Final thesis report

Vision-Based Reinforcement Learning for the guidance of an AR Drone 2

M. Siddiquee

Supervised by:
Dr. ir. Erik-Jan van Kampen
Jaime Junell, MSc.

17th April 2018

Picture source: https://cdn3.img.sputniknews.com/images/102227/55/1022275589.jpg

TU Delft

Acknowledgements

I would like to thank my family for their guidance, patience and support without which I would not be me. Special thanks to my mother for convincing me to study for a master's degree and believing in me.

For bearing with my tardiness, making sure I am in the right direction and for teaching me everything I have learned about reinforcement learning and the creation of knowledge in general, I want to express my eternal gratitude to my supervisors, Erik-Jan van Kampen and Jaime Junell. Besides my supervisors, the other person who significantly contributed (and hopefully will keep on contributing) to my understanding of reinforcement learning and love of knowledge in general is my close friend Imrul Kayesh Ashraf.

From the MAVLab, I would like to express my deepest gratitude towards Erik van der Horst. He helped realize the goals of this project by making a tethered power supply for the AR Drone 2. His approach to solving problems inspired me and is something I hope to incorporate in myself. Furthermore, I want to thank Christophe de Wagter for his work on the computer vision module and Ewoud Smeur for his work on the INDI controller of PaparazziUAV.

I am also very grateful for the work carried out by the late Pascal Brisset for the open source autopilot project PaparazziUAV, Guido van Rossum for giving humanity Python and Linus Torvalds for giving us Linux. This project would not have been possible without the vision and sacrifice of these pioneers.

Abstract

This report presents the work carried out for a research into the application of Reinforcement Learning (RL) and Computer Vision (CV) for the autonomous flight of Unmanned Aerial Vehicles (UAVs). The goal of the thesis is to "**design a reinforcement learning based guidance controller for an AR Drone 2 that uses vision sensed features for state estimation and reward generation**".

There is a growing demand for the autonomous functioning of UAVs. UAVs that can autonomously find its own way can be useful in the case of surveillance, inspection of infrastructure, or in search and rescue operations. However, current autonomous systems rely on an Integrated Localization System ,such as the Global Positioning Sysytem (GPS), for guidance and navigation. This can limit the UAV's autonomy by reducing its capacity to travel in GPS denied environments such as indoors or in tunnels.

This project contributes to the autonomy of UAV's by implementing an RL based guidance controller for a quadcopter that does not need an Integrated Localization System for guidance and navigation. Instead, visual information is encoded to define a state which is a "relative description" of the quadcopters's position. The vision-based relative state information is used by the RL agent to make guidance and navigation decisions.

The developed vision-based RL controller is implemented in a Parrot AR Drone 2 and tested in a robot testing facility of the Aerospace Faculty of TU Delft. First, a preliminary study into the general characteristics of vision-based relative description of th state is performed in a gridworld simulation. Consequently, concepts for the vision-based state, the guidance task and the RL agent are created and a learning scheme selected. Following this, the learning of the guidance by the designed agent is simulated using PaparazziUAV and FlightGear. During implementations the guidance task is simplified and the RL agent is redesigned. The defined tasks consist of travelling to a red goal in a room (one goal task), travelling to a red goal and a blue goal in a room (two goal task) and turning past a corridor to travel to a red goal (corridor task). After redesigning and simulating the agent, it is implemented in an AR Drone 2 and flight tests performed.

The gridworld study shows that the agent is unable to learn if there is insufficient information in the relative state description. Attempting to learn a fully greedy policy causes the learning to diverge. The simulation of the task in PaparaziiUAV and FlightGear shows that the agent can learn the task, and the learning can be made faster by incorporating expert knowledge. An action that lasts over multiple steps, called an *option* is implemented to overcome the problems associated with ambiguity in the state description, and the agent is encouraged to turn when not seeing a goal through the reward function. The ability of the agent to learn other, similar tasks, is demonstrated by teaching the corridor task to the agent for the two goal task. The performance of the RL agent is compared to a rule based autopilot and an absolute position based autopilot. The trained agent performs nearly as good as the rule based autopilot, but it takes over twice the amount of time to perform the task than the absolute position based autopilot.

