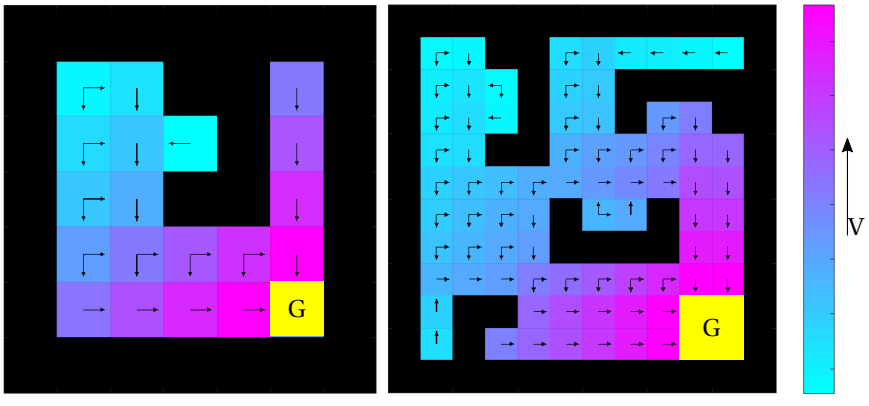


Figure 3.6: Local value-function and foward policy of the small and medium mazes. The “Local Value-Function" represents the Q-table values summed over all actions and headings in that locale, where the warmer colors are more desirable states to be in. The arrows indicate “forward policy" i.e. if the greedy action is to go forward in that state with that heading.

in Figure 3.8 for the medium maze. These figures plot the average number of steps to reach the goal during an evaluation run which always starts at the same initial state. To find the average and standard deviation of the data, the simulation was run 100 times. For clarity, the standard deviation errorbars are omitted from these figures, but can be found in Appendix A.3. Note that in discussion of these plots, results will be discussed using the average values. It is, however, duly noted that the standard deviations are often large and small differences in the performance between the optionsets may not be completely accurate and will certainly vary by individual run. However, with 100 statistical sample runs, the trends are valid so it is fair to make conclusions with the averaged values as plotted. A study to defend the number of statistical sample size was conducted and is displayed in Appendix A.2.

In Figure 3.7 we see that the agent in the small maze is quick to find the optimal path (9 steps) regardless of the optionset or flat Q-learning. The state space is small and it is easy to find the goal. Some of the optionsets are more inclined toward exploration which means that even with the initial zero Q-table at epoch 0 it is, on average, faster at finding the goal. Optionsets which go forward more often are, on average, quicker to find the goal when there is no Q information. This is shown at epoch 0 in Figure 3.7. Even so, some of the sample runs do not find the optimal path. Optionset 2a, in 100 sample runs doesn't always find the optimal solution in 200 epochs. The average performance is around 11 steps. The other optionsets find the optimal path in 100% of the sample runs within 70 epochs. For the small maze, the flat case finds the optimal path 100% of the time by epoch 20; faster than any of the optionsets. However, up until epoch 10, all the optionsets besides 2a, are performing better on average than the flat case, and it is not until epoch 15 that the flat case “catches up".

The medium maze is bigger, and therefore takes more steps to explore the environment. The results are shown in Figure 3.8 in the same manner as the small maze. In Zoomview-A we see the benefit of the HRL options; most of the optionsets perform bet-

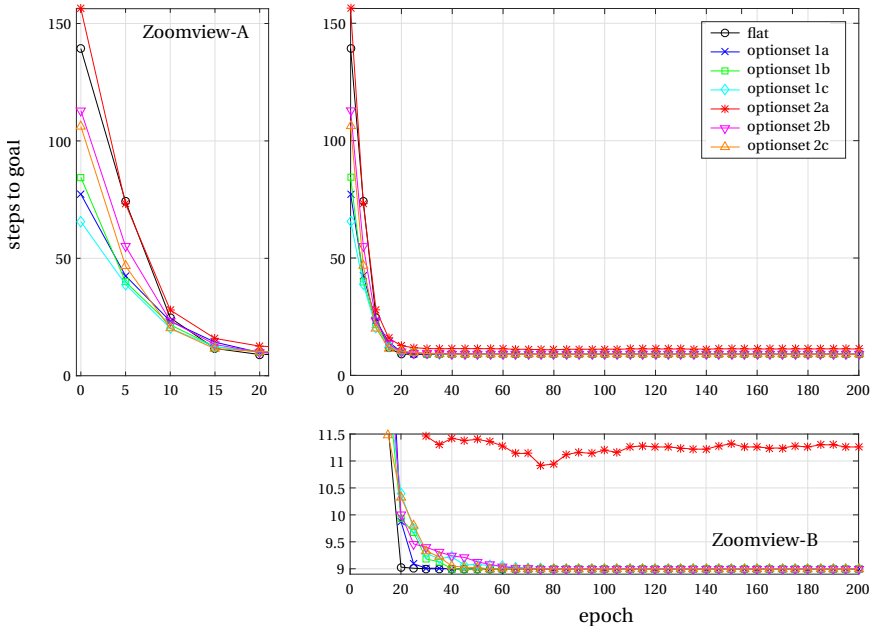


Figure 3.7: Evaluation results for small maze.

ter than the flat case from the start and continue to perform better until around epoch 35 when the flat case starts to outperform half of the optionsets. By epoch 45, the flat case is outperforming all the HRL cases as can be seen in Zoomview-B. The flat case converges first to the minimum number of steps (18), optionset 1a converges to the optimal solution (100% of the time) by epoch 180, and the other optionsets converge to a suboptimal solution, on average.

With the small and medium mazes it is shown in two separate environments that the benefits of HRL is faster learning in the early exploratory epochs, but the weakness as compared to the flat case, is that options may find suboptimal solutions. The larger the environment, the more epochs that the HRL options holds onto its benefit.

There are 6 different optionsets used in this comparison. For the medium maze, options 1c and 2c which have the option to move forward until a wall is in sight 2 squares in front, perform best in the earliest epochs but also settle more often on suboptimal solutions. Optionsets 1b and 2b which each have an option to go forward 3 times, perform about the same in the middle of the other configurations. Optionset 2a performs the worst. This could be because it has the least proportion of forward moving actions. Which means that with a random policy, it is often choosing to turn which leads to slow progress towards the goal. When another option is added to increase forward motion (optionset 2b), the performance already improves. The optionset which acts the most like the flat case is optionset 1a. This is because the options consists of the primary actions of the flat case plus one option which goes forward twice. Because of this additional

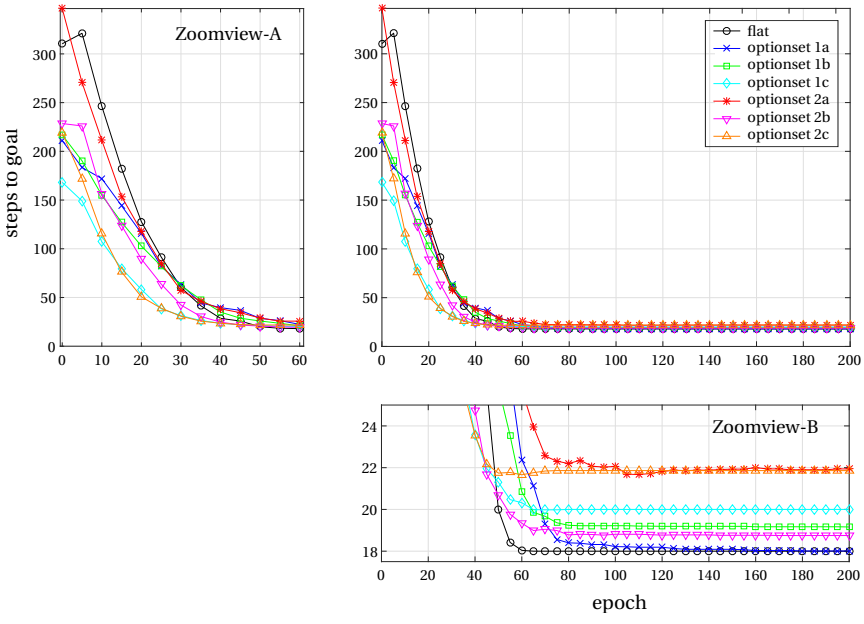


Figure 3.8: Evaluation results for medium maze.

option, it is better performing in the early epochs, but learns the optimal solution slower than the flat case.

Optionsets 1a-c (in cool colors), all contain the primitive actions as part of the optionset. Therefore, it is surprising that in the medium maze that these optionsets do not eventually find the optimal path in the evaluation. This can be because of the training schedule. The training schedule starts with completely random actions for the first third epochs, then ramps up the ϵ of the ϵ-greedy action policy, and settles for the last third of the training epochs on $\epsilon = 0.8$. Flaws in the Q-table can occur due to unequal visitation of all the state-action pairs. If the training is not random enough, the flaws may not be fixed in a timely manner because certain actions are being selected much more often and the better options are not getting the chance to update. This problem is exacerbated when the state space gets larger unless more random training epochs are added to account for size increase. In the case that time is restrictive, a quickly-learned, suboptimal solution may be preferable to a slowly-learned optimal solution. This comes into play much more with extremely large environments such as the Parr's maze, for which the results will now be discussed.

3.4.2. PARR'S MAZE

In Figure 3.9, after 200 epochs the Q-table is not fully converged but a general trend is clearly seen. The gradient of the colormap shows that the local value-function is higher at states near the goal and lower further away from the goal. Furthermore, local states near walls and obstacles are valued less than interior states. This is due to the penalized

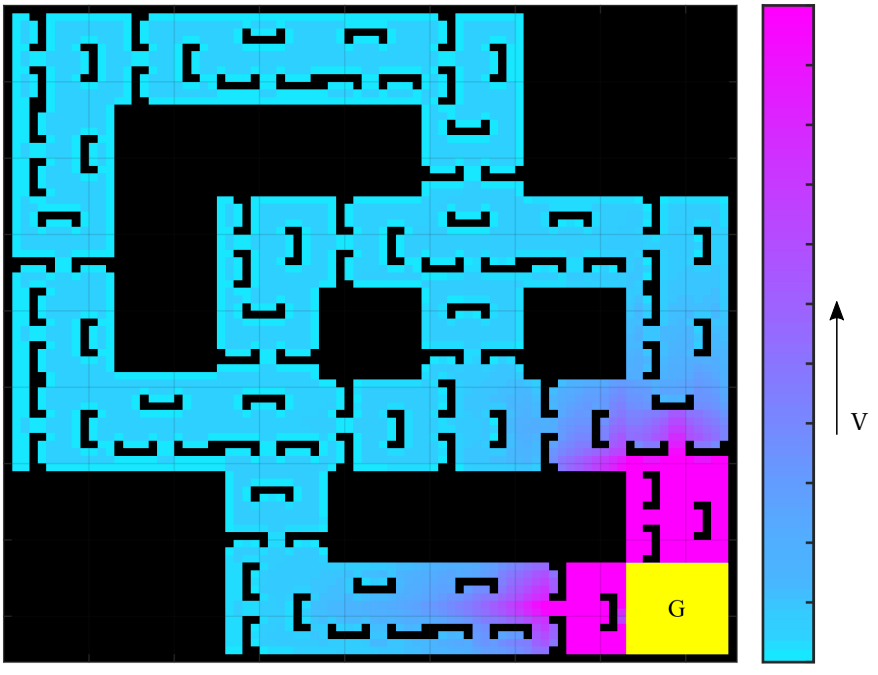


Figure 3.9: Q function represented as a V function by averaging over all possible actions. 200 epochs completed.

actions which run into walls. From this visualization, it can be said that an agent can utilize the information to find the goal more quickly than random-action, and especially will have better information the closer it is to the goal.

Figure 3.11 shows one example, for the flat case, of the evaluation runs over the course of its training. It shows the evaluation runs after epoch 0(no training), epoch 25, 50, 100, 150, and 200. Not every epoch improves the performance of the evaluation run but it is clear thatthe training is producing better policies starting near the goal and then extending outward as expected. In the last evaluation run, most of the path is optimal.

Figure 3.12 shows an evaluation example from optionset 2c. Over the epochs, the evaluation improves as with the flat case, but what is more interesting to see is the exploration. There are more long lines of forward movement covering more distance. With a random policy, of course, the agent may go a far distance just to randomly turn around and go back, but it is consistently faster to the goal in the early training epochs. However, in the later epochs, the solution looks to be converging to something non-optimal.

The evaluation results in Figure 3.10 shows the averages over 100 sample runs. The training for all these simulations was fully random. With a zeroed Q-table at epoch 0, all the optionsets, except 2a, perform find the goal significantly faster than the flat configuration. Optionsets 1c and 2c which encourage long stints of forward movement improve faster than the other optionsets. The flat configuration improves quickly in epochs 1-30, but slows its progress and only catches up to the performance of the “c" configurations at epoch 175.

The results of the Parr's maze environment coaberate the results of the small and medium maze in that options can lead to better performance with less training, but if enough training can be had, then flat Q-learning will still result in a closer-to-optimal solution.

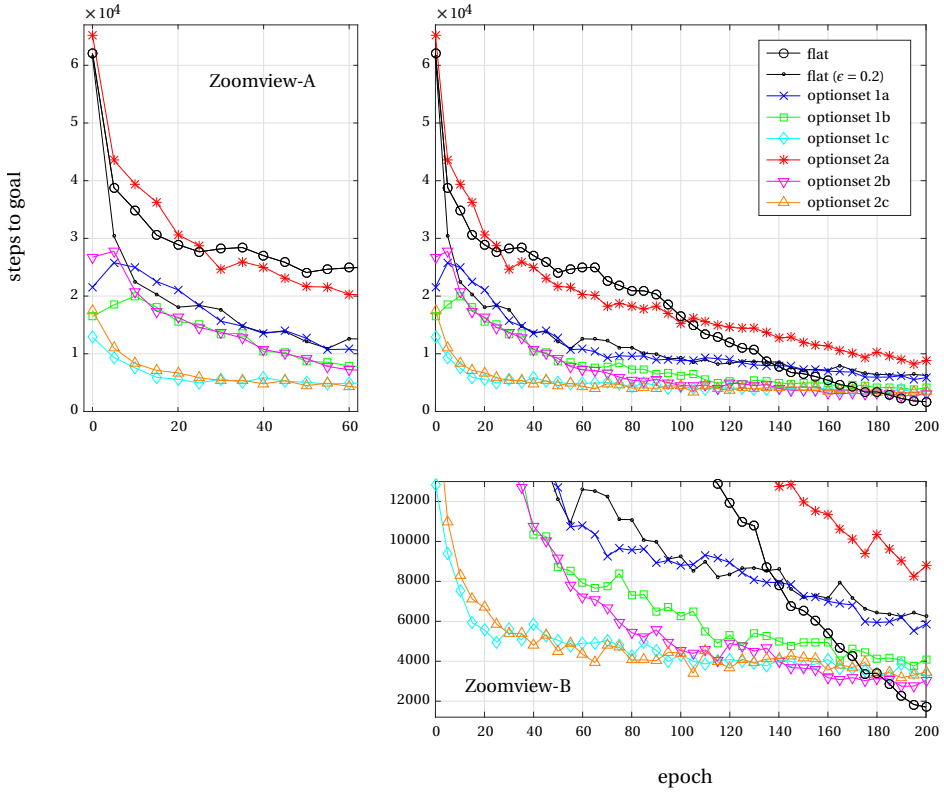


Figure 3.10: Evaluation results for Parr's maze.

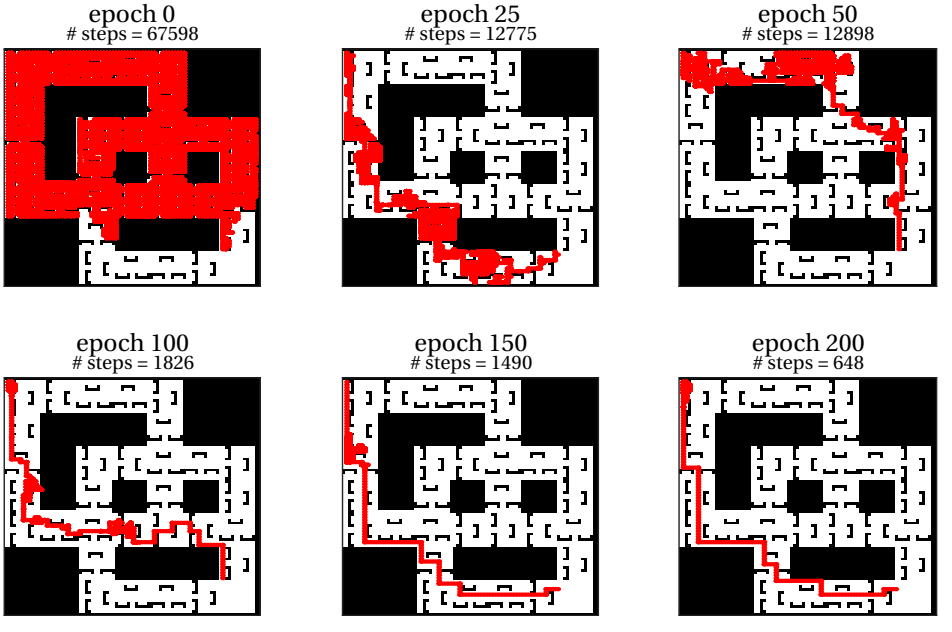


Figure 3.11: Evaluation paths to goal: flat case

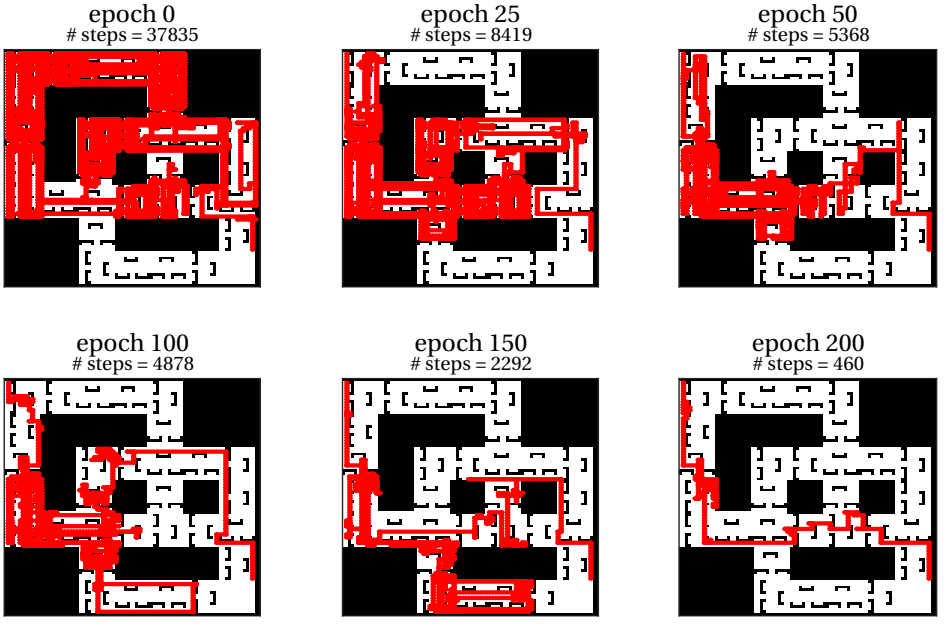


Figure 3.12: Evaluation paths to goal: optionset 2c

3.5. CONCLUSIONS

The findings in this chapter have shown that there are both benefits and disadvantages regarding the use of hierarchical reinforcement learning over a standard flat Q-learning approach. By evaluating the learned Q-function of several different configurations, we are able to observe the performance of a goal-seeking agent in an unmapped environment and make comparisons between flat Q-learning as well as different pre-defined optionsets in the HRL *options* method.

The results of a guidance task in three different sized environments show that HRL methods have the greatest advantages early in the learning process of an unknown environment. The results showed that in the largest maze environment, the best performing optionset was almost 5 times faster to find the goal than the flat Q-learning case at epoch 0 (before any learning). It was only after 175 epochs that the flat case could, on average, outperform this optionset. In the smaller mazes, the HRL methods have benefit over the flat case for less time, therefore demonstrating how temporal abstraction is specifically useful against the curse of dimensionality.

The disadvantage of the HRL method is that, given enough time to train, the flat case surpasses the performance of the HRL method and in every environment converges to the optimal solution fastest. Furthermore, the results show that not all optionsets work equally well. Some optionsets are not well suited for every task and so care must be taken to design an effective options configuration or to make automatically trim optionsets or subgoals [71, 83, 124].

Many learning parameters and different features were explored to find a good version of the flat Q-learning case. For example, eligibility traces and ϵ-greedy training variations were explored (see Appendix A). However, parameter tuning was not performed for the HRL configurations and therefore, the benefit could be greater than what is shown in the results.

The findings in this chapter demonstrate the strengths and weaknesses of the HRL method and set a base for the further work with a hierarchical approach. In application, this chapter represents a guidance task where GPS is available and in some optionsets a crude obstacle detection sensor is also available. As the end goal is to implement this for a UAV guidance task where minimal information and minimal sensors are available, the next step will be to strip the vehicle of all sensors but a camera.

4

HIERARCHICAL REINFORCEMENT LEARNING: RELATIVE STATE REPRESENTATION – A STATE ABSTRACTION APPROACH

In the previous chapter, a hierarchical reinforcement learning (HRL) approach was used to address the curse of dimensionality found with temporal difference reinforcement learning in large state spaces. An expansion of *options* [129] HRL was explored for a large maze guidance task. The hierarchical approach was able to improve upon "flat" Q-learning by introducing temporal abstraction to the available actions.

In this chapter, *state abstraction* is added to the hierarchical approach, to better represent the state-knowledge of a vision-based GPS-denied MAV. This can also be beneficial to combat the curse of dimensionality by limiting the growth of the state space as the environment size increases.

4.1. INTRODUCTION

The core concept of Reinforcement Learning is to learn desirable behavior for an agent using interaction with the environment. The ability of any decision maker to choose successful behaviors depends not only on the quality of the decision process, but also on the quality of the information on which the decision will be made. In the case of reinforcement learning, that is the *state information*. If the state information is irrelevant to the task at hand or too difficult to interpret, then the decision maker will not be able to correlate that information into a meaningful behavior policy.

The ideal input to a decision maker is relevant and directly correlated to the desired output behavior. Complexity of the real physical world and the limitations of current sensing technology make this ideal difficult to attain; however, state abstraction methods take a step in this direction.

State abstraction is a method where state representation is designed around what information is available and what information is deemed most useful for desired outcomes. The desired outcome of a reinforcement learning Q-learning approach is to converge (as quickly as possible) a Q-function so that the greedy policy is an optimal solution.

In Chapter 3, a hierarchical reinforcement learning (HRL) approach was used with an *absolute* state representation. With this state representation, there is an assumption that the location of the agent is aways known. Furthermore, scaling up the size of the environment quickly grows the state space, making the problem intractable.

Real life agents, such as UAVs, function in a continuous (large scale) world and are often limited in their sensing capabilities. Therefore, to reduce the challenges associated with scalability and to better represent the state knowledge of a real-life UAV within the reinforcement learning problem, a state abstraction approach is implemented in this chapter.

A *relative* state representation is an abstraction using the state knowledge which can be obtained from sensors such as cameras or ultrasonic sensors. Therefore, the state information is relative to the view of the agent. In this way, the state space size is no longer dependent on the size of the environment as there are a limited number of views the agent can have (based on the discretization of the state space). The relative state abstraction can be modified in many ways in order to design a state representation as close to the ideal as possible: fully relevant and directly correlated to desired behavior.

In this chapter, the relative state representation is explored as a means to improve the reinforcement learning approach of a maze guidance task by reducing the state space size. Using the discretized distance from the agent to an obstacle, state representations are paired with an HRL approach to learn navigation to a goal in the maze.

The most notable finding of the study is the effect of relative state ambiguity on learning. While the relative state abstraction creates challenges for convergence and is prone to suboptimality, it is concluded that hierarchical reinforcement learning can effectively mitigate these issues. Contributions for this chapter include an HRL approach with a relative state representation which can perform as well as an absolute state representation. Another contribution is a set of guidelines for relative state abstraction design.

In the next section, previous work in reinforcement learning with state abstraction is discussed. Next, a state abstraction approach is developed within the context of the

large maze guidance task from the previous chapter. The simulation results will then be presented showing a comparison between “Flat" Q-learning reinforcement learning and the Hierarchical approach of options with several option sets. The results from Chapter 3 will also be re, followed by a comparison to the HRL results. This chapter concludes with a discussion over the results as well as some insights related to the previous chapter.

4.2. RELATED WORKS

One of the indicators of intelligence is the ability to distinguish between relevant and irrelevant information when making decisions. As humans, our senses inundate us with information which we parse through and interpret to find what is relevant for the decision at hand. For example, a person is tasked to choose a coat for the evening. He has the whole internet at his disposal, but knows that most of the content is irrelevant. The relevant information is in the weather app, and most specifically he needs to know the low temperature for his city. If he lives in a small town for which the weather forecast is not available, he will likely make the connection that the weather in the most nearby city is sufficient to make a decision on his coat selection. This is all logical for humans; however, for machines with access to an excess of information, finding what is relevant for a given task is not trivial.

Abstraction in the context of reinforcement learning is the selection of relevant information to represent the Markov Decision Process (MDP) framework [1]. Abstraction comes in several forms, each with its own benefits. The previous chapter introduces hierarchical reinforcement learning (HRL) *options* which utilizes *temporal abstraction* to create extended actions, therefore abstracting the action space. Another form of abstraction, is *state abstraction*.

STATE ABSTRACTION FOR REINFORCEMENT LEARNING

One of the earliest challenges identified within reinforcement learning, and Artificial Intelligence in general, is the challenge of state and action space construction [7, 50].

State abstraction is the mapping of one state space to another in order to most effectively represent the MDP of a task. State abstraction, also known as feature selection in the continuous domain, is usually employed in RL to ensure as small of a state space as possible for quicker convergence of the value function [5, 27, 36]. Usually, the state representation is determined by the designer.

One interesting class of research is automatic state abstraction, where irrelevant states are automatically eliminated so as to reduce the state space size and therefore speed up learning [27]. Others focus on automated feature selection for function approximation in continuous state spaces [49, 119], and can include human demonstration for quicker learning of correct feature [30].

STATE ABSTRACTION FOR VISION-BASED REPRESENTATION

When implementing reinforcement learning on a mobile robot there are other challenges related to state representation. Firstly, the only information is that which can be sensed. Consider a GPS-denied robot with only a camera for input. Using a value

[1] See Section 3.2 for the background of semi-MDPs and HRL

for every pixel in the camera view would result in a huge, intractable state space. Consequently, state abstraction is almost always a requirement for a vision-based reinforcement learning approach.

The effective state representation will depend largely on the task. For example, Michels et al. abstracts a monocular vision system to estimate the depths of obstacles at high speeds and learn to avoid collision[86]. Gaskett et al. uses a continuous actor-critic framework with separate state abstractions for the subtasks of “wandering" and “servoeing" [49]. These are just some examples of many.

COMBINATION STATE AND TEMPORAL ABSTRACTION TOWARDS REINFORCEMENT LEARNING

The most promising of the abstraction techniques is a combination of abstractions for both state and actions (or critic and actor for continuous domain applications). A state + temporal abstraction approach has been very successful in reducing state space size and tackling the challenge of high dimensionality.

Dietterich has taken his HRL MAXQ formalization [37] and expanded the idea to include state abstraction; where each subtask of the MDP uses its own relevant subset of the state space. He outlines 5 conditions for the state abstraction which should be followed to still guarantee convergence of the MAXQ-Q algorithm and applies the state abstraction to the taxi-domain problem[36]. With the same taxi domain problem, Andre et al. demonstrates an increase in learning speed with guaranteed hierarchical optimality using state abstraction in the programming language ALisp[5].

Asada et al.(1996) presents a vision-based reinforcement learning method with both temporal and state abstraction. Learning is performed in simulation and the policy is then tested on a real goal-shooting robot, where the focus is on the observation of the reality gap. Automatic state construction for subtasks of the hierarchy was attempted but was found to be the main challenge of the study [8].

THIS CHAPTER

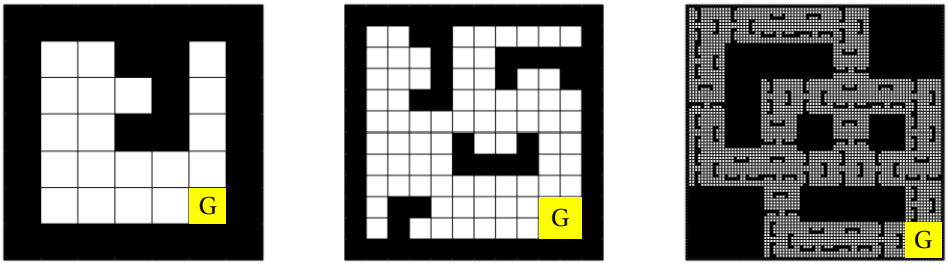
The content of this chapter differs from other studies in that it is a combination of abstraction in the relative state space with a guidance task where the reward is only known at the end. Vision-based state representation has been paired with reinforcement learning to learn behaviors such as obstacle avoidance or wandering, and state abstraction has been used with reinforcement learning to learn end-goal guidance tasks. The task in this chapter utilizes vision-based state representations with end-goal guidance, which leads to the challenge of *state ambiguity*. [2]

4.3. STATE ABSTRACTION SETUP

The same discrete grid-world mazes (Figure 4.1) from Chapter 3 are used as test environments for this chapter. The goal locations remain the same and the evaluation method is also the same.

[2] Unlike Partially observable MDP’s, this study does NOT use the POMDP formal framework, instead using the semi-MDP framework for HRL as described in Chapter 3

In the previous chapter, the state of the agent in the maze is based on the *ground state*, meaning it is represented by the location in X-Y cartesian coordinates in the maze. In a practical sense, this state representation presumes GPS sensor information or another kind of location estimation knowledge.



(a) (b) (c)

Figure 4.1: Test environment mazes: small, medium, and Parr's (large) mazes

Assuming the absolute state cannot be known or is an undesirable state representation, a different state representation can be used with a reinforcement learning approach. In this section we introduce a vision-based state representation designed to resemble a discretized form of what an agent can see from on-board sensors. In other words, a state from the perspective *relative* to the agent. For instance, there may be one or 2 cameras on-board with a 180° field-of-view which can use computer vision techniques to detect obstacles. A UAV may also carry lightweight ultrasonic sensors to estimate distance from obstacles or walls [47]. We do not take into consideration the strengths and weaknesses of the types of obstacle detection methods but assumes that the method will be accurate for the purpose of state acquisition.

The different state representations used in this chapter are defined as follows:

Heading state The heading state, x_h represents the heading the agent is facing from the possibilities of North, East, South, and West.

Ground state The ground state, x_g, of the agent is defined in terms of its X-Y cartesian coordinates in the grid world.

Vision state The vision state, x_v, is represented in terms of what is visible to the agent in its own vicinity or field-of-view. The size of the vision state will depend on d_s and n_d as described in Figure 4.2. Examples of the vision state array can be seen in Figure 4.3.

Absolute state Consists of the ground state and heading. $x_a = [x_g \; x_h]$

Relative state Consists of the vision state and heading. $x_r = [x_v \; x_h]$

The research in this chapter investigates HRL effectiveness of relative state representations. as demonstrated in Figure 4.2. The vision state is an array of length n_d, where each element of the array represents the view of each division with respect to the forward

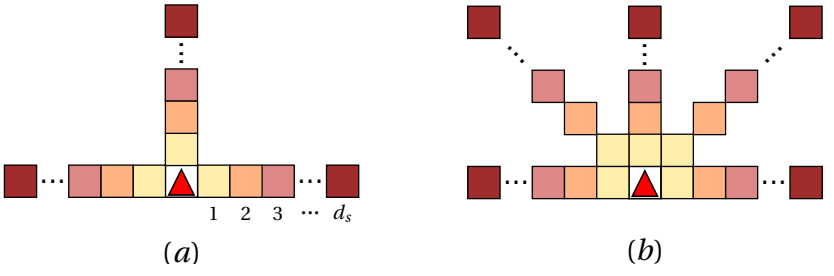


Figure 4.2: The **relative state representation** is partially defined by the distance the sensor detect obstacles, d_s, and the number of divisions its field-of-view is divided into, n_d. (a) $n_d = 3$, (b) $n_d = 5$. The default representation used for this chapter is $d_s = 3$, $n_d = 3$

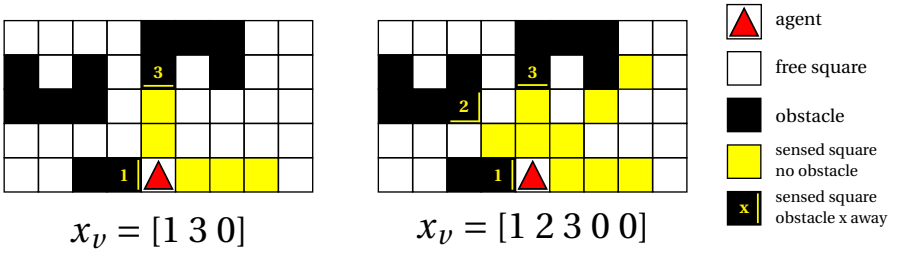


Figure 4.3: Vision state examples for relative state representation with (*left*) $n_d = 3$ [left front right] and (*right*) $n_d = 5$, x_v = [left diagonal-left front diagonal-right right]

facing orientation of the agent, starting from the left and going clockwise. The value of each element is the distance (in discrete grid squares) from the agent to the nearest obstacle. If, there are no obstacles within the maximum range of sight, d_s, that is "no obstacle" state. Examples for $n_d = 3$ and $n_d = 5$ are shown in Figure 4.3 ((*left*)and(*right*) respectively). In practice, the field-of-view of a camera can be split into any number of divisions, but in the discretized gridworld, it only makes sense to split it into 3 and 5, where 5 divisions includes the diagonals. The last element of the relative state array is the heading state, x_h as North, East, South, or West.

STATE SIZE

The relative state size trends with respect to d_s and n_d can be seen in Figure 4.4. For each maze, the number of absolute states is constant. Greater n_d will also lead to more information about the state when d_s is held constant.

Although the state matrix of the relative state will get very large as d_s increases, the number of states possible to encounter within the maze will never surpass the number of absolute states. Therefore, using relative state representation reduces the effective state space.

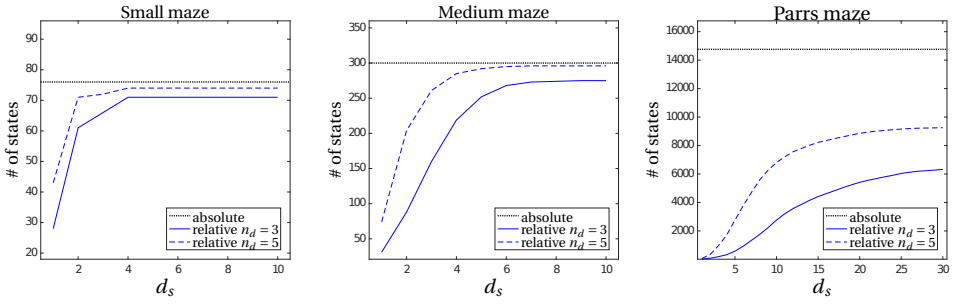


Figure 4.4: State size trends for each maze with respect to vision parameters n_d and d_s.

4.3.1. STATE AMBIGUITY

The relative state representation creates a new challenge. Using an absolute state representation, every state is unique. That is, state s for all states $s \in \mathcal{S}$ occurs only once in the environment. Furthermore, taking action a from state s, will always result in a transition to state, s', assuming the transition probability for the MDP is 1, or $P^a_{ss'} = 1$. However, a relative state can occur in multiple locations in a maze environment and therefore, taking action a from state s may result in a different s', affecting the Q-function convergence. In such cases, the state is denoted as being *ambiguous*.

Figure 4.5 demonstrates the ambiguity found in the small maze. The values in the grid represent how many times that relative state occurs in the absolute representation of the maze. The rows show the different ambiguous states for values of $d_s = [2, 3, 4]$ (top to bottom), demonstrating how expanding the field-of-view for the vision can decrease the ambiguity. The columns represent the heading state. Take for example $d_s = 2$ facing south. There are 3 grid squares which occur 3 times within the maze. This is because for all three squares, the left and right views have adjacent obstacles, and the front facing view is in the "no obstacle" state since it can only see 2 squares ahead. As the vision distance d_s increases, the agent can see the southern wall earlier, and the ambiguity for these states is removed if $d_s \geq 4$. As seen with some of the states in Figure 4.5, some of the ambiguity will always exist with a relative state representation.

An analysis was made into the ambiguity levels of each of the mazes for varying relative state representations. The histogram equivalent of the small maze in the previous figure can be seen in Figure 4.6 on the top row. The medium maze is analyzed in the middle row at $d_s = [2, 4, 6]$. The Parr's maze analysis at $d_s = [3, 5, 10]$ is shown in the bottom row. The histogram counts the ambiguity level which is the number of times each relative state occurs in the maze. For example, the ambiguity level of a unique state is 1, and any other value is classified as ambiguous to the degree of its value. With all the mazes, the distribution of the state ambiguity moves toward becoming more unique as d_s increases. In the case of the Parr's maze, the ambiguity histogram at $d_s = 3$ shows that there are 4 relative states which occur over 750 times in the maze and several occur between 200-400 times. When d_s is expanded to 10, the highest prevalence of a relative state is around 125.

The trends of state ambiguity with respect to varying the state representation can

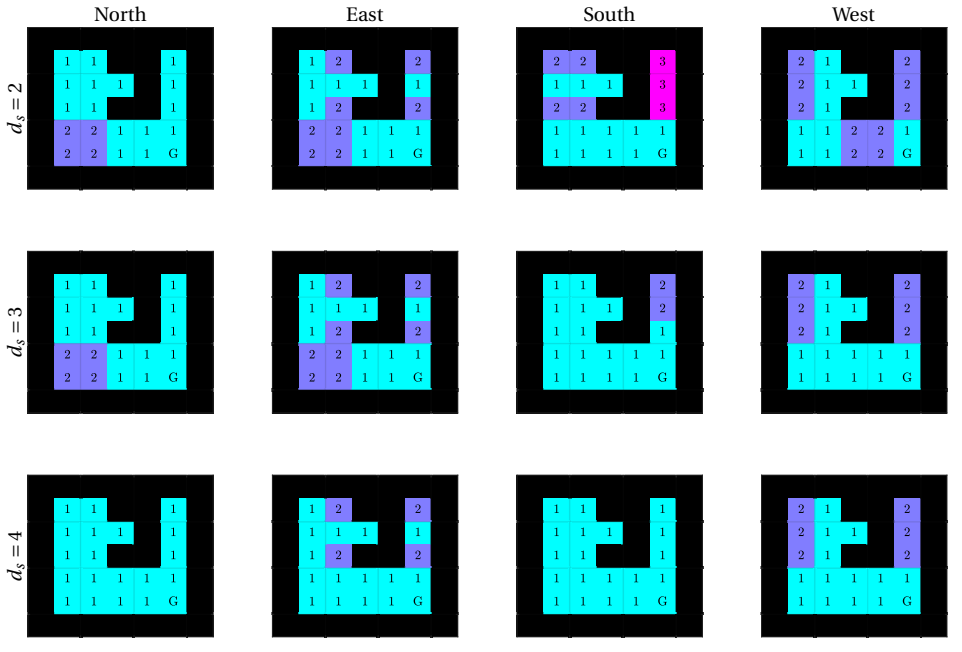


Figure 4.5: Ambiguity levels in small maze with $n_d = 3$ and varying d_s. The value indicates how many times that same relative state occurs in the maze. (top row) $d_s = 2$, (middle row) $d_s = 3$, (bottom row) $d_s = 4$.

be seen in Figure 4.7. The figure plots the ambiguity percentage trend for each maze at $n_d = 3$(solid lines) and $n_d = 5$(dashed lines) as d_s is varied on the x-axis. The ambiguity percentage is the (number of absolute states which have a corresponding relative state reapeated elesewhere in the grid) / (the total number of absolute states).

From Figure 4.4, we see that as d_s increases, so does the number of relative states within the maze. Therefore, it can also be noted that the ambiguity percentage decreases as number of relative states increase.

4.4. REINFORCEMENT LEARNING ALGORITHMIC SETUP

The algorithmic approach for the flat and hierarchical reinforcement learning is the same as in Chapter 3, with a few notable exceptions now presented.

4.4.1. HRL OPTIONSETS

In addition to the options described in Section 3.3, Table 3.2, two new options are added for use with the relative representation. These options are called “go around - left(L)” and “go around - right(R)”. They use the benefit of the vision state to go around obstacles either on the right or left. For example, the option “go around - L” can be chosen, which will cause the agent to turn left and keep going forward until there is no longer an obstacle on it’s right side, then turn right and continue going forward until there is another obstacle in the way. This option is more specifically targeted to this task and therefore,

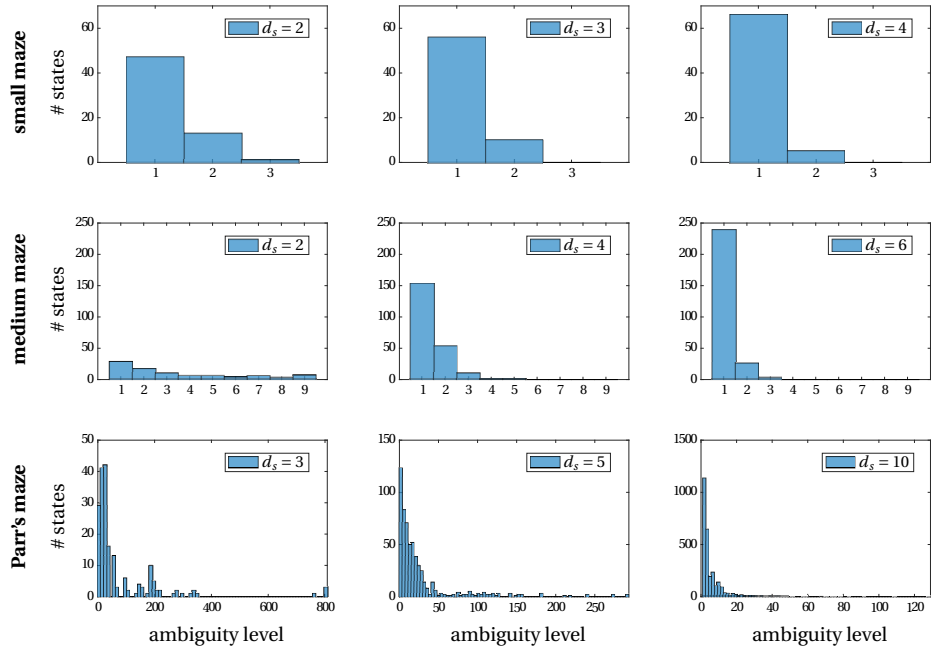


Figure 4.6: Histogram of ambiguity distribution over various maze sizes and d_s. Constant $n_d = 3$.

may not be as well suited for solving other guidance tasks. However, generalization and effectiveness of the approach is often a trade-off.

The new optionset is summarized in Table 4.1. The descriptions of Option set series 1 and 2 are condensed "N^*" denotes going forward until an obstacle is detected 2 or less spaces in front.

Table 4.1: Configurations of flat Q-learning and HRL optionsets.

flat	Optionsets			
(primitive)	1a-c	2a	2b-c	3
turn left	left	[left + fwd]	[left + fwd]	left
turn right	right	[right + fwd]	[right + fwd]	right
fwd×1	fwd×1	fwd×1	fwd×1	fwd×1
-	fwd×(2/3/N^*)	180^o turn	180^o turn	fwd×N^*
-	-	-	fwd×(3/N^*)	go around - R
-	-	-	-	go around - L

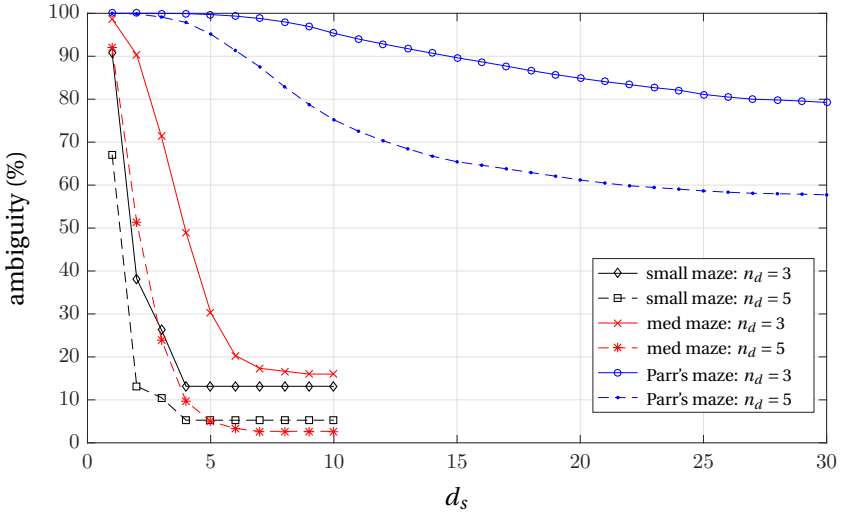


Figure 4.7: Ambiguity level trends.

4.4.2. PARAMETER TUNING

Tunable parameters in the Bellman equation and in the training policy have an effect on the convergence and therefore on the performance of the final Q-function. For this reason, two parameter schemes were developed to show how tuning can affect the learning and the trends found within the various configurations. The first scheme, is simply the same parameters as was found to be effective with the absolute state representation in the previous chapter. To find a new parameter scheme more suited for the relative state representation, a short study was conducted and the results can be found in Appendix B.

From the study into the ϵ value of the ϵ-greedy training in Appendix A, it was found that $\epsilon = 0.2$ resulted in a positive learning rate effect at the cost of having a less optimal solution than a purely random training.

With the ambiguity present along with the relative state representation, convergence can be an issue. While the first scheme uses constant $\alpha = 0.9$, the second scheme uses a variable $\alpha(s,a)$ which decreases over k visits to (s,a).

The two parameter schemes are defined as in Table 4.2.

The results analyzed in Section 4.6, use the parameters which were tuned for the relative state representation (scheme 2).

4.5. RESULT PRELIMINARIES: PLOT SELECTION AND ANNOTATION

4.5.1. PLOT SELECTIONS

In order to show clear and nonredundant results, only a representative selection of the results are presented. A study into which results are most representative is given in Appendix B. The studies determine which HRL optionsets to display as a comparison

Table 4.2: Parameter schemes for absolute and relative state representations

parameter	Absolute state representation (scheme 1)	Relative state representation (scheme 2)
Bellman equation		
γ	0.9	0.9
α	0.9	$1/\sqrt[5]{k}$
training policy		
ϵ	0.0 (random)	0.2

against the flat case, as well as a determination of the parameter scheme.

4.5.2. CONFIGURATION NOTATION

For practical purposes the configuration names in the results section sometimes need to be abbreviated.

State and configuration abbreviations used to label results in this chapter contain some or all of the following information and may be abbreviated as indicated:

maze size	small, medium (med), or Parr's
state representation	absolute(abs) or relative(rel), default: rel
optionset configuration	optionset(set) or flat
vision state representation	(d_s/n_d)
parameter tuning scheme	scheme (sch) 1 or 2, default: sch. 2

example 1: med-abs-optionset1a (2/3) - sch.1
example 2: parrs-rel-set3 (3/5)

4.6. RESULTS

The aim of this section is to analyze performance trends within flat Q-learning and hierarchical reinforcement learning relating to state representation and environment size.