After simulations, the agent is implemented in an AR Drone 2 and trained to perform the one goal task. The real life agent learns to perform the task, albeit with greater variance in the performance than the simulated agent. This is believed to be a result of the real life noise that is unmodeled in the simulations. The real life agent has a delay in the perception of the vision states. This delay causes an offset in the performance between the simulated agent, and the real life agent when Q-values learned from simulation are implemented in the AR Drone 2.

The work in this project shows that it is possible to train simple guidance and navigation tasks to quadcopters using only vision information and RL.

List of Tables

List of Figures

List of abbreviations and symbols

Table 1: Abbreviations

Abbr	Description
AI	Artificial Intelligence
CPU	Central Processing Unit
CV	Computer Vision
DP	Dynamic programming
GNC	Guidance Navigation & Control
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LSPI	Least-Square Policy Iteration
MAV	Micro Aerial Vehicle
MC	Monte Carlo
MDP	Markov Decision Process
PID	Proportional, Integral and Derivative Controller
POMDP	Partially Observable MDP
PTAM	Parallel Tracking and Mapping
RL	Reinforcement Learning
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded-Up Robust Features
TD	Temporal Difference
UAV	Unmanned Aerial Vehicle

Table 2: Symbols

Symbols	Description
π	Policy
π^*	Optimum policy
s	State
a	Action
r	Reward
γ	Discount factor
α	Learning rate
R	Return
ϵ	The probability of acting greedily in an ϵ-greedy policy

Table 3: Functions

Functions	Description
$\pi(s,a)$	Probability of selecting action a in state s
$V(s)$	Value function; reward for each state
$V^\pi(s)$	Value function for policy π
$V^*(s)$	Optimum values
$Q(s,a)$	Action-value function; reward value for each state action pair
$Q^\pi(s,a)$	Action-value function for policy π
$Q^*(s,a)$	Optimum action-values

Table continued

Functions	Description
$\mathcal{R}^a_{ss'}$	Reward model; reward for taking action a in state s and transitioning to s'
$\mathcal{P}^a_{ss'}$	Transition model; the probability of ending up in s' given the agent took action a in state s
$\mathcal{A}(s)$	Available actions in state s

Contents

1 Introduction

This document presents the design and testing of a vision-based Reinforcement Learning (RL) agent for the guidance of a quadcopter. The motivation for the design of such a system, the goals of the overall project, the scientific context of the project, and finally the structure of this report are presented in this chapter.

1.1. Motivation

Recent developments in electronics and software have led to UAVs becoming cheaper(Kendoul, 2012). Although the early drive for UAVs was from the military, these developments have led to the emergence of a consumer market for UAVs for either recreational or professional uses. Hobbyists are buying them for the pleasure of flying them or using them for photography. Businesses, research institutions and governments are buying them for mapping, monitoring and logistics. The widespread demand has been fueling a push to UAV research in order to improve them and enable them to perform even more tasks (Valavanis, 2008; Kendoul, 2012).

One of the points of attention for the development of UAVs is their autonomy (Salichs & Moreno, 2000; Kendoul, 2012). Not only does a more autonomous UAV provide significant advantages over a human controlled one, it opens up venues of applications which are inaccessible to them due to current limitations in the technology. For instance the ability to autonomously adapt the internal controller to various circumstances may lead to UAVs which can work autonomously under all weather conditions, adapt the control action to changing aerodynamic model with ageing or adapt its control action to mitigate some damage to itself. A UAV capable of autonomous navigation and survey may regularly launch itself, monitor an area, perform detailed surveillance on areas of interest and alert human operators in case it detects any problems. This will decrease the costs of manually monitoring and surveilling an area. Such systems can be used by security companies, maintenance companies or natural preserves.

Driven by these needs, a study is performed to improve the autonomy of UAVs. The focus of this work is on the autonomous flight of UAVs in GPS-denied environments. The reason for this is the limitations of current methods of autonomous flight for UAVs in GPS-denied environments and the potential gains in UAV autonomy as a result of such a system. A lot of the work towards solving this challenge uses vision and laser sensors to simultaneously locate the UAV and map its environment using Simultaneous Localization and Mapping (SLAM) (Bachrach, Prentice, He, & Roy, 2011; Grzonka, Grisetti, & Burgard, 2012; Tomic et al., 2012). This work aims to explore other tools from the field of Artificial Intelligence (AI) to solve this problem.

Namely, the goal of this work is to use RL and Computer Vision (CV) to teach a UAV to perform a guidance task without GPS information. RL has been used by the AI and the Control Engineering community to approach numerous problems (Tesauro, 1995; Abbeel, Coates, Quigley, & Ng, 2007; Mnih et al., 2015). In comparison to other AI methods, RL has the distinct advantage of being able to learn without supervision (Sutton & Barto, 1998). This feature is useful for autonomy as it provides a level of autonomous behavior unattainable by supervised learning. However, there are numerous complications and challenges with the applications of RL which must be kept in consideration. For example, dealing with partial observability, the exponentially increasing search space with increasing dimensions of the state-action space and the trade-off between exploring the environment against exploiting learned knowledge leads to complications in robotics applications (Kober, Bagnell, & Peters, 2013).

Over the past few decades numerous innovations in CV (Zhang, 2000; Lowe, 2004; Viola & Jones, 2004; Davison, Reid, Molton, & Stasse, 2007) have made it one of the most information rich sensor for the control of robots. The enabling technologies of CV and the potential benefits of using vision has encouraged numerous studies into the application of CV for UAV guidance and navigation(Bonin-Font, Ortiz, & Oliver, 2008). Due to the capacities of CV and the intuition that vision is important for guidance and navigation, it is selected as the alternative to GPS for this study.

1.2. Research Goals

The research goal for the thesis is to **"design of reinforcement learning guidance controller for an AR Drone 2 that uses vision sensed features for state estimation and reward generation"**. In order to accomplish this goal, the following research questions are defined:

RQ1 What is the current state of the art for RL in UAVs and vision-based RL for robotics applications?

RQ2 What will be the task and the architecture the controller simulated and tested in this study?

- **RQ2.1** What are some options for vision-based states that can be used by RL for learning a guidance and navigation task?
- **RQ2.2** How should the guidance and navigation task be framed to make it achievable by a vision-based RL quadcopter?
- **RQ2.3** What type of learning scheme are suitable for this study?

RQ3 Which learning schemes and vision-based states are simulatable and testable within the resources of this project?

RQ4 What is the performance of the designed controller?

- **RQ4.1** How does the controller perform in terms of (a) rate of learning, (b) time taken to perform the task?
- **RQ4.2** How does the developed algorithm compare to other guidance methods?
- **RQ4.3** Can the developed agent be used to learn other guidance tasks?
- **RQ4.4** How well does the information learned in simulation transfer to real-life?

1.3. Scientific Context

The work carried out in this project is related to two fields of knowledge; namely, Aerospace Engineering and Artificial Intelligence. Specifically it focuses on the control task of UAVs which is a sub-topic of Aerospace Engineering. The sub-categories of Artificial Intelligence pertinent to this work is reinforcement learning and computer vision.

Since it is an applied research it makes no theoretical contributions to the aforementioned fields. It's goal is to combine the knowledge present in these fields and apply them to the improvement of the autonomous flight of UAVs. The intended contribution is the design of a vision-based RL guidance controller for a quadcopter.