For each test case, the performance is evaluated every 5 epochs with an **evaluation run**. An evaluation run counts how many steps it takes to reach the goal from the corner furthest from the goal (top-left facing south) using a greedy policy and the most recently learned Q-function. The whole 200 epoch training is run multiple times using different seeds to find an average and standard deviation over several sample runs (100 times for the small and medium mazes, and 10 times for the Parr's maze due to time constraints).

When analyzing the data, two values relating to the performance metric should be kept in mind. The optimal number of steps in an evaluation run is the best performance the evaluation can achieve. The "flat random" performance is the average number of steps the evaluation run takes if a random policy of primitive actions were taken. Deviating from this value over training epochs means that the Q-function is learning (for better or worse). See Table 4.3 for these values specific to each maze.

Table 4.3: Evaluation run performance: optimal # steps and average/standard deviation of random policy

	Small	Medium	Parr's (Large)
optimal # steps	9	18	162
flat random # steps (mean± std)	160 ± 57	352 ± 91	75370 ± 29031

The results of the evaluation runs are presented in Section 4.6.1 where a comparative analysis aims to find the trends between differing relative state representation and the subsequent performance. The results for the relative state representation are also compared to that of the absolute state representation results from Chapter 3.

In Section 4.6.2, the performance of one run from each maze is shown in a visually informative form. The path of the agent is plotted during the greedy evaluation throughout the training and the final best path found by the reinforcement learning algorithms.

Finally, the trends and observations seen throughout Sections 4.6.1-4.6.2 will be critically discussed in Section 4.6.3.

4.6.1. STATE REPRESENTATION

In order to effectively use state abstraction to improve the performance of (hierarchical) reinforcement learning, it is helpful to understand how different types of state representations affect the performance and why. This section presents the results of the various relative state abstractions. The ways we can change the state representation (see Figure 4.2) include enriching the field-of-view by either 1) increasing the state space size with a larger line-of-sight distance, d_s , or 2) changing the state information structure by increasing the number of division of the field-of-view, n_d . Lastly, the performance of the relative state representation is compared to that of the absolute state representation.

INCREASING STATE SIZE WITH SENSOR RANGE, d_s

As the state space size increases, the ambiguity of the state in the maze is decreased. Therefore, it is expected that as the state space grows by increasing the distance of the sensor view range (represented as number of discrete squares), $d_s \rightarrow \infty$, then the ambiguity will decrease to its minimum and the performance should improve. However, this is not always the case.

Each maze environment is analyzed separately. The evaluation results from the small maze are shown in Figure 4.8, the medium maze in Figure 4.9, and the Parr's maze in Figure 4.11. The plots are zoomed in to more conveniently view what is most important. The flat case results are in the top plot and the results using HRL optionsets are shown in the plots thereunder. Each plot will compare varying d_s and keep the number of field-of-view divisions constant at $n_d = 3$.

For the **small maze**, the sensor view distance is varied as $d_s = \{1,2,3,4\}$. For both flat Q-learning and optionset 1b, $d_s = 1$ performs the worst as there is not enough state information to make informed decisions. With $d_s = 1$ there is about 91% ambiguity. Refer to Figure 4.7 for ambiguity trends with respect to d_s. For the optionset 1b configuration,

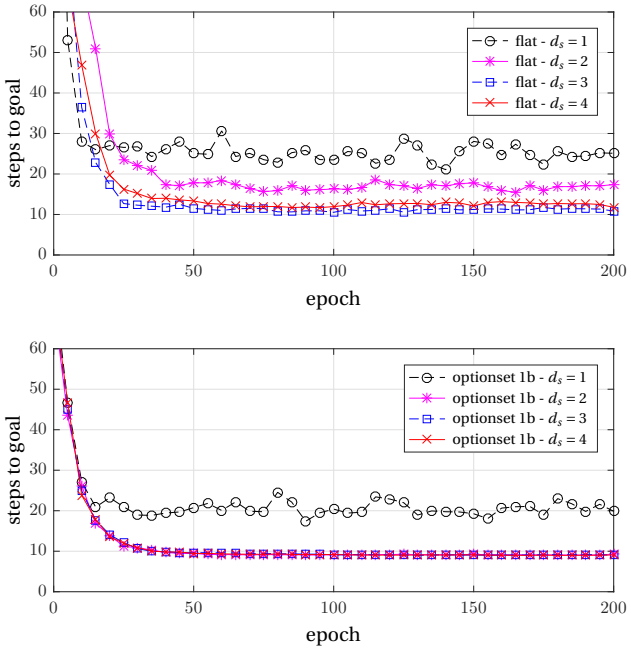


Figure 4.8: **small maze:** Evaluation result trends varying d_s for flat case(top), and optionset 1b configuration (bottom). Averages over 100 sample runs.

state representations with $d_s = \{2,3,4\}$ all converge on average at more or less the same speed to the same average number of steps to goal of between 9 and 9.25 over 100 sample runs. This shows that given a certain level and quality of state information, the optionset (as compared to the flat case of the same state space size) is able to most often find the optimal solution within a small environment.

For the flat case, increasing $d_s = 1 \rightarrow 2$ and $d_s = 2 \rightarrow 3$ improves the performance as expected. However, increasing from $d_s = 3 \rightarrow 4$, actually worsens the performance in both convergence speed and average converged performance even though the ambiguity is also reduced. This result is not intuitive.

With the **medium maze**, the same unintuitive result occurs for the flat case as seen in Figure 4.9. Here, the state space representation explored is defined by varying $d_s = \{2,3,4,5\}$. For the flat case, increasing $d_s = 2 \rightarrow 3$ improves the performance, but increasing $d_s = 3 \rightarrow 4$ and $d_s = 4 \rightarrow 5$ only hurts the performance even though the ambiguity percentage is decreasing. Therefore, there is some aspect other than just ambiguity percentage which influences the quality of the state representation. Furthermore, there must also be an explanation that certain options are able to circumvent this limitation and produce more optimal results with intuitive trends regarding state representation and performance.

To analyze this unintuitive trend in the flat case, we must first recall what the per-

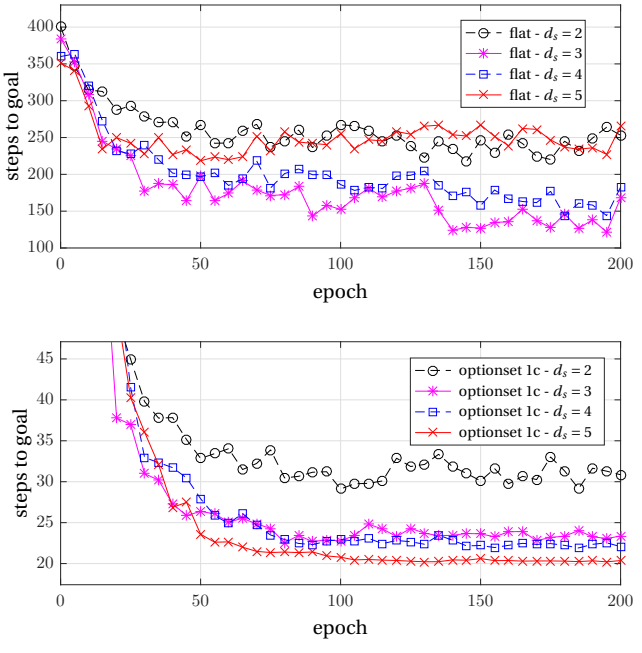


Figure 4.9: **medium maze:** Evaluation result trends varying d_s for flat case (top), and optionset 1c configuration(bottom). Averages over 100 sample runs.

formance metric is. The goal of this reinforcement learning task is to learn a Q-function which an agent can use to find the optimal path to the goal given a greedy policy. The metric by which we measure the performance is "number of steps to the goal". A full explanation of the evaluation run is found in Section 3.3.4.

The evaluation run up until now has always initialized the agent in the top-left corner of the maze. Using the same initialization point for every sample run, a good performance would be characterized by a convergence over training epochs to the minimum number of steps to goal and a standard deviation which goes to zero. However, even with such perfect behavior, starting from the same initial state means the performance being evaluated is only that of a portion of the state space. Therefore, we can only conclude that this unintuitive trend for the flat case is occurring in that portion of the state space between the initial state and the goal. In Figure 4.10 other initial evaluation states are explored.

Evaluation runs do not affect the Q-function learned from training, therefore the same Q-function can be evaluated with the agent being initialized at different states. Figure 4.10(a) is the default (and the same as Figure 4.9(top)), which uses the initial evaluation state in the top-left corner of the maze (facing south). Figure 4.10(b) shows the results when evaluated from the initial state in the top-right (facing west). This evaluation has the same optimal number of steps to the goal as Figure 4.10(a), 18 steps. For

Figure 4.10(a) and (b), the stated initial state is used for all 100 sample runs. The averages are plotted without standard deviation for clarity in comparison. In the third case, as shown in Figure 4.10(c), there is a different initial state used each sample run. The initial states are randomly generated so it should evaluate a larger portion of the state space; however the optimal number of steps to the goal in this case is unknown so the performance metric is limited[3].

Several conclusion can be drawn from Figure 4.10, which help to understand the unintuitive trends for d_s with the flat Q-learning case. In Figure 4.10(a) and (b), starting from two different initial states which have the same optimal number of steps to the goal, we see that evaluating different parts of the state space will result in different trends. This is because the ambiguity has a different effect in different regions. Also, in certain regions of the maze, certain ambiguous states have a beneficial effect. This is seen in Figure 4.10(a) and (c). In the medium maze, as a whole, the ambiguity configuration of the state representation $d_s = 4$ results in a Q-function which makes a greedy agent more prone to non-optimal paths than $d_s = 3$. It should not be forgotten, however, that this is only for the flat case, and the HRL approach is able to remedy this problem.

The state representations used for comparison within the **Parr's maze** are defined by $d_s = \{3, 5, 11\}$. The larger maze is different from the other smaller environments in that the flat case performance doesn't learn over the training period. In Figure 4.11, the flat case (top) shows no improvement over training epochs with any of the state representations. Furthermore, it is worth noting that a random primitive action selection would result in a better performance (as stated in Table 4.3).

Using HRL optionset 1c there may be some learning in the largest state space size of $d_s = 11$ as can be seen in Figure 4.11(middle) that the evaluation performance improves in the first 50 epochs and converges to a slightly faster path to the goal on average. However, the state reps defined by $d_s = \{3, 5\}$ do not improve.

Optionset 3 has the longest and most complicated options. The Q-function is able to learn and converge to a better solution than random. Although the averages of the various d_s converge to about the same number of steps to goal, the standard deviation is smaller with larger d_s, indicating that ambiguity leads to greater variance in the performance. The final average performance over 10 sample runs is about 360 steps to the goal, with the best evaluation performance found as 179 steps to goal.

[3] The limitation of the default evaluation is noted, however still defended as a good performance metric since it is useful to know the quantifiable qualities of that run, such as the optimal or random performance. Evaluating from multiple intial locations would be best but also time consuming and would add unnecessary convolution. Therefore, it was decided to stay with the default evaluation over a more all-encompassing approach.

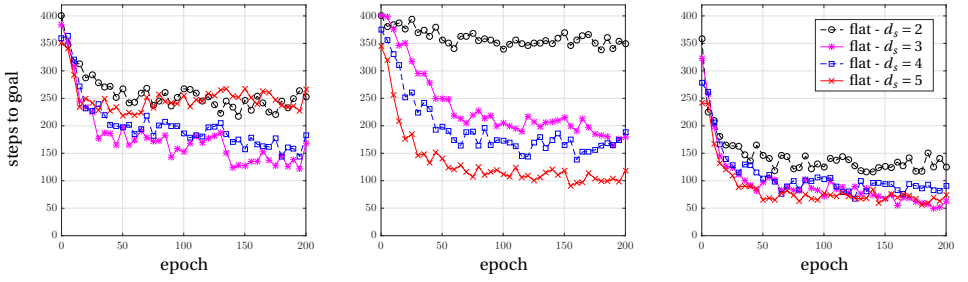


Figure 4.10: **medium maze:** Flat case results evaluating the number of steps to the goal from different initial states. **(a)** initial evaluation state is the top-left state facing south (averaged over 100 sample runs), **(b)** initial evaluation state is the top-right state facing west (averaged over 100 sample runs), **(c)** initial evaluation state is different every sample, averaged over 100 sample runs.

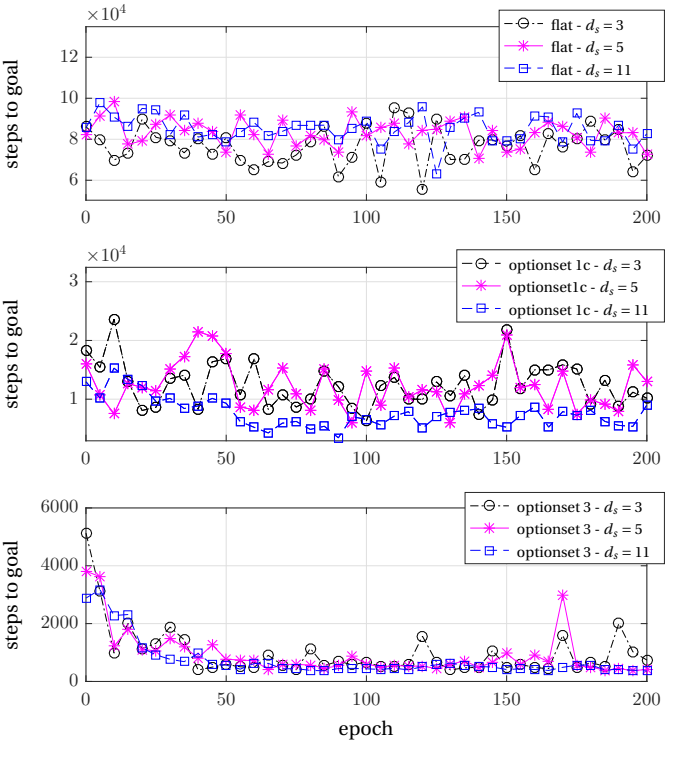


Figure 4.11: **Parr's maze:** Evaluation result trends varying d_s for flat case (top), optionset 1c(middle) and optionset 3 (bottom) configurations. Averages over 10 sample runs.

FIELD-OF-VIEW DIVISIONS, n_d

In this section, state abstraction in another form is explored. Rather than increasing the size of the state space, the structure of the state is changed by dividing the field-of-view of the agent into $n_d = 3$ and $n_d = 5$ divisions (see Figure 4.2). In each maze, there is a relative state representation of ($d_s = x/n_d = 3$) and ($d_s = y/n_d = 5$) which will have about the same number of states. It is easy to visualize which state representations will have similar number of states using Figure 4.4. The state representations which were chosen to be compared are summarized in Table 4.4.

Table 4.4: State and ambiguity information on compared relative state representations for each maze

	small maze		medium maze		Parr's maze	
	$n_d = 3$	$n_d = 5$	$n_d = 3$	$n_d = 5$	$n_d = 3$	$n_d = 5$
d_s	4	2	5	3	10	5
# relative states	71	71	261	252	2770	2767
% ambiguity	13	13	24	30	95	95

For the **small maze**, the flat case and optionset 1b are plotted with representations ($d_s = 4/n_d = 3$) and ($d_s = 2/n_d = 5$) in Figure 4.12. Although there are the same number of states in each of the representations and the ambiguity percentage is also the same, the two representations will have a different distribution of ambiguity as previously demonstrated in the histogram of Figure 4.6. The flat case shows a clear benefit using the $n_d = 5$ representation. The performance for optionset 1b is not as affected, but shows that the $n_d = 5$ rep is, on average, slightly faster to learn.

The same trends are observed for the **medium maze**, where the flat case and optionset 1c are plotted with representations ($d_s = 5/n_d = 3$) and ($d_s = 3/n_d = 5$) in Figure 4.13. In this case, the $n_d = 5$ representation has less total relative states but still performs better than the $n_d = 3$ cases, on average. The "jumpy" behavior of optionset 1c-(3/5) can be an indication that there are ambiguous states that have conflicting optimal actions. This means the performance can differ epoch to epoch based on which states were randomly visited in the trainings since the last evaluation run. A more detailed discussion of this behavior is in Section 4.6.3.

For the **Parr's maze**, the flat case, optionset 1c, and optionset 3 are plotted with representations ($d_s = 10/n_d = 3$) and ($d_s = 5/n_d = 5$) in Figure 4.14. The flat case still does not learn within the 200 training epochs, although the $n_d = 5$ case inherently finds the goal faster. The performance for flat-(10/3) fluctuates around an average of 82,465 steps, and flat-(5/5) fluctuates around an average performance of 67,818 steps. Since the greedy evaluation runs learn online in order to avoid getting stuck in loops, this means that the Q-function learned online for flat-(10/3) is actually learning a greedy behavior worse than random, and the Q-function learned online for flat-(5/5) is learning some behavior which is marginally favorable over random.

For optionset 1c, both representations learn during the 200 training epochs. Although both cases struggle to converge, as shown by the standard "jumpy" behavior, it's clear that the $n_d = 5$ learns faster initially and converges to a fewer number of steps to goal on average. Specifically, $n_d = 5$ outperforms the $n_d = 3$ case by an average of 3451

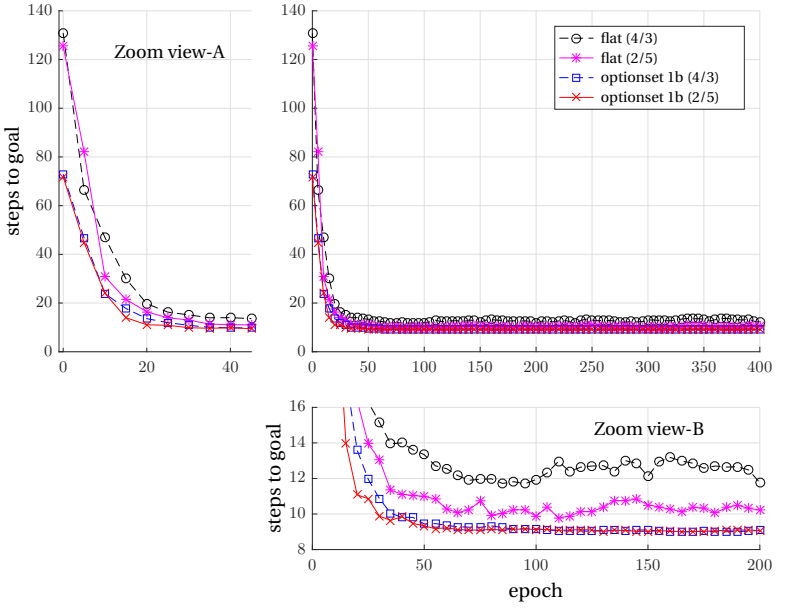


Figure 4.12: **small maze**: Evaluation results with flat case and optionset 1b with ($d_s = 4/n_d = 3$) and ($d_s = 2/n_d = 5$) relative state representations. Averages over 100 sample runs.

steps in the first 50 epochs and once it converges, the performance is better by 3077 steps on average in the last 50 epochs.

Optionset 3 is by far the best performing configuration. Between the representations using $n_d = 3$ *or* 5, they are close but $n_d = 3$ outperforms $n_d = 5$ by 462 steps on average in the first 50 epochs, and a 69 step average in the last 50 epochs.

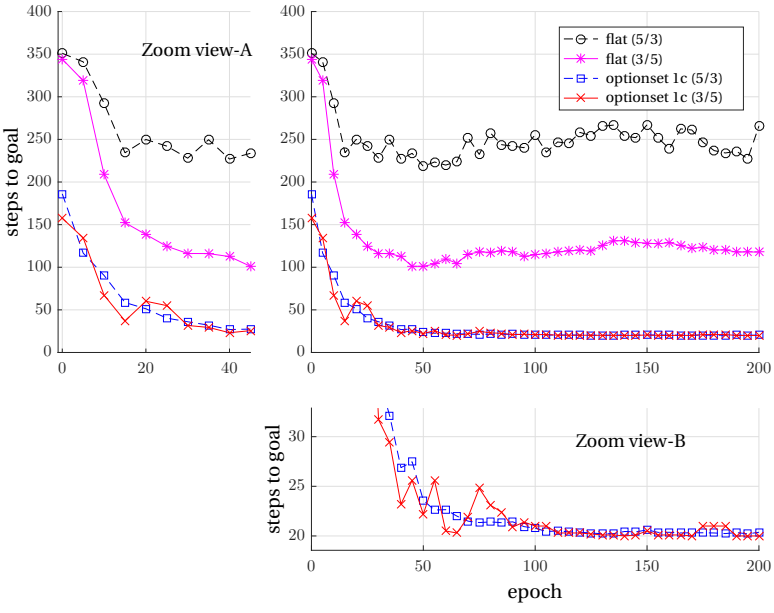


Figure 4.13: **medium maze**: Evaluation results with flat case and optionset 1c with ($d_s = 5/n_d = 3$) and ($d_s = 3/n_d = 5$) relative state representations. Averages over 100 sample runs.

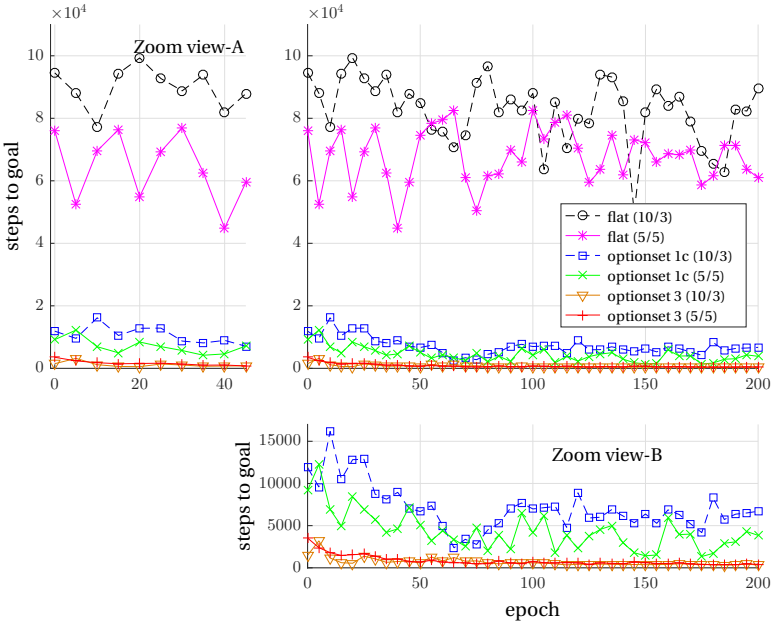


Figure 4.14: **Parr's maze**: Evaluation results with flat case, optionset 1c and optionset 3, with ($d_s = 10/n_d = 3$) and ($d_s = 5/n_d = 5$) relative state representations. Averages over 10 sample runs.

ABSOLUTE VS. RELATIVE

Now that the various relative state representations have been examined, it is interesting to see how even more fundamentally different state abstractions compare against each other. In the last chapter, the hierarchical reinforcement learning problem was analyzed using the absolute "ground" state. With absolute state representation, every state in the environment is unambiguous. Due to this, the state space size increases as the maze does and so learning speed also suffers. Therefore, the expectation is that as the mazes get bigger the relative state representation will have some benefit in early learning speed over the absolute state representations.

In this section, the absolute state results from the previous chapter are shown against the evaluation performance of one of the relative state representations from this chapter.

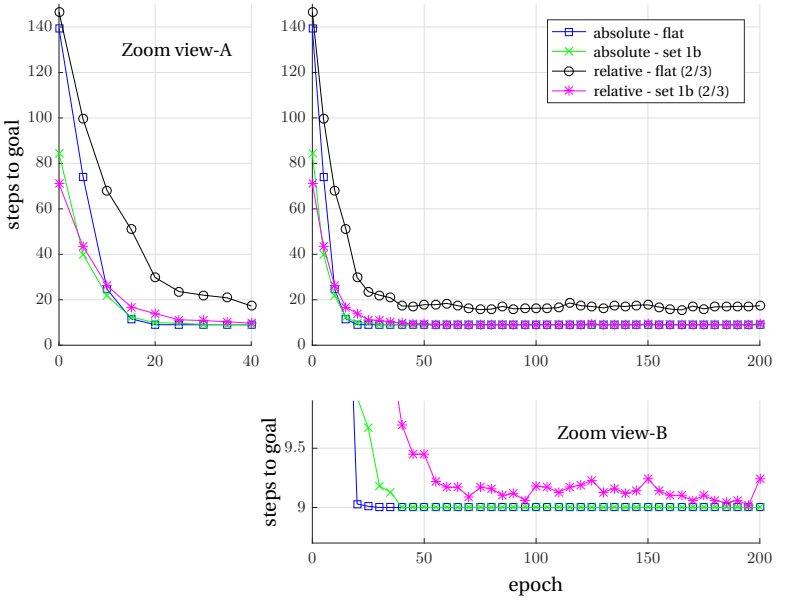


Figure 4.15: **small maze**: absolute vs relative state representation - ($d_s = 2/n_d = 3$). Averages over 100 runs.

The comparison of the **small maze** is shown in Figure 4.15. The flat case and optionset 1b are plotted for the absolute state representation and the relative state ($d_s = 2/n_d = 3$) representation. The relative representation has 61 states to the absolute's 76; about 20% less states. While the benefit for relative representation is that there are less states, the drawback is that the solution is less than optimal, on average. Absolute state representation using flat Q-learning or optionset 1b will find the optimal solution 100 out of 100 times after 45 epochs. The relative state representation has no such guarantee. As with all other results, the relative representation with flat learning is the poorest performer and the HRL optionset learning method improves upon it greatly, in this case cutting the number of needed steps to goal in half. Furthermore, the hierarchical approach with relative state representation finds the optimal at *at least one evaluation run*

100 out of 100 times. This means, that an optimal Q-function is found at some point and then further training causes it to become non-optimal due to the ambiguity and the random nature of the training. In a practical application, the optimal Q-function could thus be remembered and more effectively utilized if desired.

The comparison of the **medium maze** in Figure 4.16 plots the flat case and optionset 1c for the absolute state representation and the relative state ($d_s = 5/n_d = 3$) representation. The relative representation has 252 states to the absolute's 300; about 16% less states. The results for this maze just solidify that conclusion that the relative flat case will not perform better than the absolute flat case, even in the early epochs where it was predicted that the relative state representation could lead to faster learning rate. However for HRL, the optionset 1c does show a learning rate advantage over the absolute state representations in the early epochs: from epoch 5-25. In convergence, the absolute flat learning configuration finds the optimal path 100% of the time by epoch 60 while the relative state representation finds it slower. Both absolute and relative representations with optionset 1c configuration find the suboptimal path of 20 steps to goal.

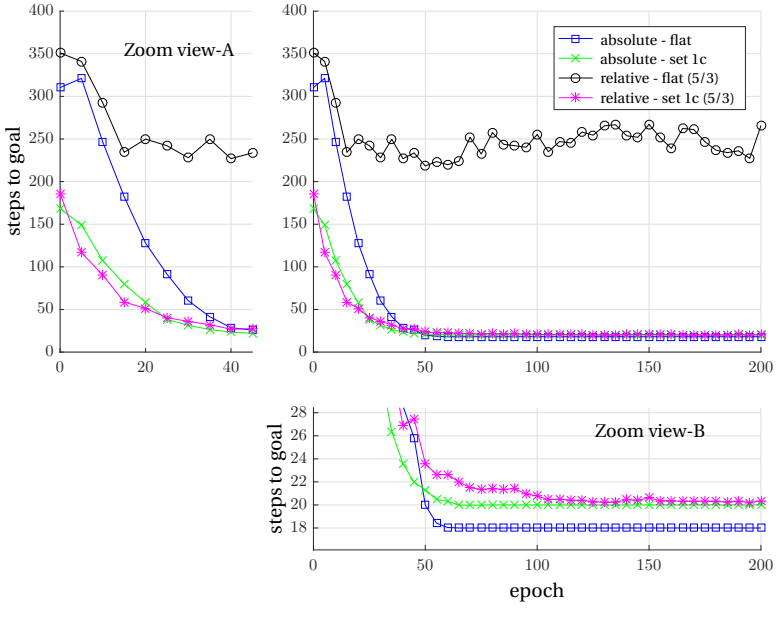


Figure 4.16: **medium maze**: absolute vs relative state representation ($d_s = 5/n_d = 3$). Averages over 100 runs.

The comparison of the **Parr's maze** is shown in Figure 4.17. The flat case and optionset 1c are shown for the absolute state representation and the relative state ($d_s = 5/n_d = 3$) representation; the same representation scheme as the medium maze displayed. Additionally, optionset 3 results are shown only for the relative state representation since the absolute state is assumed to not have the state information to run option "go around". The relative representation has 582 states to the absolute's 14760; about 96% less states. The relative flat case and relative optionset 1c do not learn, as previously observed. The absolute flat case will always find the optimal given enough time

and enough random exploration. At the end of the 200 epochs, the absolute flat case is approaching the optimal of 163. However, for the first 175 epochs the absolute optionset 1c and the relative optionset 3 performs much better than the absolute flat; on average 5 and 22 times faster, respectively, in the first 50 epochs.

The relative HRL optionset 3 configuration has the advantage of containing the longest extended action option. The options "Go around - Left" and "Go around - Right" are also more customized toward this particular problem and only available using the relative state representation. Nevertheless, Optionset 3 is consistently superior to all the other configurations for the Parr's maze. Even though the absolute flat case will eventually reach the goal faster and more consistently than any of the relative representation configurations, the HRL approach using optionset 3 will find the goal over 20 times faster in the first 50 epochs, and will do it with only 4% of the states needed by the absolute state representation.

More conclusions from these results will be discussed in Section 4.6.3.

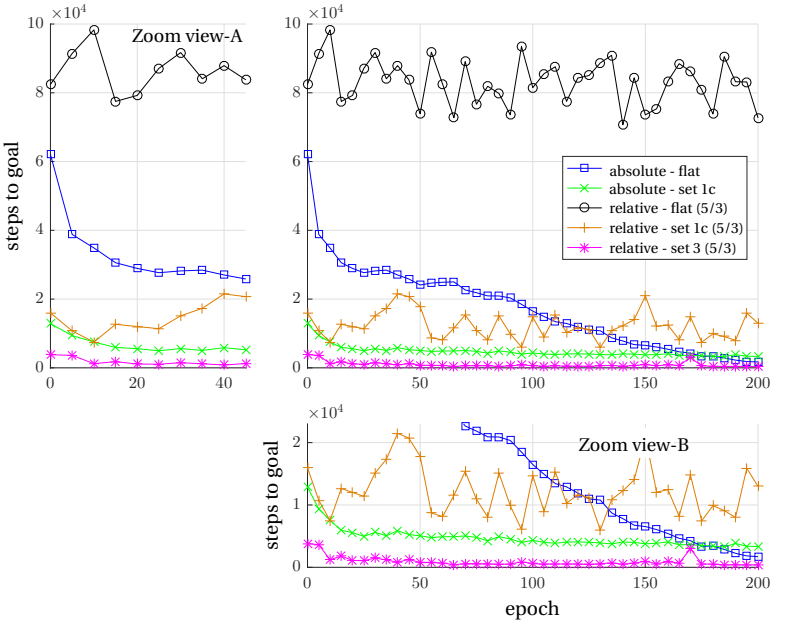


Figure 4.17: **Parr's maze**: absolute vs relative state representation ($d_s = 5/n_d = 3$). Averages over 10 runs.

4.6.2. FINAL LEARNED PATHS

In this section, the results from a single run are presented for each maze. These figures give a visually intuitive explanation of how the agent learning progresses throughout the training. The markers on the images in Figures 4.18-4.20 (a), represent all location states which were visited in select evaluation runs throughout the 200 epochs. The number of steps to the goal for that evaluation are indicated in the figure. The arrows on the larger (b) parts of the figures go into more detail by showing the direction of the heading. Each (b) plot is the same as the final path at epoch 200. The initial evaluation state is always in the top-left corner facing south.

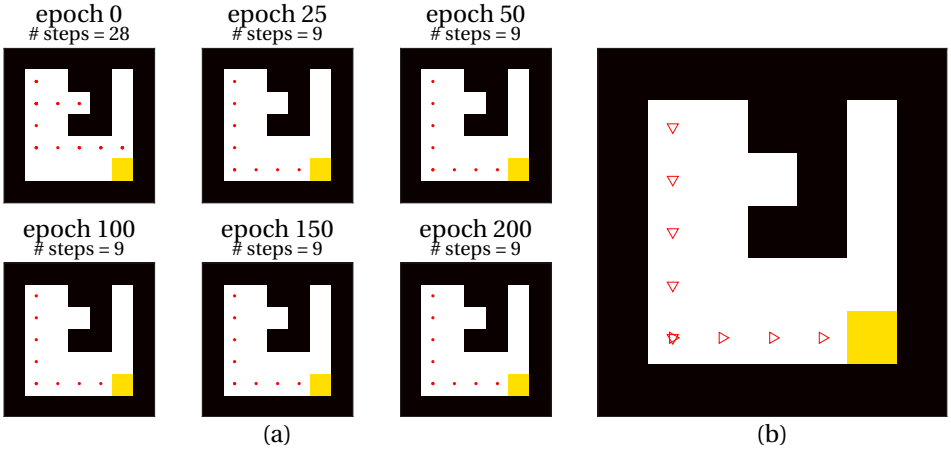


Figure 4.18: Performance progression of a typical sample run in the **Small maze** with relative state representation - ($d_s = 3/n_d = 3$) - optionset 1b

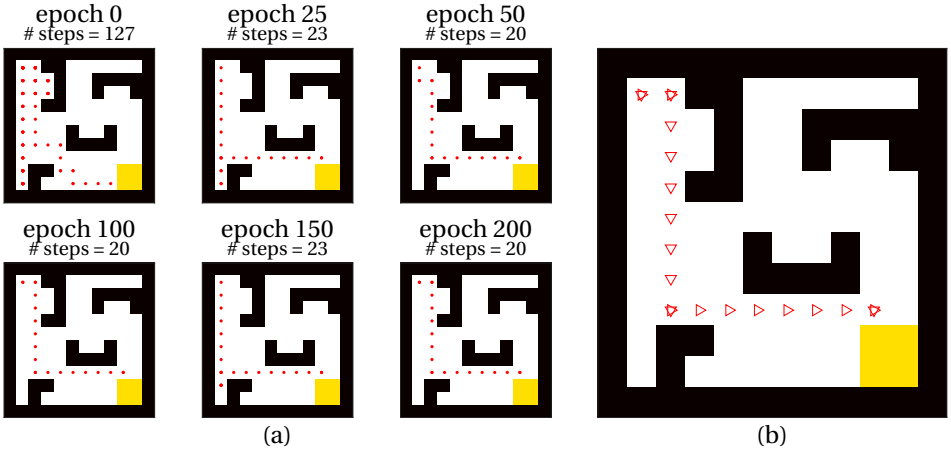
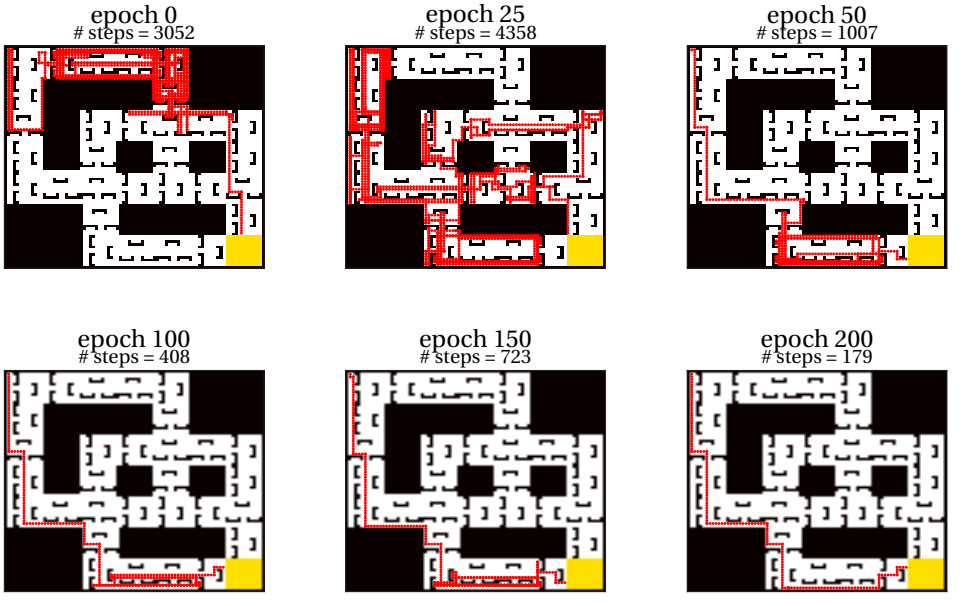
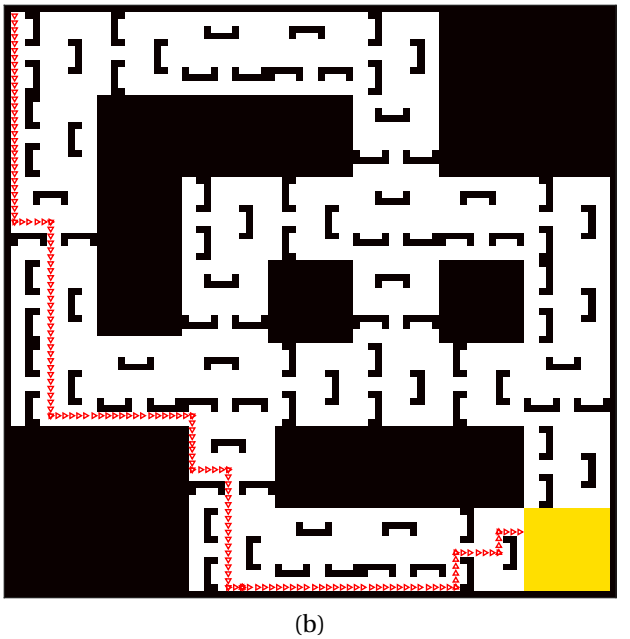


Figure 4.19: Performance progression of a typical sample run in the **Medium maze** with relative state representation - ($d_s = 3/n_d = 3$) - optionset 1c



(a)



(b)

Figure 4.20: Performance progression of the sample run with the *best* result (of 10 samples) in the **Parr's maze** with relative state representation - ($d_s = 5 / n_d = 5$) - optionset 3

4.6.3. RESULTS DISCUSSION

The trends and observations of relative state representation and this performance metric are now discussed.

By evaluating the Q-functions which are learned given different state representations, several take-away points have been compiled in the following topics discussed in this section:

- Understanding the (side)effects of relative state representation
- The trade-offs of relative state representation
- The benefits of Hierarchical approaches with relative state representation
- Guidelines for designing a relative state representation

EFFECTS OF RELATIVE STATE REPRESENTATION: NON-SMOOTH CONVERGENCE, AMBIGUITY, AND SUBOPTIMAL SOLUTIONS

Using a relative state representation will result in a non-smooth convergence as seen in the previous results. This "jumpy" behavior and a continued large standard deviation in later epochs occurs because of ambiguity. The performance can differ epoch to epoch based on which states were randomly visited in the trainings since the last evaluation run.

Say there are two locations in the gridworld, x_1 and x_2. Both have the same relative state, x_r, despite being geographically far away. The state, x_r is ambiguous with a level of 2. This kind of situation will happen easily in the large environment and not as often in the small maze environment. State x_1 is close to the goal and therefore no matter which primitive action is taken, the next state will have a relatively high $\max Q(s,a)$ value. State x_2 is not very close to the goal, and therefore no matter what primitive action is taken, the $\max Q(s,a)$ value will not be as high. Even if it turns out both grid locations have the same optimal action, the value in the Q-function will fluctuate depending on which state-action pairs happen to be visited during that training. The value of this state will also propogate its influence outward to the neighboring states as per the RL update law (Eq. (3.2)). Extend this situation to a state space where several ground states are represented by the same relative state and the optimal actions are conflicting. The problem then compiles with several geographical locations influencing each other in hard-to-predict ways.

This ambiguity arrangement will lead to not only "jumpy" behavior, but also to suboptimal convergence; where some states will give the correct greedy action and others not. It will also be possible to be stuck in action loops (ie. turning right then left infinitely) which will take time to accrue enough penalty to break out of this behavior when updating the Q-function during the evaluation. The more an ambiguous state-action value has been influenced in a negative way, the longer it will take to accrue the necessary penalty, all the while updating the Q-value in other locations which may be visited later in the evaluation run. This explains why some of the configurations actually perform worse than a random policy. The ambiguity arrangement of that state representation and maze is prone to Q updates which cause action loops which take longer to break than random choice. Getting trapped in an action loop is more likely in the larger mazes

since the likelihood of the evaluation path taking the agent by many of these ambiguity traps is also higher. In the Parr's maze, the flat case results did not learn despite increasing the state space size of the relative representation. It could be that increasing the state space size further could have led to an improvement, however the time required in the simulation was deemed too taxing.

Another observation which supports this claim of "ambiguity arrangement" importance, is that the $n_d = 5$ representation tends to perform better in the small and medium maze given the same state space size as the $n_d = 3$ case. Therefore, the ambiguity arrangement proves to be more favorable for the $n_d = 5$ representation in the small, medium maze, and optionset 1c in the Parr's maze, and to not make much of a difference for optionset 3 in the Parr's maze.

Based on the results and this explanation, the conclusion can be drawn that ambiguity *percentage* is not the only influencing factor. If there is a particular representation which has a more favorable ambiguity percentage but the ambiguous states are situated in an unfavorable way, the convergence of the Q-function will be affected and the performance will suffer. However, the flat Q-learning case is primarily influenced by this negative trait of relative state representation.

ADVANTAGES AND CHALLENGES OF RELATIVE STATE ABSTRACTION WITHIN REINFORCEMENT LEARNING

As previously discussed, relative state representation is necessary when no other state estimation can be obtained and advantageous in cases when an environment is too large for absolute state representation and limited state space size is desired.

The advantage of relative representation and its associated smaller state space is demonstrated in Section 4.6.1 with the comparison between relative and absolute state representations. When used in combination with HRL options, the relative representation is able to learn with a fraction of the states needed for the absolute representation. However, this comes at a cost.

The disadvantages of relative representation are the product of ambiguity, as discussed in the previous subsection. To combat these disadvantages, the challenge is to find a representation which is least prone to the pitfalls as well as an appropriate optionset for the task at hand. This challenge is not easy when applied to general problems where environment is assumed to be unknown; however, if knowledge of the problem task is available it can be used to guide the design of the approach.

EFFECTS OF HIERARCHY

For the relative state representation, the hierarchical reinforcement learning approach is vital. With absolute representation, the flat Q-learning RL performs the best in terms of final converged solution but with a trade-off of learning speed where HRL excels. It is a trade-off. However, with relative representation, there is no trade-off; the results show that the HRL options approach outperforms flat RL in every metric.

Optionset selection also proves to be an important aspect of relative as well as absolute representation. In the Parr's maze with certain state representations, the optionset selected can make the difference between learning and not learning. For example, in Figure 4.11, the flat case doesn't learn with any representation, optionset 1c can only learn using the representation with the largest state space, and optionset 3 learns well with all

the representations. In fact, optionset 3 performs better than any of the absolute representations do for the first 175 epochs. However, claiming superiority for relative over absolute state representation with optionset 3 is not a fair claim due to the "go around" option being unavailable in the absolute case.

The consistent superiority of optionset 3 in the Parr's maze infers that in large environments, the optionset with the longer extended actions performs the best. This is likely because taking extended actions bypasses the action loop traps. The poor performanc of the flat cases show that taking only primitive actions compounds the problems associated with ambiguity since updates are made at every time step. When actions are extended, there are fewer decisions made and therefore fewer updates and fewer chances for the Q-values of the state/action pairs to become obscured with its other ambiguous states. Even so, the best solution for optionset 3 in the Parr's maze (Figure 4.20) gets caught in one action loop trap which can be seen by the extraneous turns before continuing on the path towards the goal.

DESIGN GUIDELINES FOR A RELATIVE STATE REPRESENTATION

To overcome the challenges of relative state representation and still reap the benefits of reinforcement learning, the following are suggestions for formulating the state representation:

Use hierarchical reinforcement learning: Use available knowledge of the problem to create (or learn) helpful options. If the environment is large, longer extended-action options are beneficial.

Choose the representation with favorable ambiguity arrangement: This will change from task to task and may be hard to predict. In general, try to reduce the amount of extreme "ambiguous state value separation". For example, $n_d = 5$ performed better in most cases than the $n_d = 3$ representation with of the same state space size.

4.7. CONCLUSIONS

The content of this chapter explored the effect of state abstraction and state ambiguity on a guidance task solved with hierarchical reinforcement learning.

The extended action concept of *options* proved to be a capable approach for the task. For the relative state representations there is a clear benefit over "flat" Q-learning in all environments and all representations. However, HRL is not a perfect solution to ambiguity. Performance of a relative state representation fluctuates due to the ambiguity, never fully converging to a single solution. Sub-optimality depends on the ambiguity arrangement associated with the state representation or if the optionset selected is not equipped to overcome the ambiguity. Therefore, this chapter contributes a **guideline** for designing a relative state representation and optionsets for specific tasks.

With a favorable relative state representation and optionset, and **96% less states**, hierarchical reinforcement learning can perform on par with or better than RL or HRL with an absolute state representation in the Parr's maze. The flat Q-learning absolute representation is still the only configuration which, though slowly, finds the optimal solution. However, for most applications the benefit of improved performance early in training

outweighs the promise of optimality in the long run. In future work, a sensor suite could be assumed which combines absolute and relative state knowledge. A combined state abstraction or state/reward abstraction could prove to have benefits from both worlds.

This chapter concludes that state abstraction to a relative state representation creates an advantage of reduced state space but at the cost of the challenges associated with ambiguity. These challenges, however, can be mitigated using carefully thought-out state representation and hierarchical reinforcement learning with effective *options* selection. The new guidelines for state representation design can be applied to a reinforcement learning quadrotor agent, in a large scale GPS-denied environment, for effective and adaptive decision making in a guidance task, using only camera sensors. The end goal of implementation onto a UAV guidance task is now within reach and is the subject of the upcoming chapter.

5

SELF-TUNING GAINS OF A QUADROTOR USING POLICY GRADIENT REINFORCEMENT LEARNING

In the previous chapters, reinforcement learning was used to address problems within the scope of high-level guidance. This chapter demonstrates one way in which to implement reinforcement learning into a low-level control of a quadrotor.

Model-free reinforcement learning needs time to explore states and actions in order to find the best action for each state. When it comes to a low-level control policy, random unconstrained exploration of actions would inevitably result in unsafe scenarios for a quadrotor. Therefore, in this chapter a method is developed which uses a simple model to intelligently direct exploration. A “tried and true" PID controller is implemented as a low-level controller and reinforcement learning policy iteration is used to tune the gains. First, this method is validated in simulation using an F-16 aircraft model as a platform. The approach is then tested experimentally with a real quadrotor to self-tune the PID gains of the vertical control loop during a take-off maneuver.

The content of this chapter has been published in:

J. Junell, T. Mannucci, Y. Zhou, and E. van Kampen. Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, San Diego, CA, USA, 2016. [66]

5.1. INTRODUCTION

Autonomous flight of Unmanned Aerial Vehicles (UAVs) and, especially, Micro Aerial Vehicles (MAVs) is challenging in the realm of control for many reasons. Dynamics of the system can be difficult to model, the vehicles are often more susceptible to unforeseen disturbances, and they can have sensor or actuator limitations due to weight and size requirements. One method to address these problems is to create learning or adaptive controllers. Reinforcement learning (RL), for example, can make a system learn desired actions with little or no a priori knowledge of its dynamics or environment and can adapt to changing conditions [130]. For these reasons, RL has become a promising tool for improving autonomous flight in many different types of UAVs and MAVs [1, 18, 79, 139]. This chapter will focus on hybrid approach which uses a simple or incomplete model of a system in order to decrease the number of real-life trials needed in reinforcement learning policy improvement.

Policy improvement for a finite number of states and actions is an attractive method for policy search since it guarantees convergence to the optimal policy, π^* as well as an optimal value function, V^* for all states [130]. However, sweeps of the whole state set are required each iteration. Modern computers can now handle the computational load of large state sets but of course there are limits and methods have been explored to cope with high dimensionality [12, 73]. For continuous problems, function approximations have been used with success [38, 128]. Guarantees of local optimum policies are still present but a near-perfect model of a system is necessary. By using a crude model in simulation to do some of the heavy computational work, it is possible to decrease the number of real-life trials and still guarantee convergence to a locally optimal policy [2, 142].

This hybrid model-based approach is advantageous for vehicles which need local tuning of gains, since it is guaranteed to find the local optimum but not the global. The AR.drone 2 quadrotor, for instance, could benefit from this since the blades of the rotors can become less efficient over time, therefore decreasing the maximum amount of thrust available or adding additional actuator delay. An optimal policy can be found and used by some means but will shift over time. By using the old optimal gains as a starting point, the new optimum can be found with this approach. Another vehicle which could benefit is a flapping-wing MAV such as the Delfly. [1] [2] Efforts are being made to model this complex system [6, 24], however inconsistencies in the manufacturing make each individual vehicle unique. Hand tuning the gains of the controller gives desirable performance, however it is time consuming work. This approach could use the finely tuned gains from, say, vehicle A as a starting point, and as long as the newly manufactured vehicle B is not too different from A, the optimal gains for this new vehicle should be reachable with only a few real-life trials using a “good enough" model and policy gradient RL.

A background of reinforcement learning policy iteration and other necessary algorithmic information as it is applied to this problem is outlined in Section 5.2. The experimental setup with regard to how the policy improvement algorithm will be applied to the F-16 and quadrotor platforms is described in Section 5.3. The F-16 results and anal-

[1] TU Delft Robotics Institute. http://robotics.tudelft.nl
[2] http://www.delfly.nl/index.html

ysis are in Section 5.4. The quadrotor simulation results are found in Section 5.5. The setup and results from the real-life quadrotor flight tests are presented in Section 5.6. An overview of all the results is discussed in Section 5.8.

5.2. BACKGROUND

Since PID controllers are heavily relied upon in industry, there has been no shortage of methods which have tried to improve upon gain tuning. In this section, we take a brief look into the background of gain tuning, policy improvement reinforcement learning, and the hybrid approach which inspired this research.

5.2.1. PID GAIN TUNING

The performance of controllers for aircraft and other applications are often reliant on the tuning of their gains. While PID controllers are prevalent in many fields due to its simplicity and reliable performance, poorly tuned gains can detract from these benefits. In order to improve upon this control strategy, researchers throughout the last several decades have come up with a number of successful procedures or tools.

Rule-based gain tuning is one of the most simple approaches, where a set of rules are determined which can easily be followed. One of the oldest and well-known approaches is the Ziegler-Nichols method where gains can be calculated using measurements of the step response of the plant [147]. In other methods, if the plant's model is well known, gains can be tuned based on performance and stability criteria. Traditional examples include pole placement and designs based on gain and phase margin [31].

When an accurate model of the plant is not available, the options for gain tuning are less well-known. If the hardware for the system is available and easily tested, a plant model can be created using the input-output data from the tests and system identification methods. With the created model, the gains can then be tuned by traditional model-based methods. If the system cannot be tested, or is too complex to be easily modeled with system identification, there has been some research into model-free gain tuning and gain scheduling. A non-linear adaptive gain tuning method uses iterative feedback tuning [82]. Another approach called “model-free control" uses“intelligent" PID controllers (iPIDs) by replacing a complex mathematical model with what they call an “ultra-local model" which uses a piecewise function approximator to estimate the model dynamics [43].

The method proposed in this chapter is a hybrid approach which lies inbetween model-dependent and model-free and will now be introduced.

5.2.2. GRADIENT POLICY ITERATION IN REINFORCEMENT LEARNING

The basic framework of RL consists of a learning agent and its interaction with a finite Markov decision process (MDP). Let S and $A(s)$ be defined as finite sets of states and actions, respectively. $P(s'|s,a)$ is the state transition probability from state s to s' given action a, and R is the reward function. The policy, π, is a mapping between states and actions. The value function, $V^{\pi}(s)$, represents the value of being in state s given that policy π is being followed. The goal of reinforcement learning, in general, is to gain information about V^{π} using interaction with the environment in order to find an optimal policy from

which to determine desired actions. Policy iteration, iteratively improves upon V^{π} and π in phases as seen in Eq. (5.1) [130]. Where $\xrightarrow{E}$ denotes an evaluation phase of the policy to obtain $V^{\pi}(s)$ for all $s \in S$, and $\xrightarrow{I}$ denotes a policy improvement phase in which $\pi(s)$ is found for all $s \in S$.

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi^* \xrightarrow{E} V^* \tag{5.1}$$

Since $V^{\pi}(s)$ for every state in the finite state set is evaluated each iteration, this approach is guaranteed to converge to the global optimum, V^* and π^*. However, one can see how a large state set can lead to an unmanageable amount of computation to reach this optimum. Additionally, this guarantee only applies to discrete cases where as most real world problems are in the continuous domain. Therefore, this concept can be used with function approximation for the value prediction in order to alleviate the number of necessary evaluation computations in a continuous state problem. A simple and widely used parameter update method for function approximation is gradient-descent. In this case, the policy, π_θ is represented by parameters $\vec{\theta}_t = (\theta_t(1), \theta_t(2), ...\theta_t(N))$, and V_t is any differentiable function of $\vec{\theta}_t$.

$$V_t = f(\vec{\theta}_t) \tag{5.2}$$

With this, one can setup a parameter update rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \alpha \nabla_{\vec{\theta}_t} f(\vec{\theta}_t) \tag{5.3}$$

Where α is the stepsize and $\nabla_{\vec{\theta}_t} f(\vec{\theta}_t)$ denotes the partial derivative vector for any function f,

$$\nabla_{\vec{\theta}_t} f(\vec{\theta}_t) = \left(\frac{\delta f(\vec{\theta}_t)}{\delta \theta_t(1)}, \frac{\delta f(\vec{\theta}_t)}{\delta \theta_t(2)}, ..., \frac{\delta f(\vec{\theta}_t)}{\delta \theta_t(N)} \right) \tag{5.4}$$

The function f can also be denoted as a performance measure, ρ. The performance measure used for this research is the mean absolute error(MAE) evaluated at a fixed timestep over the run. The derivative vector will point in the direction the error decreases most rapidly.

This approach has been proven [128] to converge to a *local* optimal policy for the performance measure. [3]

Even though the local optimum is guaranteed, and even though it applies to continuous space using a function approximation, the optimal policy is solved for the *model* of the true system, with which the performance measure is evaluated. If the policy is learned on a model and intended for use on a real system, the guarantee is no longer valid unless the model is a perfect representation. Otherwise, the optimal policy can be learned on the real system if that is an option, but to find a crude derivative vector will require at least as many real-life trials as number of parameters in $\vec{\theta}_t$ for *each* iteration.

[3] The step-size, α, must satisfy the stochastic approximation conditions $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\lim_{k \to \infty} \alpha_k = 0$.

Since a perfect model doesn't exist and even an adequate model is sometimes not available, this method needs some adjustments in order to account for model inaccuracies. The policy iteration algorithm which works on continuous domain problems and doesn't require a perfect model is found in gradient decent policy improvement using inaccurate models. The background for this approach will now be discussed.

5.2.3. POLICY IMPROVEMENT USING INACCURATE MODELS

The use of gradient policy iteration is promising in that it guarantees to find a locally optimal policy for some performance measure, $\rho(\pi_\theta)$. However, this is only guaranteed if a perfect representation of the process is known and the performance measure is differentiable by the policy parameters, $\vec{\theta}_t$. The derivatives found are then used to update the policy parameters in the direction of steepest improvement. The computational cost accrues from the derivative calculations and policy evaluation. Policy evaluation is just one run per iteration to find the performance metric; however, the more policy parameters, the more derivative calculations are necessary and so this part can make up a large portion of the computation needs. Since only the direction of the derivatives is needed for the update, it is possible to find the direction, $\nabla_{\vec{\theta}_t} f(\vec{\theta}_t)$, using a model. Updates from the derivatives, along with policy evaluation from real-life trials can converge in the same way to a locally optimal policy with less real-life trials since the model has taken the workload of the derivative calculations.

In Abbeel et al.(2006) [2], an approach is presented which learns near-optimal policies for a non-stationary MDP described by $M = (S, A, T, H, s_0, R)$, and its inaccurate counterpart defined as $\hat{M} = (S, A, \hat{T}, H, s_0, R)$, where S and A are the state and action sets, respectively, T and $\hat{T}$ denotes the true and model-based process, respectively, H is the number of time steps, and R is the reward function. They explain the algorithmic approach using the following steps:

1. Initialize model for $i = 0$. $\hat{T}^{(0)} = \hat{T}$.
2. Find a locally optimal policy $\pi_{\theta^{(0)}}$ for $\hat{T}$ using a policy search algorithm.
3. Evaluate $\pi_{\theta^{(0)}}$ in the real MDP, M, and record the state-action trajectory and the performance, $\rho(\pi_{\theta^{(0)}})$.
4. Construct a new model, $\hat{T}^i$ by adding a time-dependent bias term outside of the control loop, so that the model outputs an estimated real system output from which to calculate derivatives. Specifically set $\hat{f}_t^{(i+1)}(s,a) = \hat{f}_t(s,a) + s_{t+1}^{(i)} - \hat{f}_t(s_t^{(i)}, a_t^{(i)})$ for all times, t.
5. Find the derivative vector $\nabla_{\vec{\theta}_t} \hat{\rho}(\pi_\theta)$ such that the real performance measure $\rho(\pi_{\theta^{(0)}})$ improves from the former iteration.
6. Solve for the next policy by a line search in the gradient direction. $\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \alpha \nabla_{\vec{\theta}^{(i)}} \rho(\pi_\theta)$. Use multiple step-sizes, α and choose the best performing policy as policy $\pi_{\theta^{(i+1)}}$.
7. If the line-search does not return an improved policy, then return the current policy and exit. Otherwise, $i \leftarrow i + 1$ and continue at step 3.

The steps are visualized in Figure 5.1, which shows the necessary experimental setup, where the *model* plant is represented by the simulation model MDP, $\hat{M}$ and the *real/true* system is represented by the MDP, M. In this paper, M is called the *true* system if it is a simulation or the *real* system if it is a real-life flight. The policy improvement analysis uses the bias from the $\pi_{\theta^{(i)}}$ evaluation flights along with many runs of the model at varying policies to find the partial derivative vector used for policy updating.

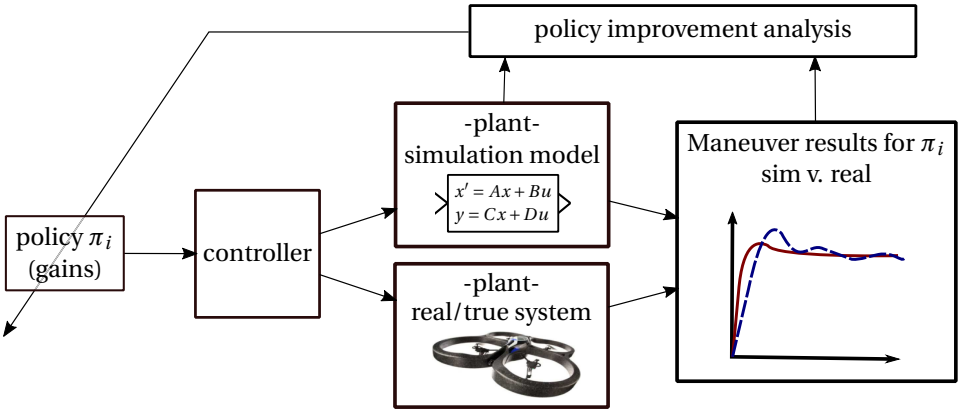


Figure 5.1: The policy, π_i, is implemented in both the system model and the real-life process. The data from the real-life trial will be used in collaboration with the mathmatical model to find a policy, π_{i+1} which improves the performance of the real system.

The research in this chapter takes this concept and expands it to a real-life quadrotor application using a very simple model. The extreme simplicity of the model gives a greater challenge to the approach as it may not capture the necessary derivatives of the real dynamics.

5.3. EXPERIMENTAL SETUP

Three experiments are presented in this chapter. The first two are fully within simulation and the last is with a real system. In this section, the descriptions of the mathematical models and true systems for each of the experiments are described along with controllers and explanations of the reasoning behind the decisions which were made.

5.3.1. NOTES ON NOTATION

To avoid symbol confusion between pitch angle θ and the parameters of the policy θ (used in the theory presented in Section 5.2), the following nomenclature will be used for the remainder of this chapter. The pitch angle for the F-16 will be denoted as θ, and the policy parameters denoted as $\vec{k}$, since in this case all the parameters are gains. For the F-16 four-gain controller, $\vec{k} = [k_q,\ k_\theta,\ k_\gamma,\ k_h]$, and for the quadrotor PID controller, $\vec{k} = [k_p,\ k_i,\ k_d]$. The performance metric, ρ, is the mean absolute error calculated with a sample time $\Delta T = 0.01s$, over a 1 minute simulation for the F-16 reference tracking task and $\Delta T = 0.04s$ over a 30 second run for the quadrotor takeoff task.

5.3.2. F-16 IN SIMULATION

The first system explored as a proof of concept is a model of the F-16 jet fighter. The choice to use this system was based on convenient access to a comprehensive non-linear model of the aircraft.

As described in Section 5.2, a *true* system and an inaccurate *model* are needed for the experiment. The model will be simulated thousands of times so that the real-life system only needs a few trials. However, a real-life flying F-16 was not readily available, even for a few trials. Therefore, it was decided that the F-16 non-linear model would serve as the true system and the linearized state space model of the longitudinal dynamics would be the mathematical model.

TRUE SYSTEM

The non-linear model used as the true system simulates the dynamics of an F-16 jet. The low fidelity model [4] as explained in a manual by Stevens and Lewis [112, 123] was simulated consistently while trimmed at 15000 ft altitude and 500 ft/s.

INACCURATE MODEL

The inaccurate model was created by linearizing the non-linear model flight dynamics about a trim condition. A linear time-invariant state space model of the longitudinal dynamics of the F-16 was created with the states: altitude h [ft], pitch angle θ [rad], velocity V [ft/s], angle of attack α [rad], and pitch rate q [rad/s]. The inputs are thrust F_T [lbs], and elevator deflection δ_{elev} [deg]. The state space equations can be seen in Eq. (5.5). Note that the actuator dynamics were not linearized and so a non-linear aspect in the elevator and thrust saturations remains in the inaccurate model. The actuator model from the F-16 model was implemented into the inaccurate model.

One aspect of interest about this policy improvement RL approach, is *how* inaccuracy of the mathematical model can affect the capabilities of the algorithm. To this end, a few linearized models were created about different operating points. Since the non-linear model true system is trimmed at 15,000/500 (altitude [ft] and velocity [ft/s] respectively), the model was first linearized about the same conditions and then about 15,000/600 and 15,000/400. These state space models are denoted through the chapter as '15000/400', '15000/500', and '15000/600'. The resulting state space models with the values of their A and B matrices are listed in Appendix C.

$$\dot{\vec{x}} = A\vec{x} + B\vec{u} \quad \text{where} \quad \vec{x} = \begin{bmatrix} h & \theta & V & \alpha & q \end{bmatrix}^T \quad \text{and} \quad \vec{u} = \begin{bmatrix} F_T & \delta_{elev} \end{bmatrix}^T \tag{5.5}$$

CONTROLLER SETUP

To explore the capability of this approach on high dimensional problems, controllers of 2-gains, 3-gains, and 4-gains were created. The greater the number of gains to be tuned, the higher the dimension of the optimization problem and therefore the more difficult it is to tune by hand or with other methods.

A 2-gain pitch controller was created to control the elevator deflection. The designated task is to track the pitch angle, θ, which is set to be a sinusoidal-wave or block-wave with respect to time. The relationship between states, outputs, and gains can be

[4] The low fidelity model (as opposed to the high fidelity model), does not include leading edge flaps

seen in Figures 5.2-5.3. Though there are actually 3 gains, the velocity gain which regulates the thrust is kept constant at $k_V = -50$, while the policy parameters to be improved, $[k_q k_\theta]$, control the elevator deflection, δ_{elev}. The performance metric is the mean absolute error (MAE) between the actual pitch angle and the reference over a 1 minute simulation.

A 3-gain controller (Figure 5.17), tracks a reference flight path angle, γ. A 4-gain controller (Figure 5.20), tracks an altitude reference. These controllers are displayed along with the F-16 experimental results in Section 5.4.

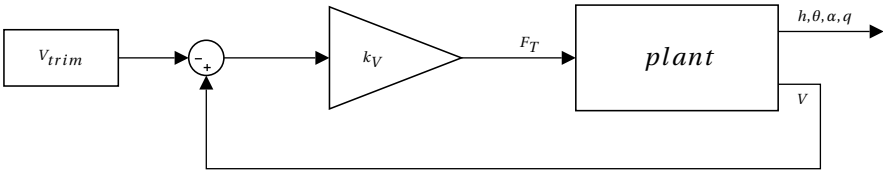


Figure 5.2: The velocity regulator, drives velocity towards its trim condition. k_V is constant.

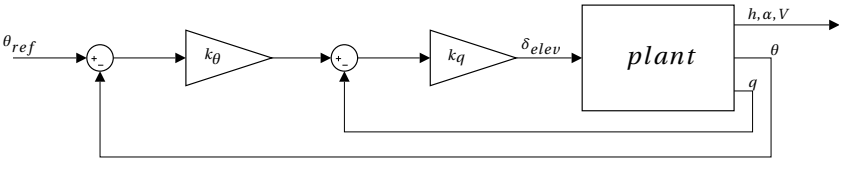


Figure 5.3: The two gain controller tracks the pitch angle reference, θ_{ref}

5.3.3. QUADROTOR IN SIMULATION AND REAL FLIGHT TESTS

One experiment in simulation and one on a real-life quadrotor are setup using a simple model of vertical dynamics. Both use the Paparazzi autopilot software (see Section 2.4.1) to control the *true/real system*. The quadrotor simulator built in to Paparazzi uses JSBsim, an open source Flight Dynamics Model, and is used as the *true system* in simulation. The real-life flight experiment uses an AR.drone 2.0 quadrotor as the *real system*. In these experiments, the quadrotor will perform a takeoff maneuver from the ground to an altitude of 3 meters. The details of the simple model created and the controller setup will now be described.

QUADROTOR SIMPLE VERTICAL MODEL

The simple model of the quadrotor is very crude. The only forces considered are forces of gravity downwards and the upward thrust (which is also the controller output). It is assumed that all rotors give equal thrust and therefore no roll or pitch will develop to tilt the lift vector. The state space matrices for the the plant model are derived from the equations of motion in Eq. (5.6), where F_T is the force of the thrust, m is the mass of the

vehicle, and g is gravitational acceleration. z is the axis of altitude, therefore $\ddot{z}$ is vertical acceleration with positive direction up.

$$\sum F = m\ddot{z} = F_T - mg \tag{5.6a}$$

$$\ddot{z} = F_T \frac{1}{m} - g \tag{5.6b}$$

$$\begin{pmatrix} \dot{z} \\ \ddot{z} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} z \\ \dot{z} \end{pmatrix} + \begin{bmatrix} 0 & 0 \\ 1/m & -g \end{bmatrix} \begin{pmatrix} F_T \\ 1 \end{pmatrix} \tag{5.7}$$

In a real flight there will be disturbances to the attitude of the quadrotor. Therefore, the horizontal controller will remain active during the flight tests and only the vertical loop gains will be tuned. The attitude angles should never get too big since the horizontal loop controller will be actively eliminating it. However, there could be repercussions for this omission in the model. For this approach to succeed, the simple quadrotor model only needs to have similar derivatives with respect to the policy parameters, θ, to find the direction of policy improvement for the vertical loop PID gains.

CONTROLLER SETUP

In order for the model to be effective at improving the policy parameters, the design of the controller must be the same for the model simulation as it is for the true system. Otherwise, the gains won't represent the same thing in both controllers and the wrong policy will be learned.

For the quadrotor experimental setup, the controller for the true system within the Paparazzi autopilot software is recreated within Simulink. The details of the controller can be found at the Paparazzi webpage [5] [6]

Since the simple model is only used for the vertical loop and assumes no attitude disturbances, there may not be a guarantee that the gains found will be optimal once horizontal movement is needed for other maneuvers. In future work, horizontal gains can be added to the policy parameters for a full controller policy improvement.

IMPLEMENTATION: FROM SIMULATION TO REAL-LIFE TESTS

The benefit of the Paparazzi autopilot is that the same controller software runs with the built-in simulator and the real-life flights. Switching from simulation to an actual flight needs no changes. The learned gains will be different because the simulator is not exactly the same as a real-life quadrotor, however experience shows the simulator to be an accurate predictor of the AR.drone vertical takeoff behavior.

5.3.4. LINESEARCH METHOD

The linesearch method deserves special note because it has a large impact on how quickly the optimal policy is learned and has not been described in detail in the method of

[5] Paparazzi Free Autopilot. http://wiki.paparazziuav.org

[6] To reproduce, use Paparazzi v5.4.2 stable version. vmode = "alt".

Abbeel et. al's version as described in step 6 of the algorithmic approach (Section 5.2.3). Furthermore, in the results section we will see that the linesearch method will play a big role in safety when it comes to learning policies for unstable systems.

For example, moving along the gradient line too conservatively, means more trials will be needed to converge, while too aggressive movement might result in an overshoot of the optimal solution or in a worst case: instability. A method should be used which can find a balance.

A concept developed for this research is demonstrated in Figure 5.4. As described in Section 5.2, the derivative direction, $\nabla_{\vec{k}}\hat{\rho}_i(\vec{k})$, over all the parameters, $\vec{k}$, is found via simulation of the *inaccurate* model, $\hat{T}_i$. The model changes each iteration since the bias from the last true-system trial at k_i is used to calculate the performance metric to determine the gradient.

Taking a pre-defined constant stepsize in that direction will lead to inefficiency since we aim to reach the optimum in as few steps as possible. Therefore, a new linesearch method is proposed where the gradient direction line is extended out until the error is zero. The corresponding parameter value is called the *zero-error gain*, denoted in the figure as $k_{\rho=0}$. Note that the example in the figure is simplified by making the policy, π_k only consist of one parameter, $\vec{k} = k$. Normally, this will be a multidimensional problem.

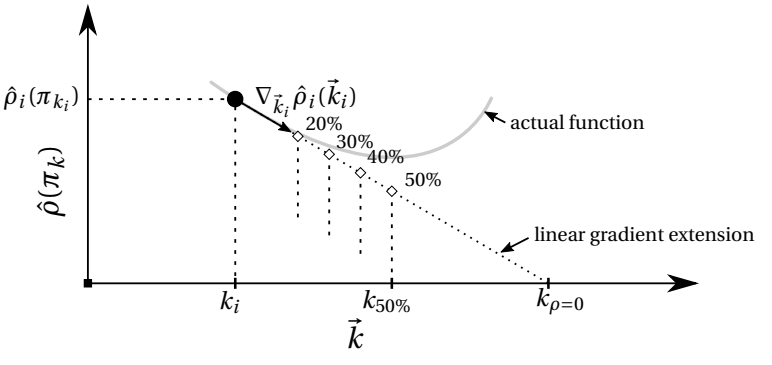


Figure 5.4: Example of the linesearch method. The concept developed for this research takes the gradient direction, $\nabla_{\vec{k}_i}\hat{\rho}_i(\vec{k}_i)$ for iteration i, and extends it out linearly until the error, ρ is equal to zero. A percentage along that line will be chosen by the linesearch method and the corresponding parameter values, $\vec{k}$ are used for iteration $i+1$. Notation: $\vec{k}$ represents all the parameters in policy π_{k^i}, but is shown here as a scalar for simplicity.

Since the actual function is non-linear and a gain set resulting in $\rho = 0$ is not possible, the next iteration's parameter value should lie somewhere between the current value and the zero-error gain value. How far to move along the line so as to not require too many iterations, and to not overshoot the optimal point is a key challenge. To solve this challenge, two linesearch methods are used, depending on the faults of the model, the task and the stability issues inherent in the task. A description of those linesearch methods now follow.

SIMULATION-BASED

The simulation-based linesearch method was developed for this problem to find the optimal solution as quickly as possible. It uses the advantage of the hybrid approach by using the inaccurate model to simulate several policies along the gradient line, $\nabla_{\vec{k}}\rho(\vec{k})$. The policies to simulate are determined by percentage along the line with a higher density of simulations nearby the current policy since the model becomes less accurate the further it goes, due to the bias term being calculated with policy π_{k_1}.

In tasks where there are no safety risks for the true system, or if the inaccurate model predicts the safety hazards of the true system with relative accuracy, then this simulation-based linesearch method will decrease the amount of true system trials by taking bigger steps toward the optimal solution. However, since the model is by definition inaccurate, it is risky to follow its suggestions so far away on the line from the last policy. If safety is an issue, a more conservative linesearch method should be used.

The simulation-based method was used for the quadrotor takeoff task (Sections 5.5-5.6) and initially for the F-16 pitch angle tracking task (Section 5.4) before determining a more conservative approach was needed.

STEPSIZE-BASED

Where the simulation-based linesearch can lead to quicker convergence, it can be dangerous if the task has safety or instability issues. Therefore, a more conservative linesearch can be used. The stepsize-based linesearch moves a smaller percentage of the line distance towards the zero-error gain and can use a consistent stepsize, or a stepsize which decays over iterations. The initial stepsize and decay rate is a matter of tuning which is a major pitfall of this method. The designer must choose between either a slow convergence or a potential overshoot of the optimum which could mean danger.

For the F-16 task, a linear decay rate was used, meaning that each iteration of a true system trial, the stepsize decreased by a constant rate. As a measure of insurance, the model was simulated at the new gain set and if it appeared that the performance would be significantly worse, then a smaller stepsize would be taken. The parameters of the linear decay used will be discussed in the results section.

END CONDITIONS

In order for the policy improvement to eventually stop, some end conditions must be determined. The desirable outcome is to stop exactly when the local optimum is reached in the fewest amount of true system trials. However, it might be beneficial to stop the optimization before the optimum is found if the cost of the trials is greater than the added benefit of the performance improvement.

For each task, the search stops if:

1. A local minimum has been reached. This is true if neither direction along the gradient line results in a performance improvement.

2. The improvement rate is too slow. The last n trials have improved by less than some task specific limit.

3. Other task specific conditions discussed further in the result sections.

5.3.5. TABLE OF EXPERIMENTS

Table 5.1 summarizes the experiments which will be presented in the upcoming results section.

The *platform* column indicates if it was a test with the F-16 or the Quadrotor platform. Each test required an inaccurate or simple model which is indicated in the third column.

For the F-16 experiments, descriptions such as '15000/500 sim' specify a model where the state space matrices were found by linearizing the non-linear *true* model about the trim conditions 15,000 [ft] altitude and 500 [ft/s] velocity. The inaccurate models vary in the velocity trim condition where as the true system of the F-16 is consistently represented by the nonlinear model with trim conditions 15000/500. Furthermore, there are 2 different types of control tasks performed and 3 controllers which informs the number of policy(gain) parameters to be learned. Each time, 2 linesearch methods are compared.

Tests 5 and 6 for the quadrotor are the same except for the true system. A simple model of the vertical dynamics was created to simulate a takeoff maneuver with a 3-gain PID altitude controller. The only change was the true system, where the Test 5 uses the built-in paparazzi (pprz) simulator and Test 6 uses a real-life AR.Drone2 quadrotor.

More details about all these parameters are described in Section 5.3.2 and Section 5.3.3 for the F-16 and AR.Drone2 quadrotor respectively. More information about the linesearch methods are found in Section 5.3.4.

Table 5.1: Table of experiments

no.	Platform	Inaccurate model	True system	Task	# of policy parameters	Linesearch method(s)
1	F-16	15000/400 sim	nonlinear sim	sinusoid	2-gain (θ-ctrl)	stepsize & sim-based
2	F-16	15000/400 sim	nonlinear sim	block-wave	2-gain (θ-ctrl)	stepsize & sim-based
3	F-16	15000/500 sim	nonlinear sim	block-wave	3-gain (γ-ctrl)	stepsize & sim-based
4	F-16	15000/500 sim	nonlinear sim	block-wave	4-gain (alt-ctrl)	stepsize & sim-based
5	Quadrotor	simple mdl	pprz sim	takeoff	3-gain(PID)	sim-based
6	Quadrotor	simple mdl	AR.Drone	takeoff	3-gain(PID)	sim-based

5.4. F-16 SIMULATION RESULTS

The first task we analyze for the F-16 is to track a reference sine wave in pitch angle, θ. This task corresponds to the 2-gain controller introduced in Figure 5.3 of Section 5.3. For the following results, a simulation based linesearch was used.

In Figure 5.5, the results show that the policy improvement algorithm is working as it should. The arbitrarily chosen initial gains of $\vec{k} = [k_q,\ k_\theta] = [-10,\ 10]$ produce a non-optimal response which is then iteratively improved upon until the locally optimal gains are found. The figure shows that the response has been improved so that it matches more closely with the reference sine wave. However, when the results are more closely scrutinized, we see that some undesirable behavior has occurred.

In Figure 5.6 we can see the run history of the gains and the performance metric as it converges to its locally optimal value. The optimal gains found were from the 5th true-system trial and the policy improvement algorithm finished after the 8th trial since it wasn't improving significantly anymore. As seen in the top plot, the performance metric, ρ, calculated as the Mean Absolute Error (MAE), is not consistently improving. On trials number 2, 4, and 6, the ρ of the true system increases to a very large error while the inaccurate model still improves from the previous trial.

Even though in the end, the approach works to find a good policy, this kind of behavior is not desirable because the exploration process should also be safe to test. In order to explain this behavior, an in-depth analysis of the responses with the 2-gain controller was performed.

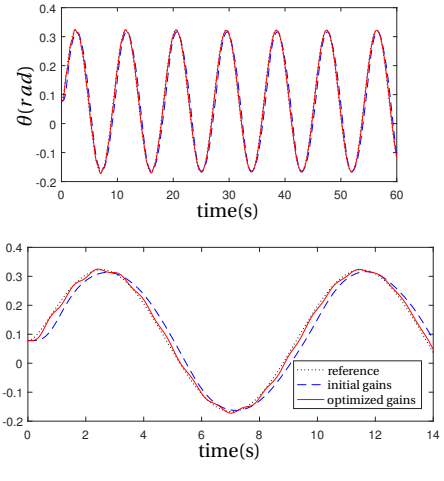


Figure 5.5: Time response of the policy improvement algorithm run on the true system for the initial gains of $\vec{k}_0 = [-10,\ 10]$ and the optimized gains. (*(left)*) full 60 second run, (*(right)*) zoomed-in to see tracking improvement.

5.4.1. ANALYSIS OF THE 2-GAIN PITCH ANGLE CONTROLLER

The first step to analyze the policy improvement results is to see what is happening in the true system trial when it is responding with a high error. Trial number 2, with the high

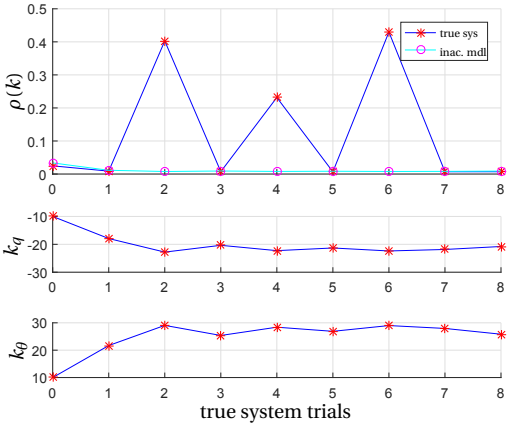


Figure 5.6: Performance and gain history of the policy improvement algorithm run from the initial gains of $\vec{k}_0 = [-10,\ 10]$. The gradient of the inaccurate model is driving the gains toward a policy where the true system has an unstable response.

error as seen in Figure 5.6, has a time response as seen in Figure 5.7. The gains for this run are $\vec{k} = [-22.7,\ 29.0]$. We can see from the response that the true-system response is unstable while the inaccurate model is still able to track the reference signal well.

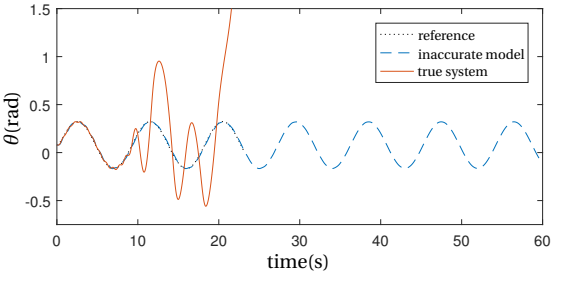


Figure 5.7: Time response of inaccurate model and true system with gains of Trial # 2 of the policy improvement algorithm history shown in Figure 5.6.

It can be concluded that the true system becomes unstable at gains which do not cause instability in the inaccurate model. To validate this conclusion and to learn to what extent this is a problem, a few approaches were used. Simulations were run through whole sweeps of gain values for both models and plotted as surface plots. Figures 5.8 - 5.9, show the results. For the range $k_q = [-500,\ 0)$ and $k_\theta = (0,\ 50]$, a simulation was run and the performance metric plotted. In total, each model was simulated 5,151 times to find the relationship between policy and performance. The error is depicted in a color gradient from 0 to 0.05 so that the gradient can be seen in the low-error region.

The orthogonal view is seen in Figure 5.8. Here we see clearly an “instability cliff" where the error increases significantly due to an unstable response. The inaccurate

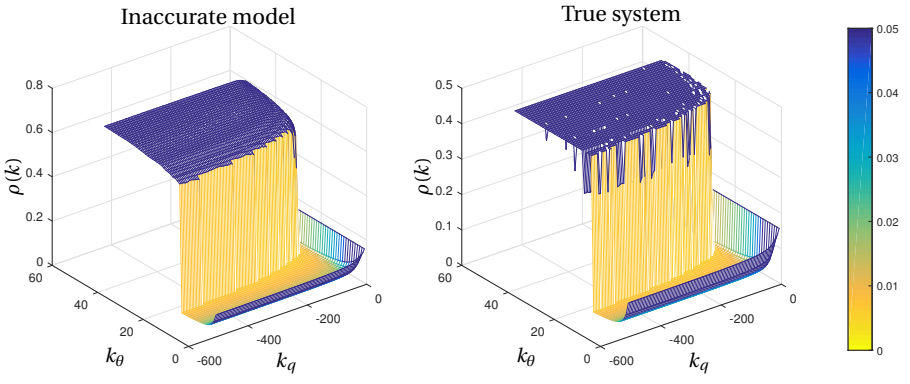


Figure 5.8: A surface plot of the performance metric as a function of the two policy parameters, $\vec{k} = [k_q,\ k_\theta]$.

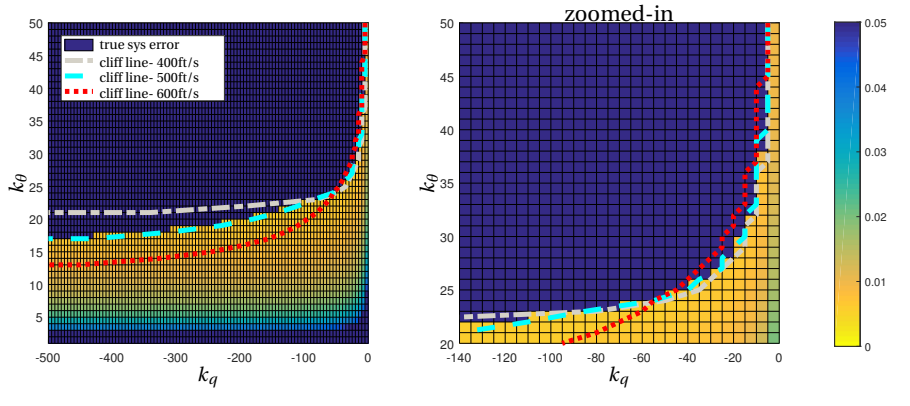


Figure 5.9: Top view of surface plot with lines overlaid representing the “cliffs" of the inaccurate models linearized about 400ft/s, 500ft/s, and 600ft/s respectively. *(right)* zoomed view

model doesn't actually become unstable, but has a sharp transition to a generally bad performance. The simple model does not become unstable due to the model inaccuracies incurred during linearization. The true-system does become unstable and the simulation must stop. In the cases that the simulation was stopped due to instability, the performance metric value is no longer meaningful and is therefore plotted in the surface plot as a value just above the highest error value.

In Figure 5.9(*left*), we see the top view of the same data showing more clearing where the “unstable zone" begins for the true system model. As a general trend, the error decreases with increasing k_θ, until it reaches a point where the controller leads to unstable behavior. The lines overlaid on the plot represent the “cliff" of the inaccurate models which were linearized about different trim conditions: 15000ft altitude and a velocity of 400ft/s, 500ft/s or 600ft/s. We can see that the “cliff" is reached at a different point by each of the inaccurate models and the true model. The points where the true system becomes unstable at an earlier gain than the inaccurate model are where the problem

lies.

Since the optimization is gradient-based using the *inaccurate model*, the policy improvement algorithm will suggest to move to the edge of the cliff where the error is lowest, even though the true model has already transitioned off the cliff to instability. Figure 5.9(*right*) is a zoomed in version to show that none of the inaccurate models are a perfect representation of the true model regarding the “cliff" location. Even the model which was linearized about 500ft/s (the same as the true model trim condition), has a gain region where it will have an excellent performance and the true system will be unstable. This conflict results in the gradient of the inaccurate model advising the wrong direction for an improved policy.

PROPOSED SOLUTION

For this method to be effective, the inaccurate model cannot have an inaccuracy which leads the policy to instability. One possible solution could be to impose a stability margin condition and calculate it with the inaccurate model. However, depending on how inaccurate the model is, it will not guarantee stability of the true system. Furthermore, it could impose unnecessary restrictions which can prevent the optimal solution of the true system from being explored.

Another solution could be to decrease the step size when changing the policy along the gradient line. In that way, logic can be implemented to stop the policy iteration when the improvement becomes negligible, or if the error has already reached a desired limit set a priori. The policy improvement run from the results above, would then find its final gain set in only 2 or 3 trials.

BLOCK WAVE TASK

Not all tasks will have the same problem as the sinusoid tracking task. Only tasks where the performance gradient pushes the policy towards instability and where the true system transitions to instability earlier than the inaccurate system.

The block wave task is the second task looked into for this algorithm. The time response of the block wave is shown in Figure 5.10 for the true system as well as the inaccurate models linearized about 15000ft altitude and velocity 500ft/s or 400ft/s. The block wave is only one period so as to demonstrate response to a step up and a step down.

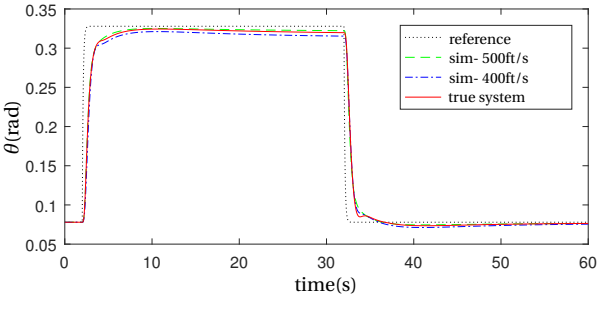


Figure 5.10: Block wave task time response of the true system and 2 inaccurate models using gains of [-50, 2]

We analyze the surfaces plots of the block wave task in Figure 5.11, in the same way as with the sinusoid. The left figure is the orthogonal view and the right is the top view with superimposed lines representing the transition “cliff" of the inaccurate models. There is indeed still a distinct cliff, but it is more gradual than with the sinusoid, therefore creating a valley where a gradient-based optimization will perform better. Furthermore, even the worst case gains are still marginally stable for the true-system as can be seen in Figure 5.12 where the time response of the worst case gains (in this gain range) is plotted.

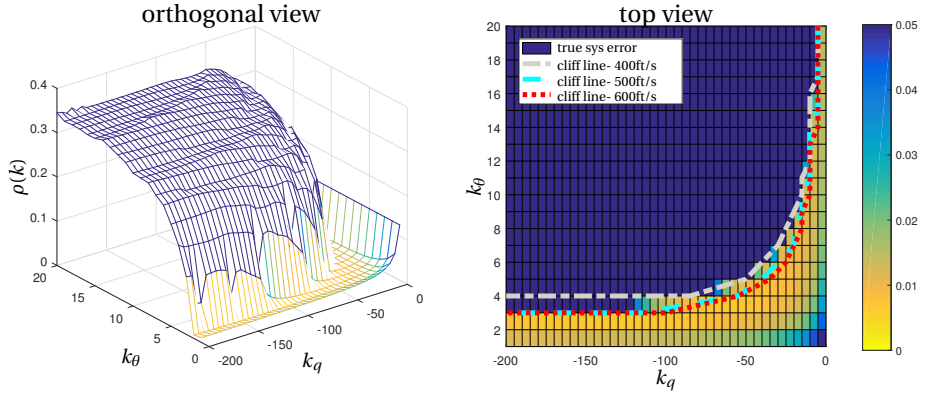


Figure 5.11: Block wave task surface plots. (*left*) orthogonal view, (*right*) top view

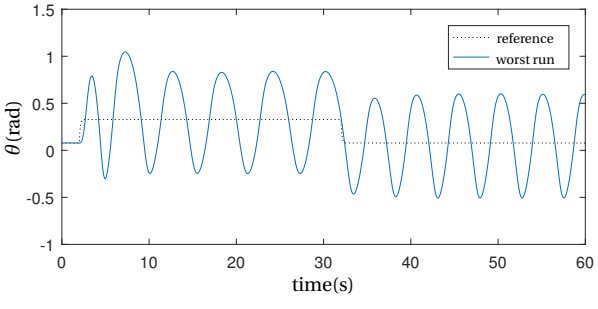


Figure 5.12: Block wave task time response with the worst gain policy found during the gain sweeps shown in Figure 5.11. For the block task, even the worst run remains stable. gains $\vec{k} = [-180,\ 20]$

In the remainder of the F-16 simulation results sections, the sinusoid task results will be presented for the 2-gain scenario only since it has already been shown that this task is not a good case study for this algorithmic approach. The 2-gain results are maintained to show how possible solutions can be used to make this approach still tangible. The block wave task results for the policy improvement algorithm will be presented for a 2-gain, 3-gain and 4-gain controller.

5.4.2. RESULTS: 2-GAIN PITCH ANGLE CONTROLLER

The 2-gain controller was used on two tasks: to track a sinusoid and to track a block wave. The results that follow show the ability of the reinforcement learning policy improvement algorithm to improve the gain policy of the controller for the two tasks.

For each task, results are shown for different linesearch methods and starting from different initial gain conditions. Since the algorithm finds only a local –and not a global– optimum, the initial policy will have a great effect on the final results. The policy for the pitch angle controller consists of the gains $\vec{k} = [k_q,\ k_\theta]$, as seen in the controller block diagram in Figure 5.3. The initial gains are found by hand-tuning and in the 2-gain case the surface plots could also be used for guidance. For demonstration purposes in this paper, the initial gains shouldn't have a response which is too good or there will be no room for improvement, but it also must be a stable solution for the algorithm to succeed in learning.

For the pitch angle sinusoid task, the initial gains are $\vec{k}_0 = [-10,\ 10]$ and $\vec{k}_0 = [-10,\ 5]$. For the pitch angle block wave task, the initial gains $\vec{k}_0 = [-10,\ 5]$ and $\vec{k}_0 = [-50,\ 2]$ were selected. These gains were used since they all resulted in a reasonable performance; however, one of the policies is closer to instability than the other, therefore giving the opportunity to show how initial conditions can pose challenges for linesearch properties which must balance a desire to reach the optimum quickly without overshooting it or risking instability.

The 15000ft-400ft/s inaccurate model is used to produce the results shown. This is used because it is slightly more inaccurate than the 500ft/s model and therefore is more of a challenge.

SINUSOID TASK

The previous section already discovered the limitations with the sinusoid task. One proposed method to help avoid an unstable trial is to use a more conservative linesearch method, or use a priori knowledge of the system to determine an end condition based on the desired performance metric.

Figure 5.13 shows the error and policy history results of the policy improvement algorithm when using a stepsize-based linesearch and starting from different initial policy conditions. The stepsize-based linesearch will approach the (local) optimum point more slowly and conservatively, however it still does not guarantee that the optimal will not be overshot to the instability zone. Figure 5.14 shows a comparison of linesearch methods starting from the same initial gain. In the figure and in the data in Table 5.2, we ascertain that the stepsize-based linesearch takes 4-6 steps to reach the performance which was reached in the first step of the simulation-based linesearch. However, the simulation-based linesearch also tries an unstable policy much earlier. The stepsize based linesearch for initial policy $\vec{k}_0 = [-10,\ 10]$ eventually has an unstable trial, but the initial policy $\vec{k}_0 = [-10,\ 5]$ does not. The reason for this is that the later is further away from the "cliff" and written into the linesearch properties is an inherent slowing down after several trials and an end condition which stops the run when the improvement rate $\frac{d\rho}{dt}$ reaches a limit.

Any of the linesearches can be tuned so that it stops when it reaches a performance metric which has been determined to be sufficient a priori. As an example, see Table 5.2

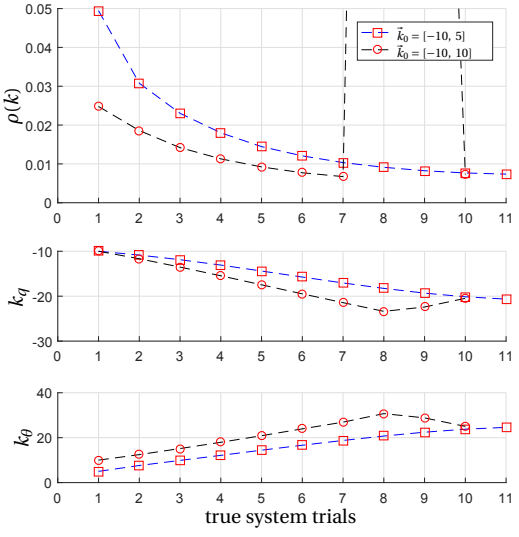


Figure 5.13: Performance and gain history of policy improvement algorithm run using the stepsize-based linesearch method. Comparing different initial gain policies.

Table 5.2: Number of true system trials in the policy improvement algorithm run and the resulting performance metric given different linesearch methods and end conditions. $^{\triangle}$ indicates a run which became unstable.

Case	ρ^*(ave)	initial policy $[-10, 10]$	initial policy $[-10, 5]$
w/o threshold			
sim-based	0.0068	$9^{\triangle}$	$6^{\triangle}$
stepsize-based	0.0071	$10^{\triangle}$	11
w/ threshold			
sim-based	0.0079	2	3
stepsize-based	0.0092	5	8

where the results have been compiled with and without such a threshold. Without a threshold, an end condition of $\frac{d\rho}{dt} \leq 0.001$ was used to end the run. Runs which had any unstable true system trials are denoted with a triangle next to the number of trials in the run. If a user decides that a threshold of $\rho \leq 0.01$ is sufficient as a stopping condition, then none of the trial runs from the results above would have become unstable. Furthermore, the number of true system trials would be dramatically decreased. Most of the improvement is won in the first few iterations anyway, especially with the simulation-based linesearch. The performance improvement stagnates when it approaches the local optimum, so the performance benefit is usually small in the last trial runs. In the table, we see that the average best performance, ρ^*(ave), is still good when the run is stopped at the threshold.