1.4. Report Structure

The next chapter (Chapter 2) presents a stand alone article which provides a brief introduction to the ideas used in this project and documents its most interesting results. The RL agent and the validation of the simulation as described in Chapter 2 provide answers to **RQ3** ,**RQ4.1** and **RQ4.4** (and partly to **RQ4.2**)..

Backgrounds on the concepts and theories used in this project are described in Chapter 3. Namely, it discusses the autonomy of UAVs, explains the working principle of RL, and describes some computer vision methods relevant for vision-based state design. The discussion about the state of the art in RL and CV in Chapter 3 addresses **RQ1**. In Chapter 4, a gridworld simulation is carried out to obtain ideas and insights about the design of a vision-based state. Concepts for the state description, the guidance task and the initial design of the RL agent are presented in Chapter 5. The concepts for the state, task and agent answers **RQ2**.

Experimental results answering some of the remaining research questions are described in Chapter 6. The RL controller is compared to an absolute position based controller to complete the answer to **RQ4.2**. **RQ4.3** is answered by using the designed RL agent to learn a different navigation and guidance task. Following Chapter 6, conclusions from this project and recommendations about future work are presented in Chapter 7.

Readers with prior knowledge of RL and CV are recommended to start by reading the article in Chapter 2 and than move onto Chapter 6. These two chapters contain all the results from the vision-based RL agent that has been developed in this project.

2

Article

Flight test of Quadcopter Guidance with Vision-Based Reinforcement Learning

Manan Siddiquee*Faculty of Aerospace Engneering, TU Delft

Reinforcement Learning (RL) has been applied to teach quadcopters guidance tasks. Most applications rely on position information from an absolute reference system such as Global Positioning System (GPS). The dependence on "absolute position" information is a general limitation in the autonomous flight of Unmanned Aerial Vehicles (UAVs). Environments that have weak or no GPS signals are difficult to traverse for them. Instead of using absolute position, it is possible to sense the environment and the information contained within it in order to come up with a "relative" description of the UAV's position. This paper presents the design of a RL agent with relative vision-based states and rewards for the teaching of a guidance task to a quadcopter. The agent is taught the task of turning towards a red marker and approaching it in simulation and in flight tests. A more complex task of travelling between a blue and a red marker is trained in simulation. This work shows that relative vision-based states and rewards can be used with RL to teach quadcopters simple guidance tasks in simulations and in real flights. The performance of the trained agent is inconsistent in simulation and flight test due to the inherent partial observability in the relative description of the state.

Nomenclature

IMU	Inertial Measurement Unit	UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System	MAV	Micro Aerial Vehicle
UAS	Unmanned Aerial System	RL	Reinforcement Learning
UDP	User Datagram Protocol	MDP	Markov Decision Process
HTTP	Hypertext Transfer Protocol	INDI	Incremental Non-linear Dynamic Inversion
RGB	Red Green Blue	YUV	

I. Introduction

APPLICATIONS such as urban search and rescue, surveillance and infrastructure monitoring require Unmanned Aerial Systems (UAS) which can safely and autonomously fly in unknown environments without position information from the Global Positioning System (GPS). For these types of applications the Unmanned Aerial Vehicle (UAV) must perceive the environment to get an idea of where it is, identify where it needs to go and guide itself to its goal. Perception uses vision, distance sensors and other on-board sensors to come up with a relative description of the position and attitude of the UAV. These approaches are either computationally expensive or they require heavy hardware (for example laser distance sensors or stereo-vision setups) which can be challenging for Micro Aerial Vehicles (MAVs) with limited payloads. Furthermore, if the UAV is limited to using pre-programmed flight plans and routines for autonomous behavior, it cannot adapt to changing environments or mission requirements without human supervision. The limitations of current methods and lack of methods which enable higher levels of autonomy constrain their use in the aforementioned applications.