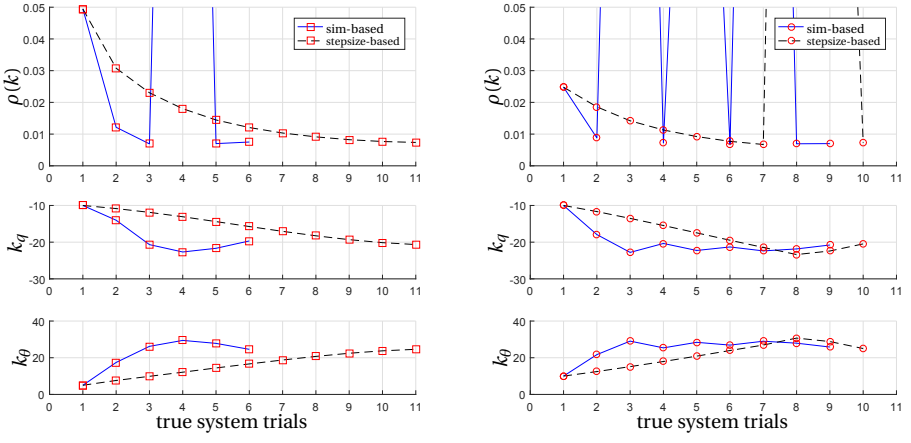


Figure 5.14: Performance and gain history of policy improvement algorithm run for the sinusoid pitch angle tracking task. Comparison of the 2 different linesearch methods. (*left*) Initial gains, $\vec{k}_0 = [-10,\ 5]$ and (*right*) Initial gains, $\vec{k}_0 = [-10,\ 10]$.

5

It is not very useful to use this threshold concept as a possible solution since knowledge of the system was used to generate the threshold value. In most cases, we assume that the desired performance metric is not known a priori. The result is presented because it demonstrates that using knowledge of the system can help with the effectiveness of this approach.

BLOCK WAVE TASK

The same policy improvement trials were conducted with a block wave task. The same improvement rate end condition was used as with the sinusoid task. No performance threshold as an end condition was implemented (only improvement rate related end conditions) since instability was shown to not be present in the state space explored for the surface plots.

The results in Figure 5.15 show the performance metric and gain histories of each of the runs. For initial gains $\vec{k}_0 = [-10,\ 5]$, both linesearch methods have the greatest improvement in the first step and then improves slightly over the next trials. The initial gains $\vec{k}_0 = [-50,\ 2]$ start out with a better performance and after the improvement in the first step the simulation-based linesearch is unable to find a better solution. The stepsize-based linesearch also struggles to find a better solution after the first step but finds a policy slightly better. This means that the initial policy was near a local optimum. The results are summarized in Table 5.3.

The initial and resulting optimal time response from the initial gains $[-50,\ 2]$ and simulation-based linesearch can be seen in Figure 5.16. From the plot we can see that the solution which optimizes the performance metric based on the mean absolute error (MAE) may not necessarily be desirable. The optimized policy indeed results in a smaller MAE, but with more overshoot, which may not be the most desirable solution for a real application. This result demonstrates the importance of a well-defined performance metric for the task.

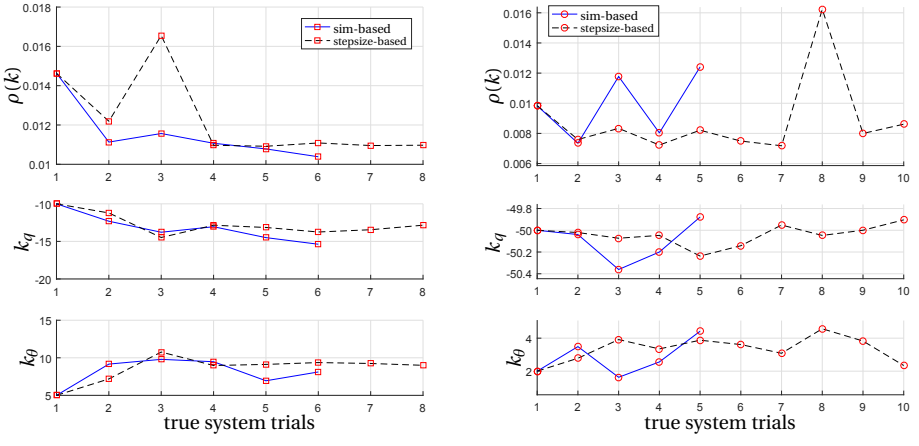


Figure 5.15: Performance and gain history of policy improvement algorithm on the 2-gain pitch angle block-wave task using inaccurate model linearized about 15000ft/s altitude and 400ft/s velocity. Comparison of 2 different linesearch methods from 2 different initial gain policies. (*left*) Initial gains, $\vec{k}_0 = [-10,\ 5]$. (*right*) Initial gains, $\vec{k}_0 = [-50,\ 2]$.

Table 5.3: Results of policy improvement algorithm with block wave task comparing different linesearch methods and initial gain policies.

Case	ρ^* (ave)	initial policy $[-10,\ 5]$	initial policy $[-50,\ 2]$
w/o threshold			
sim-based	0.0089	6	5
stepsize-based	0.0181	8	10

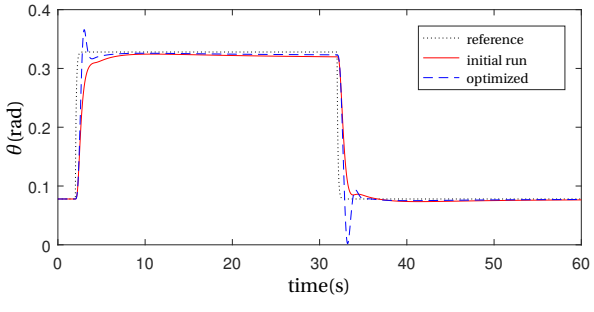


Figure 5.16: Initial and optimized time response of the 2-gain pitch angle controller block-wave task using the policy improvement algorithm with simulation-based linesearch. $\vec{k}_0 = [-50,\ 2]$.

5.4.3. RESULTS: 3-GAIN FLIGHT PATH ANGLE CONTROLLER

Controllers with higher dimensions become more difficult to tune by hand since each of the gains influence the others. Therefore, it is important to test this approach on tasks of

higher dimensions.

The first example is of a 3-gain flight path angle controller which tracks the flight path angle, γ. See the controller block diagram in Figure 5.17. The controller is similar to the 2-gain controller with one additional feedback loop for γ. The flight path angle is calculated using known states with: $\gamma = \theta - \alpha$.

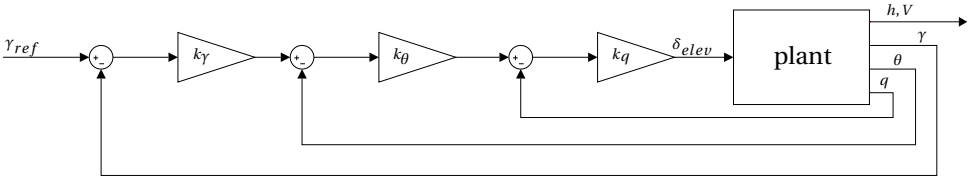


Figure 5.17: The three gain controller tracks the flight path angle reference, γ_{ref}

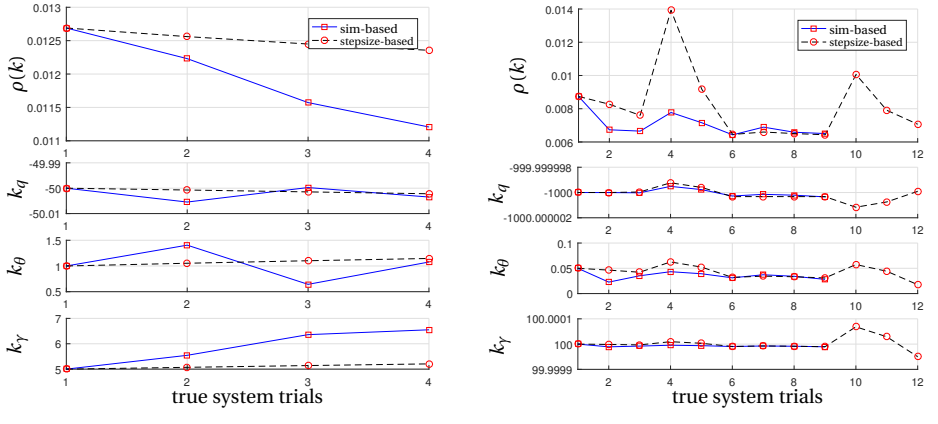


Figure 5.18: Performance and gain history of policy improvement algorithm run on 3-gain flight path angle tracking task using an inaccurate model linearized about 15000ft/s altitude and 500ft/s velocity. Comparison of 2 different linesearch methods from 2 different initial gain policies (*left*) Initial gains, $\vec{k}_A = [-50,\ 1,\ 5]$. (*right*) Initial gains, $\vec{k}_B = [-1000,\ .05,\ 100]$.

Once again, the results from two linesearch methods and two initial policy cases are presented. The initial policies are $\vec{k}_A = [-50,\ 1,\ 5]$ (Case A) and $\vec{k}_B = [-1000,\ .05,\ 100]$ (Case B) and were found through hand-tuning to find two cases where the policies were in a different gain envelope region, stable, and had room for improvement.

Figure 5.18(*left*) shows that there was a small improvement from both initial policies. The region of policy $\vec{k}_A$ has a gradual gradient with k_γ as the most prominent influence. The algorithm stopped due to a slow improvement rate end condition being met. The increases in error indicate that the region around policy $\vec{k}_B$ is most likely near a point of disagreement between the true system and inaccurate model.

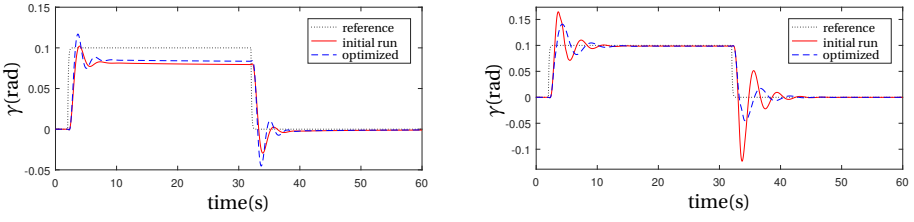


Figure 5.19: Initial and optimized time response of the 3-gain flight path angle tracking block task using the policy improvement algorithm with simulation-based linesearch. (*left*) Initial gains, $\vec{k}_A = [-50,\ 1,\ 5]$. (*right*) Initial gains, $\vec{k}_B = [-1000,\ .05,\ 100]$.

Figure 5.19 shows that from the initial gains $\vec{k}_A$, the direction of improvement corresponds to an increase in the overshoot of the step response and an effort to decrease the steady state error. There is obviously further room for improvement in case A, however it is a limitation of the algorithm that it will not find the global optimum. The local optimum for Case A is different than the local optimum in Case B, and therefore it is not surprising that the optimized policy for Case A is not even as good as the initial policy for Case B. Furthermore, the policy was still improving in Case A when it was stopped, but it was improving too slowly.

From the initial gains $\vec{k}_B$, the direction of improvement corresponds to a decrease in the overshoot of the step response. The example shows an already good initial policy and how the algorithm is good for finding a small local improvement

5.4.4. RESULTS: 4-GAIN ALTITUDE CONTROLLER

Exploring one dimension higher, a 4-gain altitude tracking task was examined. See the controller block diagram in Figure 5.20. Once again, a feedback loop was added for altitude, h.

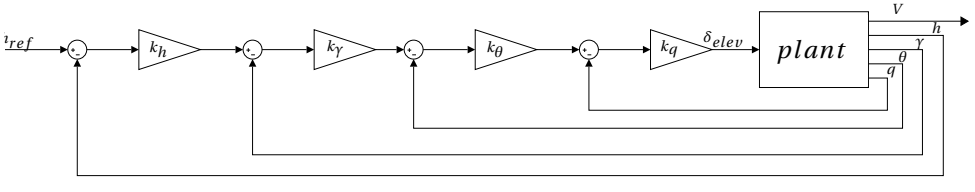


Figure 5.20: The four gain controller tracks the altitude reference, h_{ref}

Again, the results from two linesearch methods and two initial policy cases are presented. The initial policies are $\vec{k}_A = [-100,\ 2,\ 0.1,\ 0.005]$ (Case A) and $\vec{k}_B = [-500,\ 0.5,\ 0.01,\ 0.1]$ (Case B) and were found through hand-tuning to find two cases where the policies were in a different gain envelope region, stable, and had room for improvement.

Figure 5.21(*left*) and Figure 5.22(*left*), Case A, shows a good response, which may be overdamped, is optimized to increase its rise time but also its overshoot for a better MAE performance. However, an average passenger (or pilot in the case of the F-16) would

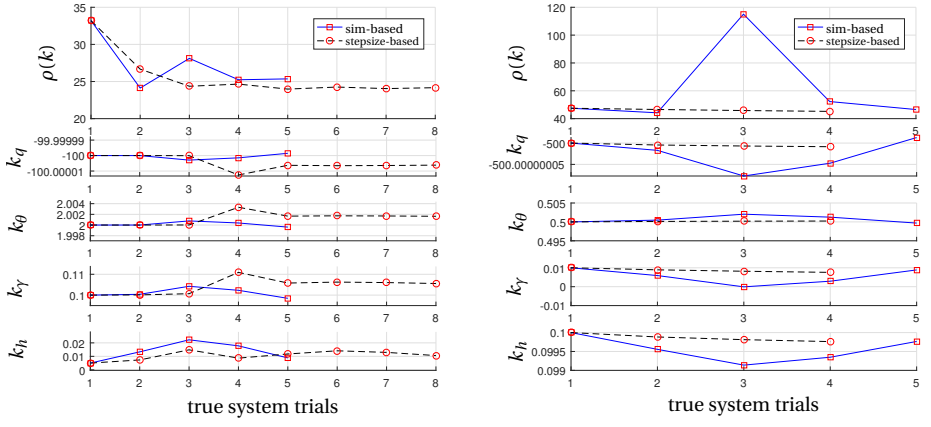


Figure 5.21: Performance and gain history of policy improvement algorithm run on block-wave 4-gain altitude tracking task using inaccurate model linearized about 15000ft/s altitude and 500ft/s velocity. Comparison of 2 different linesearch methods from 2 different initial gain policies (*left*) Initial gains, $\vec{k}_A = [-100,\ 2,\ 0.1,\ 0.005]$. (*right*) Initial gains, $\vec{k}_B = [-500,\ 0.5,\ 0.01,\ 0.1]$.

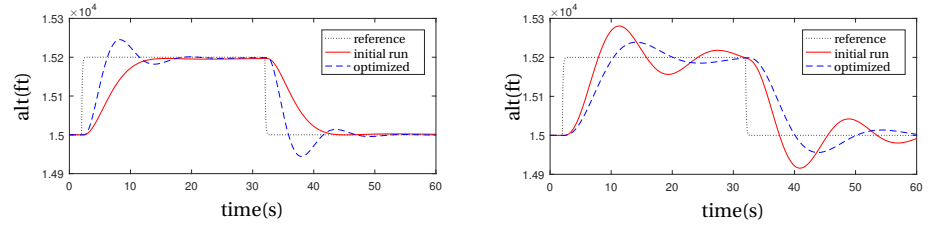


Figure 5.22: Initial and optimized time response of block-wave 4-gain altitude tracking task using the policy improvement algorithm with simulation-based linesearch. (*left*) Initial gains, $\vec{k}_A = [-100,\ 2,\ 0.1,\ 0.005]$. (*right*) Initial gains, $\vec{k}_B = [-500,\ 0.5,\ 0.01,\ 0.1]$.

prefer the initial policy. This again highlights the importance of a relevant performance metric in which to optimize.

In Case B, shown in Figure 5.21(*right*) and Figure 5.22(*right*), the initial policy gives a step response with large oscillations. Even though the error did not improve by much, the optimized solution response is quite different. It has a slower rise time but is more damped and would also be a more comfortable flight for the passenger.

5.5. QUADROTOR SIMULATION RESULTS

The results for the quadrotor simulation policy improvement will now be presented. The simulator used to produce data for the true system trials, is that of JSBsim within the Paparazzi autopilot software. Therefore, it is vital that the Simulink controller which is used for analysis of the derivatives is exactly the same as the controller implemented in Paparazzi. The validation of the simulink controller against the Paparazzi vertical-loop controller will be presented first, followed by the results for the policy improvement based self-tuning gains.

5.5.1. SIMULINK CONTROLLER VALIDATION AGAINST PAPARAZZI AUTOPILOT

Before implementing the policy improvement algorithm, it must first be ensured that the simulink controller (with which the gradient is found) is the same as the Paparazzi controller which controls the real-life trials. The full simulink model of the vertical loop controller and the simple quadrotor model can be seen in Figure 5.24.

Since we cannot validate the model with the simple quadrotor model inside the closed loop, we must validate the controller in sections. The inputs and feedback signals were taken from a Paparazzi trial, so as to confirm that the simulink model will have the same outputs out of the reference model as well as the feedforward and feedback commands. The validation of same outputs can be seen in Figure 5.23.

Small disagreements between the Paparazzi and simulink model are due to differences in sample times and round off error. The larger disagreement at the beginning of the feedback command (fb_{cmd}) plot is due to the summed error integrator not being initialized with the value from Paparazzi. Over time, the plots merge and the discrepancy doesn't effect the performance of the policy improvement algorithm.

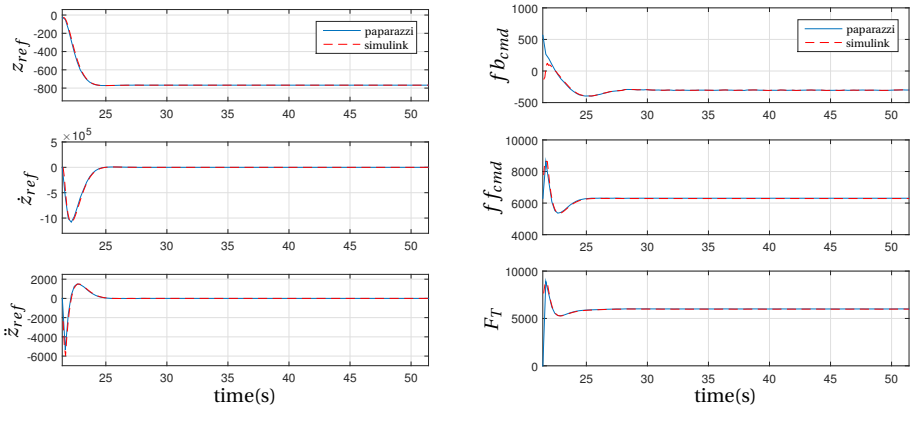


Figure 5.23: Validation that simulink controller produces the same outputs as Paparazzi. The simulink model was validated using known inputs from a Paparazzi trial. The Paparazzi trial started with its takeoff command at 21.4 seconds. note: positive z is downward. The units for these plots are from bitwise operations and therefore do not represent a physical value. That the two outputs match is the only important take away.

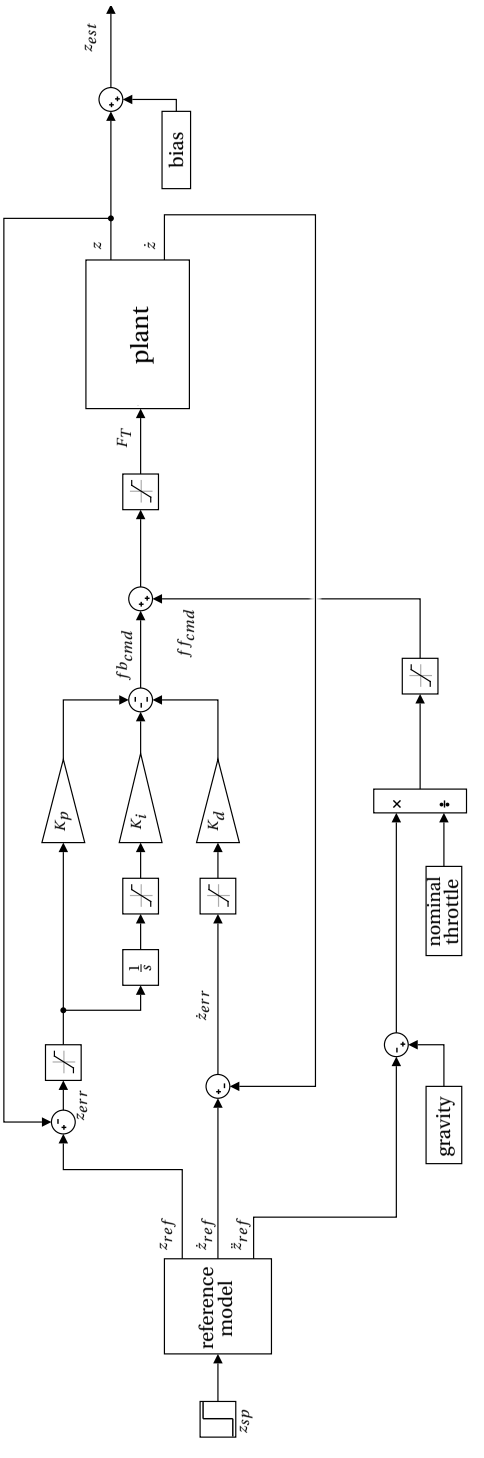


Figure 5.24: The quadrotor's full controller used for policy improvement analysis. It is a replica of the controller from Paparazzi. A bias (determined from previous trials) is added to the final model output to more accurately represent a real-life trial.

5.5.2. POLICY IMPROVEMENT

The successful policy improvement to a local optimum is demonstrated in Figure 5.25 and the corresponding tables in Table 5.4. The figures and tables show the results from 2 different initial gain sets. The plot on the left starts with Trial 1, where the policy is the default gains used within the Paparazzi software for the AR.drone. The take off response is far from optimal as it overshoots its intended altitude and then slowly descends to its setpoint, z_{sp},. After each iteration of policy improvement, the gains move toward the locally optimal policy and settle with the policy in Trial 5. The ending criteria is met when the performance fails two iterations in a row to improve by greater than 5% from the previously best solution, or if 7 true system/real-life trials have been performed. The plot on the right shows what can be considered a near global optimal. First, the optimal policy solution for the model is found using the Nelder-Mead simplex search. These gains are used as an initial policy, Trial 1. The solution found with Nelder-Mead is a very good solution with only $0.04m$ of overshoot and quick settling time. The error is calculated against the altitude setpoint, z_{sp}, however the controller follows the reference trajectory, a 2nd order system. The altitude reference has an error of $0.0913m$ and therefore, we can expect this error to be the best possible performance. The best policy found with our policy improvement algorithm, Trial 3, results in an error of $0.0940m$.

Table 5.4: Gains used in each iteration of the policy improvement results. These values correspond to the plots in Figure 5.25. (*left*) Initial trial uses default Paparazzi gains. (*right*) Initial trial uses resulting gains from Nelder-Mead simplex search solution.

Trial #	error(m)	k_p	k_i	k_d
Trial 1	0.3775	283	13	82
Trial 2	0.2006	285	44	82
Trial 3	0.1596	286	102	82
Trial 4	0.1733	499	177	42
Trial 5	0.1521	324	116	75

Trial #	error(m)	k_p	k_i	k_d
Trial 1	0.1006	1624	1356	195
Trial 2	0.0988	1644	1365	155
Trial 3	0.0940	1649	1365	175
Trial 4	0.0946	1652	1365	195

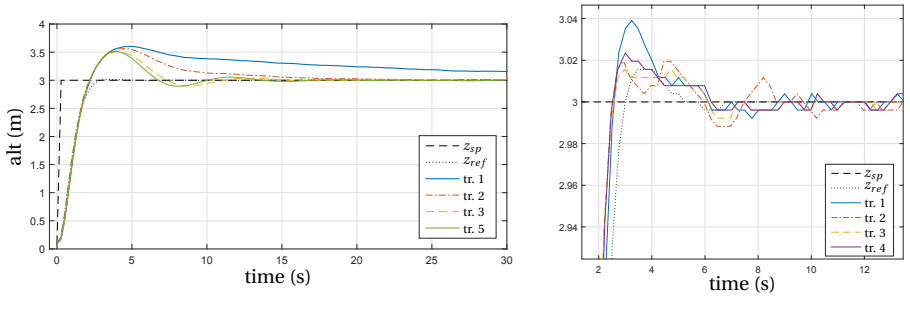


Figure 5.25: Take off trajectory of true system flights using policy improvement in Paparazzi simulation. The figures correspond to the values from Table 5.4. (*left*) Initial trial uses default Paparazzi gains [283, 13, 82]. (*right*) Initial trial uses resulting gains from a Nelder-Mead simplex search solution [1624, 1356, 195]. Zoomed-in to overshoot.

5.6. QUADROTOR FLIGHT TEST RESULTS

In the flight test, the quadrotor performs a takeoff maneuver from the ground to an altitude of 3 meters. The performance metric to be minimized is the mean absolute error of the altitude against the altitude setpoint. The necessary state information is collected from the Cyber Zoo simulated GPS estimate. Paparazzi, an open source autopilot, is used for the control with the policy parameters represented by the gains of the vertical loop PID controller. Data from the flights is used to run the policy improvement algorithms offline.

The resources used in this process, including the AR.drone 2.0, the TU Delft Cyber Zoo, and Paparazzi autopilot, can be reviewed in Section 2.4.1 and can also be found in Junell et al. [67].

The results of the flight test experiment are now presented.

5.6.1. FLIGHT TEST RESULTS

The results for the quadrotor policy improvement with real-life flights will now be presented.

Table 5.5: Gains used in each iteration of the policy improvement results. These values correspond to the plots in Figure 5.26. (*left*) Initial trial uses default Paparazzi gains. (*right*) Initial trial uses the resulting gains from a Nelder-Mead simplex search solution.

Trial #	error(m)	k_p	k_i	k_d
Trial 1	0.5102	283	13	82
Trial 2	0.2537	286	60	81
Trial 3	0.2265	301	133	65
Trial 4	0.2652	301	135	59
Trial 5	0.2483	301	134	62

Trial #	error(m)	k_p	k_i	k_d
Trial 1	0.1218	1624	1356	195
Trial 2	0.1249	1599	1366	130
Trial 3	0.1278	1612	1361	163

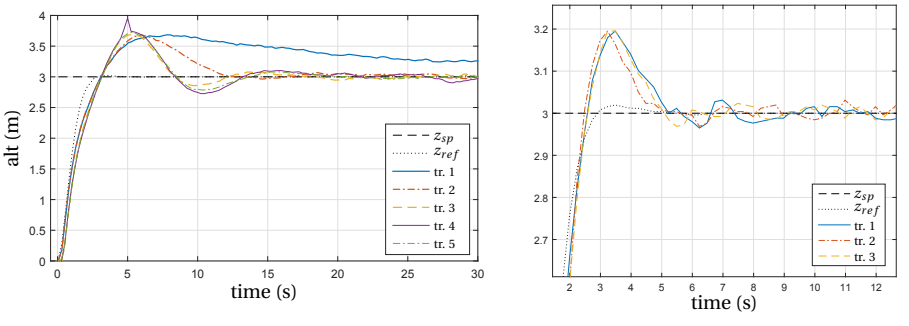


Figure 5.26: (*left*) Policy improvement starting from initial policy of default Paparazzi gains [283, 13, 82]. The little peak in Trial 4, is not physically possible and is likely a small glitch in the tracking system where perhaps tracking was momentarily lost. (*right*) Policy improvement starting from gains optimized by Nelder-Mead search [1624, 1356, 195]. Zoomed-in to overshoot.

The structure of the test setup and the results are the same as the quadrotor simula-

tion. Policy improvement with a simple model and real-life trials was performed starting from 2 different initial policies, $\pi_{\theta^{(0)}}$. The first series of flights started from the default gains for the AR.drone in the Paparazzi autopilot software. The second series of flights used an initial policy which was found by the Nelder-Mead simplex search to be the optimal policy for the simple model. The results of these two series are shown in Figure 5.26, and the corresponding table, Table 5.5.

Compare the takeoff trajectory of the default Paparazzi gains, Trial 1, in real-life flight (Figure 5.26(*left*)) to that of the same flight in simulation (Figure 5.25(*left*)) and the same kind of behavior is seen. The quadrotor overshoots its setpoint and then slowly descends towards it. After one iteration of the policy improvement algorithm, a new policy is found. Trial 2 performs with about half the error as Trial 1 in both the simulation and real-life case. Trial 3 shows improvement but not as substantial. Trial 4 is not an improvement, which means the step size along the derivative direction was too large. So far, the simulation and the real-life flights show the same progression. At Trial 5, they differ. Trial 5 is along the same derivative line as Trial 4, just not as far. For the simulation, it was an improvement but for the real-life trial it is again not as good as Trial 3.

The reason that Trial 5 doesn't perform as well as Trial 3 could possibly be because Trial 3 is already the local optimum, too large of a step size was taken and Trial 5's policy is indeed not as good, or external influences have influenced the results. To take a look at the last possibility, an experiment was conducted to see what effects the battery life and variability have on the performance metric. This variability can be inherent in the quadrotor or from external disturbances.

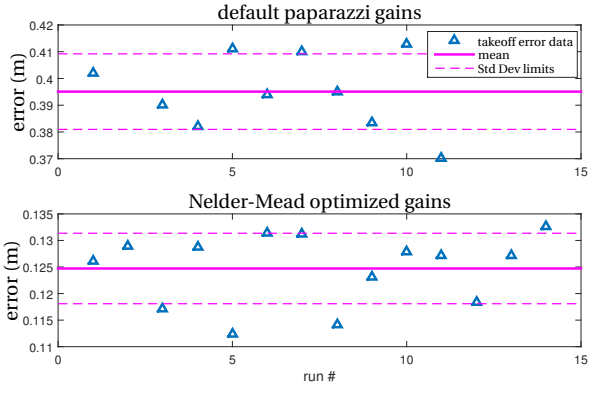


Figure 5.27: Multiple takeoff maneuvers were performed successively with the same gains during the life of a single battery on the quadrotor. Tests were performed at the default Paparazzi gains [283, 13, 82] (*top*) and the Nelder-Mead solution gains [1624, 1356, 195] (*bottom*). The results of this test shows that 1) the battery level does not influence the performance of the takeoff, and 2) the standard deviation of the performance metric depends on which gains are being used and can be used to find the statistical significance of the real-life tests.

Multiple takeoff maneuvers were performed successively with the same gains during the life of a single battery on the quadrotor. Tests were performed at the default Paparazzi gains [283, 13, 82] Figure 5.27(*top*) and the Nelder-Mead solution gains [1624, 1356, 195] Figure 5.27(*bottom*). The results show that the battery life does not have an affect on the

performance metric. Since the real-life experiment is non-deterministic, it is beneficial in the analysis of the results to know the standard deviation of a takeoff maneuver. The standard deviation of the takeoff performance with the default paparazzi gains is $0.0141m$ and the standard deviation with the Nelder-Mead gains is $0.0066m$. Therefore, it can also be concluded that fine-tuning of the gains on a real-life system is rather difficult with this method given the variability.

While this approach is able to improve the policy from the default paparazzi gains, it fails to improve upon the Nelder-Mead optimized gains. However, in Table 5.5(*right*) it is shown that the trials after the initial trial are all within the standard deviation of a takeoff maneuver. The initial policy is likely already too good of a performer and therefore, is not able to improve by an amount greater than the variability.

5.7. RESULTS OVERVIEW

The final results from all the policy improvement experiments have been compiled in Table 5.6. The percent improvement of the performance metric from the initial gainset to the optimized gainset is shown for each of the experiments.

Reference the corresponding *no.* in Table 5.1 for a full description of the experimental setup. An expanded table of results can be found in Appendix C.

Table 5.6: Compiled results

no.	initial gains, $\vec{k}_0$	linesearch	improvement(%)
1	*F-16: θ-ctrl, sinusoid*		
	[−10 5]	sim	86
		stepsize	85
	[−10 10]	sim	73
		stepsize	73
2	*F-16: θ-ctrl, block-wave*		
	[−10, 5]	sim	29
		stepsize	25
	[−50, 2]	sim	26
		stepsize	27
3	*F-16: γ-ctrl, block-wave*		
	[−50, 1, 5]	sim	12
		stepsize	2
	[−1000, 0.05, 100]	sim	26
		stepsize	26
4	*F-16: altitude-ctrl, block-wave*		
	[−100, 2, 0.1, 0.005]	sim	27
		stepsize	28
	[−500, 0.5, 0.01, 0.1]	sim	7
		stepsize	5
5	*quadrotor: takeoff simulation*		
	[283, 13, 82]	sim	60
	[1624, 1356, 195]	sim	7
6	*quadrotor: takeoff real-life*		
	[283, 13, 82]	sim	56
	[1624, 1356, 195]	sim	0

5.8. CONCLUSIONS

Learning through interaction with the environment is one of the key features of model-free reinforcement learning but can be time consuming or dangerous when learning from scratch on a real system.

This chapter has presented a hybrid approach which is not completely model-free but which utilizes a simple model in a way which assumes model inaccuracies and compensates accordingly. Simulation of the simple model takes on much of the computational burden in order to quickly converge to a good policy with only a few real-life trials. This method has been shown to be particularly effective with policy improvement where

the policy consists of the gain set of a real-life quadrotor. A local optimum can be found within 1-3 real-life trials and further improvement is only limited by the variability inherent in a real-life application.

The policy improvement approach has been implemented on an F-16 model in simulation as well as an AR.drone 2 quadrotor in simulation and in real-life.

The experimental results from the F-16 simulation show that for a 60 second simulated run, the algorithm is able to improve upon the initial policy. Improved policies produce a time response with a better performance metric for a sinusoid wave and a block wave reference. In this case, the performance metric is the mean absolute error between the reference and the response. Tracking a sinusoid for the pitch angle, θ, can be improved upon by up to 80% depending on the initial policy and the tracking of a block-wave for pitch angle can be improved by up to 29%, based on the experimental results.

The number of true system (real-life) trials that are needed depend on how strict the end conditions of the algorithm are. Having strict end criteria means that the policy will converge closer to the local optimum, but will take more trials to get there. Easily satisfied end criteria means that less trials will be run, but the final solution may be short of the optimal.

From the F-16 results of the sinusoid tracking task, a problem arises which gives insight into an important limitation of this approach. The policy improvement algorithm can be driven to policies which lead to instability as a result of the model inaccuracies. Such safety implications will limit the use of this approach for certain applications where instability cannot be tolerated. In this case, some knowledge of the system is required. Knowledge of the system can be used in different ways to prevent unstable policies from being selected. For example, performance thresholds, or system stability criteria can guide restrictions in the linesearch for safe exploration. Conservative exploration may lead to a suboptimal policy, but the results in this chapter indicate that most of the improvement is gained in the first

Two linesearch methods are proposed and compared: One which conservatively moves in small stepsizes along the gradient, and another which uses simulation results of the inaccurate model to determine the next policy. Moving along the line conservatively means more trials are needed to converge; while aggressive movement might lead quickly to overshooting the optimal solution and possibly to instability. Furthermore, the conservative approach still doesn't guarantee stability. The balance of this trade-off should be determined by the needs of the task.

If instability is not an issue for the task or if an unstable trial can be tolerated, the simulation-based linesearch is the best and fastest option to find a (locally) near-optimal solution in only a few trials, usually with the largest part of the improvement in only one trial after the initial one. This is using the inaccurate model to its fullest potential for the fastest learning.

The simulation and real-life quadrotor take-off experiments also shows promising results and draws its own conclusions. The performance metric of mean absolute error over a 30 second run of a takeoff maneuver is improved by the reinforcement learning policy improvement algorithm.

By starting from two different initial gain sets, it is shown that a locally optimal pol-

icy can be found for both and that the initial gain set effects the performance of the final policy since it is only optimized locally using a gradient descent based method. The Nelder-Mead simplex search on the simulated simple model provides a starting policy with a better performance than the default gains from Paparazzi: reducing the initial performance metric by 4 times in the real-life trial and therefore proving to be an effective method for finding an initial policy from which to start the policy improvement algorithm. When starting in the convex region of the global optimum, it is guaranteed that the near-optimal solution will be found. How near it settles to the optimum depends on the ending criteria.

The initial policy will also effect the ability of the algorithm to improve the performance. Policies which have more room for improvement, can benefit more from this method. The simulated quadrotor which started from the default paparazzi gains improved the performance by 60% while the quadrotor which started from the Nelder-Mead optimized gains improved by only 7% upon its initial performance. For the real-life quadrotor, the algorithm improved the performance by 56% starting from the paparazzi default gains, and was not able to improve the performance at all when starting from the Nelder-Mead optimized gains. This is due to the initial policy being so close to the optimum that the difference may likely lies within the variability from external disturbances.

From the results of the real-life quadrotor, an end criteria for the algorithm can be determined. Fine-tuning of the gains on a real-life system are shown to be non-beneficial since the performance can be more affected by externally influenced variability than by the gains used. By finding the standard deviation of the task in question, an end condition in the form of improvement rate limits can be calculated and extraneous trials eliminated. Taking this into account, the best policy solution is found for the quadrotor task after only 1-3 real-life trials.

Even though the conclusions of this approach imply some limitations, there are still many applications where this approach would be useful. The original vision for this type of local optimization was to fine-tune the gains of the Delfly flapping wing MAV. Every Delfly vehicle is slightly different due to small variations in manufacturing and what is an optimal gainset for one may be slightly different for another. As long as the gains found to be optimal for Vehicle A are within the same convex region as those of Vehicle B, it should give a good enough starting point for the optimal gains of Vehicle B to be found within 3-5 trials. Furthermore, there will be some knowledge of the attainable performance of the vehicle and that knowledge can be used to guide the linesearch and the ending conditions of the policy improvement algorithm. Therefore, the original vision for this approach is still intact.

6

TRANSFER LEARNING OF A QUADROTOR – FOR A NON-MARKOV TASK

The previous chapters demonstrated the learning and optimization capabilities of various reinforcement learning approaches in order to address the challenges within autonomous guidance or control in real-life flight. This chapter combines those separate approaches onto a single quadrotor platform in order to test reinforcement learning knowledge transfer within a non-Markov honeybee task.

6.1. INTRODUCTION

The use of MAVs in real-life applications, to solve real-world problems, is growing in demand as the technology becomes more widely known and accessible. The field of machine learning in robotics is more popular than ever as a potential pathway to autonomy; however, despite the desire to ultimately apply these algorithms, flights conducted with reinforcement learning (RL) are rare. The limitation is due, in part, to the extensive time commitment for *tabula rasa* learning – that is, learning from scratch with no a priori knowledge of the environment. While many RL methods focus on speeding up the learning rate of tabula rasa learning, several other methods have been developed which focus on reducing only the amount of *in-flight* training time necessary in reinforcement learning; acknowledging that simulations, though imperfect representations, can be used to take on the time burden of training. One such framework is transfer learning.

Transfer in reinforcement learning is the study of transferring knowledge learned in a *source* domain or task, to a different *target* domain or task. Learning first in the source domain can result in improved speed of convergence in the target domain, over tabula rasa learning. Different types of RL-based knowledge can be transferred, such as value function or policy [133]. Understandably, the two domains must be related to some degree in order for the transferred knowledge to have a benefit in the target domain; but how much commonality is needed is not well formalized or easily generalized as most examples within robotics are empirical in nature [75, 103, 133].

For MAV applications, the target domain is in a real-world environment under the setting specific to the task at hand. The source domain can also be in the real world and still benefit the overall mission. The source domain can, for example, be in a safer or more controlled environment than the intended target environment. More often, the source domain is in simulation and the target domain is in the real world. This approach can reduce the amount of total flight time needed to learn a near optimal behavior and in some cases help to avoid dangerous scenarios [1, 32].

In this chapter, a new honeybee task is defined to be tested in the TU Delft Cyber Zoo flight testing facility. The honeybee's mission is to collect nectar from flowers and bring it back to the hive as efficiently as possible. The task is designed to fit in the limited space available, yet still be high-dimensional to maintain its applicability to large-scale discrete problems. As a departure from the standard Markov decision process (MDP) formalization – from which the other chapters in this dissertation and most RL research are structured – this task is non-Markov with one hidden state in order to limit the size of the state space. Real-world tasks are often non-Markov which makes this problem setup a more realistic representation of the world as it is not always possible to ascertain all relevant state knowledge.

The honeybee task is addressed by a reinforcement learning approach which incorporates methods from each previous chapters; including an extension of vision-based rewards from Chapter 2, and hierarchical methods with *options* from Chapters 3 and 4 [1]. This chapter builds upon those methods with the addition of transfer learning and a brief analysis into the effect of hidden states (which is also compared to a similar MDP task). With the non-Markov task, learned solutions of the honeybee task show that in

[1] Gains tuned in Chapter 5 for the vertical loop control are used during flight test, but do not have an influence on the RL methods of this chapter.

most situations, transferred knowledge from a source task improves the speed of convergence for the target task. However, as the commonality between the source and target domains decreases, the benefit of transfer learning over tabula rasa learning instead becomes a trade-off. One result shows that initializing with inaccurate knowledge can give an initial advantage over tabula rasa, but will hinder the learning speed of the training so that the performance advantage is lost later in the training period.

In Section 6.2 a more in-depth introduction to transfer learning and non-Markov tasks for MAVs and robotics in general is presented and the contributions of this chapter are put into the context of recent advances in the field. In Section 6.3, the experimental designs of the honeybee task for both simulation and flight tests are defined. The results of a substantial simulation study are then presented in Section 6.4. The flight test results are presented in Section 6.5, followed by a discussion of the results and conclusions.

6.2. BACKGROUND AND RECENT ADVANCES

This chapter integrates the methods from previous chapters and builds upon those methods with a transfer learning approach. Furthermore, a look into the effectiveness of these method on systems with a hidden state is explored. It is assumed that the reader is informed on temporal difference reinforcement learning (TDRL), Q-learning, and hierarchical reinforcement learning (HRL) from previous chapters or external sources. This section will give an introduction to methods which address hidden states for reinforcement learning problems, transfer learning methods, and the state-of-the-art in transfer learning for MAVs and robotics.

6.2.1. HIDDEN STATE TASKS AND NON-MARKOV DECISION PROCESSES

Markov decision processes (MDPs) are the formal foundation for reinforcement learning; however, it is not hard to find applications where the Markov property does not hold true. Micro aerial vehicles, depending on the size and class, may not be able to carry all the sensors needed for a complete or accurate state representation. A state which is relevant to the decision process but is not represented in the state input vector is called a *hidden state.* Formalizations for Partially observable Markov decision processes (POMDPs) handle non-Markov tasks by trying to estimate the missing state [19, 87]. POMDP approaches have had successes applied to navigation tasks on ground robots [44, 45, 51, 135]. Another method is to predict the value for the missing state in the Q table, such as with recurrent neural networks (RNN) [10, 56, 114].

Not all hidden state tasks require state estimation or value function prediction. In some cases the non-Markov task can still be solved with standard temporal different RL or Q-learning, or can be aided in hierarchical methods; as in Chapter 4 where "ambiguity" in the relative state representation is one way a system can be partially observable. In this chapter, the reinforcement learning algorithms solve a honeybee MDP and then a non-Markov version of the task with an added hidden state. A hierarchical reinforcement learning over *options* approach is able to solve both version, but is on average slower to learn the non-Markov task.

6.2.2. TRANSFER LEARNING

Transfer learning for reinforcement learning is an extensive field of study. The field is defined, generally, as methods which take advantage of knowledge learned in a source task in order to gain some benefit when that knowledge is used in a different, but related, target domain. Lazaric [75] defines a taxonomy which classified each transfer learning approach based on 3 main dimensions: the *setting*, the transferred *knowledge*, and the *objective*. One interpretation can be visualized as in Figure 6.1. Taylor and Stone [133] created a slightly different classification in their survey. Both taxonomies will be referenced in this section.[2]

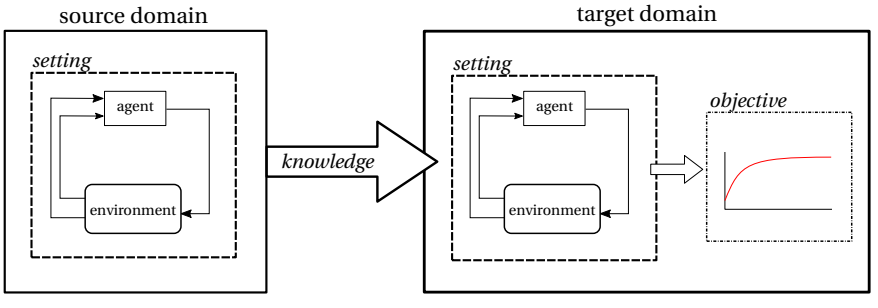


Figure 6.1: A visualization of the taxonomy of transfer learning in 3 main dimensions: the setting, the transferred knowledge, and the objective.

Due to the large range of different transfer approaches that exist, the *setting* is defined by several elements which define the nature of the task(s) to be solved; the features and formulations that define the setting include the state or action space [54, 69], reward structures [74, 85], transition function [102, 105], start state and/or goal state [41] of the task, among others. Any one of these elements may be changed between the source tasks and target tasks. In some cases, a single agent in the target domain can receive knowledge gathered from multiple agents in the source domain [98, 132]. Lazaric categorized settings into 3 groups: Transfer from source task to target task with fixed domain, Transfer across tasks with fixed domain, and Transfer across tasks with different domains. Taylor and Stone name it *transfer dimensions* and categorize the methods into 5 classification groups defined according to how the source and target domains differ from each other: Same state variables and actions, Multi-Task learning, Different state variables and actions with no explicit task mappings, Different state variables and actions with inter-task mappings, and Learning inter-task mappings.

The transferred *knowledge* is the information which is learned in the source domain and transferred to the target domain. The type of knowledge which can be transferred include: Q value functions [11], policies [40, 41], action or option sets [85, 116], task features [54, 74], among others. Lazaric further generalizes the types into: Instance transfer,

[2] This section gives only a brief introduction to transfer learning. For a more complete look into transfer learning, the two referenced surveys are highly recommended. Taylor and Stone (2009) [133], give an in-depth look into the transfer learning algorithms implemented as well as a classification into the types of information that can be transfered from source to target domain; and Lazaric (2012) [75] focuses on a taxonomy of transfer learning within the broader scope of reinforcement learning research.

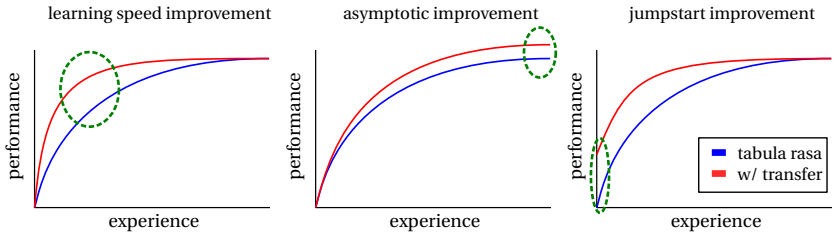


Figure 6.2: The main objectives of transfer learning as described by Lazaric [75]. The contribution of each learning objective is identified by the dashed circle.

Representation transfer, and Parameter transfer.

The *objective* is defined by the metric(s) with which one can determine the benefit (if any) of the transfer learning method over tabula rasa. The ways which the performance can be improved can be measured in different ways. Figure 6.2 displays the different metrics as described by Lazaric. The *jumpstart* demonstrates an improved performance at the start of training in the target domain. The *learning speed* is the rate in which the performance improves, and the *asymptotic improvement* is the amount by which the transfer method improves the convergence of the final learned behavior. Taylor and Stone add that the *total reward* (the area under the curve), can be used as a metric; and with that, a *transfer ratio* can be calculated equal to the total reward of the transfer method over the total reward accumulated by the tabula rasa learning.

According to the above classification scheme, this study can be categorized as having a setting where the transition function of the task is changed. The state and action/option spaces are unchanged. The transferred knowledge is the Q-function for HRL options. The main objective metrics referenced are the convergence to the optimum (if or when it converges), the learning speed, and the jumpstart improvement – all analyzed over 50 sample runs.

RECENT ADVANCES IN APPLICATION

While it is interesting and useful to classify the vastly different algorithmic approaches of transfer learning, we are also interested in the applications where these methods are beneficial for the improvement of reinforcement learning. Without a well-established theoretical foundation, empirical studies are the main driving force behind transfer learning for RL [133].

Transfer learning has been successfully applied to physical robots in the robosoccer [126] keepaway benchmark problem, by learning inter-task mappings from a simple source domain in simulation to the more complex keepaway domain [122, 134] or with "Q-reuse" and SARSA where the source domain is a reduced version of the target domain; both on a physical humanoid robot [11]. Recently, inverse dynamics of neural networks have been learned on a simulation in order to generate actions for the target domain of a Fetch robotic arm in a continuous domain task [28]. A multi-stage deep reinforcement learning agent consisting of a camera sensor and robotic arm uses transfer learning to more quickly identify and pick up specific objects [57].

Due to the time commitments needed for training there are less physical applications for UAVs and MAVs. One study trained a neural network policy for a quadrotor stabilization control task in a simple simulation and successfully transferred the policy to a quadrotor; testing the stabilization by throwing it in the air and by waypoint navigation [62]. Another study with a quadrotor, learned obstacle avoidance through forest in the summer and then transferred the policy to a target domain in the winter in different weather conditions (snow) [32].

In this study aims to demonstrate RL transfer learning on a non-Markov task using a quadrotor and vision-based rewards. The detailed setup for the experiment will now be explained.

6.3. EXPERIMENTAL SETUP

A new reinforcement learning “honeybee” task was designed for the experiment presented in this chapter. The reinforcement learning methods from the previous chapters are consolidated onto a single platform in order to give a demonstration of a real-life robotic system utilizing reinforcement learning to learn a guidance and control application. In this section, details and motivation are given for the design of the task and solution methods, as well as a description of the setup of the simulation and real-life flight tests.

6.3.1. HONEYBEE TASK DESIGN

A new honeybee task is defined for the purposes of this experiment. The task is modified from the design in Chapter 2 to be more realistic, higher dimensional, and more complex, yet still be spatially compact so as to fly in the space of the Cyber Zoo. This section will discuss the state representation of the new task, as well as the physical rules of the world the bee is operating in. There are two world designs: one which maintains the properties of a Markov decision process (MDP) but is not a realistic depiction of real-life, and another which incorporates a “hidden state", resulting in a loss of the Markov property and is a more realistic model of the real world but more difficult to solve.

STATE REPRESENTATION

In order to create a more challenging and relevant task, a new state representation was necessary. In Chapter 2, the agent had no knowledge of visitation history and a non-realistic feature had to be added to prevent the bee from continuously visiting the same flower. After each visit to a point of interest (a flower or the hive), the agent was regenerated randomly within the grid world, thus the agent learned, in effect, how to get from its current location to the nearest reward spot. This simplification is not realistic, but requires less states and therefore was used as an intermediary step.

To remedy this unrealistic characteristic of the problem setup, new information is introduced which incorporates a limited amount of historical knowledge in the state, so that the bee knows which point of interest (POI) was most recently visited. This change is added to reflect the reality that nectar will not be immediately available at the last visited flower. The optimal solution will involve a sequence of visited flowers, and knowing which flower was visited last will be instrumental in learning this sequence.

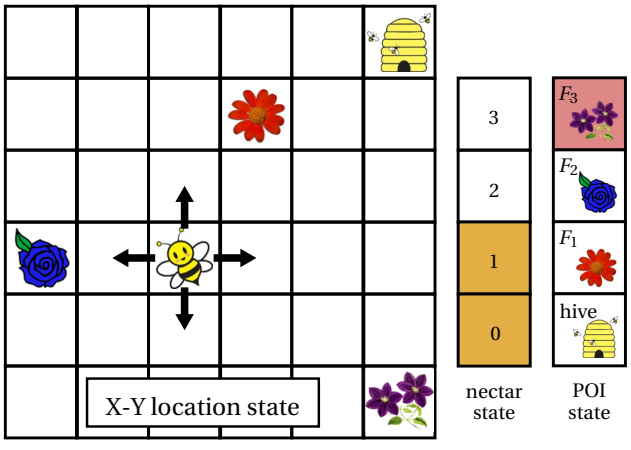


Figure 6.3: New honeybee task concept: The new honeybee task is based on the task from Chapter 2, with extended state space to incorporate the knowledge of the last-visited point of interest.

The new state representation can be seen in Figure 6.3. The location state is represented in X-Y coordinates within a discrete 6×6 gridworld. The nectar state indicates how many nectar units the bee is carrying. The bee can carry a maximum of 3 nectar units. The POI state represents the POI which was last visited: the hive, or one of the 3 flowers. With 36 location states, 4 nectar states and 4 POI states, there are **576 total states**. However, since there are certain states which will never occur, for example the nectar state must be zero if the hive was the last visited POI and the nectar state can not be zero if a flower was last visited, the effective number of states is actually **360 states**.

BEEWORLD 1

In Beeworld 1, nectar cannot be retrieved from the same flower twice in a row. Nectar is not available at last visited flower (known by POI state). As soon as another POI is visited, nectar is assumed to be available again. The criterion for nectar availability is directly represented in the state, therefore maintaining the Markov property.

The rules of this MDP world are, however, unrealistic since the amount of nectar available at a flower doesn't have to do with visiting other flowers in the meantime, but rather many factors including: the species of flower, how old the bloom is, and many other unknown factors [35]. Therefore, this Beeworld domain only serves as a demonstration of learning the task as an MDP. In reality, there will be many unknowable *hidden states* which cause the problem to be non-Markov.

BEEWORLD 2

Beeworld 2 is more realistic, at the expensive of losing the Markov property. Nectar is depleted at a flower when the agent visits, but the nectar regenerates after t_{nr} timesteps. There is a counter for each flower tracking the timesteps since it was last visited, however this information is not known to the agent. This sort of non-Markov task is known as a "hidden state" task [144]. It is left as a hidden state since the required increase in state space would be prohibitive for learning, and also because in real-life there are instances

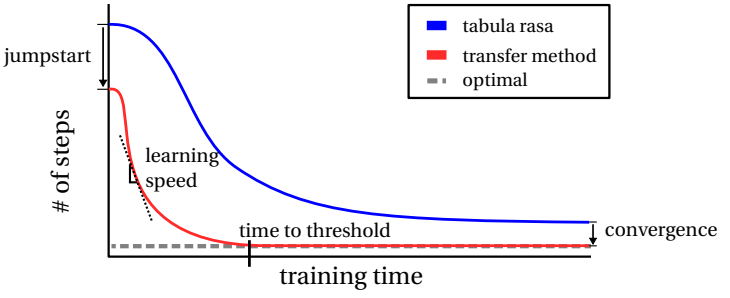


Figure 6.4: Evaluation metrics: Several aspects of the performance can be evaluated to determine the benefits of one method over another. In this case, the performance goal is to minimize the number of evaluation steps.

where a state can not be known. This study aims to demonstrate that the task can still be learned with reinforcement learning techniques.

The nectar regeneration time, t_{nr}, in this task will determine the optimal path sequence between the flowers. If the nectar regenerates quickly, it is beneficial to visit only the flowers nearby the hive. As it takes longer to regenerate the nectar, it becomes more beneficial to travel and collect the nectar at further flowers.

The task is sequential, but can also have more than one optimal path. For example, if the optimal solution is to loop through all three flowers and return to the hive, the agent can go either:

- counter clockwise, as in: $hive \rightarrow F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow hive$
- clockwise, as in: $hive \rightarrow F_3 \rightarrow F_2 \rightarrow F_1 \rightarrow hive$

Optimal evaluation scores and the corresponding path(s) as a function of t_{nr}, can be seen in Table 6.4 (Section 6.4).

PERFORMANCE EVALUATION

The evaluation of a learned value function is determined by how many iterations it takes to get 12 nectar units to the goal using a greedy policy($\epsilon = 1$). Each evaluation run is capped at 120 steps.

As discussed in Section 6.2, benefits of one method over another can be measured in several ways. In this study, the performance as measured in number of discrete steps, is to be minimized. The metrics which will be analysed and discussed in the results are shown as they apply to this problem setup in Figure 6.4.

The *jumpstart* demonstrates an improved performance from the very beginning of training. This can be achieved, for example, by the transferred Q-function or from an optionset well-suited to the problem (such as shown in Chapters 3 and 4). The *learning speed* is the rate in which the performance improves. The rate can be defined as the slope of the line, so the learning speed is changing throughout the learning and can be compared to the learning speed of the tabula rasa line. A more compact metric is the *time to threshold*. In many applications, the optimal performance cannot be known, so a threshold performance is defined in order to compare the time it takes for different methods

to reach it. For this study the optimum is known so observations of the time to the optimal behavior are made. The *convergence* metric of the method describes the asymptotic behavior or the performance at the end of the training period. For the purposes of this study, we are interested in the convergence to the optimum and how *consistently* it finds the optimum over 50 sample runs.

6.3.2. REINFORCEMENT LEARNING METHODS

The reinforcement learning methods used in this experiment have been described in previous chapters.

The value function temporal difference RL update law from derivations in Section 2.2, was originally introduced in Eq. (2.6). In this chapter, this method is referred to as V-TDRL. The "flat" Q-learning update law from derivations in Section 3.2, was originally introduced in Eq. (3.2). The hierarchical reinforcement learning (HRL) Q-learning over *options* update law from derivations in Section 3.2, was originally introduced in Eq. (3.5).

The state space or state/action space sizes depend on the RL method used and the number of primitive actions or options available. A summary is below:

V-TDRL 360 states

Q-learning 1440 state/action pairs

HRL 2880 state/option pairs (with 8 options)

HRL OPTIONSET

The design of the optionset is informed by the lessons learned from Chapters 3 and 4 and the nature of this task.

Table 6.1: Configurations of flat Q-learning and HRL optionsets.

flat (primitive)	HRL options
North (N)	N+E
East (E)	E+S
South (S)	S+W
West (W)	W+N
-	$N \times 3$
-	$E \times 3$
-	$S \times 3$
-	$W \times 3$

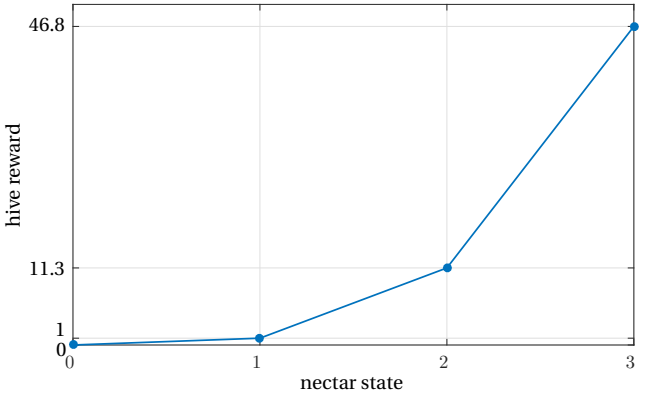


Figure 6.5: Hive reward: The reward at the hive POI is a function of nectar state. $r = ns^z$. The value $z = 3.5$ was tuned for this task.

REWARD STRUCTURE

Designing an effective reward structure is vital for most forms of reinforcement learning. The reward structure for this task is as follows:

−1 Penalty for each timestep

−3 Penalty for state-action resulting in a wall hit

***f*(*ns*)** The reward at the hive as a function of the nectar state (Figure 6.5).

To learn the sequential path between flowers in this task, the main positive reward must come from the hive at the higher nectar states. The reward is only collected at the end once the nectar is brought to the hive and therefore has to propagate through the Q-function's multiple dimensions, where visiting flowers is the only way to transition to the next nectar state. The hive reward, as in Figure 6.5, was tuned by trial and error against Beeworld 2 with $t_{nr} = 12$ so that Path C (from Figure 6.9) could be found.

6.3.3. PARAMETER SELECTION

Observation of training runs indicated sensitivity to: the learning rate, α, the discount factor, γ, and the greediness factor, ϵ, of an ϵ-greedy policy.

To ensure a fair comparison between the various configurations, a parameter search was conducted using 50 simulated runs for statistical significance and observing the convergence behavior. The resulting parameters for each configuration are shown in Table 6.2. The discount factor, γ, was found to be best at $\gamma = 0.90$ consistently, and so is left out of the table. A more detailed analysis is shown in Appendix D.

The results of the parameter study for the MDP of Beeworld 1, showed that the solution is robust to a large range of parameters and learns very easily within that range.

The results of the parameter study for the non-Markov Beeworld 2 showed that the performance of the RL task is sensitive to each of the parameters varied; however, the

Table 6.2: Summary of parameter selection for each configuration

	bee-world 1			bee-world 2		
parameter	V	Q	HRL	V	Q	HRL
ϵ	0.6	0.6	0.6	0.8	0.8	Scheduled
α	0.2	0.3	0.3	0.3	0.2	$1/\sqrt[4]{k}$

discount factor γ and the learning rate, α are consistently reliable in the same range of values for each of the configurations. While a constant α was usually sufficient, there were some occasions where a variable α improved convergence greatly. These cases will be further discussed in the results sections.

The study into the parameter ϵ of the $\epsilon-$greedy policy had surprising results for Bee-world 2. Fully random exploration resulted in poor learning, signifying that greediness is needed to overcome the presence of the hidden state since it adds time as a factor. The HRL approach benefited from scheduling of the greediness over the course of its training. Since there are infinite ways to schedule the ϵ-greedy policy, a few profiles were tried and the resulting schedule was selected (see Appendix D).

6.3.4. FLIGHT TEST SETUP

Two flight tests are performed using the resources available in the Cyber Zoo. A position tracking system is used for location state estimation for the grid world as well to collect the takeoff position for the self-tuning gains. The points of interest are represented by colored paper on the ground of the arena. They are detected use a color-based vision capability as already described in Chapter 2; however there are now 4 colors to detect, with each color indicating a different POI. Figure 6.6 gives an overhead view of the Cyberzoo setup, where the yellow paper represents the hive, and the red, blue, and purple papers represent flowers 1, 2, and 3, respectively.

Figure 6.6: Overhead view of Cyber Zoo with quadrotor and color-based vision features

The flight tests are conducted for the hierarchical reinforcement learning method in Beeworld 2 in order to demonstrate the transferability of the simulation learned guidance policy to a real-world target task. The task is slightly changed, by way of changing the nectar regeneration time, t_{nr}. The first flight test has the quadrotor agent learn a Q-function in a simulated source domain with $t_{nr} = 5$ and transfers that knowledge to the Cyber Zoo with target domain of $t_{nr} = 12$. In the second flight test, the Q-function is learned via a simulated source domain with $t_{nr} = 12$ and is transferred to a $t_{nr} = 5$ target domain in the Cyber Zoo. In each case within the target domain, the training remains at an ϵ-greedy policy of $\epsilon = 0.8$.

6.3.5. TABLE OF EXPERIMENTS

Table 6.3 summarizes the experiments which will be presented in the upcoming results section.

Table 6.3: Table of experiments

#	Domain	Q_{init}	Methods	Figure
	Simulation			
1	Beeworld 1	tabula rasa	V-TDRL Q-learning HRL	Figure 6.7
2	Beeworld 2 $t_{nr} = 12$	tabula rasa	V-TDRL Q-learning HRL	Figure 6.10
3	Beeworld 2 various t_{nr}	transfer see Table 6.5	HRL	Figures 6.12 - 6.13
	Flight test			
4	Beeworld 2 various t_{nr}	transfer see Table 6.6	HRL	Figures 6.18 - 6.15

6.4. SIMULATION RESULTS

Simulation is used in this chapter in two ways: First, as a representation of the real-world in order to predict the behavior of a learning agent when learning from tabula rasa; second, as a tool to learn knowledge of the simulated source environment in order to transfer the knowledge to the real-life target environment and possibly benefit from the prior knowledge (Flight test results in Section 6.5).

The results of learning from tabula rasa are presented first in the Beeworld 1 MDP (Experiment 1) and then in the non-Markov Beeworld 2 (Experiment 2). Results are shown for the 3 RL methods: Value function temporal difference RL (denoted as V-TDRL),

Q-learning, and hierarchical reinforcement learning (HRL) over *options*. The main interest is in the HRL method, of which the benefits and limitations are described in detail in Chapters 3 and 4. The V-TDRL method requires a transition model for action selection, and therefore has limited applications; however, it is kept in the study as a visualization aid since the larger dimensionality of Q-learning and HRL is difficult to visualize. "Flat" Q-learning is presented for comparison purposes.

Results of the Q-table transferability study are presented only for Beeworld 2 using HRL over *options* (Experiment 3). Knowledge transfer necessitates that there exist two different domains: the source and the target. For knowledge transfer to be beneficial over tabula rasa, the source and target domains must be similar enough that the knowledge gained in the source domain is still useful in the target domain. For this study, the source and target domains differ in the nectar regeneration time, t_{nr}. The contribution of this parameter to the "hidden state" means that the change will not be reflected in the state input vector and will therefore only be learned by interaction with the target domain environment.

6.4.1. BEEWORLD 1

The Beeworld 1 MDP is easily learned by each of the methods. The performance – the number of steps it takes to get 12 nectar units to the hive during a greedy evaluation run – of each method over training iterations is shown in Figure 6.7. The average of 50 runs is plotted. The standard deviations are not shown, but each of the methods' average performance reaches the optimum, meaning that the standard deviation goes to zero. The V-TDRL approach is the fastest to learn since it has, by far, the least number of states it must explore. The HRL option/action space has twice as many values to learn as the flat Q-learning, but still finds the optimal solution faster (with standard deviation of 0, over 50 runs). This task is still small in its physical space, therefore the V-TDRL is the best approach but with a larger physical space it is likely that the HRL approach would gain the advantage.

To demonstrate the learned value function, the V-function table learned by V-TDRL is shown in Figure 6.8 along with notation (in yellow) showing the resulting optimal path of a fully greedy evaluation run. Each square grid represents the discrete gridworld within the "plane" of the indicated nectar state/POI state. The solid yellow circle denotes the starting location within the plane, the arrow is the path it will take to the next flower or hive denoted by a dashed-line circle, which will in turn transition it to a different nectar state/POI state "plane". Q-learning and HRL over *options*, results in approximately the same path, but visualization of the Q-function would take N times as many "planes" to plot, where N is the number of actions or options.

The greedy evaluation run will result in the path: $hive \rightarrow F_1 \rightarrow F_2 \rightarrow F_1 \rightarrow hive$ (Figure 6.9-Path B). The furthest flower is not part of the optimal path with this setup and cannot be without adding additional state variables and new rules about nectar availability.

6.4.2. BEEWORLD 2

Beeworld 2 introduces the non-Markov property and the "hidden state" property as described in Section 6.3. The effect of the hidden state will first be discussed, followed by

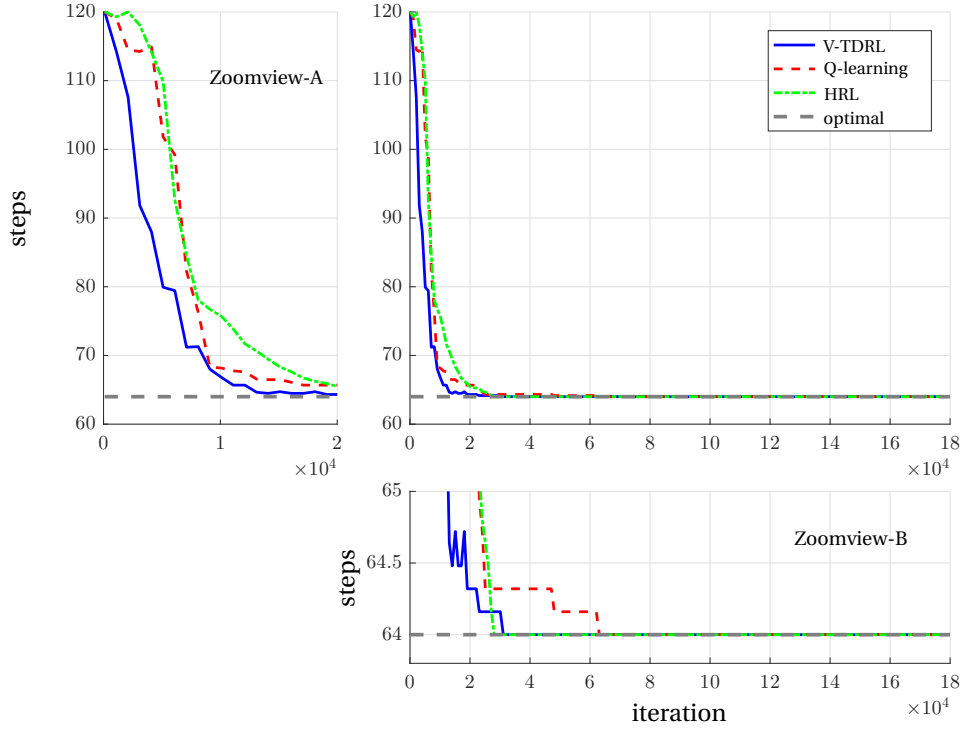


Figure 6.7: Experiment 1: comparison of performance for V-TDRL, Q-learning, and HRL approaches

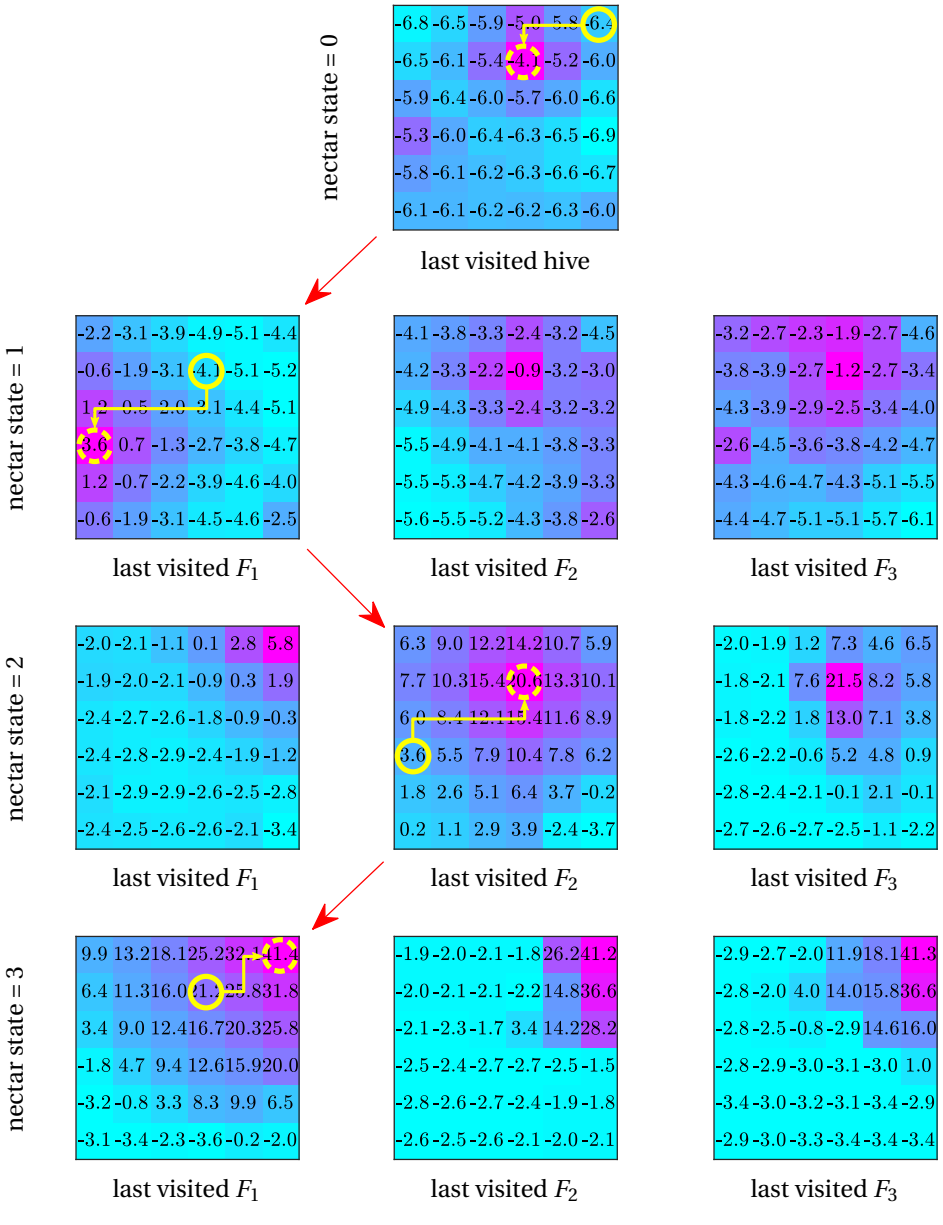


Figure 6.8: Experiment 1: Value function (numbers) and greedy path (yellow path)

the results from the tabula rasa learning of each of the RL methods and the results of the transferability study with the HRL approach.

HIDDEN STATE EFFECT ON OPTIMAL PATH

The hidden state in the form of time-sensitive state transition makes the optimal path dependent on the nectar regeneration time, t_{nr}. For this specific task, the ranges of t_{nr} and the corresponding optimal paths are noted in Table 6.4. The resulting paths, are visualized in Figure 6.9.

Certain ranges of t_{nr} will have the same optimal path but with a higher number of evaluation steps than other ranges. This is because the agent will wait around that flower until the nectar is regenerated. In certain ranges of t_{nr}, a "wait around" sort of policy is faster than making the trip to a further flower.

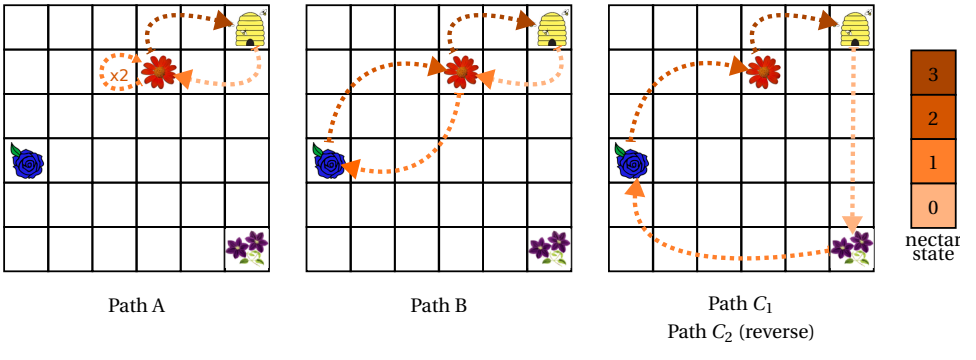


Figure 6.9: Beeworld 2: Possible learned paths

Table 6.4: Beeworld 2: nectar regeneration time ranges and corresponding optimal path (with primitive actions and no wall-hits)

t_{nr}	evaluation optimal # steps	path	description
$0 \leq t_{nr} < 2$	40	Path A	$hive \rightarrow F_1 \rightarrow F_1 \rightarrow F_1 \rightarrow hive$
$2 \leq t_{nr} < 4$	56	Path A	$hive \rightarrow F_1 \rightarrow F_1 \rightarrow F_1 \rightarrow hive$
$4 \leq t_{nr} < 6$	64	Path B	$hive \rightarrow F_1 \rightarrow F_2 \rightarrow F_1 \rightarrow hive$
$6 \leq t_{nr} < 10$	76	Path B	$hive \rightarrow F_1 \rightarrow F_2 \rightarrow F_1 \rightarrow hive$
$10 \leq t_{nr} < 20$	80	Path C_1	$hive \rightarrow F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow hive$
			or
		Path C_2	$hive \rightarrow F_3 \rightarrow F_2 \rightarrow F_1 \rightarrow hive$

RL METHODS COMPARISON

A comparison of the RL methods is shown in Figure 6.10 using $t_{nr} = 12$ so that the optimal path will be either path C_1 or path C_2, circling through each of the flowers in either a clockwise or counter-clockwise direction, respectively. The direction will be determined by the random actions taken. The parameters chosen for each of the methods were summarized in Table 6.2 and help to understand the results, especially regarding the scheduled ϵ value used for the HRL methods.

The V-TDRL approach finds the optimal path consistently after about 150,000 iterations. Q-learning has a slower learning rate, and after 180,000 iterations still has 2 runs out of 50 which don't converge to the optimum. The HRL with options approach performs on average worse on all metrics, with a slower learning rate, and converges to a solution slightly worse than optimum (but still within 1 step of the optimum on average). The results from Chapter 3 showed that the benefit of the temporally extended options only became apparent in the largest maze. Therefore, it is likely that the option-set chosen is not well suited for the size of the discrete gridworld since the Q-learning with only primitive actions can perform better. However, given that the benefits for HRL were already discussed in Chapters 3 and 4, we can hypothesize that the HRL method will surpass the flat Q-learning method as the problem scales up.

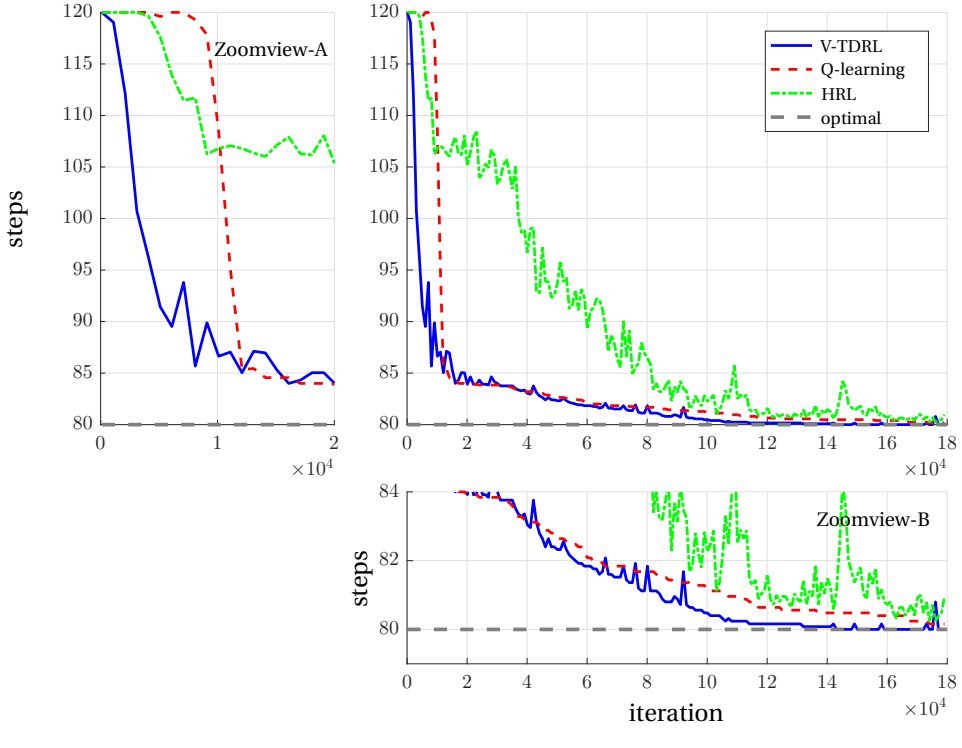


Figure 6.10: Experiment 2: Comparison of performance for V-TDRL, Q-learning, and HRL approaches

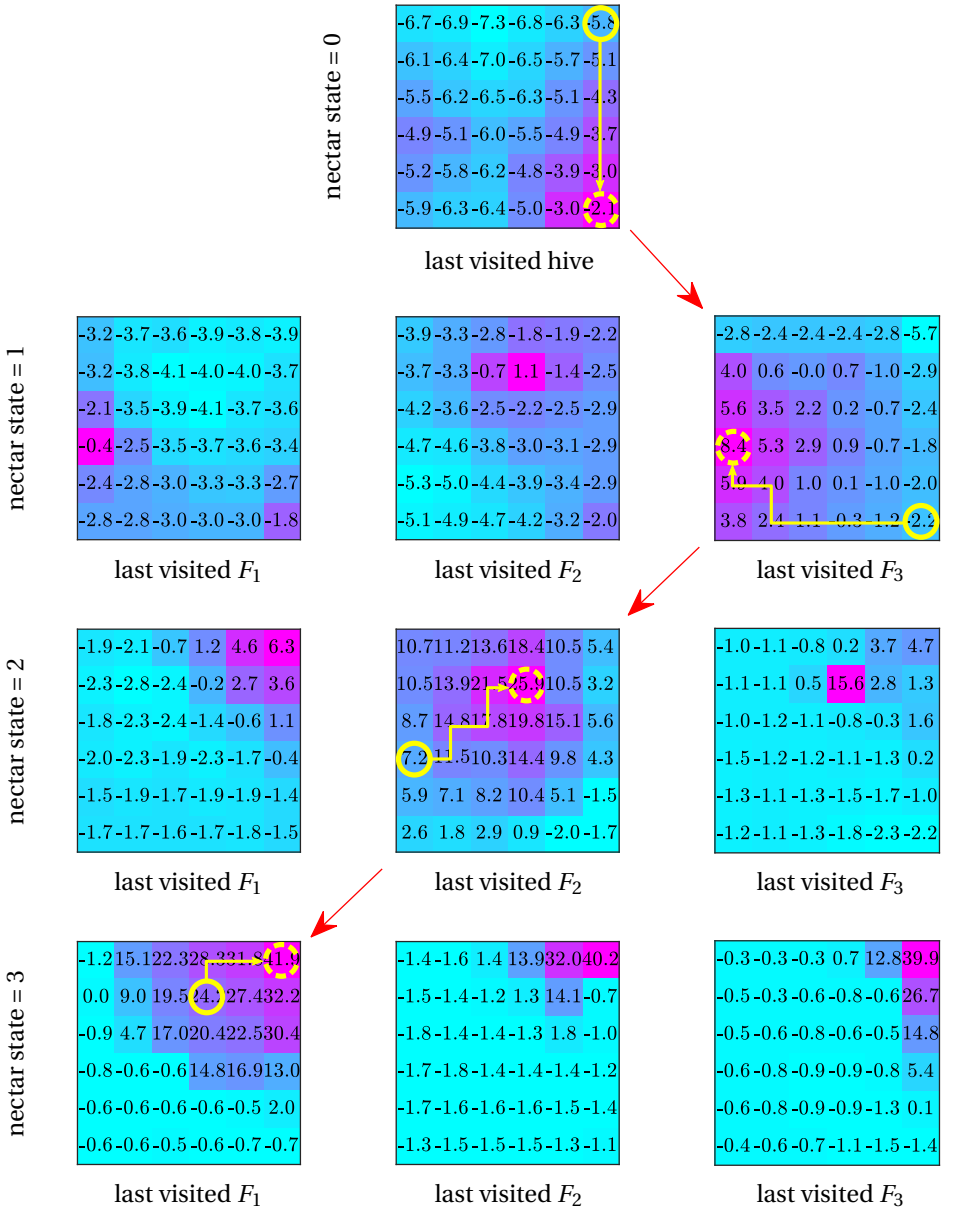


Figure 6.11: Experiment 2: Value function (numbers) and greedy path (yellow path) – Path C_1

6.4.3. TRANSFERABILITY STUDY

To study the transferability of the HRL Q-table initialized from the source domain and executed within a target domain, the two domains are defined as having different t_{nr}. The tests are described in Table 6.5. The source domain provides a converged $Q-$table to use as an initial Q_{init} for the target domain. The alternative to using a sourced Q_{init} is learning tabula rasa, in other words from a blank slate with the Q-table initialized to all zeros as already explored in the previous sections.

The tests from Table 6.5 are selected such that learning of the target domain starts from a variety of source Q_{init}'s. The target domains of $t_{nr} = 5$ and $t_{nr} = 12$ are selected based on Table 6.4. They result in different optimal paths, Path B and Path C, respectively, so that experiments can be set up which approach the target domain from various degrees of commonality for the setting. Target domain $t_{nr} = 5$ corresponds to optimal Path B and is initialized with source value functions which correspond to Path C_1 (3_b), Path B(3_c), and Path A(3_d). Target domain $t_{nr} = 12$ corresponds to optimal Path C and is initialized with source value functions which correspond to Path C (3_x), Path B(3_y), and Path A(3_z). The transfer learning evaluation results are also compared to tabula rasa learning (3_a and 3_w). It is hypothesized that transfer can be highly beneficial over tabula rasa learning in all of the metrics described in Figure 6.4; however, it is also hypothesized that there will be some point where the source and target domains are too different and transfer can become detrimental for certain metrics in the way that "bad habits" can be harder to break than learning from scratch.

Table 6.5: Experiment 3: Simulation transferability study–Figure descriptions

#	t_{nr} (source)	t_{nr} (target)	optimal paths source → target
	Figure 6.12		
3_a	tabula rasa	5	→ Path B
3_b	12	5	Path C_1 → Path B
3_c	9	5	Path B → Path B
3_d	3	5	Path A → Path B
	Figure 6.13		
3_w	tabula rasa	12	→ Path C
3_x	15	12	Path C → Path C
3_y	5	12	Path B → Path C
3_z	3	12	Path A → Path C

The evaluation results for the transfer experiment with target domain $t_{nr} = 5$ (tests 3_{a-d}) are plotted in Figure 6.12, averaged over 50 runs. The dashed grey line is the optimal number of evaluation steps for the target domain. Each source is represented by a different color, where the diamond (to the left of iteration 0) is the optimal performance of Q_{init} in the respective source domain, and the solid line is the performance starting with Q_{init} at iteration 0, and evaluated as it is trained in the target domain. The results show that RL transfer is beneficial over tabula rasa learning, within some limitations.

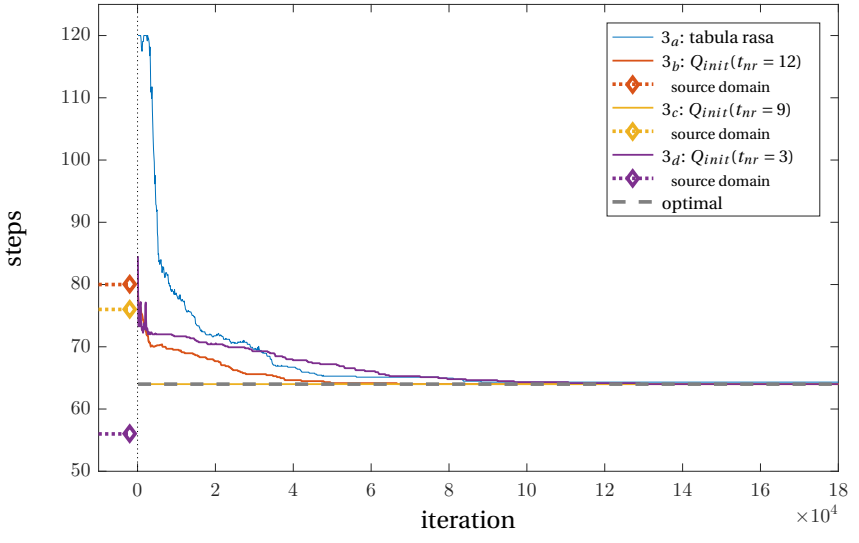


Figure 6.12: Experiment 3_{a-d}: transfer study at $t_{nr} = 5$ with Q_{init} from source domains at $t_{nr} = \{12, 9, 3\}$. Compared with tabula rasa learning. The figure plots the number of steps in the greedy evaluation run averaged over 50 runs.

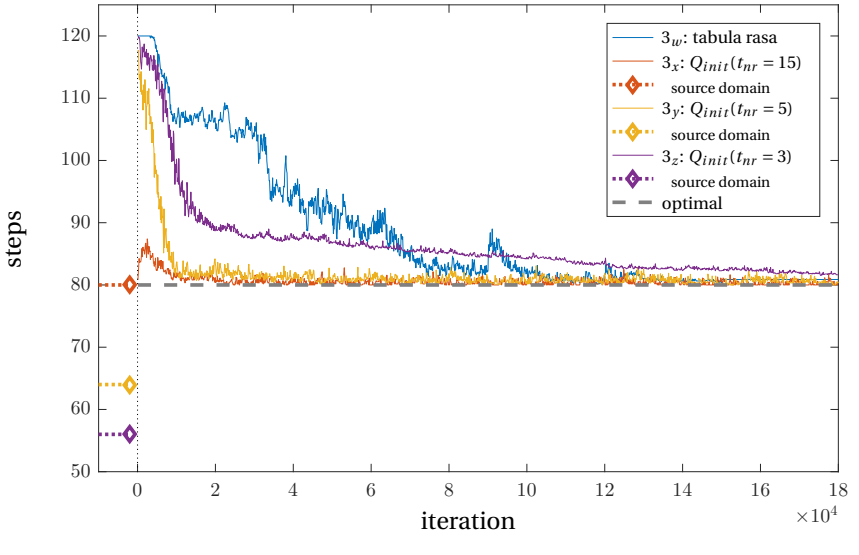


Figure 6.13: Experiment 3_{w-z}: transfer study at $t_{nr} = 12$ with Q_{init} from source domains at $t_{nr} = \{15, 5, 3\}$. Compared with tabula rasa learning. The figure plots the number of steps in the greedy evaluation run averaged over 50 runs.

The jumpstart improvement by transfer learning is in each case highly beneficial. Test 3_c immediately starts and maintains the optimum. The one metric which is not better than the tabula rasa results is test 3_d, where the source has a lower number of optimal steps. Initially, the agent is operating in the target domain as if it were still in the source domain. The previously rewarding path (Path A) results in flowers without nectar available and it has to wait to take enough random actions to find Path B, and reinforce that path through continuous interaction with the new environment. This can take time to override the previously known knowledge within the value function and as a result it takes longer than tabula rasa to find the value function which leads to the optimal path.

The evaluation results for the transfer experiment with target domain $t_{nr} = 12$ (tests 3_{w-z}) are plotted in Figure 6.13, averaged over 50 runs. The benefits of transfer learning again show in the jumpstart and in a faster learning rate, early in the training. The one exception is again with source domain $t_{nr} = 3$ (test 3_z) where it learns at a faster rate than tabula rasa early on in training, but then slows while the tabula rasa learning finds the optimal path earlier, on average. An interesting result from test 3_x, indicates that even though the source and target domains share the same optimal path (Path C → Path C), the transfer learning introduces opportunities to stray from a Q-function which results in an optimal path. This is contrary to test 3_c, where the "Path B → Path B" transfer continued to be optimum for the whole course of training in the target domain. One difference between the results seen in Figure 6.12 and the results in Figure 6.13 is the noise. When the optimal path is Path C for the target domain, there is more noise in convergence because there is more than one optimal route which gives conflicting state/option updates. The greedy performance is most stable when one option is clearly the best choice for a given state and is reflected as such in the values of the Q-function. However, when several options give similarly high-valued expected returns, it only takes a few random actions during training to give an unfavorable value function update which results in a non-optimal decision during a greedy evaluation. Because the learning rate parameter $\alpha = 1/\sqrt[4]{k(s,o)}$, [3] and $k(s,o)$ is reset to zeros at the beginning of training in the target domain, the relatively larger bellman update stepsize will increase the occurrences of these temporary deviations from the optimal performance. As the α decreases, the performance converges again toward the optimum. This explains the general noisiness of the evaluation performance for target domain $t_{nr} = 12$, as well as the temporarily worse performance of test 3_c.

6.5. FLIGHT TEST RESULTS

Flight tests were performed for two transfer scenarios as described below in Table 6.6. The flight tests, Tests 4_a and 4_b, have the same configurations as the simulation tests 3_b and 3_y, respectively(see Table 6.5). Therefore, the result is expected to be similar with two differences: 1) The flight test presented in this section will be a single run instead of an average over 50 runs, and 2) Due to time constraints, the run will only last for about 4500 iterations (3.25 hours) – as compared to the 180,000 iterations in simulation – which means it is likely that there will be noticeable learning, but may not converge to the optimal in that time. In Chapter 2, we saw that a near-optimal greedy evaluation

[3] $\alpha = 1/\sqrt[4]{k(s,o)}$, where $k(s,o)$ is the number of times each state/option is visited

could be found within the battery life of the AR-drone 2.0 (10-12 minutes) and long before the value function was completely converged. This honeybee task is larger in scale and uses transfer learning and HRL methods to combat this larger scale. Results from simulation show how many iterations it will take on average (over 50 runs); the flight tests presented in this section provide insight about what those results mean in real time on a real-life platform, and the associated challenges.

Table 6.6: Experiment 4: Flight test transferability study–Figure descriptions

#	t_{nr} (source)	t_{nr} (target)	optimal paths source → target	Figure	Figure description
4_a	12	5	Path C_1 → Path B	Figure 6.15 Figure 6.17	evaluation results tracked flight
4_b	5	12	Path B → Path C	Figure 6.16 Figure 6.18	evaluation results tracked flight

The results are shown in two different formats. The evaluation results as in Figure 6.15 and Figure 6.16 are familiar from the simulation results section. In the flight tests, the Q-function was saved and evaluated every 10 iterations. The tracks shown in Figure 6.14, Figure 6.17 and Figure 6.18 show the position tracking of the quadrotor throughout the flight tests. The flights in Figure 6.17 and Figure 6.18 have been divided into "rounds" groupings for clarity. Each round, as numbered in the figure, consists of the path starting from the hive with no nectar, and ending at the hive when it delivers the nectar there. The early portion of the round is dark blue and over time transitions in color to yellow. These figures are from training with $\epsilon = 0.8$, and therefore 80% of the actions are greedy and 20% of the actions are random. Figure 6.14 shows examples of a greedy evaluation using the Q-function from the *source* domains of Tests 4_a and 4_b.

We will first discuss the results from transfer learning Test 4_a, as seen in Figure 6.15. The source domain is represented by $t_{nr} = 12$, results in the optimal Path C and 80 steps is the optimal evaluation performance (Figure 6.14(a)). The 80 step optimal for the source domain is visualized as the red diamond left of the line at iteration 0. The target domain $t_{nr} = 5$ has an optimal Path B and 64 steps is the optimal evaluation performance, which is visualized by the grey dashed-line. The training within the target domain begins at iteration 0 using $\epsilon = 0.8$ throughout the whole training period. The Q_{init} provided is a product of the source domain learning and will therefore choose an action consistent with Path C_1, 80% of the time, as per the ϵ-greedy training policy. This behavior for the training can be seen in Figure 6.17, where Path C_1 is followed with the exception of a few random option selections. For example, Rounds 1, 3, and 13 have completed the round in the minimum number of iterations, while the other rounds took some random actions in order to explore. In Round 12, the agent explores with a series of random actions which find Path B, the new target domain path.

For this test, the critical juncture is at the hive with 0 nectar, where the agent chooses an option which either moves toward the purple flower (as was the optimal state/option from the source domain), or moves towards red flower (which is the desired path in the

target domain). When the agent takes the target domain desired path, it has an effect of increasing those state/option values in the Q-function, as compared to taking Path C. The greedy action is taken 80% of the time, and 20% of the time one of the 7 remaining options will be taken. Therefore, it is only a matter of *when* the right option will be taken and how many times that option needs to be selected for its state/option value to overtake the previously greedy option's value which was reinforced in the source domain. This effect can be seen in Figure 6.15. The initial Q function is the biggest influence to the greedy evaluation until around iteration 1720, where one of the state/options was updated enough that the associated value became larger than the option which moved the agent to the purple flower. This resulting greedy path, included an option which resulted in a wall hit but was still valued higher than the Path C option. This has some safety implications, and indicates that the wall hit penalty should be greater. Again, around iteration 4400, another option was sampled enough to overtake the wall hit option. It results in more greedy evaluation steps, but doesn't not hit the wall at any point so it ended up having a higher value. The optimal option was still not sampled enough within the 4500 iterations; however this result for a single run is consistent with the trends seen in the simulation results in Figure 6.12 where 50/50 runs found the optimal after about 50,000 iterations.