Presently, there are three main approaches to guidance in GPS denied environments. The earliest and simplest approach is dead reckoning. Estimates for the acceleration is obtained from the IMU, but it can also be obtained using novel methods such as visual odometry [1]. Using dead reckoning alone is problematic for UAVs due to sensor noise and drift. Therefore, it is often fused with other localizing systems in order to improve their

*MSc. Student, Department of Control & Simulation, Faculty of Aerospace Engneering, TU Delft

accuracy [2–5]. Another approach is replacing GPS with some other local integrated positioning system [5,6]. The problem with this method is the requirement of an external system. The final approach is using a laser rangefinder or a camera to create a map of the environment and localize the UAV within the generated map. This approach is called Simultaneous Localization and Mapping (SLAM) [7]. The drawback of this method is its computational cost which makes real-time applications challenging. Research into this method has yielded numerous guidance and navigation systems which use some form of speeded up SLAM, enabling GPS free navigation [8–10]. An example of a map generated using SLAM is shown in Figure 1.

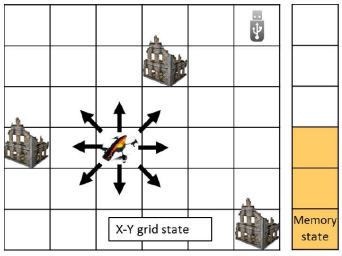
Figure 1: Maps of the first floor of MIT's Stata Center as presented in [8]. The left image shows the map created by the autonomously flying UAV developed in [8] using laser range finders. The right image shows the architectural floor plan.

Vision is often a key element in many of these non-GPS localizing methods. The volume of information in vision and the lightweight nature of most cameras make it a desirable source of information required for guidance and navigation [5, 9, 11–13]. Bipin, Duggal and Krishna [14] used supervised learning to train depth estimates from visual data which is later used for trajectory planning. Purely vision-based SLAM is not very common. Weiss et al. [9] and Shen et al. [15] use purely vision-based SLAM for the autonomous navigation of MAVs. Research efforts to incorporate vision into the autonomy of UAVs has focused on specific tasks [9] such as landing [16, 17], hovering [18], corridor following and obstacle avoidance [19]. Some of these methods calculate optic flow [20] to estimate the attitude of the UAV and generate control actions. The accessibility of visual data and the relatively smaller amount of work carried out on the guidance of UAVs using vision-based reinforcement learning (RL) motivates its use in this study.

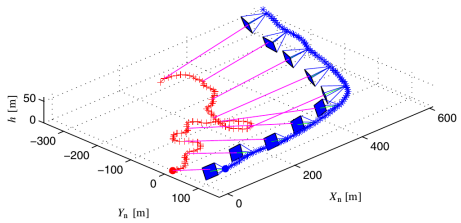
The core characteristic of autonomous systems is their ability to perform tasks with limited or no human interaction. Many of the work on the autonomous flight of UAVs in the previous decade used hard-coded software that has little or no learning capacity [13]; this limits their autonomy. In this regard, techniques from the field of Artificial Intelligence (AI) and robotics have been applied in UAS to incorporate learning and adaptation into its flight. Out of the machine learning methods available, RL promises the distinct advantage of being able to learn without supervision [21].

The increase in sensing and acting potential of UAVs brought on by improved electronics has encouraged the application of RL in its guidance and control. Abbeel, et al. [22] demonstrated autonomous aerobatic maneuvers by a helicopter using inverse RL. It has been used to train optimal shapes for a morphing UAV at different flight conditions by Valasek et al. [23]. Valasek et al. [24] has also used RL to teach a simulated fixed wing UAV to track a target for surveillance using vision feedback; the path taken by one of their trained controller is presented in Figure 2b. The performance of a non-linear autopilot is compared to a controller learned using RL in [25]. The study found that the designed non-linear controller slightly outperformed the RL taught controller. It has been used by Sharma [26] to train a UAV autopilot using a method called Fuzzy Q-learning and by Junell et al. [27] to tune the gains of a quadcopter using policy gradient RL. Further, it has been used for exploring an unknown environment and find paths to defined goals in [28] (see Figure 2a). The ideas of RL have been used in [29] and [30] to come up with methods that enable UAV guidance and navigation in unknown environments with obstacles in the former study and cooperatively plan paths to avoid threats in the latter study.