Since t_{nr} is a hidden state which directly influences the nectar availability, and therefore the state transitions, the agent won't "know" that the domain has changed unless there are some indications when the agent visits a flower. That is to say, when the target domain t_{nr} is lower than the source domain t_{nr}, as in Test 4_a, the greedy path can be followed and the updates of those state/option values will be small. The state transitions and Q-values continue as they did in the source domain, so the only way to find the quicker path in the target domain is through random exploration to find the state/option values which are different from the source domain. Figure 6.15 demonstrates this behavior, as the source domain path is continually executed during the greedy evaluation and only after several opportunities of the *right* random actions does it find a better policy.

If the target domain t_{nr} value is higher than the source domain t_{nr} value, as in Test 4_b, there will be implications for the state transitions of the greedy path since flowers which previously had nectar after 5 timesteps, will not have nectar for an addition 7 timesteps. This change in the hidden state explains the results for Test 4_b as presented in Figure 6.16 and Figure 6.18. The agent tries to follow Path B, but at the flowers finds that there is no nectar and therefore no state transition. The agent is then "stuck" in that nectar state plane until if finds another flower which can transition it to the next nectar state, or until enough time has past that the original flower has run out the clock and has nectar again. In the short term, the agent will perform the latter, since the source Q-function is leading the agent to that flower. The result of this behavior is large updates to the Q-table on the state/option values which have state transitions affected by the change in the hidden state – that is, around the flower location states, and eventually all the surrounding states.

As a comparison between the two test configurations, say $\Delta Q_i = \sum |Q_i - Q_{i+10}|$; where i is the iteration at each evaluation which occurs in increments of 10, and $\sum$ is the sum over all the elements in the matrix. Then ΔQ represents the total changes in the Q-function made by updates over the span of 10 iterations. The average ΔQ for flight test

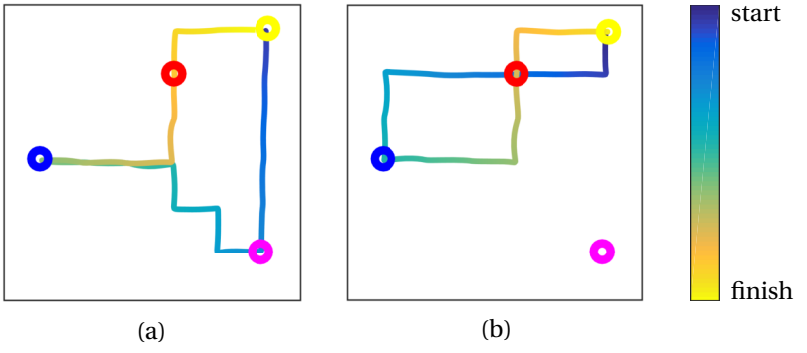


Figure 6.14: Experiment 4_a and 4_b: The tracked path of the greedy evaluation at the start of (a) Test 4_a, resulting in Path C_2 and (b) Test 4_b resulting in Path b.

4_a was 3.4, while the ΔQ for flight test 4_b was 11.1; which reflects how the 4_b transfer configuration instigates more active value function updating. While the critical juncture for Test 4_a was to find the better option during the random actions, the critical juncture for Test 4_b is to *either* find the better option through random action, or keep penalizing all the other options until the best option is the highest value by process of elimination.

In the position tracking in Figure 6.18, Rounds 4 and 6 look on first glance like they are following Path B perfectly, but a close look shows that it has to double back on top of its own tracks in order to wait for the red flower to regenerate nectar. The active updating of the value function is demonstrated in the greedy evaluation runs shown in the noisiness of Figure 6.16. In training, the greedy options are taken 80% of the time, so the greedy state/options values will be the first to significantly deviate from the values of the source domain Q-table. In this case, the values will decrease until a different option from a given state has a greater state/option value. This change in the greedy option causes the large jumps in the evaluation performance. Eventually, the purple flower is visited and Path C discovered. With enough samples, the value function will be updated enough to reinforce the new path. In the 4,500 iterations which were performed, there is already an improvement seen (on average).

Since there is also more than one optimal route for Path C (clockwise or counter-clockwise), the Q-function gets some contradictory updates, which is why the evaluation results are noisy, even in the simulation results where it was averaged over 50 runs. The result in the flight test run proves to be representative what was seen in simulation.

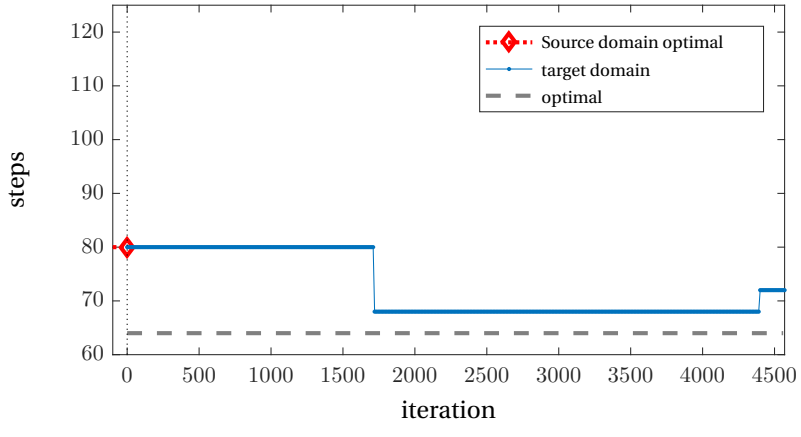


Figure 6.15: Experiment 4_a: Flight test greedy evaluation performance results.

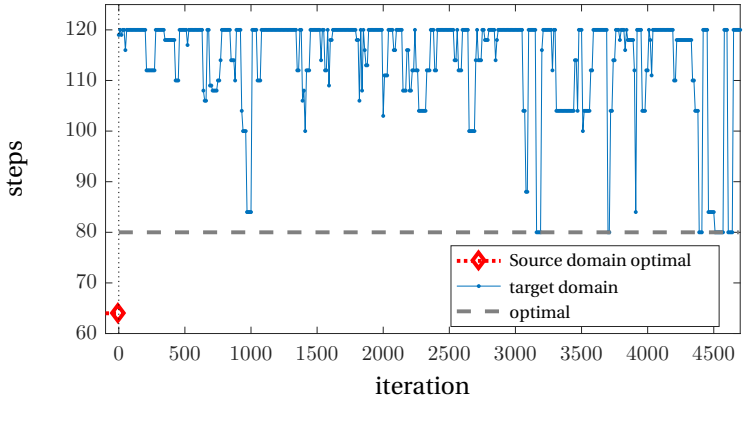


Figure 6.16: Experiment 4_b: Flight test greedy evaluation performance results.

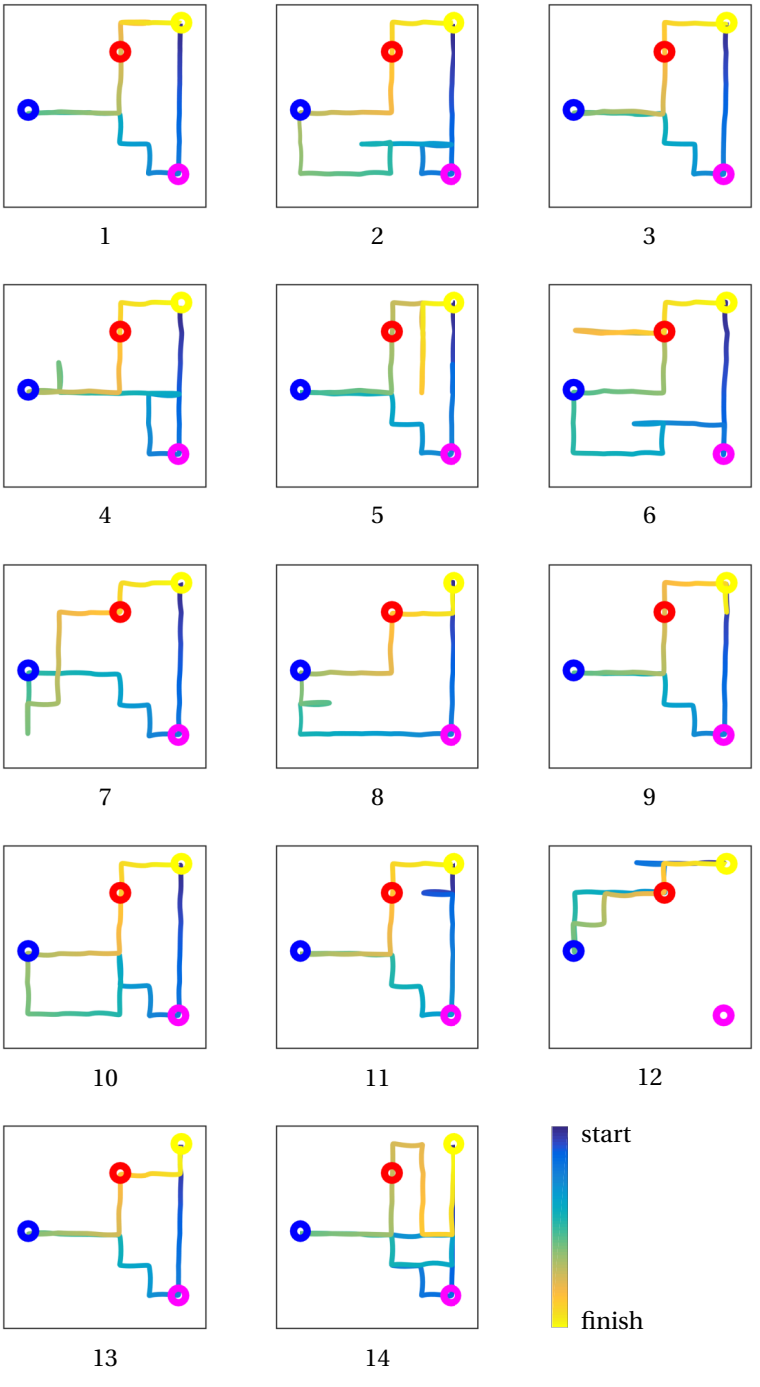


Figure 6.17: Experiment 4_a: Flight test training position tracking for the first 300 iterations.

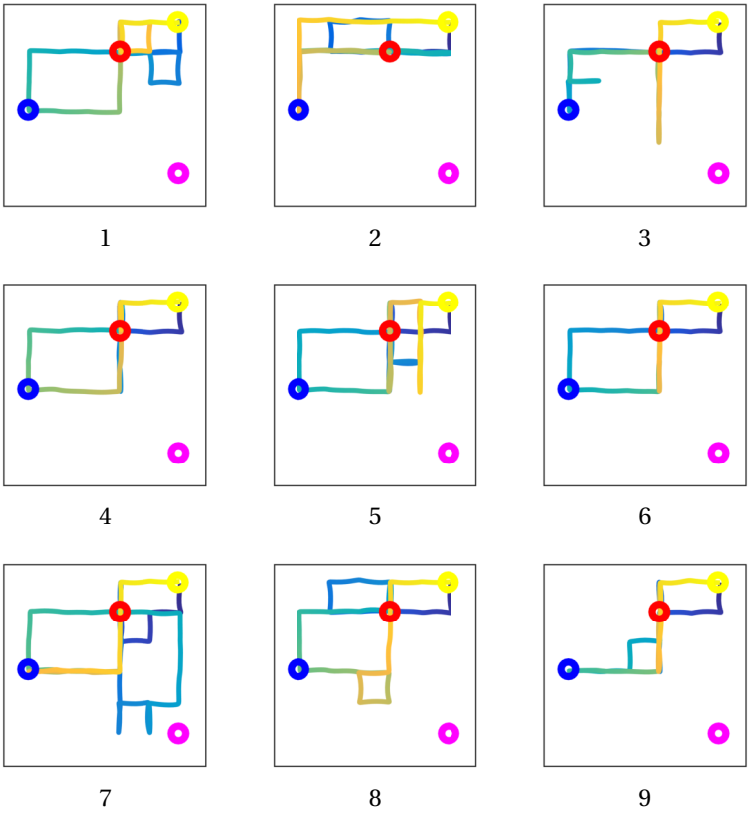


Figure 6.18: Experiment 4_b: Flight test training position tracking for the first 300 iterations.

6.6. CONCLUSIONS

This chapter presented a new reinforcement learning task which imitates a bee collecting nectar from flowers and returning it to the hive, in perpetuity. The task was designed so as to culminate all the approaches studied in previous chapters and to be tested on a quadrotor in the TU Delft Cyber Zoo facility. Furthermore, a new feature of reinforcement learning was explored in a real-life application by transferring knowledge from a source domain within simulation to a target domain via a quadrotor test flight.

The results first show the relative ease of learning within a Markov decision process (MDP), where all states are known and observable. However, in the real-world there are many scenarios where one or more states cannot be obtained, resulting in a "hidden state". Literature indicates that non-Markov properties can result in the necessity for more complex solutions, but the results show that in this task it is still solvable by the RL methods explored throughout this dissertation.

The hidden state changed between domains was the time variable, t_{nr}, which represents the time it takes to regenerate nectar at a flower. By changing this variable, but hiding its influence on the state vector, the effectiveness of reinforcement learning transfer in non-Markov hidden state domains was explored. This study showed that reinforcement learning Q-table knowledge transfer has benefits over learning *tabula rasa* on a non-Markov task using HRL with *options*. However, this benefit holds true only until a certain point when the source and target domain become too different from each other. If the source domain results in a substantially different optimal path than the target domain, it can be time consuming to "unlearn" the knowledge embedded in the value function from the source domain.

Another notable conclusion from the transfer study, is that transfer is more beneficial when going from a higher valued t_{nr} to a lower valued; that is, if the sourced behavior can still function well in the target domain, there will be greater benefit in the jumpstart performance metric, and similar benefit in time to a near optimal solution. Therefore, for a task like this one, it is prudent when choosing parameters for the source domain to err on the side of the more complex or more easy to adapt from, before deploying the agent into the real-world target domain.

The real-life flight tests built on the results from Chapter 2 by increasing the scale of the environment, and by applying the HRL methods from Chapter 3 and 4 and the transfer learning method from this chapter. The real-life flights are similar to the simulation transferability study, but give further insight by identifying the challenges associated in the real-world. Designing the simulation study involved making decisions on several parameters including: the number of iterations to train, the scheduling for ϵ, α, and many other parameters. It is easy to plan for a simulation which requires millions of training iterations and for which the parameters and rewards are tied to simulated cues; however, implementing in real-life inevitably guides these design decisions in directions not previously considered. For example, a better way to implement the discrete steps of an option in the flight test would be to execute the option in one smooth command and stop only if a point of interest (POI) is spotted. However, as it was simulated and knowledge transferred, a diagonal move might miss the POI where it would have been "seen" in the simulation. This could have led to better design or better implementation of the options within the HRL algorithm.

Further improvement with transfer learning against tabula rasa learning might have been accomplished with the transfer of a different hierarchical RL approach. Further subdivisions of the task, such as in HRL HAMs [101], might have further sped up the learning within the target task.

7

CONCLUSIONS

Reinforcement learning is an ever expanding field of research with a wide breadth of potential applications. Due to the algorithm's notable utility in optimization with little or no a priori information, the reinforcement learning approach is especially popular with tasks where little or no information is known or where the information is unreliable and adaptive behavior is needed. Such tasks are widely desired for unmanned aerial systems (UAS) and micro aerial vehicles (MAVs) in particular, which have many uses for autonomy in unknown environments. However, several inherent limitations arise with *tabula rasa* learning that limit the practical usefulness of the approach for an MAV with limited resources. This thesis aimed to address some of the practical challenges related to using reinforcement learning within the scope of MAV guidance and control tasks. This led to the primary research question stated in Chapter 1:

Primary Research question

How can reinforcement learning contribute towards the goal of autonomous flight for micro aerial vehicles?

From this question, a number of reinforcement learning approaches for MAV applications were explored, with each chapter contributing a different approach. The final experiment in Chapter 6 combined all the approaches onto one platform with the additional contribution of a transfer learning analysis; creating a quadrotor platform which makes decisions using reinforcement learning in both guidance and control. The challenges and successes of the experiments guide this final chapter where the main findings are summarized and discussed in Section 7.1. The contributions of this thesis are put into context of the greater goal of MAV autonomous flight in Section 7.2, and finally, recommendations for future work are proposed in Section 7.3.

7.1. MAIN FINDINGS AND CONCLUSIONS

In Chapter 1, an approach was set out to answer the primary research question; first by identifying the several challenges associated with reinforcement learning as they combine with the physical limitations and desired applications of MAVs, and then by establishing how these challenges would fit into the scope of suitable guidance and control experiments.

The resulting focus was determined to be the RL challenges of: Slow learning due to *tabula rasa* learning, and the Curse of dimensionality; and the MAV challenges of: limited resources, and complex dynamics. Each chapter in this thesis presented a potential solution to one or more of these categorical challenges, which will now be discussed.

7.1.1. SLOW LEARNING DUE TO TABULA RASA LEARNING

> Q1. What RL methods are available to overcome the practical challenges associated with slow learning due to *tabula rasa* learning on an MAV?

One of the greatest advantages of reinforcement learning over other methods is to learn through interaction with the environment *without* any need for prior knowledge. Therefore, *tabula rasa* learning is at once both an attractive feature and also one of the main causes of slow learning speeds since the agent must train with random actions until each state or state/action has been sampled a sufficient number of times (sufficient to form a reasonable estimate of the value function to result in acceptable behavior). This thesis considers these as trade-off points and will further discuss the trade-offs made in each of the approaches.

Since we are interested in physical systems which will take a fixed amount of time to execute an action, we focus on decreasing the number of iterations (or timesteps) needed to learn a task. And furthermore, the time spent *in-flight* is the most important aspect when considering the major limitation of battery life of an MAV. Therefore, methods which take advantage of simulation can do so, in this respect, at little or no "cost".

The most obvious way to speed up learning is to tune the parameters. This was done by hand for each experiment in order to give the methods a good starting point. Since optimal parameter tuning was not one of the focuses of this thesis, most of the parameter tuning studies are done in the background, or presented in appendices when there was an interesting or important finding.

One way to avoid the slow learning of *tabula rasa*, is by **not** learning *tabula rasa*. In that case, some prior knowledge of the task or system must be available. By utilizing an inaccurate model [2], **Chapter 5** demonstrated that the policy gradient reinforcement learning could iteratively improve upon a PID gain policy for a takeoff task of a quadrotor with just a few real-life trials. Due to safety concerns, it would not be advisable to try a randomly generated policy, so this approach for gain tuning is one work-around which shows that reinforcement learning can be used in a variety of ways. In this case, the characteristic interaction with the environment comes in the form of real-life trials which informs the inaccurate model of its own inaccuracy in order to calculate a bias and therefore improve the model (locally). The gradient which advises the direction of change for the policy is calculated in simulation of the model, and therefore does not

add to the *in-flight* time cost. However, this method necessitates prior knowledge of the system and it also has the disadvantage of using gradient-based improvement – which is susceptible to local minima.

In **Chapter 6**, another method was explored which takes advantage of simulation to learn a good starting point for a state/value Q-function; instead of using a zeroed, *tabula rasa* Q-function. Transfer learning recognizes that as long as there is some commonality between two different domains, the knowledge learned in one can improve the learning speed in the other by giving a "better than random" starting point. Again, this method most often assumes that certain knowledge is known about the tasks, such as, state transitions, where the reward states are, what actions will be available to take with the agent. Requiring that this knowledge is known detracts from the goal of generalization for model-free, tabula rasa reinforcement learning; but in most cases some knowledge of the task is known. It is shown in this chapter that utilizing task-specific knowledge in a simulated "source" domain, can substantially reduce the training iterations needed to find a near-optimal behavior in the in-flight "target" domain. Chapter 6 varied state transitions between the source and target domains to test how differences in the model affected the learning in the target domain from the starting point Q-function learned within the source domain. A set of metrics to analyze the performance improvement was introduced including: jumpstart, learning speed, time to threshold, and convergence By these metrics, the results of the *best* case showed that a similar source domain can result in the immediate optimal solution in the target domain, bypassing the millions of iterations needed for training with *tabula rasa* techniques to find the optimal evaluation performance. In the worst case studied, the learning speed was faster than *tabula rasa* in the early stage of training, but in the later stages of training, the *tabula rasa* method was faster and ultimately converged sooner to the optimal, on average. This sort of result shows that transfer learning can sometimes transfer knowledge which, like bad habits, can take longer to break than learning from scratch.

To keep the *tabula rasa* aspect of learning, **Chapters 3 and 4** showed that temporal abstractions could be used to reduce the number of timesteps needed to find a goal in a large maze. Temporal abstractions can be considered an approach to address slow tabula rasa learning, but can also be considered a solution to address the curse of dimensionality. Therefore, these two chapters will be discussed in the next section.

7.1.2. CURSE OF DIMENSIONALITY

> Q2. What RL methods are available to overcome the practical challenges associated with the curse of dimensionality on an MAV?

Addressing the curse of dimensionality for practical applications, ultimately, has the intension to speed up the learning for large dimensional state or state/action spaces; same as in Section 7.1.1. However, the classification of the research questions in Chapter 1 categorizes it as a different challenge because the *approach* taken to address the problem in this thesis is different from that of speeding up *tabula rasa* learning. The curse of dimensionality is addressed here by temporal and state abstraction in order to make the state space smaller or decomposed, instead of aiming to minimize the time

commitment to random-action exploration.

Conclusions about the effect of temporal abstraction can be drawn from **Chapter 3**, and the effect of state abstractions can be analyzed by comparing the results of Chapter 3 against Chapter 4. In Chapter 3, hierarchical reinforcement learning over *options* was used as a way to speed up *tabula rasa* Q-learning with an absolute state representation. Starting from a null Q-table in a large obstacle-rich maze where the only reward is at the goal state, extended actions can find the goal much faster (i.e. in less iterations) than flat Q-learning even at epoch 0, when no knowledge of the environment is known because less of the state space needs to be sampled before finding the goal. For example, in Chapter 3 with the absolute state representation, the best optionset in the large Parr's maze made it to the goal in almost 5 times less timesteps than the flat case and roughly half the number of timesteps for the small and medium sized maze. The benefit depends on three factors: 1) the optionset, 2) the size of the state space, and 3) the epoch of training (how long it has been learning). Some optionsets resulted in poorer performance in all metrics (ie. learning speed, convergence) compared to the flat case. This shows that the expert knowledge introduced into the problem within the design of these optionsets is not fail-safe. Like with the reward structure, the designer of the optionset has an effect on how well the RL algorithm will work. Furthermore, the benefit is not consistent over all environments or throughout the training epochs. In the small maze, the performance of the flat case quickly improves and can become more optimal than the optionsets in a greedy evaluation. The analysis of the three factors described are best encompassed in the evaluation performance shown in Figure 3.7 for the small maze, Figure 3.8 for the medium size maze, and Figure 3.10 for the Parr's maze.

In **Chapter 4**, state abstraction was introduced as a way to **1)** better represent the limitations of quadrotor camera-based state acquisition (discussed soon in Section 7.1.4), and **2)** use a state representation which is independent from the size of the environment. Uncoupling the state space size from the size of the maze environment, directly breaks the curse of dimensionality, but results in state ambiguity – where not every unique position has a unique representation in the abstracted state input. Several state representations were explored to find a relationship between representations and its effects on convergence, ambiguity, and suboptimality. Comparing the absolute state representation of Chapter 3 to the relative state representation in Chapter 4, the relative state abstraction uses 96% less states to represent the large maze problem. Using this many less states takes a toll on the performance of the flat Q-learning configuration, where the relative state abstraction fails to improve upon a random policy. However, once temporal abstraction methods in the form of HRL *options* is added to the state abstraction methods, the improvement in terms of greedy evaluation performance is improved over the absolute state representation across the whole learning time. This result is best seen in the comparative plot of Figure 4.17.

7.1.3. MAV COMPLEX DYNAMICS

Q4. What RL methods are available to overcome the practical challenges associated with MAV complex dynamics?

Chapter 5 demonstrated an RL approach that was both low on number of in-flight trials (addressing the slow learning challenge as discussed in Section 7.1.1) and also focused on a solution for improving control performance of MAVs with complex, difficult to model dynamics. Complex dynamics of an MAV can be difficult and expensive to model accurately; however, a "bad" or inaccurate model, is relatively easy to come by. The approach of policy improvement reinforcement learning in Chapter 5 was found to be a good solution for fine-tuning gains, though not in all cases.

The policy improvement reinforcement learning approach was first shown to be effective on the simulation of an F-16 tracking task; however, concerns arose about stability when the gradient decent method pushed the policy into areas which were stable for the inaccurate model, but unstable for the real system. This phenomenon is best demonstrated in Figures 5.8-5.9. A solution as found to minimize the possibility of choosing an unstable policy by decaying the step size and choosing conservative end conditions. In simulation several different tracking tasks were tested which involved 2, 3, or 4 gains; in order to show that the approach works with larger dimensioned policies.

Lastly, the approach was applied to a quadrotor takeoff task using the PID gains for the vertical control loop to minimize the mean absolute error (MAE) calculated against the altitude setpoint. This task did not have the same kind of system dynamics which would lead a gradient toward an instability and therefore was safe to use on a real quadrotor. The results showed that the number of trials taken until the end conditions were met and the final solution, were dependent on the starting policy. Because the optimization follows a gradient, the best policy which can be found is confined to local optima. In two cases (starting from different starting policies), a local optimal was found within three trial runs on the real quadrotor. This result can best be seen in Figure 5.26.

7.1.4. MAV LIMITED RESOURCES

> Q3. What RL methods are available to overcome the practical challenges associated with MAV limited resources?

Ideally, all the experiments conducted would have been designed under the assumption that only MAV outfitted sensors were available – with the quality or accuracy alloted to the current day sensors. However, due to time and/or resource limitations some concessions were made in the name of incremental progress. In order to demonstrate the usefulness of RL methods within the context of an MAV with limited resources, the issues were addressed one at a time in several experiments throughout this thesis.

Chapter 2 sets the stage for the challenges faced regarding MAVs and reinforcement learning. A simple example is presented of a guidance task through learning of a temporal difference value function. Position of the state is commanded using waypoints and the Optitrack position tracking system. The bottom-facing camera on-board the quadrotor was used to identify the locations of rewards by placing red paper at the reward states and setting a red color filter threshold. The guidance task in an environment of reduced size was learned using *tabula rasa* temporal different learning with random actions for the amount of time of the battery life – about 10 min at the time of the experiment. Even with the reduced size, there was not enough time to learn a converged value function.

However, when looking at the greedy policy map, we can see that the greedy policy is optimal after 400 iterations; therefore demonstrating how improvement of the performance does not necessitate full convergence. This result is shown best in Figure 2.10. In **Chapter 6**, the premise for Chapter 2 is expanded with the vision-based rewards differentiating between colors to determine which reward state it has just visited.

In **Chapter 4**, the relative state abstraction was designed based on one possible camera-based state input. The input required the ability to estimate the distance to obstacles within a 180° field of view. This state vector could be achieved through machine vision techniques such as stereo vision [34, 84] or using light weight ultrasonic sensors, for example.

7.2. MAIN CONTRIBUTIONS

This thesis aimed to address the goal of autonomous flight of MAVs through contributions of reinforcement learning experiments. The main contributions of this thesis towards this purpose, will now be inferred from the above conclusions and itemized.

- For RL to be effective using *tabula rasa* learning, it is not necessary to fully converge the value function to gain the benefits of an improved greedy policy (Ch. 2).
- The combination of state and temporal abstraction together outperforms either one on their own for an obstacle-rich maze task (Ch. 3 & 4).
- Relative state representation is necessary when no other state estimation can be obtained and advantageous in cases when an environment is too large for absolute state representation and limited state space size is desired (Ch. 3 & 4).
- As far as state ambiguity affects the learning of a maze task, the *distribution* of the ambiguous states can be more of a negative influence than the *number* of ambiguous states (Ch. 4).
- Gradient-based policy improvement reinforcement learning can be used to good effect for fine-tuning gains when starting from a reasonably good starting point, but care should be taken to ensure the task is not prone to sharp transitions into unstable regions within the policy-space (Ch. 5).
- Successful transfer of value function knowledge within reinforcement learning is aided by (and sometimes predicated by) well-tuned learning parameters (Ch. 6).
- Including a priori knowledge with reinforcement learning is a double-edged sword: It can aid in faster learning rate but undermines the benefits of RL as a *tabula rasa* learning method, and can limit the freedom of the algorithm to find its own (possibly more optimal) solution (Ch. 6).

7.3. RECOMMENDATIONS AND FUTURE WORK

This thesis addresses the challenges of reinforcement learning for MAV autonomy by conducting experiments within the research scope laid out in Chapter 1, and within the limits of the time and resources available. This thesis contributed to several novel

insights about how reinforcement learning can be used towards the autonomy of MAV flight. However, studies are always ongoing, and answering some questions tends to lead to many more.

This section presents a number of recommendations which follow from the lessons learned in this thesis as they relate to the reinforcement learning, autonomy, and MAV communities.

DIRECTION FOR REINFORCEMENT LEARNING RESEARCH

The methods presented in this thesis are all based in reinforcement learning which, under certain conditions, has theoretical convergence guarantees; however, these guarantees do not apply to any of the experiments used in this thesis. Steps are taken to foster a setup which empirically tends to converge, but certain practical conditions which exist in the real-world just don't fit into the box of the theoretical convergence criteria. This has led to many empirical studies in RL working without guarantees. This is one approach. With a large enough amassment of successful empirical studies, RL could become commonplace enough to be a staple of intelligent decision-making for robotics. However, theoretical frameworks which include more practical applications within their influence could do even more to focus the research within the RL community.

Further, some important research in RL already aims towards generalization. Much of the RL problem depends on choices made by the designer. A streamlined approach for parameter tuning, reward shaping, and state abstraction could take away much of the current guesswork for the problem setup and would make reinforcement learning more accessible.

TOWARDS AUTONOMOUS FLIGHT FOR MAVS

The field of MAV guidance, navigation, and control focused towards MAV related physical and task-based demands is growing fast as the popularity of MAVs broadens and diversifies. This thesis has focused on the application of reinforcement learning to real-life platforms – something which is still rare as there are many more studies which only use simulation methods. Discovering the unforeseen problems encountered in flight tests created many of the biggest challenges and formed or limited the approaches which could be accomplished. For reinforcement learning to form methods which are meant for application, then those applications – not just a representation of them – must be part of the process which molds them.

APPLICATIONS AND SOCIETAL IMPACT

Several applications for autonomous MAVs have been proposed in numerous fields (and sometimes in literal fields). Fields of interest include farming [46], ecology [115], pest control [88], rescue [111], and artificial pollinators [145], among others. With such a large scope of possible applications, spanning a large amount of area over the earth, there will need to be considerable discussion into the potential of societal, moral, and natural repercussions. It is in the best interest of humanity and the earth in general to ensure that MAVs – or any technology – are designed and operated in a safe and ethical way; promoting privacy, antipollution, and conservation of nature.

APPENDICES

A

CHAPTER 3 SUPPORTING STUDIES

A.1. FLAT Q-LEARNING IMPROVEMENT

In Chapter 3, for a fair comparison against the hierarchical reinforcement learning results, a good faith effort was made to get the best result using flat Q-learning methodologies. We decided to explore different training policies and eligibility traces. These studies will now be presented. The choices regarding which results were presented in the main body of the chapter are also justified.

A.1.1. ϵ-GREEDY TRAINING POLICY

The training and evaluation methodologies of the various algorithms are described in Section 3.3.4. The parameter ϵ, for the ϵ-greedy action policy, can drastically affect the performance as we first saw in Chapter 2. Therefore, it was important to study the effect of ϵ schedules for the training epochs.

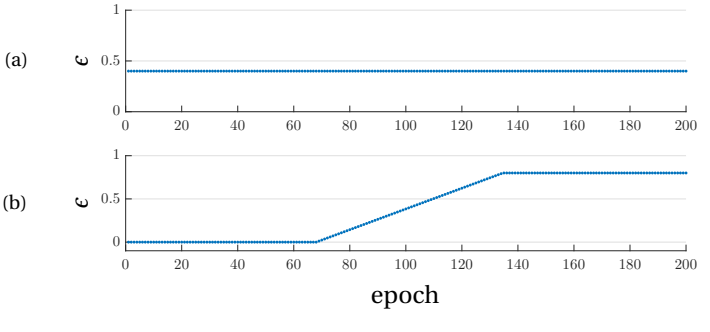


Figure A.1: ϵ-greedy training schemes. (a) constant ϵ for training, (b) scheduled ϵ for training.

For completeness, Figure 3.3 is repeated here as Figure A.1. This shows the different ways we have scheduled ϵ in Chapter 3. There are innumerable ways to change the parameter, and so just a couple simple ways were explored. First, a constant ϵ was used as

A

in Figure 3.3(a). Several values were looked at for the constant value. A ramp-scheme was also explored as in Figure 3.3(b), where the policy is completely random for the first third of the training (67 epochs), increasing for the next third, and then stays at $\epsilon = 0.8$ for the last third. This scheme is chosen because randomness is required to find the best solution, but exploiting knowledge gained is beneficial for convergence speed and penalty avoidance.

Figure A.2 shows the results for the analysis on the medium maze. The ϵ values that perform the best are the fully random (constant $\epsilon = 0.0$), and the ramp-scheme. The medium maze is small enough that all the state-action pairs had been visited enough within the first 67 random epochs to have a good policy. Therefore, the Q-function's will have different values, but the result of the greedy evaluation plotted here will be the same. Added exploitation in the form of $\epsilon > 0$ was not helpful. Therefore, for the small and medium mazes, a ramp-scheme was chosen to be used.

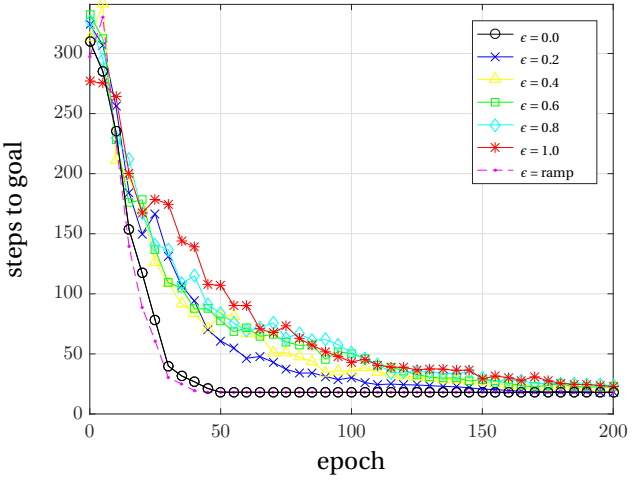


Figure A.2: ϵ parameter study: Medium maze. Evaluation results with different ϵ-greedy training schemes. All cases are averaged over 25 statistical sample runs.

Figure A.3 shows the results using the Parr's maze. With a larger environment, there is a different trend. The fully random policy learns the slowest because it explores more than it exploits, but also gets the best result at the end of the 200 epochs for the same reason. All the cases where $\epsilon > 0.0$, the performance improves faster, but doesn't explore enough to continue that improvement in the later epochs. The ramp-scheme for the first third follows the results of the fully random scheme, as expected because they are both fully random in these first epochs. In the second third, when the ϵ starts to ramp-up, the performance improves quickly as it progressively exploits more of its knowledge. However, since the ramp-scheme stops exploring so much, it also falls short of the fully random performance at epoch 200. This is one of the standard RL trade-offs.

Comparisons (with standard deviations) are shown between the fully random case and the ramp-scheme (Figure A.4) and the fully random case against the $\epsilon = 0.2$ case

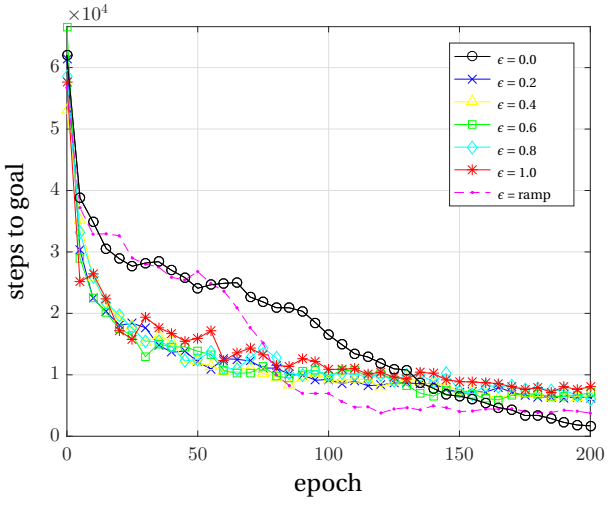


Figure A.3: ϵ parameter study: Parr's maze. Evaluation results with different ϵ-greedy training schemes. All cases are averaged over 25 statistical sample runs.

(Figure A.5). Larger values of ϵ result in convergence to suboptimal solutions, but also to less variability in the evaluation result.

CONCLUSION

The results shown in this section are interesting to note, but ultimately the goal is to compare a flat Q-learning case with HRL techniques and not to have to do an exhaustive analysis into parameter tuning. The ϵ-greedy parameter tuning explored for the flat case can also be used for the HRL options, so while it is beneficial to compare the HRL results to the best possible flat case, it is also reasonable to make comparisons using the same parameters. Therefore, the main body of the chapter will use the fully random ($\epsilon = 0.0$) training scheme and the $\epsilon = 0.2$ training scheme as comparison against the HRL methods for the Parr's maze.

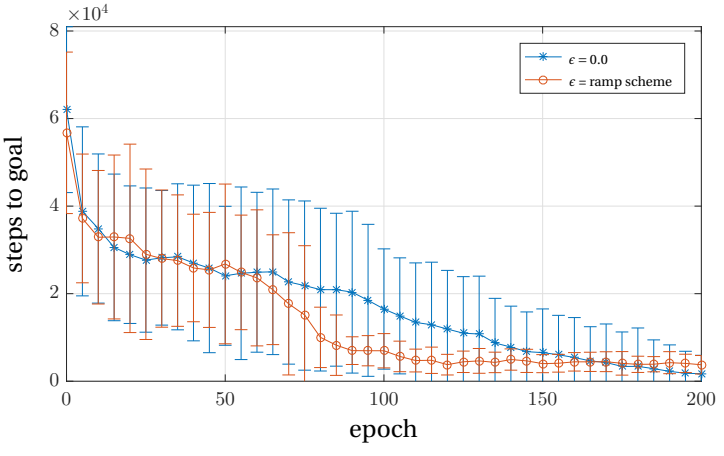


Figure A.4: ϵ parameter study: Parr's maze. Evaluation results with standard deviation errorbars. ϵ-greedy training schemes. All cases are averaged over 25 statistical sample runs.

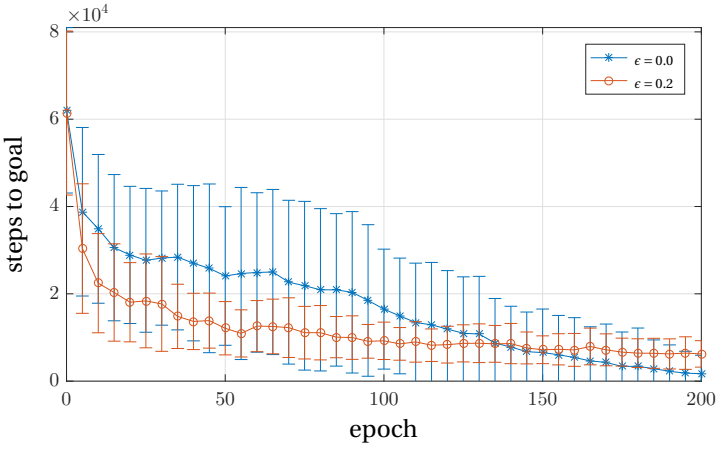


Figure A.5: ϵ parameter study: Parr's maze. Evaluation results with standard deviation errorbars. ϵ-greedy training schemes used for comparison in the main body of Chapter 3. Cases are averaged over 100 statistical sample runs.

A.1.2. ELIGIBILITY TRACES

Eligibility traces are explored for the possibility to improve the flat Q-learning case. The final conclusion is that eligibility will not be used in the main body analysis because the improvement is too small and unreliable. Information on eligibility traces can be found in Sutton and Barto (1998) [130].

Update calculations on large matrices with eligibility is more computationally taxing than without. Therefore, for the sake of time, only 10 statistical samples were ran.

Figure A.6 shows the comparison of the no-eligibility case with 10 and 100 sample runs and the eligibility cases with 10 sample runs. The standard deviations are not plotted for chart clarity. From this figure we see that eligibility traces with $\lambda \leq 0.4$ has a slight benefit for the cases with 10 sample runs. However, when comparing to a no-eligibility run of sample size 100, the improvement is all but gone. Furthermore, when $\lambda \geq 0.5$ the performance is very poor. This is likely because the training is random and an increasing λ increasingly rewards suboptimal actions.

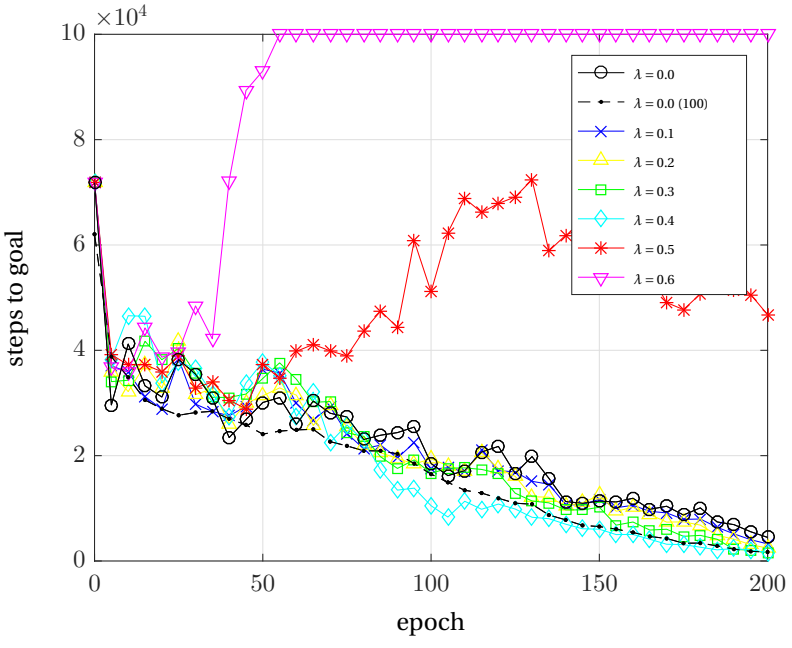


Figure A.6: Eligability traces study: Evaluation results with several different λ values. All cases are averaged over 10 statistical sample runs, except the one specified as (100) sample runs.

After this analysis it was decided to not make more runs for a better comparison since, but to just do without eligibility traces in the main body of the chapter for several reasons. Considering that large λ values result in poor or even unstable results, the slight improvement is not worth the risk, especially considering the improvement is slight and uncertain given the high standard deviations. Furthermore, The ϵ-greedy parameter for training has a larger impact on the performance.

A

CONCLUSION

The conclusion is that eligibility traces with carefully selected λ values have a slight improvement over the case with no eligibility when run with 10 statistical samples. However, larger λ values cause a sharp decrease in performance. Furthermore, the trend of the evaluation metric is the same with a small λ as it is with no eligibility at all, so adding eligibility to the flat case doesn't add much value when comparing to the HRL methods. Because of the lack of added value, it is determined to not implement

A.2. STATISTICAL ANALYSIS SAMPLE SIZE

A study was performed to justify the statistical sample size used throughout Chapter 3. To a certain extent, the number of sample runs is limited because simulation on large environments is time consuming.

Figure A.7 (medium maze) and Figure A.8 (Parr's maze) show the flat Q-learning performance mean and standard deviation(std) with respect to the number of samples. The different color data lines represent a particular epoch.

For the medium maze, the mean and std at epoch 200 converges immediately. Every case finds the optimal solution by epoch 200. The earlier epochs are more variable and due to time constraints, too many samples would be required to converge at every epoch, especially for the Parr's maze. Priorities must be decided. Standard deviations are not that important in order to compare the performance of one approach to the other. As long as the trends of the mean values at each epoch are in the correct order, the sample size should be sufficient to draw conclusions when comparing the flat Q-learning to the HRL methods. Therefore, based on the results in these figures, **it was determined that 100 sample runs is sufficient**.

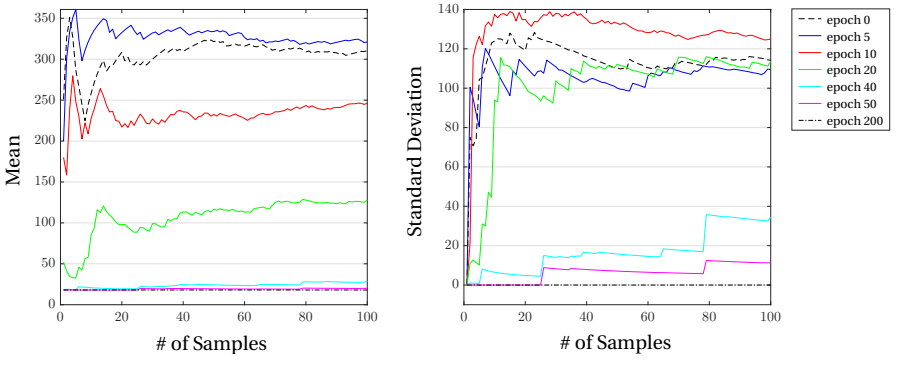


Figure A.7: Sample size analysis: Medium maze

A.3. STANDARD DEVIATIONS FROM STATISTICAL ANALYSIS

This series of charts, expand on the results from Section 3.4. Figures A.9- A.11 show the results from the evaluation runs of the learned Q on small, medium and Parr's mazes. The average number of steps to reach goal is plotted with the standard deviation dis-

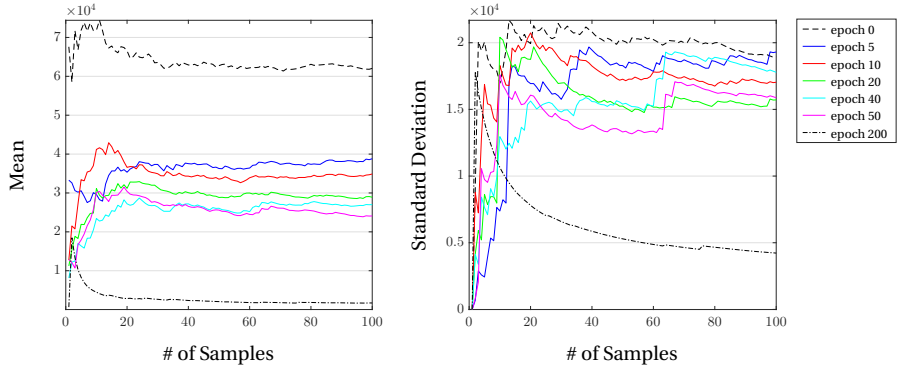


Figure A.8: Sample size analysis: Parr's maze

played as errorbars. The data consists of 100 statistical sample runs. The small and medium mazes were trained with a "ramped" epsilon-greedy policy, and the Parr's maze was trained with completely random policy.

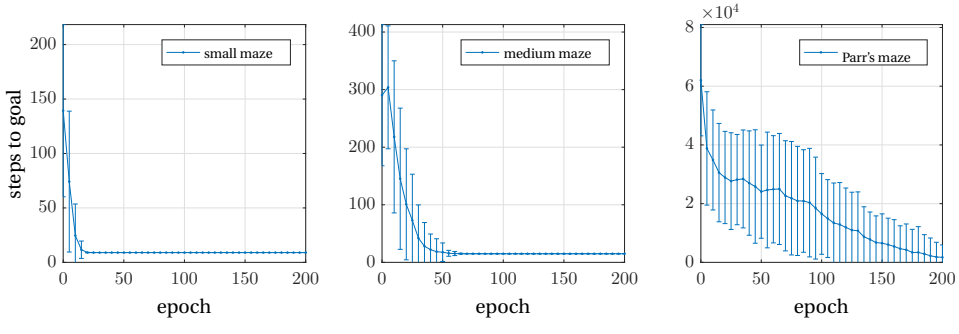


Figure A.9: Flat Q-learning

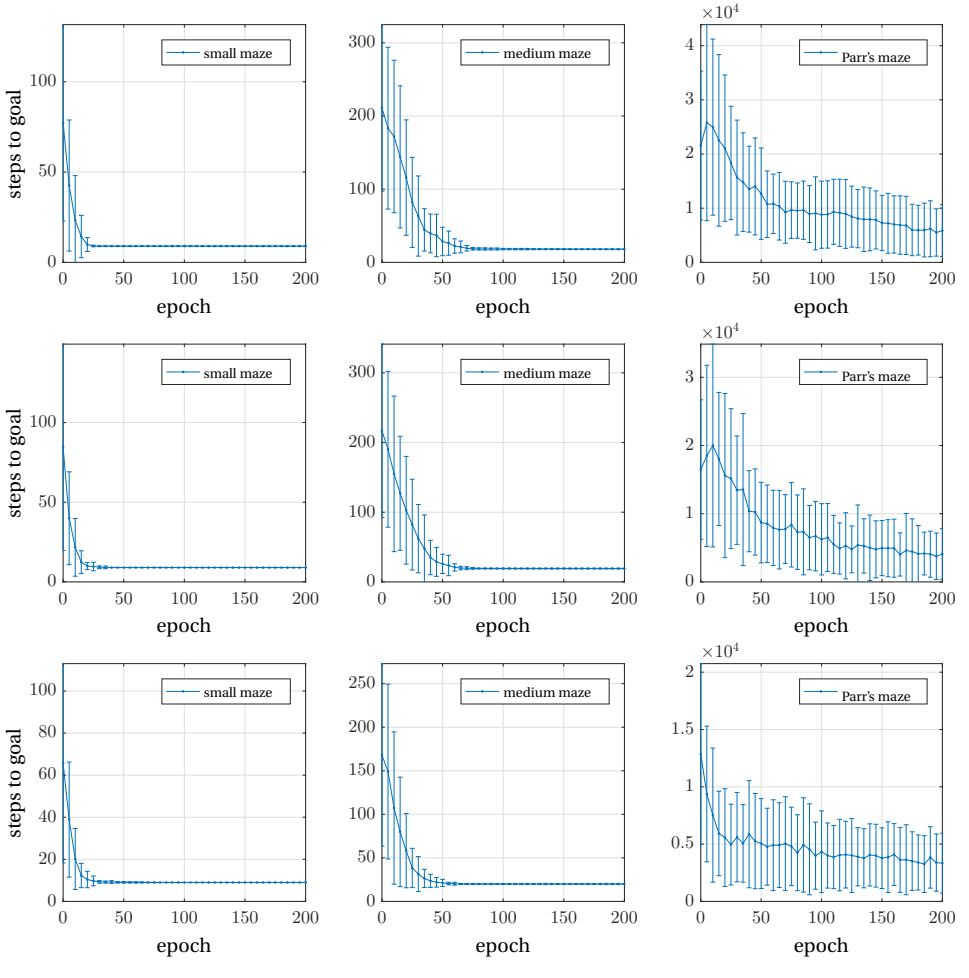


Figure A.10: HRL options with optionsets 1a(top row), 1b(middle row), 1c(bottom row)

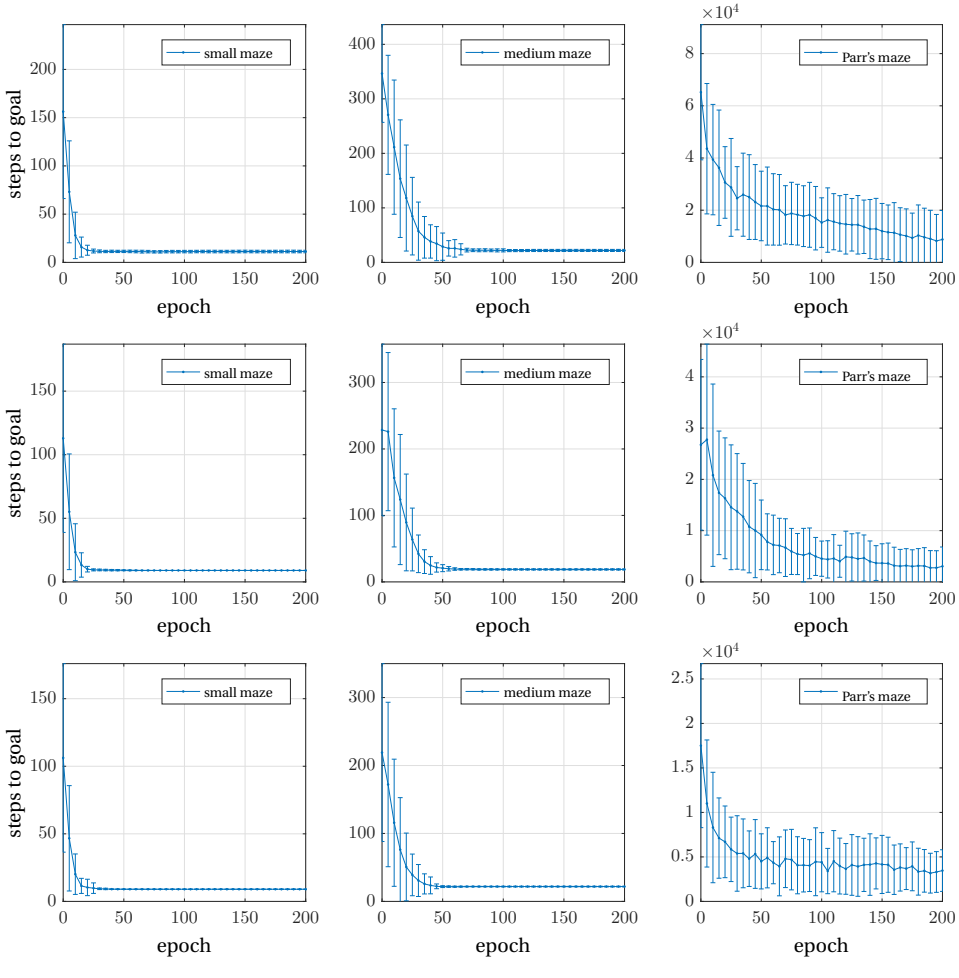


Figure A.11: HRL options with optionsets 2a(top row), 2b(middle row), 2c(bottom row)

B

CHAPTER 4 SUPPORTING STUDIES

B.1. OPTIONSET SELECTION

In Chapter 4, seven optionset configurations and the flat Q-learning configuration were tested on 3 different mazes with 2 different parameter schemes. From the analysis of the results several take-away conclusions are determined but much of the data support redundant conclusions and only serve to clutter the plots. In order to show clear and concise results it was determined that one HRL options configuration would be sufficient to represent the HRL approach against the flat case for the small and medium mazes. The Parr's maze, being a larger and more complex problem is sufficiently explained with 2 optionsets and the flat case.

Several arguments are made to select the representative optionsets. For example, an optionset which performs well is obviously desirable as it is fair to show the best that HRL has to offer. Furthermore, to accommodate a fair comparison against the absolute state representation results from Chapter 3, a consistently good performer in both absolute and relative state representations is selected. Rationale for the final selected optionsets for each maze are found in Table B.1.

Table B.1: Rationale for selected optionsets of each maze size

Maze	Reference Figure(s)	rationale	selected optionset(s)
general trends	B.1-B.2	Set 2a is generally the worst of the optionsets. Set 1 series includes all the primitive actions and is desirable over series 2 because it is simpler to design and converges faster.	-
small	B.1 3.7	Sets 1a and 1b have the best solutions for relative state. In absolute state, all configurations perform well.	set 1b
medium	B.2 3.8	Set 1c learns fastest and converges to the best solution. Learns fast in absolute state rep. but doesn't converge to the optimal.	set 1c
Parr's	B.5 3.10	Bigger mazes work best with longer temporally extended options. Therefore, Set 3 was especially made for Parr's maze to tackle the complex problem. Also select set 1c to compare to other cases.	sets 1c & 3

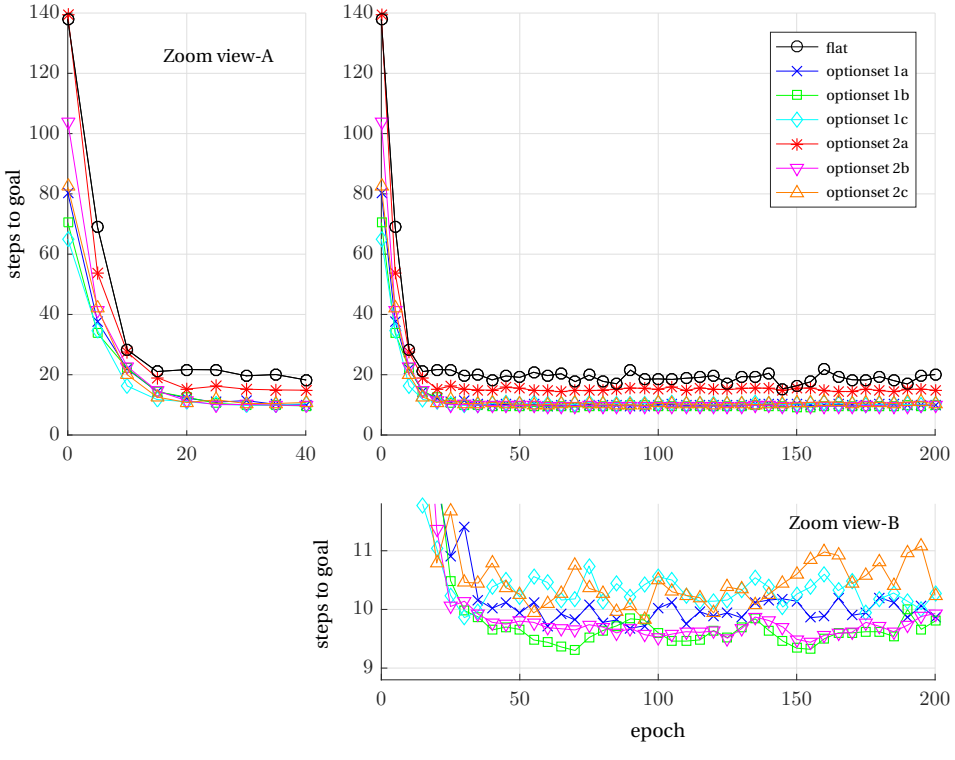


Figure B.1: **Small maze** - Evaluation performance option set comparison: Average over 100 sample runs. relative state representation - ($d_s = 3 / n_d = 3$) - scheme 1.

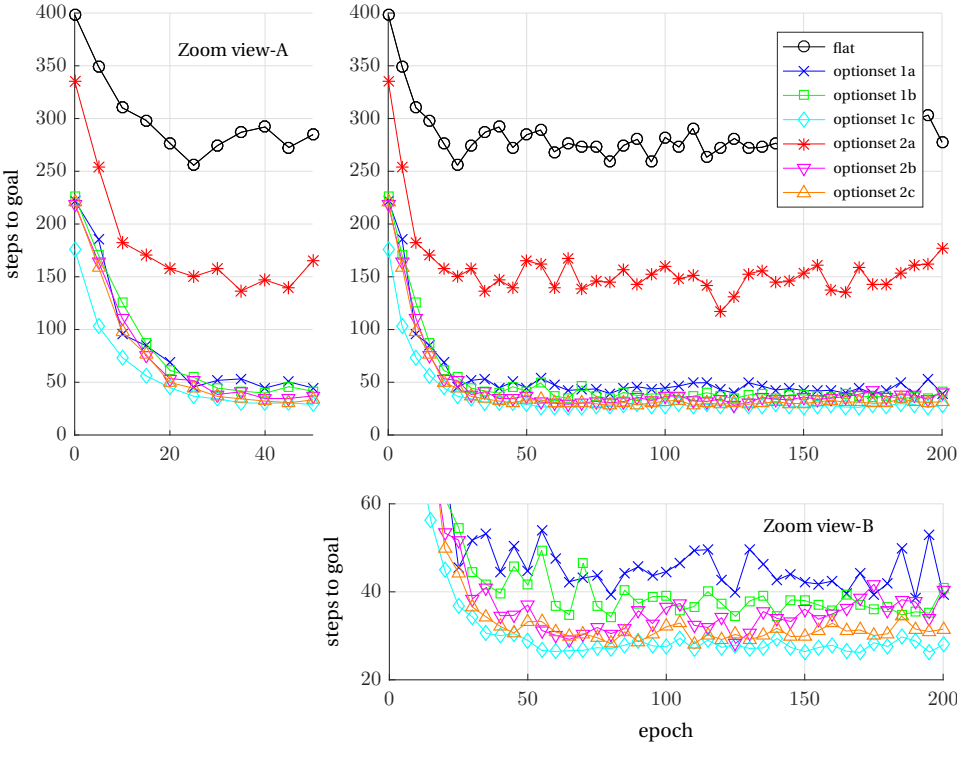


Figure B.2: **Medium maze** - Evaluation performance option set comparison: Average over 100 sample runs. relative state representation - ($d_s = 3 / n_d = 3$) - scheme 1.

B.2. PARAMETER TUNING

In Chapter 3 with absolute state representation, the flat Q-learning and HRL configurations learned well with a random training policy and a constant step-size, α in the bellman update equation. Using a relative state, particularly for the flat Q-learning configurations, the reinforcement learning approach is not as successful with these parameter selections. When evaluating the benefit of Hierarchical RL over flat RL, it is important that the flat configuration is not shown in a disadvantaged light. Therefore, in order to present a more complete picture of the different representations and configuration abilities, it was determined that it would be beneficial to tune the parameters to come up with a parameter scheme for the relative state representation scenario.

The process to choose these values will now be explained. First, the parameters to modify for a relative state representation (scheme 2) will be determined by starting from the absolute state representation parameters (scheme 1) and modifying one parameter at a time (Figure B.3). Then a comparison will be made for each maze between the two schemes (Figures B.4-B.5).

The parameters used for absolute state representation and relative state representation are as follows :

parameter	Absolute state representation (scheme 1)	Relative state representation (scheme 2)
Bellman equation		
γ	0.9	0.9
α	0.9	$\frac{1}{\sqrt[5]{k}}$
training policy		
ϵ	0.0 (random)	0.2

TRAINING ϵ-GREEDY POLICY

In Appendix A.1.1, the ϵ value of the ϵ-greedy training was studies for the absolute state representation and it was found that $\epsilon = 0.2$ resulted in a positive step-size effect at the cost of having a less optimal solution than a purely random training. The same $\epsilon = 0.2$ is used as one of the tuned parameters to explore for the relative state case.

BELLMAN UPDATE PARAMETERS

From the study in Appendix A.1.2, it was found that adding eligibility traces was not desirable. It would be even less likely to be helpful with a relative state space due to state ambiguity.

With the ambiguity present along with the relative state representation, convergence can be an issue. While the scheme 1 configuration uses constant $\alpha = 0.9$, the scheme 2 configuration uses a variable $\alpha(s,a)$ which decreases over k visits to (s,a). Several RL algorithms such as Policy Iteration RL, have convergence guarantees conditional on the step-size where α should be positive and satisfy $\lim_{k\to\infty}\alpha_k = 0$ and $\sum_k \alpha_k = \infty$ [128].

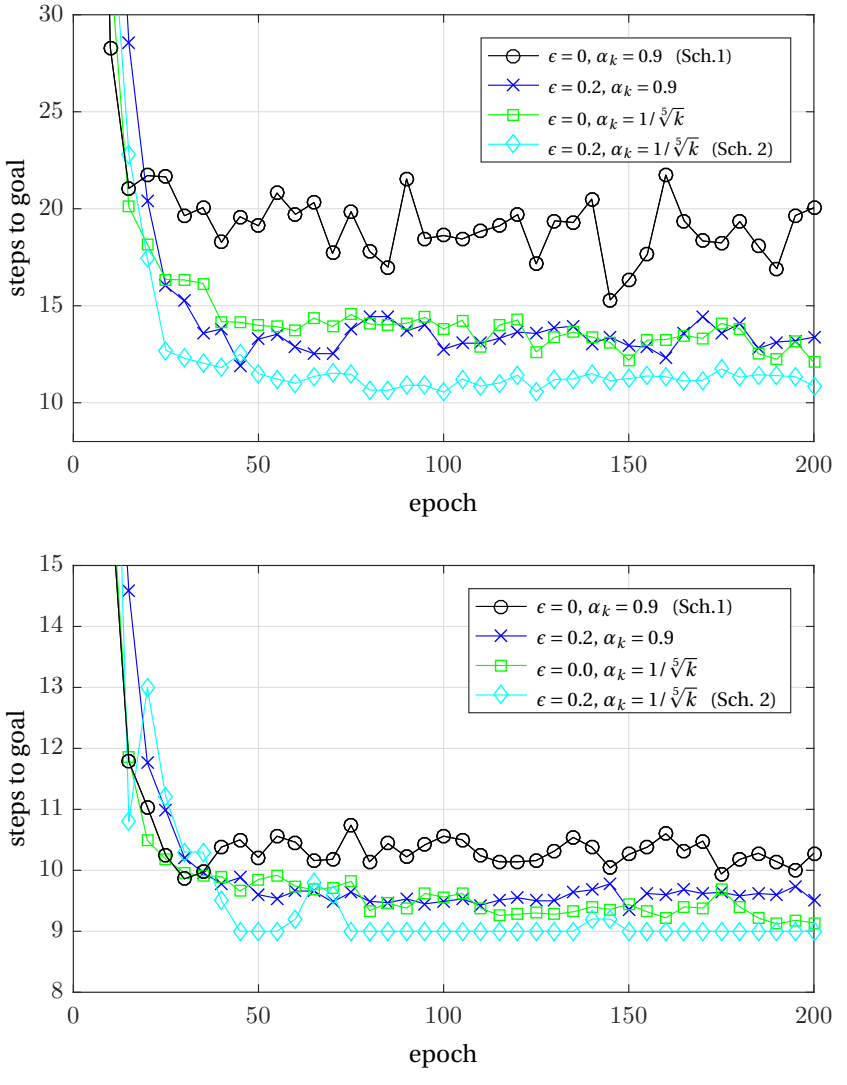


Figure B.3: Evaluation performance of small maze with various parameter modifications: averaged over 100 sample runs. (*top*) flat Q-learning configuration, (*bottom*) optionset 1c configuration. ($d_s = 3/n_d = 3$)

While there is no mathematical guarantee for convergence, one can see from the update equation that if the step-size is constant, then all visits to the state-action pairing will have equal influence on the Q-value and when a state can exist in several different locations in the maze, the Q-value will jump around and affect the performance in a sporadic way. A desirable trend for step-size was determined to be given by $\alpha_k = \frac{1}{\sqrt[5]{k}}$.

B.2.1. RESULTS

DETERMINING PARAMETERS

The plots in Figure B.3 compare the small maze's evaluation results of the absolute representation parameters against parameters with modifications to ϵ, α and both. Each figure plots the average evaluation response over 100 sample runs.

Figure B.3(*top*) takes results from the flat Q-learning configuration. The response of scheme 1 learns a better Q-function quickly but converges on a suboptimal solution around 19 steps, while the optimal is 9. Modifying ϵ and α separately improves upon the performance of the scheme 1 parameters, and modifying both parameters gives the best result. Though the optimal is still not found, on average.

Figure B.3(*bottom*) takes results from the optionset 1c configuration. Note that with the small maze, optionset 1c performs worse than 1a and 1b, so it is not the strongest optionset and therefore has room for improvement. The trends are the same as with the flat case. Modifying ϵ and α separately improves upon the performance of the scheme 1 parameters, and modifying both parameters gives the best result; even finding the optimal solution in all 100 sample runs for several evaluations epochs.

With the results of this short study it was determined to use both modified ϵ and α as a configuration tuned for the relative state representation which could then show more accurately the capabilities of this approach. Furthermore, since this parameter configuration improves the flat Q-learning response the most, it is also contributes toward an honest comparison of HRL vs. flat capabilities.

EFFECT ON RESPONSE

Parameter scheme 2 for the relative state representation is used in a comparison against scheme 1 to find the effect on each maze. Each plotted result is using the relative state representation with ($d_s = 3 / n_d = 3$).

In Figure B.4, the evaluation performance averaged over 100 sample runs are shown for the small maze with flat and optionset 1b configurations(*left*), and the medium maze with flat and optionset 1c configurations (*right*). Figure B.5 shows the results for the Parr's maze where the *left* side plot is with parameter scheme 1, and the right side uses parameter scheme 2. In each case, the runs of the flat case and optionsets 1c, 2c, and 3 and is averaged over 10 sample runs.

In the small and medium sized mazes, the tuning of scheme 2 improves all the cases, however it improves the flat case more than its HRL options counterpart. Furthermore, the new tuned parameters only help to converge the solution closer to the optimal value; it doesn't improve upon the convergence speed and is slightly slower on average (though not significantly within the standard deviation).

The performance within the larger environment of the Parr's maze doesn't easily learn which makes the comparison different but still meaningful. The flat configuration

doesn't learn; as in, the performance doesn't improve over the 200 training epochs. However it inherently converges (without learning) to some degree of poor performance and that degree of "poorness" is different between optionsets and parameter schemes. The flat case improves with scheme 2 by about 4,059 steps on average in the first 200 epochs. Optionset 2c improves with scheme 2 by about 11,810 steps on average in the first 200 epochs. Optionset 1c improves by about 3,320 steps on average in the first 200 epochs. Optionset 3 improves by about 968 steps on average in the first 200 epochs. Parameter scheme 2 also appears to enable learning in optionsets 1c and 3. Where there was no or very small improvement in the first 20 epochs for scheme 1, for scheme 2 there is a marked improvement in the first 20 epochs and convergence to a range of values better than at epoch 0.

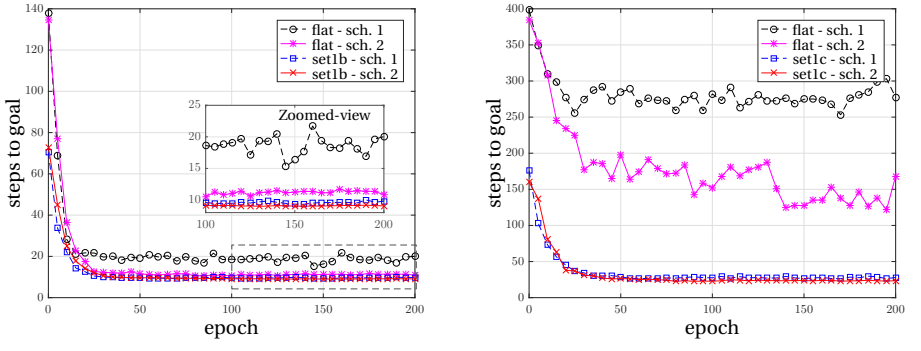


Figure B.4: Evaluation performance comparing scheme 1 and scheme 2. Average over 100 sample runs. Relative state representation ($d_s = 3/n_d = 3$) *(left)* small maze. *(right)* medium maze.

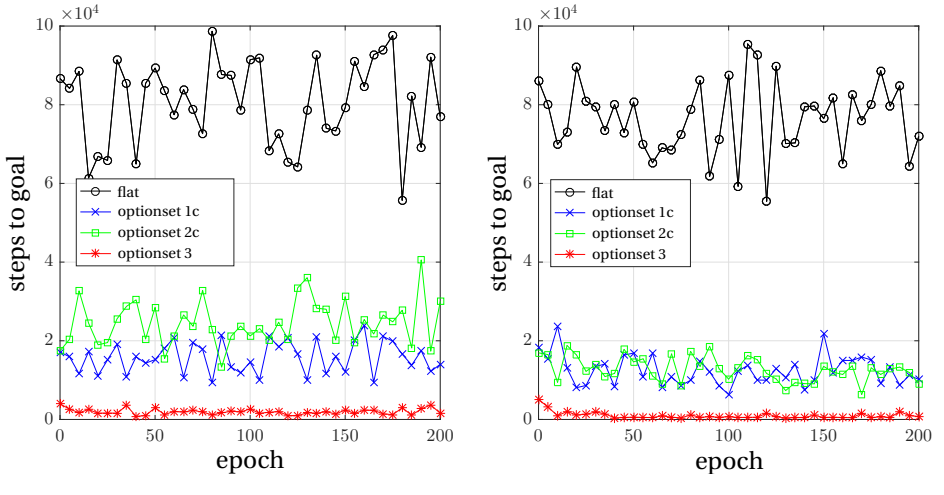


Figure B.5: **Parr's maze** Evaluation performance comparing scheme 1(*left*) and scheme 2(*right*). Average over 10 sample runs. Relative state representation ($d_s = 3/n_d = 3$)

B.2.2. CONCLUSION

The conclusion from this short study is that for the relative state representation, parameter scheme 2 is overall a better fit than scheme 1, and switching to this set of parameters results in better performance, without creating additional bias between the configurations. It actually improves the flat case most, therefore aiding in an honest comparison in the context of this research. Therefore, it is deamed appropriate to use parameter scheme 2 for all the results in Chapter 4.

C

CHAPTER 5 SUPPORTING STUDIES

C.1. STATE SPACE MATRICES FROM LINEARIZED F-16 MODEL

In Chapter 5, a non-linear F-16 model is linearized to obtain the state space matrices about different trim conditions. The linearized model of these states take the form of Eq. (5.5) from Section 5.3, which is repeated here:

$$\dot{\vec{x}} = A\vec{x} + B\vec{u} \quad \text{where} \quad \vec{x} = \begin{bmatrix} h & \theta & V & \alpha & q \end{bmatrix}^T \quad \text{and} \quad \vec{u} = \begin{bmatrix} F_T & \delta_{elev} \end{bmatrix}^T$$

For each of the conditions, the C and D matrices are the same and are as follows:

$$C = \begin{bmatrix} 1.00 & 0 & 0 & 0 & 0 \\ 0 & 57.2958 & 0 & 0 & 0 \\ 0 & 0 & 1.00 & 0 & 0 \\ 0 & 0 & 0 & 57.2958 & 0 \\ 0 & 0 & 0 & 0 & 57.2958 \end{bmatrix}; \qquad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The A and B matrices differ. The following are the A and B matrices for the stated trim conditions.

Altitude: 15,000 ft, Velocity: 500 ft/s

$$A = \begin{bmatrix} 0 & 500 & 0 & -500 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.0001 & -32.17 & -0.0133 & -7.3259 & -1.1965 \\ 0 & 0 & -0.0003 & -0.6398 & 0.9378 \\ 0 & 0 & 0 & -1.5679 & -0.8791 \end{bmatrix}; \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.0016 & 0.0740 \\ 0 & -0.0014 \\ 0 & -0.1137 \end{bmatrix}$$

Altitude: 15,000 ft, Velocity: 400 ft/s

$$A = \begin{bmatrix} 0 & 400 & 0 & -400 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.0001 & -32.17 & -0.0184 & -4.3245 & -1.9977 \\ 0 & 0 & -0.0004 & -0.5128 & 0.9376 \\ 0 & 0 & -0 & -1.3756 & -0.7532 \end{bmatrix}; \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.0016 & 0.0186 \\ -0 & -0.0011 \\ 0 & -0.0730 \end{bmatrix}$$

Altitude: 15,000 ft, Velocity: 600 ft/s

$$A = \begin{bmatrix} 0 & 600 & 0 & -600 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.0001 & -32.17 & -0.0113 & 3.8334 & -0.6570 \\ 0 & 0 & -0.0002 & 0.7679 & 0.9396 \\ 0 & 0 & -0 & -2.2510 & -1.0462 \end{bmatrix}; \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.0016 & 0.1448 \\ -0 & -0.0016 \\ 0 & -0.1642 \end{bmatrix}$$

C.2. SELF-TUNING GAINS COMPILED RESULTS

Table C.1 is a copy of Table 5.1 from Section 5.3.5, displayed here for easy reference.

Table C.1: Table of experiments

no.	Platform	Inaccurate model	True system	Task	# of policy parameters	Linesearch method(s)
1	F-16	15000/400 sim	nonlinear sim	sinusoid	2-gain (θ-ctrl)	stepsize & sim-based
2	F-16	15000/400 sim	nonlinear sim	block-wave	2-gain (θ-ctrl)	stepsize & sim-based
3	F-16	15000/500 sim	nonlinear sim	block-wave	3-gain (γ-ctrl)	stepsize & sim-based
4	F-16	15000/500 sim	nonlinear sim	block-wave	4-gain (alt-ctrl)	stepsize & sim-based
5	Quadrotor	simple mdl	pprz sim	takeoff	3-gain(PID)	sim-based
6	Quadrotor	simple mdl	AR.Drone	takeoff	3-gain(PID)	sim-based

The final results from all the policy improvement experiments have been compiled in Table C.2. This is an expanded version of Table 5.6.

Units for ρ depend on the task. Pitch angle and flight path angle tracking tasks measure in radians (*no.* 1 – 3), the F-16 4-gain altitude tracker and the quadrotor takeoff tasks are measured in meters (*no.* 4 – 6).

Table C.2: Compiled results

no.	initial gains, $\vec{k}_0$	linesearch	ρ_0	ρ^*	improvement(%)	ρ^* trial #	tot. # trials
1	*F-16 θ-ctrl sinusoid*						
	[−10 5]	sim	0.0495	0.0069	86	3	6
		stepsize	0.0495	0.0069	85	11	11
	[−10 10]	sim	0.0248	0.0067	73	6	9
		stepsize	0.0248	0.0067	73	7	10
2	*F-16 θ-ctrl block-wave*						
	[−10, 5]	sim	0.0146	0.0104	29	6	6
		stepsize	0.0146	0.0109	25	5	8
	[−50, 2]	sim	0.0098	0.0073	26	2	5
		stepsize	0.0098	0.0072	27	7	10
3	*F-16 γ-ctrl block-wave*						
	[−50, 1, 5]	sim	0.0127	0.0112	12	4	4
		stepsize	0.0127	0.0124	2	4	4
	[−1000, 0.05, 100]	sim	0.0087	0.0064	26	6	9
		stepsize	0.0087	0.0064	26	9	12
4	*F-16 altitude-ctrl block-wave*						
	[−100, 2, 0.1, 0.005]	sim	33.20	24.12	27	2	5
		stepsize	33.20	23.98	28	5	8
	[−500, 0.5, 0.01, 0.1]	sim	47.47	44.19	7	2	5
		stepsize	47.47	45.21	5	4	4
5	*quadrotor takeoff simulation*						
	[283, 13, 82]	sim	0.3775	0.1521	60	5	5
	[1624, 1356, 195]	sim	0.1006	0.0940	7	3	4
6	*quadrotor takeoff real-life*						
	[283, 13, 82]	sim	0.5102	0.2265	56	3	5
	[1624, 1356, 195]	sim	0.1218	0.1218	0	1	3

D

CHAPTER 6 SUPPORTING STUDIES

D.1. PARAMETER TUNING

In Chapter 6, a comparison between reinforcement learning methods is made. To give a fair comparison, each method should have parameters which result in a representative performance. For that reason, a parameter study was conducted for each method.

It was observed by trial and error of single runs, that the performance is sensitive to the parameters:

ϵ The greediness factor for action policy

α The learning rate

γ The discount factor

The procedure for parameter selection went as follows:

1. Begin with default parameter values based on observations from single run trial and error
2. Study statistical evaluation with various ϵ values (constant and varying)
3. Choose a default ϵ based on convergence performance
4. Use new default and perform statistical evaluation with constant and varying α
5. Choose a default α based on convergence performance
6. Use the new default ϵ and α and perform statistical evaluation with various γ values

Table D.1: Summary of parameter selection after study

	beeworld 1			beeworld 2[1]		
parameter	V	Q	HRL	V	Q	HRL
ϵ	0.6	0.6	0.6	0.8	0.8	Scheduled
α	0.2	0.3	0.3	0.3	0.2	$1/\sqrt[4]{k}$
γ	0.90	0.90	0.90	0.90	0.90	0.90

This procedure was carried out on the Value function TDRL, Q-learning, and HRL with *options* on each of the two Beeworld designs. A selection of the results from the above procedure can be seen in Section D.2. The final parameter selections are summarized below in Table D.1.

Given similar evaluation performance, preference was given to parameter values which consistently perform well across different configurations. Furthermore, easier implementation was also considered. For example, variable learning rate, α, involves storing another table with the same dimensions as the Q-function, in order to keep track of number of times each state/action pair is visited. Using a constant α, doesn't require this extra large variable. Therefore, even though convergence appears to be slightly better over 50 statistical runs for a variable α, the constant value was implemented in the study. For the purposes of comparison, the results will still be valid. In Section 6.4.3, some of the results diverge using constant α, and a variable α proves to converge. Therefore, modifying α can be seen as a useful tool in case of poor convergence.

It is a common practice to increase the greediness of an ϵ-greedy policy through training in order to begin with more exploration when little is known about the environment, and then later exploit the knowledge gained. Three "scheduled" ϵ-greedy policies were designed as another possible ϵ solution for comparison. The schedules are shown in Figure D.1. For the parameter search, Schedule 1 and 2 were compared to the constant ϵ parameters. There are infinite ways to schedule ϵ, and some other methods, such as convergence-based scheduling was also tried on specific configurations. After the parameter study and more trial and error, Schedule 3 was determined to be a good fit for the HRL configuration. It stops at $\epsilon = 0.8$ because going fully greedy only reinforces the current optimal path, which leaves no room for improvement. This could be a good thing if the current learned behavior is optimal, but since that may not be knowable, it is beneficial to train with some random actions.

[1] Beeworld 2 with nectar regeneration time, $t_{nr} = 12$ timesteps.

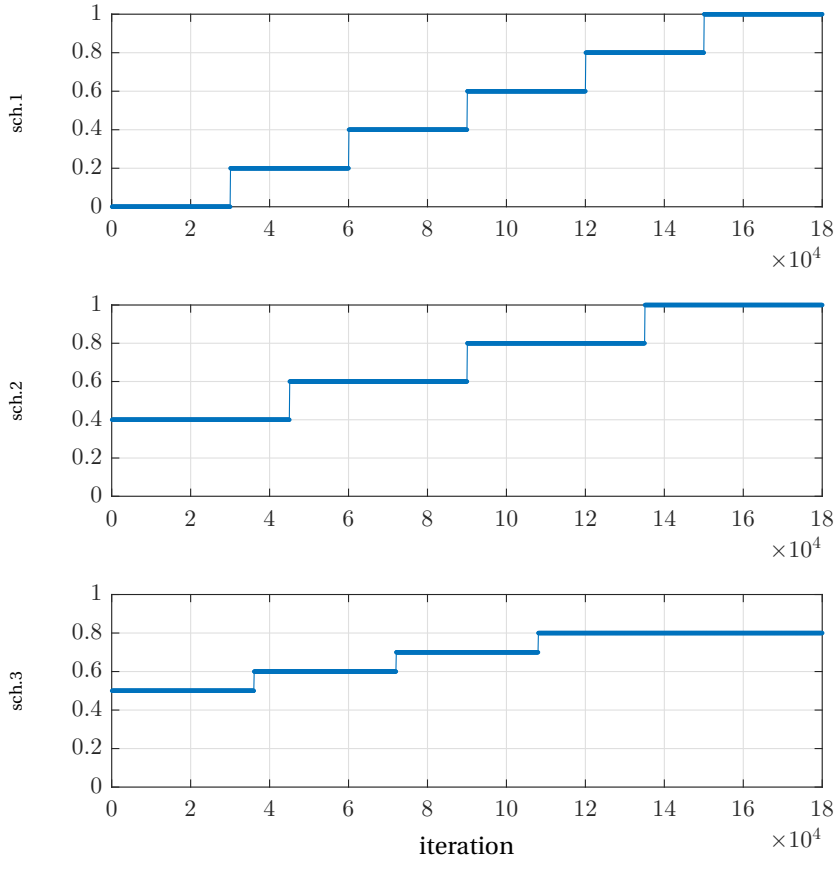


Figure D.1: Examples of time-scheduled ϵ-greedy policies. *(top)* Schedule 1 ramps up from fully random to fully greedy in order to trade-off exploration for exploitation. *(middle)* Schedule 2 is useful when fully random exploration is not beneficial, such as in beeworld 2.

D.2. SELECT INFORMATION FROM THE PARAMETER STUDY

The select information shown in the following figures are plotted in two different ways. The results of each individual parameter set, ran 50 times, can be seen in individual subplots plotted as mean and standard deviation in the shaded region. The other kind of plot shows the averages of several configurations, as a comparison.

D.2.1. BEEWORLD 1

In Beeworld 1, we see that the performance is robust to a large range of parameter values and within this range will converge to the optimal 100% of the time (the standard deviation goes to 0). We can see this through the selected plots showing a range of ϵ values for the flat Q-learning case in Figure D.2 and the HRL case in Figure D.3.

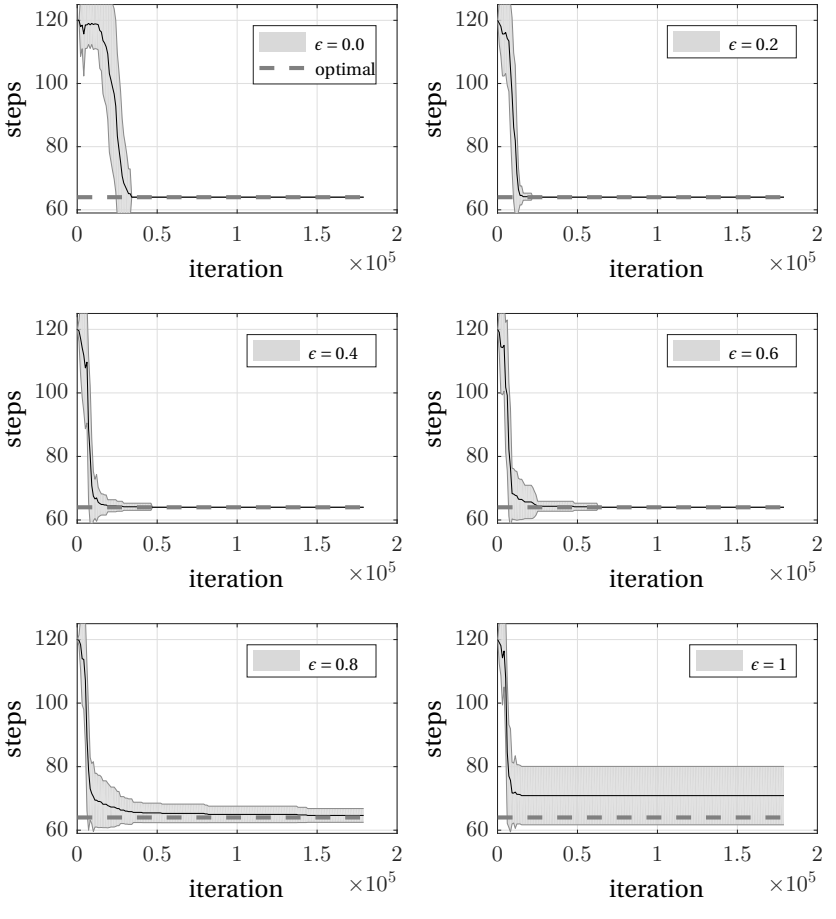


Figure D.2: Q-learning parameter study: ϵ. Evaluation results with standard deviation over 50 statistical runs.

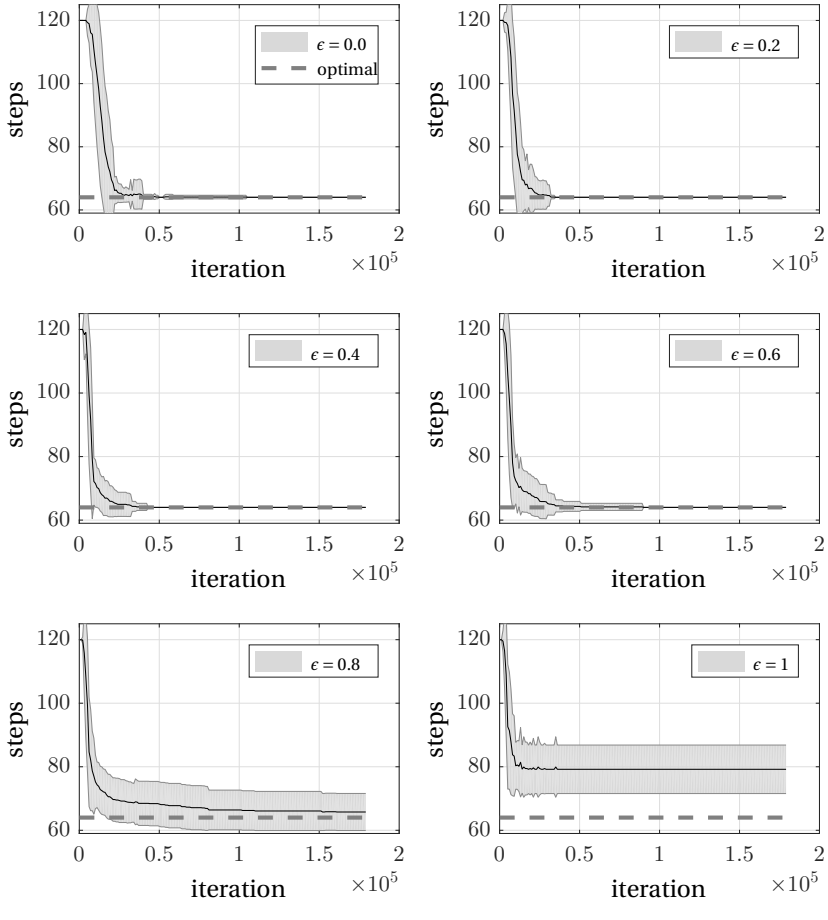


Figure D.3: HRL parameter study: ϵ. Evaluation results with standard deviation over 50 statistical runs.

D.2.2. BEEWORLD 2

For Beeworld 2, the range of robust parameters is more restrictive than for Beeworld 1.

FLAT Q-LEARNING

Figure D.4 shows the performance of various ϵ values for flat Q-learning, before α and γ are tuned. The resulting choice was to use $\epsilon = 0.8$ for this method. Figure D.5 shows how γ effects the performance on average after ϵ and α are tuned. In almost all cases, $\gamma = 0.9$ was the best choice or just slightly worse than the best performing γ; therefore, it was decided to use $\gamma = 0.9$ consistently throughout the study.

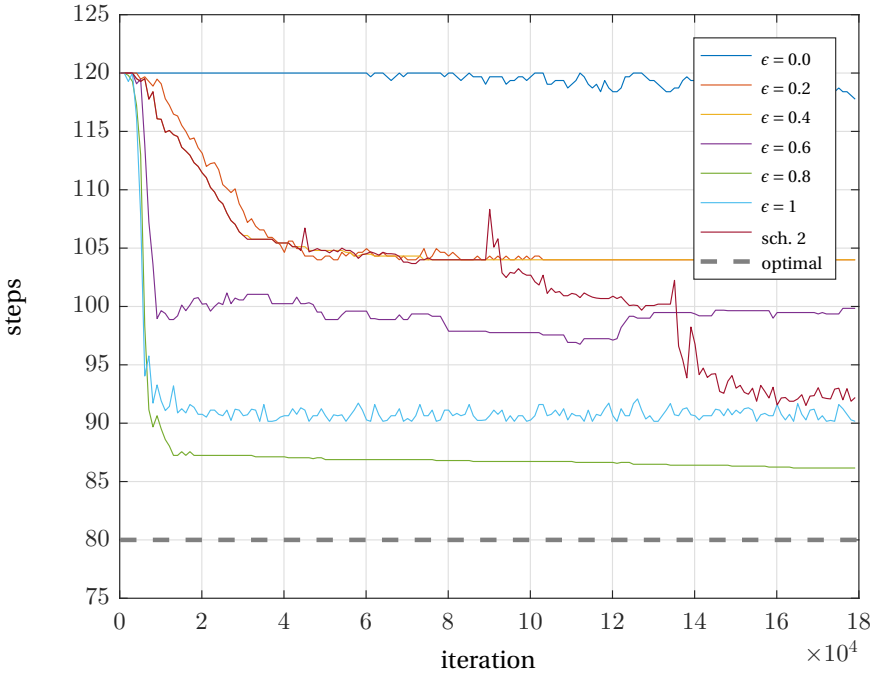


Figure D.4: Q-learning parameter study: ϵ. Evaluation average results over 50 statistical runs.

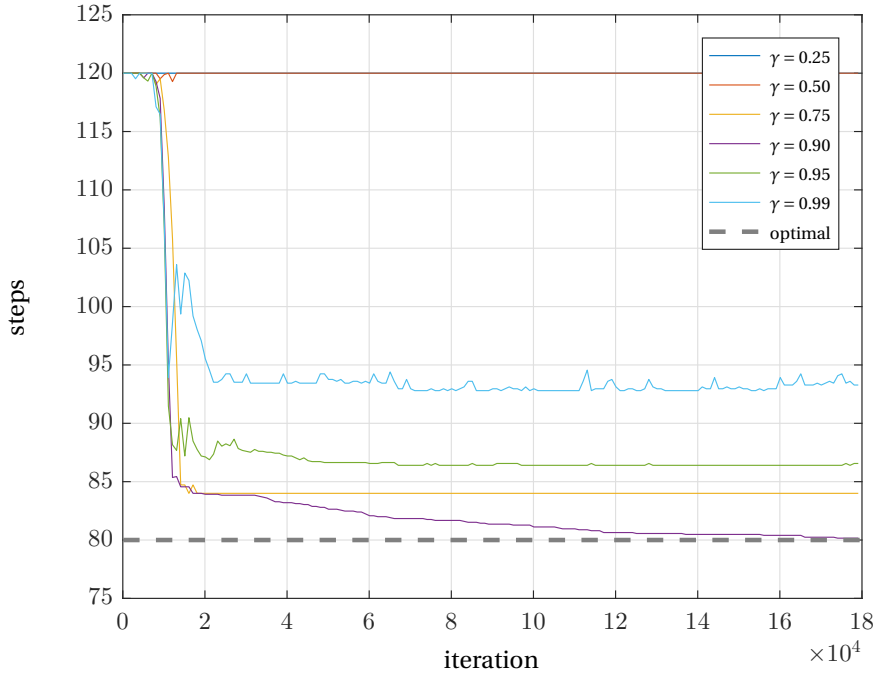


Figure D.5: Q-learning parameter study: γ. Evaluation average results over 50 statistical runs.

HIERARCHICAL REINFORCEMENT LEARNING WITH *options*

Figure D.6 and Figure D.7 show the same data in different forms: the former comparing the means of each ϵ value performance in a single plot, and the latter showing the mean and standard deviation of each dataset in its own subplot. The interesting conclusion from the parameter search for HRL was that the scheduled ϵ is far superior in finding the optimal evaluation path, where as a constant ϵ was best for V-TDRL and Q-learning. This can be because using high greed in early training, when not much is known about the value function, is more wasteful when using extended actionsets than with single step methods.

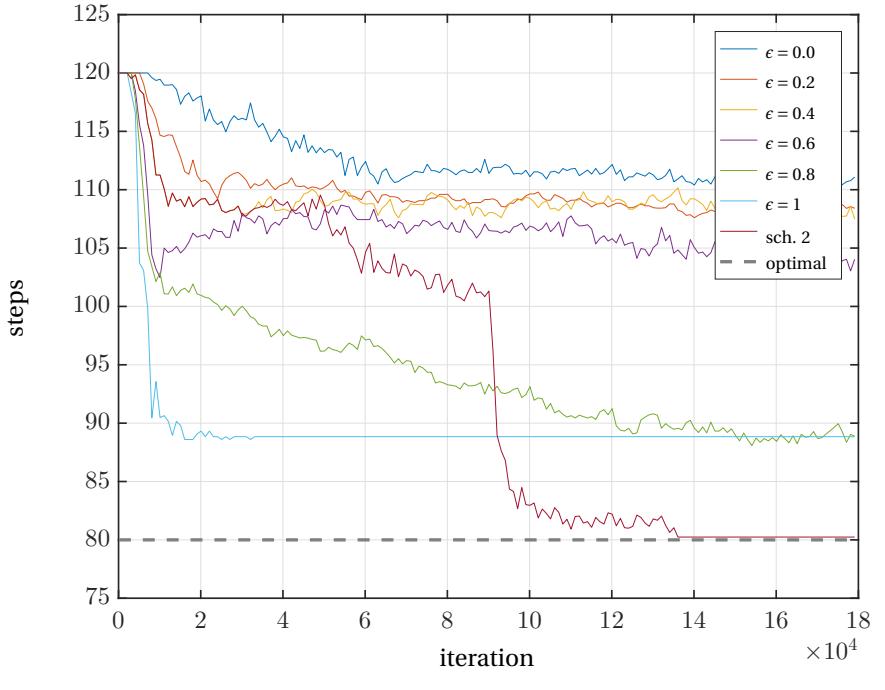


Figure D.6: HRL parameter study: ϵ. Evaluation average results over 50 statistical runs.

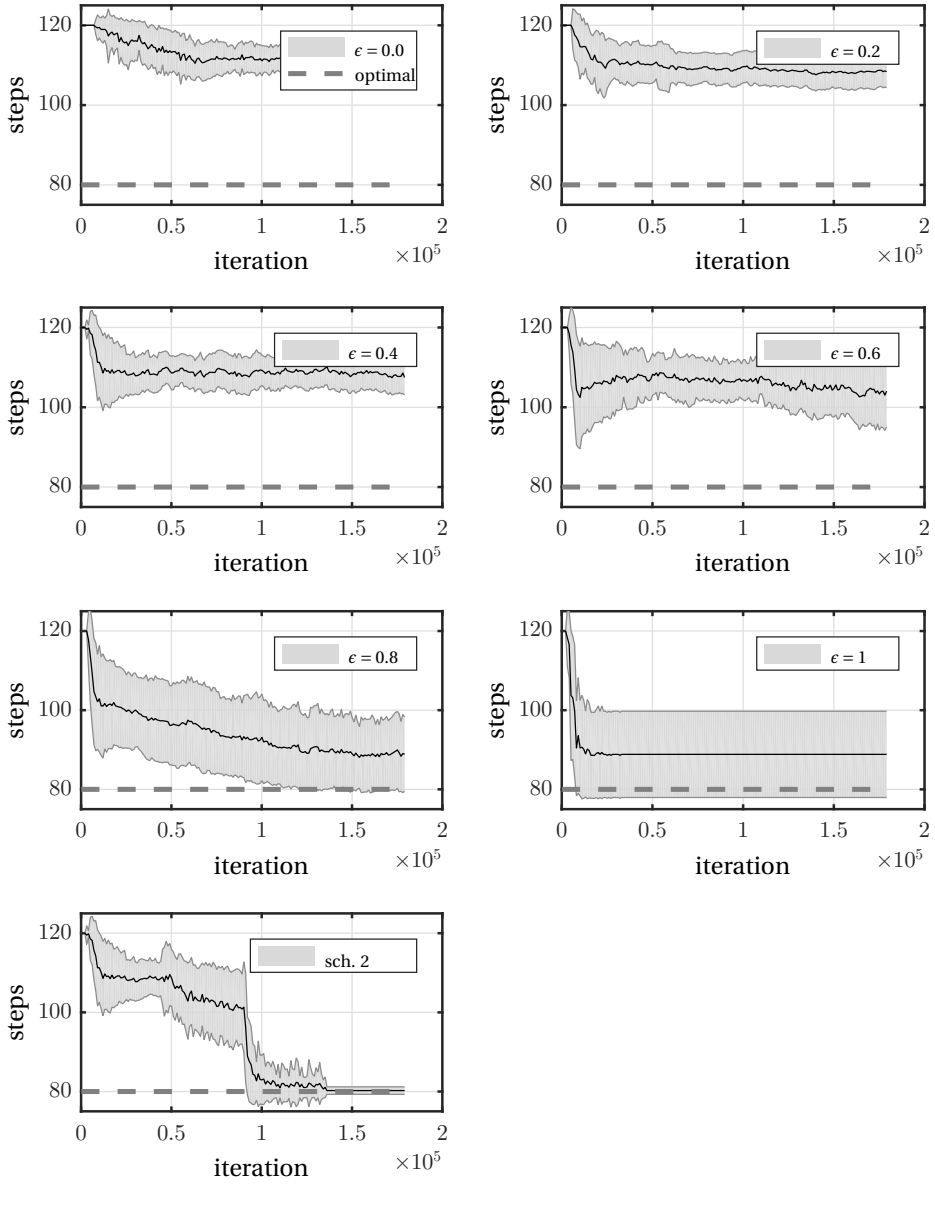


Figure D.7: HRL parameter study: ϵ. Evaluation results with standard deviation over 50 statistical runs.

D.2.3. BEEWORLD 2, VARYING t_{nr}

With the parameters tuned, it is also important to see how it affects different values of t_{nr} for the transferability study.

The results from t_{nr} values within each of the ranges are found in Figure D.8 using ϵ-schedule 2. The optimal number of steps are demonstrated with a dashed line and the average and standard deviation of 50 runs evaluated over 18000 iterations are plotted. The optimal path is normally found, except for $t_{nr} = 3$. For $t_{nr} > 20$, the optimal is not known.

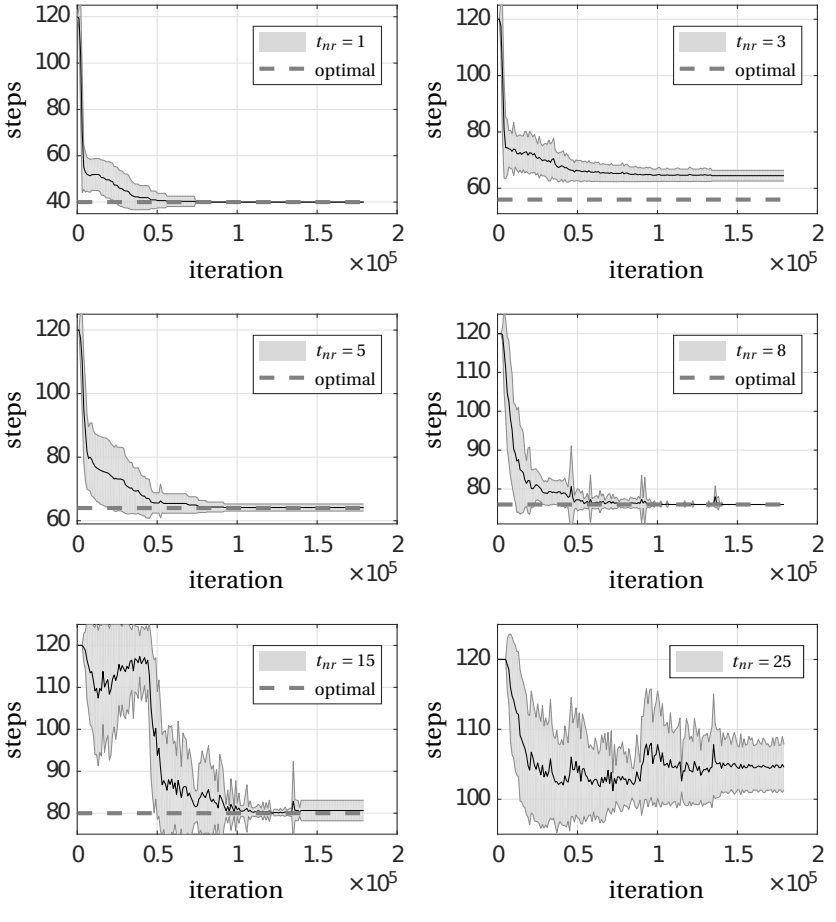


Figure D.8: Beeworld 2: study of t_{nr} ranges. Results here use ϵ schedule 2 and a modified optionset to include primitive actions. Options are: [N, E, S, W, $N \times 3$, $E \times 3$, $S \times 3$, $W \times 3$]

REFERENCES

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *In Advances in Neural Information Processing Systems*. MIT Press, 2007.

[2] P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *International conference on Machine learning*, New York, New York, USA, 2006.

[3] AIAA Intelligent Systems Technical Committee (ISTC). *AIAA Intelligent Systems in Aerospace Workshop. NASA Langley, Hampton, VA*, August 3-5, 2016.

[4] AIAA Intelligent Systems Technical Committee (ISTC). Roadmap for Intelligent Systems in Aerospace, 1st edition. C. Tschan and A. Yucel and N. Nguyen, editors, June 2016.

[5] D. Andre and S. J. Russell. State Abstraction for Programmable Reinforcement Learning Agents. In *18th National Conference on Artificial Intelligence*, pages 119–125, Menlo Park, CA, USA, 2002.

[6] S. F. Armanini, C. C. de Visser, and G. C. H. E. de Croon. Black-box LTI modelling of flapping-wing micro aerial vehicle dynamics. In *AIAA Science and Technology Forum: AIAA atmospheric flight mechanics conference*, Kissimmee, Florida, USA, January 2015.

[7] M. Asada, K. Hosoda, and S. Suzuki. Vision-based behavior learning and development for emergence of robot intelligence. In Y. Shirai and S. Hirose, editors, *Robotics Research: The 8th International Symposium*, pages 327–338. Springer, London, 1998.

[8] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2):279–303, May 1996.

[9] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó. The SLAM problem: A survey. *Frontiers in Artificial Intelligence and Applications*, 184(1):363–371, 2008.

[10] B. Bakker. Reinforcement learning with long short-term memory. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, pages 1475–1482, Vancouver, BC, Canada, 2001. MIT Press.

[11] S. Barrett, M. E. Taylor, and P. Stone. Transfer learning for reinforcement learning on a physical robot. In *9th International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS - ALA)*, 2010.

[12] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming . *Artificial Intelligence*, 72(1–2):81 – 138, 1995.

[13] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[14] A. G. Barto and S. Singh. On the Computational Economics of Reinforcement Learning. In *Proceedings of the 1990 Connectionist Models Summer School, San Mateo, CA: Morgan Kaufmann*, pages 35–44, 1990.

[15] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. Safe Model-based Reinforcement Learning with Stability Guarantees. In *31st Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017.

[16] H. Bijl, E. van Kampen, Q. P. Chu, and B. Mulder. Guaranteed globally optimal continuous reinforcement learning. *52nd Aerospace Sciences Meeting. National Harbor, Maryland*, 2014.

[17] G. Boothroyd. *Assembly Automation and Product Design* . Boca Raton: CRC Press., second edition, 2005.

[18] H. Bou-Ammar, H. Voos, and W. Ertel. Controller design for quadrotor UAVs using reinforcement learning. In *2010 IEEE International Conference on Control Applications*, pages 2130–2135, Yokohama, Japan, September 2010. IEEE.

[19] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the 13th National Conference on Artificial Intelligence*, volume 2 of *AAAI'96*, pages 1168–1175. AAAI Press, 1996.

[20] J. M. Bradshaw, R. R. Hoffman, D. D. Woods, and M. Johnson. The seven deadly myths of 'autonomous systems'. *IEEE Intelligent Systems*, 28(3):54–61, 2013.

[21] P. Brisset and G. Hattenberger. Multi-UAV control with the paparazzi system. In *Conference on Human Operating Unmanned Systems*, 2008.

[22] R. A. Brooks. Planning is just a way of avoiding figuring out what to do next. *Cambrian Intelligence the early history of the new AI*, pages 103–110, 1987.

[23] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska. Policy search with cross-entropy optimization of basis functions. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 153–160, March 2009.

[24] J. V. Caetano, J. Verboom, C. C. de Visser, G. C. H. E. de Croon, B. D. W. Remes, C. De Wagter, and M. Mulder. Near-Hover Flapping Wing MAV Aerodynamic Modelling - a linear model approach. *International Journal of Micro Air Vehicles*, 5(4), 2013.

[25] F. Cayzer, P. Chesham, and P. Wilkinson. Unmanned Aircraft Systems (UAS) – keeping the human in the picture. In M. Anderson, editor, *Proceedings of the international conference on Ergonomics & Human Factors 2012*, pages 65–66, Blackpool, UK, 2012. CRC Press 2012.

[26] S. A. Chechetka, Y. Yu, M. Tange, and E. Miyako. Materially Engineered Artificial Pollinators. *Chem*, 2(2):224 – 239, 2017.

[27] Z. Cheng and L. E. Ray. State Abstraction in Reinforcement Learning by Eliminating Useless Dimensions. In *13th International Conference on Machine Learning and Applications*, pages 105–110, 2014.

[28] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *CoRR*, 10 2016.

[29] B. Clough. Metrics, Schmetrics! How Do You Track a UAV's Autonomy? *1st AIAA UAV Systems Technologies and Operations Conference and Workshop*, May 2002.

[30] L. C. Cobo, P. Zang, C. L. Isbell, Jr., and A. L. Thomaz. Automatic State Abstraction from Demonstration. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1243–1248, 2011.

[31] P. Cominos and N. Munro. Pid controllers: recent tuning methods and design to specification. *IEEE Proceedings - Control Theory and Applications*, 149(1):46–53, Jan 2002.

[32] S. Daftry, J. A. Bagnell, and M. Hebert. Learning Transferable Policies for Monocular Reactive MAV Control. In *Springer Proceedings in Advanced Robotics*, 2017.

[33] G. C. H. E. de Croon, M. Percin, B. D. W. Remes, R. Ruijsink, and C. De Wagter. *The Delfly: Design, aerodynamics, and artificial intelligence of a flapping wing robot*. Springer, 2016.

[34] C. De Wagter, S. Tijmons, B. D. W. Remes, and G. C. H. E. de Croon. Autonomous flight of a 20-gram Flapping Wing MAV with a 4-gram onboard stereo vision system. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4982–4987, May 2014.

[35] B. Denisow, M. Masierowska, and S. Antoń. Floral nectar production and carbohydrate composition and the structure of receptacular nectaries in the invasive plant Bunias orientalis L. (Brassicaceae). *Protoplasma*, 253(6):1489–1501, 2016.

[36] T. G. Dietterich. State Abstraction in MAXQ Hierarchical Reinforcement Learning. In *NIPS*, pages 994–1000, 1999.

[37] T. G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13(1):227–303, 2000.

[38] K. Doya. Reinforcement Learning In Continuous Time and Space. *Neural Computation*, 12:219–245, 2000.

[39] D. Fanelli. Negative results are disappearing from most disciplines and countries. *Scientometrics*, 90(3):891–904, 2012.

[40] F. Fernández, J. García, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robot. Auton. Syst.*, 58(7):866–871, July 2010.

[41] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. *Proceedings of the 5th international joint conference on Autonomous agents and multiagent systems - AAMAS*, page 720, 2006.

[42] E. Feron and E. N. Johnson. Aerial robotics. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*, pages 1009–1029. Springer Berlin Heidelberg, 2008.

[43] M. Fliess and C. Join. Model-free control. *International Journal of Control*, 86(12):2228–2252, 2013.

[44] A. F. Foka and P. E. Trahanias. Real-time hierarchical POMDPs for autonomous robot navigation. *Robotics and Autonomous Systems*, 55(7):561–571, 2007.

[45] A. F. Foka and P. E. Trahanias. Probabilistic autonomous robot navigation in dynamic environments with human motion prediction. *International Journal of Social Robotics*, 2(1):79–94, Mar 2010.

[46] P. K. Freeman and R. S. Freeland. Agricultural uavs in the us: potential, policy, and hype. *Remote Sensing Applications: Society and Environment*, 2:35–43, 2015.

[47] N. Gageik, P. Benz, and S. Montenegro. Obstacle Detection and Collision Avoidance for a UAV With Complementary Low-Cost Sensors. *IEEE Access*, 3:599–609, 2015.

[48] J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, Jan. 2015.

[49] C. Gaskett, L. Fletcher, and A. Zelinsky. Reinforcement learning for a vision based mobile robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 403 – 409, 2000.

[50] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.

[51] G. J. Gordon, N. Roy, and S. Thrun. Finding Approximate POMDP solutions Through Belief Compression. *Journal of Artificial Intelligence Research*, 23, 2005.

[52] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems, Man, and Cybernetics*, 42(6):1291–1307, 2012.

[53] M. Grzes and D. Kudenko. Theoretical and empirical analysis of reward shaping in reinforcement learning. In *International Conference on Machine Learning and Applications*, pages 337–344. IEEE, 2009.

[54] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. In *International Conference on Learning Representations*, pages 1–14, 2017.

[55] M. Hardegger, D. Roggen, S. Mazilu, and G. Tröster. ActionSLAM: Using location-related actions as landmarks in pedestrian SLAM. In *International Conference on Indoor Positioning and Indoor Navigation*, pages 13 – 15, Sydney, Australia, 2012.

[56] M. J. Hausknecht and P. Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *Association for the Advancement of Artificial Intelligence Fall Symposium*, 2015.

[57] I. Higgins, A. Pal, A. A. Rusu, L. Matthey, C. P. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. *International Conference on Machine Learning*, 2017.

[58] T. Hinzmann, T. Schneider, M. Dymczyk, A. Melzer, T. Mantel, R. Siegwart, and I. Gilitschenski. Robust map generation for fixed-wing UAVs with low-cost highly-oblique monocular cameras. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3261–3268, Oct 2016.

[59] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin. The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC). *The 23rd Digital Avionics Systems Conference (IEEE)*, 2004.

[60] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[61] C. W. Hsu and A. Liu. High-level behavior control of an e-pet with reinforcement learning. In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 29–34, Istanbul, 2010.

[62] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[63] P. A. Ioannou and J. Sun. *Robust Adaptive Control*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[64] S. Jacklin. Closing the Certification Gaps in Adaptive Flight Control Software. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, pages 1–14, Honolulu, Hawaii, 2008.

[65] E. N. Johnson and M. A. Turbe. Modeling, control, and flight testing of a small-ducted fan aircraft. *Journal of guidance, control, and dynamics*, 29(4), 2006.

[66] J. Junell, T. Mannucci, Y. Zhou, and E. van Kampen. Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, San Diego, CA, USA, 2016.

[67] J. Junell, E. van Kampen, C. C. de Visser, and Q. P. Chu. Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, Kissimmee, Florida, USA, 2015.

[68] W. Knight. *The Roomba Now Sees and Maps a Home*, September 2015. [Accessed 21-Feb-2018]. https://www.technologyreview.com/s/541326/the-roomba-now-sees-and-maps-a-home/.

[69] M. L. Koga, V. F. Silva, F. G. Cozman, and A. H. R. Costa. Speeding-up Reinforcement Learning Through Abstraction and Transfer Learning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 119–126, Richland, SC, 2013.

[70] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl. AR-Drone as a Platform for Robotic Research and Education. In *Research and Education in Robotics - EUROBOT 2011*, volume 161 of *Communications in Computer and Information Science*, pages 172–186. Springer Berlin Heidelberg, 2011.

[71] R. M. Kretchmar, T. Feil, and R. Bansal. Improved automatic discovery of subgoals for options in hierarchical. *Journal of Computer Science & Technology*, 3, 2003.

[72] V. Kumar and N. Michael. Opportunities and challenges with autonomous micro aerial vehicles. In *Robotics Research*, pages 41–58. Springer, 2017.

[73] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, Dec. 2003.

[74] A. Lazaric. *Knowledge Transfer in Reinforcement Learning*. Phd thesis, Politecnico di Milano, Milano, Italia, 2008.

[75] A. Lazaric. Transfer in reinforcement learning: A framework and a survey. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, pages 143–173. Springer, Berlin, Heidelberg, 2012.

[76] J. D. Lee. Perspectives on Automotive Automation and Autonomy. *Journal of Cognitive Engineering and Decision Making*, page 155534341772647, 2017.

[77] T. Lombaerts. *Fault Tolerant Flight Control: A Physical Model Approach*. Phd thesis, Delft University of Technology, Delft, the Netherlands, 2010.

[78] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea. A platform for aerial robotics research and demonstration: The Flying Machine Arena. *Mechatronics*, 24(1):41–54, 2014.

[79] S. Lupashin, A. Schollig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648, May 2010.

[80] T. Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles*. Phd thesis, Delft University of Technology, Delft, the Netherlands, 2017.

[81] T. Mannucci, E. van Kampen, C. C. de Visser, and Q. P. Chu. Safe Exploration Algorithms for Reinforcement Learning Controllers. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–13, 2017.

[82] A. J. McDaid, K. C. Aw, S. Q. Xie, and E. Haemmerle. Gain scheduled control of {IPMC} actuators with 'model-free' iterative feedback tuning . *Sensors and Actuators A: Physical*, 164(1–2):137 – 147, 2010.

[83] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *Computer Science Department Faculty Publication Series*, page 8, 2001.

[84] K. McGuire, G. C. H. E. de Croon, C. De Wagter, K. Tuyls, and H. J. Kappen. Efficient Optical flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone. *IEEE Robotics and Automation Letters*, 2(2), 2017.

[85] N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, 73(3):289, Jun 2008.

[86] J. Michels, A. Saxena, and A. Y. Ng. High Speed Obstacle Avoidance Using Monocular Vision and Reinforcement Learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 593–600, New York, NY, USA, 2005.

[87] G. E. Monahan. State of the Art—A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, 28(1):1–16, 1982.

[88] C. G. Morley, J. Broadley, R. Hartley, D. Herries, D. MacMorran, and I. G. McLean. The potential of using unmanned aerial vehicles (uavs) for precision pest control of possums (trichosurus vulpecula). *Rethinking Ecology*, 2:27, 2017.

[89] N. Navarro, C. Weber, and S. Wermter. Real-world reinforcement learning for autonomous humanoid robot charging in a home environment. In *Towards Autonomous Robotic Systems*, pages 231–240, Berlin, Heidelberg, 2011. Springer.

[90] R. C. Nelson. *Flight Stability and Automatic Control*. The McGraw-Hill Companies, second edition, 1998.

[91] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, San Francisco, CA, USA, 1999.

[92] M. Nieuwenhuisen, D. Droeschel, M. Beul, and S. Behnke. Autonomous Navigation for Micro Aerial Vehicles in Complex GNSS-denied Environments. *Journal of Intelligent & Robotic Systems*, 84(1):199–216, Dec 2016.

[93] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437, May 2014.

[94] K. Nonami, M. Kartidjo, K. J. Yoon, and A. Budiyono. *Autonomous Control Systems and Vehicles: Intelligent Unmanned Systems*. Springer, 2013.

[95] Office Of The Secretary Of Defense. Unmanned Aerial Vehicles Roadmap 2000-2025. April 2001.

[96] Office Of The Secretary Of Defense. Unmanned Aerial Systems (UAS) Roadmap 2005-2030. August 2005.

[97] Office of the Under Secretary of Defense for Aquisition Technology Logistics. The Role of Autonomy in DoD Systems. *DoD Defense Science Board*, July 2012.

[98] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability. In *Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia, 2017.

[99] M. Pachter and P. R. Chandler. Challenges of autonomous control. *Control Systems, IEEE*, 18(4):92–97, 1998.

[100] R. E. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. Phd thesis, University of California, Berkeley, 1998.

[101] R. E. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049, 1998.

[102] T. J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning: Cmpsci technical report 99-34. Technical report, Department of Computer Science, University of Massachusetts, Amherst, MA, 1999.

[103] C. Phillips. Knowledge Transfer in Markov Decision Processes. report, Reasoning and Learning Lab, McGill University, 14 September 2006.

[104] O. Purwin and R. DÁndrea. Performing aggressive maneuvers using iterative learning control. In *IEEE International Conference on Robotics and Automation*, pages 1731–1736, Kobe, Japan, 2009. IEEE.

[105] B. Ravindran and A. G. Barto. Relativized Options: Choosing the Right Transformation. In *Proceedings of the 20th International Conference on Machine Learning*, pages 608–615, 2003.

[106] B. Ravindran and A. G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *IJCAI*, volume 3, pages 1011–1018, 2003.

[107] S. A. Raza and W. Gueaieb. Intelligent Flight control of an Autonomous Quadrotor. In F. Casolo, editor, *Motion Control*, chapter 12, page 580. INTECH, Croatia, 2010.

[108] G. Reinhart. Assembly automation. In L. Laperrière and G. Reinhart, editors, *CIRP Encyclopedia of Production Engineering*, pages 52–54. Springer, Berlin, Heidelberg, 2014.

[109] B. D. W. Remes, D. Hensen, F. van Tienen, C. De Wagter, E. van der Horst, and G. C. H. E. de Croon. Paparazzi: How to make a swarm of Parrot AR Drones fly autonomously based on GPS. In *IMAV 2013: Proceedings of the International Micro Air Vehicle Conference and Flight Competition*, Toulouse, France, 2013.

[110] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

[111] J. Roberts and R. Walker. Flying robots to the rescue [competitions]. *IEEE Robotics & Automation Magazine*, 17(1):8–10, 2010.

[112] R. S. Russell. Non-linear F-16 Simulation using Simulink and Matlab. Technical manual version 1.0, University of Minnesota, 22 June 2003.

[113] S. R. B. Santos, S. N. J. Givigi, and C. L. N. Júnior. An Experimental Validation of Reinforcement Learning Applied to the Position Control of UAVs. In *IEEE International Conference on Systems, Man, and Cybernetics(SMC)*, pages 2796–2802, 2012.

[114] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[115] C. J. E. Schulp, S. Lautenbach, and P. H. Verburg. Quantifying and mapping ecosystem services: Demand and supply of pollination in the European Union. *Ecological Indicators*, 36:131 – 141, 2014.

[116] A. A. Sherstov and P. Stone. Improving Action Selection in MDP's via Knowledge Transfer. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI, pages 1024–1029. AAAI Press, 2005.

[117] S. Sieberling, Q. P. Chu, and J. A. Mulder. Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction. *Journal of Guidance, Control and Dynamics*, 33(6), 2010.

[118] Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine learning*, page 95, 2004.

[119] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, pages 361–368, 1995.

[120] E. J. J. Smeur, Q. P. Chu, and G. C. H. E. de Croon. Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles. *Journal of Guidance, Control and Dynamics*, 39(3), 2016.

[121] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. *IEEE International Conference on Intelligent Robots and Systems*, 2017-September:4241–4247, 2017.

[122] V. Soni and S. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the 21st National Conference on Artificial Intelligence*, AAAI, pages 494–499. AAAI Press, 2006.

[123] B. L. Stevens and F. L. Lewis. *Aircraft control and simulation*. Wiley Inter-science, New York, 1992.

[124] M. Stolle and D. Precup. Learning Options in Reinforcement Learning. In S. Koenig and R. Holte, editors, *Abstraction, Reformation, and Approximation (SARA)*. Springer, Berlin, Heidelberg, 2002.

[125] P. Stone, R. S. Sutton, and G. Kuhlmann. Scaling reinforcement learning toward robocup soccer. In *Journal of Machine Learning Research*, 2003.

[126] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[127] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. PAC Model-free Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 881–888, New York, NY, USA, 2006. ACM.

[128] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[129] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181 – 211, 1999.

[130] Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[131] N. Taghizadeh and H. Beigy. A novel graphical approach to automatic abstraction in reinforcement learning. *Robotics and Autonomous Systems*, 61(8):821 – 835, 2013.

[132] A. Taylor, I. Dusparic, and E. Galván-López. Transfer learning in multi-agent systems through parallel transfer. In *Proceedings of the 30 th International Conference on Machine Learning*, volume 28, Atlanta, Georgia, USA, 2013. JMLR: W&CP.

[133] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, Dec. 2009.

[134] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 37. ACM, 2007.

[135] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observable Markov Decision Process Models for Robot Navigation. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 511–516, Seoul, Korea, 2001.

[136] S. Tijmons. *Autonomous Flight of Flapping Wing Micro Air Vehicles*. Phd thesis, Delft University of Technology, Delft, the Netherlands, 2017.

[137] Tonyle. *Creative Commons image: YUV UV plane* CC by 3.0 (modified), 2009. `https://en.wikipedia.org/wiki/YUV`.

[138] N. Topin, N. Haltmeyer, S. Squire, J. Winder, M. DesJardins, and J. MacGlashan. Portable option discovery for automated learning transfer in object-oriented markov decision processes. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI, pages 3856–3864. AAAI Press, 2015.

[139] E. van Kampen, Q. P. Chu, and J. A. Mulder. Continuous adaptive critic flight control aided with approximated plant dynamics. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, USA, August 2006.

[140] F. Wang, J. Q. Cui, B. M. Chen, and T. H. Lee. A Comprehensive UAV Indoor Navigation System Based on Vision Optical Flow and Laser FastSLAM. *Acta Automatica Sinica*, 39(11):1889 – 1899, 2013.

[141] J. Wang, F. Holzapfel, and F. Peter. Comparison of Nonlinear Dynamic Inversion and Backstepping Controls with Application to a Quadrotor. In *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control*, pages 1245–1263, Delft, The Netherlands, April 2013.

[142] X. Wang and T. G. Dietterich. Model-based policy gradient reinforcement learning. In *International Conference on Machine Learning*, pages 776–783, Washington,DC, 2003.

[143] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin. Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning, 2005.

[144] S. D. Whitehead and L.-J. Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1):271 – 306, 1995. Computational Research on Interaction and Agency, Part 2.

[145] C. Williams. Summon the bee bots: can flying robots save our crops? *New Scientist*, 220(2943):42–45, 2013.

[146] Y. Zhou. *Online Reinforcement Learning Control for Aerospace Systems*. Phd thesis, Delft University of Technology, Delft, the Netherlands, 2018.

[147] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Trans. ASME*, 64(8):759–768, 1942.

NOMENCLATURE

LIST OF SYMBOLS

$\mathcal{M}$	Markov decision Process
t	discrete time step or iteration
$s,\ s_t$	state at t
$s',\ s_{t+1}$	state at $t+1$
$\mathcal{S}$	state set
$a,\ a_t$	action at t
$\mathcal{A}$	action set
o_t	option at t
$\mathcal{O}$	option set
r_t	reward at t
$k(s,a)$	number of of updates at (s,a)
α	learning rate, step-size
γ	discount factor
π	policy
ϵ	probability of greedy action in ϵ-greedy policy (where $\epsilon = 0$ is random and $\epsilon = 1$ is fully greedy)
V	Value function
$V^\pi(s)$	Value of state s under policy π
Q	Value function for state-action paired Q-learning
$Q^\pi(s,a)$	Value of state s taking action a under policy π
λ	decay-rate parameter for eligibility traces
δ	temporal-difference error

ABBREVIATIONS

GNC	Guidance, Navigation, and Control
MAV	micro aerial vehicle
UAV, UAS	unmanned aerial vehicle/system
RL	reinforcement learning
HRL	hierarchical reinforcement learning
MDP	Markov decision process
sMDP	semi-Markov decision process
TDRL	temporal difference reinforcement learning
MAE	mean absolute error
POI	point(s) of interest
PID	proportional–integral–derivative (controller)

ACKNOWLEDGMENTS

The Dutch construct of "gezelligheid" is one of the first concepts every expat should learn upon arrival in the Netherlands. The word itself is often touted as untranslatable and, indeed, it takes some time to learn the correct usage. After over 5 years living in the Netherlands, I would venture to claim that one of the core ingredients of gezelligheid is *appreciation*. Everything which is designated gezellig has some quality which is being appreciated by the observer; and just as the famous Dutch word is hard to translate, so is appreciation difficult to convey in just a few short paragraphs – but I will try.

This is for everyone who made my work and life so very gezellig.

I would first like to extend my deepest gratitude to my promotor, Max Mulder. He has had an immeasurably positive impact on my thesis work, and on my overall experience in the Netherlands. His talent for very quickly getting to the heart of the issue, asking the most relevant questions, stimulating helpful discussions, and inspiring motivation has been invaluable. This thesis would not be possible without his unwavering support.

To my copromotor, Dr. Chu, I thank you for your technical expertise, constant cheerfulness, and wise career advice. Thank you to Bob Mulder for being the captain on our "Heidag" sailing boat as well as the captain, so-to-speak, of my defense while serving as the chairman of the committee. Thank you to Guido for your help and practical advice. To my early-days supervisor, Coen, thank you for being there to help me find my feet in the first year of the study. To my daily supervisor, Erik-Jan, thank you for your insights and critical assessments to help shape me into a reinforcement learning researcher.

Research doesn't happen in a bubble. In addition to my supervisory team, several others have helped me along the way when I was stuck or needed guidance. Anyone who wants to work with quadrotors will invariably get help from some of the capable people in the MAVlab. Thank you to Guido, Erik, Christophe, Ewoud, Freek, Elizabeth, Kirk, and the rest for guiding me through the world of Paparazzi, Git, and OptiTrack – even when the solution was literally spraying water on astroturf. To my fellow reinforcement learning buddy, Tommaso, thank you for collaborating with me on projects, review my papers, going together to conferences, and being patient as I learned the correct spelling of you name.

Thank you to those who kept my plants alive.

The greatest part of working at the TU Delft was getting to interact with my fun and impressive colleagues. We sat together every day, had coffee together, mingled at numerous Vrijmibo's and social events, and occasionally played basketball, went sailing for "heidag", and sang karaoke together at the annual BBQ. They are the kind of people you can entrust to code part of project, and also to water your plants while you're on vacation. I can't thank enough the kind PhD students and staff at Control & Simulation.

The first months are the hardest in a new place. During my start at TU Delft, I received patient guidance from my peers in order to navigate the world of dutch bank accounts, bicycle transport, OV-chipcards, DUWO housing, and learning the local language. My sincerest thanks to all my early-years colleagues who helped me feel at home in a new environment: to my two Belgian desk neighbors, Jan and Laurens, for the "woord van de dag"; to Rita for breaking the awkward silences at coffee corners; to Maarten for teaching me all the flight practical info with the "firehose" method; to Deniz for the chance to get sick in a helicopter simulation in Simona; to Yazdi for the beautiful art around the office; to Liguo and Miriam for my invaluable paranymph experiences; to Jan S. for always being up for a drink and a chat; to Rolf for an awesomely themed beachside party; to João for your sense of humor and inspiring discipline; to Dyah for riding roller coasters with me at sunny conference locals; and to Herman, Bruno, Jia, Joost, Gustavo, Peng, Sjoerd, and Hann Woei for all coming before me to help pave the way. You truly taught me the ways of the PhD student.

Thank you, for all these reasons and for the others too numerous to count.

To the PhD students who arrived after me, thank you for bringing new and fresh ideas into the office environment. As the C&S department has been steadily growing, there are many more in this category. Thank you to Tommaso, Sophie, and Tao, for all the joy that singing brings; to Kimberly for radiating your awesome presence at my back in our seating arrangement; to Ewoud for constantly impressing me with your sleeping skills (which are the first I've seen to outmatch my own); to Lodewijk for great knee injury advice; to Isabel for being the perfect house guest; to Jerom for playing a mean ping pong game; to Jelmer for being a great desk neighbor; to Henry for teaching us Volendam tradition; to Ye Zhang for your cheerfulness and style; to Ivan for sharing with me your history and point of view; to Kasper for helping me with the Dutch translation of my summary and being graduation buddies; to Diana for your fun and lively spirit and for the mixed drinks; to Matěj (our token Post doc) for being a giving friend and fearless ski leader; and to Annemarie, Yingzhi, Ye Zhou, Junzi, Wei, Sherry, Julia, Dirk, Sarah, Mario, Neno, Daniel, Paolo, Ezgi, Malik, Anne, Fede, Emmanuel, Sihao, Shuo, Tom; and all the staff: Daan, Marilena, Olaf, Joost, Jacco, Herman, Clark, Guido, Bart, Christophe, Alwin, René, Harold, and Andries, for bringing your irreplaceable contributions to each coffee corner, lunch get-together, BBQs, social events, and drinks at the atmosfeer. From the staff, a special thanks to Daan for being my "custom-made gifts" mentor, for organizing the N-team basketball tournaments, and for having such a contagious laugh.

One of the greatest opportunities I got as a PhD student in the C&S department was to be a coordinator for the flight practical. Every flight was a thrill and I'm constantly in awe as to how lucky I am to have been in that position. Thanks to Hans Mulder and Xander van 't Veld for this rare opportunity, and to Menno and Fred for your vast knowledge on aircraft and for cheerfully vouching for me at the security gate.

Next, I happily express my appreciation for the most important person in the C&S department, Bertine Markus. Without her making my life easier in practical matters, I would surely have made some small bureaucratic error resulting in my deportation.

Finally, I would like to thank Kirk for being a singularly supportive friend, inside and outside the office.

Gratias, Gratias agimus tibi, propter magnam gloriam tuam.

If you are inwardly singing the above latin text to a particular tune, then you most likely sang 'Gloria' with me. During my time in Delft, music was a constant companion. I've never felt so much love and appreciation for music as when I've been singing with Krashna Musika, the university classical music association. The joy that I feel is not just from the music, but also from the people who have shared their gift with me. I'd like to thank my conductor of 5 years, Ruben, and the several other conductors I've been lucky enough to have as a leader. Thank you to all the board members and committee members who gave their time and energy into making each concert, tour, and social gathering a fun and growing experience. I would especially like to thank my fellow chamber choir committee (kakocie) – Joris, Yuri, Nathan, and Daan – for working with me and helping me learn another aspect of Dutch student life. Thank you to Sophie and Tijmen for providing piano accompaniment for my occasional solo aspirations. From my brief, but fun, time in Delft Blue kamerkoor, a special thanks to Els and Loes for being so welcoming. Lastly, a fortissimo major chord thanks to my Krashna friends whom I shared much more than just music with. To Dena, Alexey, Ranko, Arturo, Jay, Tommaso, Susana, Encar and the rest: Let's never stop sharing music, travels, and laughter.

Thank you for making me feel at home in a foreign land.

Since moving to Delft I have been so fortunate to meet wonderful people from all around the world who are not only fantastic friends, but were also kind enough to host me in their home country. I've become spoiled with their kindness and hospitality. To see a place as it is lived in by its residents is something special that not every tourist gets to experience. It has quickly become my favorite way to travel. I've visited my friends at their homes in Belgium, Sweden, England, Germany, Russia, Portugal mainland and the Azores, Denmark, Czechia, Brazil, and of course the Netherlands. Thank you to: Mariana, Deborah, Naomi, Matěj, Riccardo, Isa, Mark, Franzi, Alexey, Felipe, Inês, Pedro, Miguel G., Miguel D., Francisco, the Furgals, Alex T., Katrina, Henry, Sophie, and Encar. I will continue to try my best to deserve you all as friends. I also want to especially thank Deborah for not just hosting me, but actually sharing a home with me as a roommate, and for always being there for me as a friend. You have influenced me in ways I didn't expect, and because of you I've seen positive changes in myself.

Far away in distance does not mean far away in heart and mind.

Lastly, let's jump across the pond where I thank my family for there continuous support and understanding. An additional thanks to my talented cousin, Jessica, for the extremely cool cover art. To grandma Grace, thank you for being an example of what a beautiful full life can be. To my family, I hope I've been able to show you that far away in distance does not mean far away in heart and mind.

To everyone mentioned above – and to those written fondly in memories if not on the page – Thank you for the gezelligheid!

Delft, November 2018

CURRICULUM VITÆ

Jaime Lin JUNELL

15-06-1986 Born in Clackamas, Oregon, United States of America

EDUCATION

2013–2018 PhD. Aerospace Engineering
Technische Universiteit Delft, Delft, the Netherlands
Thesis: An Empirical Approach to Reinforcement Learning for Micro Aerial Vehicles

2008–2009 Masters of Science, Mechanical Engineering
Oregon State University, Corvallis, OR, USA
Thesis: Adaptive Methods for Robust Commercial Vehicle Control

2004–2008 Honors Bachelors of Science, Mechanical Engineering
Oregon State University, Corvallis, OR, USA
Thesis: Optimization of Sensor/Neuro-Controller Pairings for Effective Navigation

EXPERIENCE

2010–2013 Software Engineer
Lockheed Martin Space Systems Company, Sunnyvale, CA, USA

2007 Intern
Chevron Energy Technology Co., Houston, TX, USA

2006 Intern
European Gravitational Wave Observatory, Virgo group, Rome, Italy

LIST OF PUBLICATIONS

7. J. Junell, T. Mannucci, E. van Kampen, M. Mulder. State abstraction in Hierarchical Reinforcement Learning. *Engineering Applications for Artificial Intelligence*, (to be submitted).

6. J. Junell, T. Mannucci, E. van Kampen. Self-tuning gains of a quadrotor: A reinforcement learning approach. *Journal of Guidance, Control, and Dynamics*, (submitted).

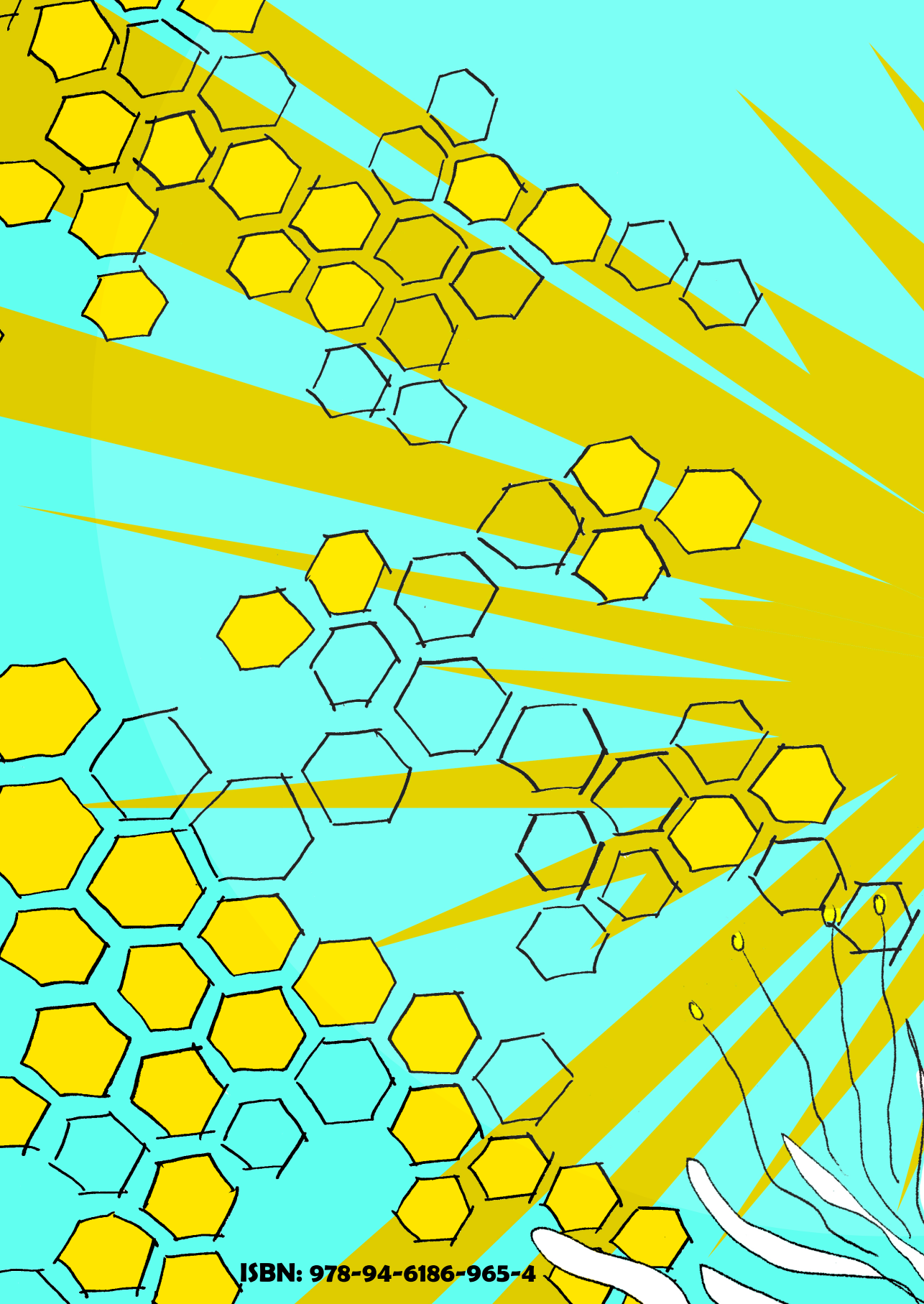
5. M. Siddiquee, J. Junell, E. van Kampen. Flight test of Quadcopter Guidance with Vision-Based Reinforcement Learning. In *AIAA SciTech Forum: Guidance, Navigation and Control Conference*, San Diego, CA, USA, 2019.

4. G. Schonebaum, J. Junell, E. van Kampen, Human demonstrations for fast and safe exploration in reinforcement learning. In *AIAA SciTech forum: Information Systems–Infotech at Aerospace*, Grapevine, TX, USA, 2017.

3. J. Junell, E. van Kampen. Adaptive path planning for a vision-based quadrotor in an obstacle field. In *International Micro Air Vehicle Competition and Conference*, Beijing, PR of China, 2016.

2. J. Junell, T. Mannucci, Y. Zhou, and E. van Kampen. Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, San Diego, CA, USA, 2016.

1. J. Junell, E. van Kampen, C. C. de Visser, and Q. P. Chu. Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, Kissimmee, Florida, USA, 2015.



ISBN: 978-94-6186-965-4

Propositions

accompanying the dissertation

AN EMPIRICAL APPROACH TO REINFORCEMENT LEARNING FOR MICRO AERIAL VEHICLES

by

Jaime Lin JUNELL

1. Including a priori information with reinforcement learning (RL) is a double-edged sword: It can aid in faster learning but undermines the benefits of RL as a *tabula rasa* learning method, and can limit the freedom of the algorithm to find its own (possibly more optimal) solution. [this thesis]
2. The combination of state and temporal abstraction together outperforms either one on their own for an RL task in an obstacle-rich maze. [this thesis]
3. State abstraction, which fights the curse of dimensionality, comes at the cost of added state ambiguity therefore leading to suboptimal solutions. [this thesis]
4. Gradient-based policy improvement reinforcement learning can be used for fine-tuning gains when starting from a reasonably good starting point, but care should be taken to ensure the task is not prone to sharp transitions into unstable regions within the policy-space. [this thesis]
5. A good reward structure is minimalist.
6. In life, as in machine learning, optimal solutions are not learned through fully greedy policies.
7. Hiding negative results within "positive spin" creates self-inflicted and unnecessary inefficiency in the research community. [Fanelli, 2012]
8. Targeted financial incentives to recruit females into engineering positions should come only from a desire to have more of the female mindset in the workplace – not from a stance of trying to even the playing field or making reparations.
9. Critics of foreign cultures often lack the resources needed to see through another cultural lens. This is true for any hot topics including Zwarte Piet in the Netherlands, or gun control in the US.
10. If the goal of automation was to provide for human's basic needs in order to reduce stress and free up time, we have succeeded as engineers and failed as a society. Therefore, the best way to honor the accomplishments and labors of previous generations is to go on vacation more often.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors Prof. dr. ir. M. Mulder and Dr. Q.P. Chu.