A broader use of RL in UAVs is restricted due to several challenges with its implementation. Issues such as dealing with partial observability, the intractable state-action space for complex tasks and the trade-off between exploration and exploitation lead to problems in robotics applications [31]. Specifically for UAVs, RL implementations that aim to carry out on-line learning needs to ensure safe exploration [32] and fast convergence. This factor discourages learning in flight tests with such systems.



(a) Representation of the guidance problem that was solved using RL in [28]. The agent is positioned in a 6x6 gridworld with the goal of finding an optimum route to photograph the ruins all the while accounting for available memory.



(b) Simulation results from [24]. Here an RL agent controlling a fixed wing UAV (blue) learns to give the right bank angle commands in order to keep a tracked target (red) in its view. This plot shows the trained agent tracking an erratically moving target.

Figure 2: Application of RL to UAV guidance

UAV guidance and vision-based RL have been researched as separate topic. However, there has been relatively little work on the application of vision-based RL for UAV guidance. This work aims to contribute to the filling of this "knowledge-gap" by implementing a vision-based RL guidance system in an AR Drone 2. First a RL agent learning a guidance task using vision-based states and rewards is simulated. Following this the agent is implemented in an AR Drone 2 and real life tests carried out. The differences between the simulation and practical tests (i.e. the "reality gap") is studied. The developed RL agent uses a vision-based state, thus there is no need for absolute position information. Furthermore, the UAV explores the environment by itself in order to find "good" paths to perform the navigation task. This makes the system more autonomous than one with a pre-programmed flight plan. The proposed method abstracts the mapping and localization by using relative vision-based states.

Simulation of the guidance tasks are carried out. The tasks consist of traveling to goal locations marked by colored rectangles. PaparazziUAV and FlightGear are used to simulate the RL agent. The learned policies are consequently uploaded to an AR Drone 2 and the performance of the real and simulated quadcopter are compared.

The next section (Section II) provides some background on RL and describes the algorithms used in this study. Section III discusses the navigation tasks, the evironment of the guidance tasks and details about the agent. After this the simulations and the flight tests are expanded upon and their results explained in Sections IV and V. Finally, the report is concluded in Section VI with a statement of the findings of this paper and directions for further work.

II. Background on Reinforcement Learning

The following subsection provides a generalized explanation of the elements that make up RL and its working principle. After that an overview of two key concepts of RL, the ideas of value and policy, are explained. Lastly the learning method used in this work is described. The information provided in this section is sufficient to create the vision-based RL agent that will be taught a guidance task.

A. Overview of RL

RL is a machine learning method which can be used to teach a software agent sequential decision making tasks that have delayed rewards. In RL, the learning entity is called the agent. It acts in an environment in order to accomplish a defined goal. While acting in the environment the agent senses its state and a scalar reward accompanied with every state transition. The reward is a scalar measure of how "good" it was to take the previous action from the previous state. The goal of an RL agents is the maximization of the sum of future rewards that it can accumulate.

One of the most basic forms of the RL problem is characterized by a finite and discrete time Markov Decision Process (MDP). A MDP is a decision process whose state and state transitions satisfy the Markov Property. For a description of the state and the consequent state transition brought on by the environment to be Markov, future states must only depend on the current state and the current action. This property is important for many RL

methods as it allows learning based on the current state. Since all future states the agent can be in are fully determinable by its current state and action, the expected sum of rewards the agent can gather from any state is also function of its current state and action.

A finite MDP can be described using the five following elements: a finite **state space** (S), a finite **action space** (A), the **transition probability** ($P(s_{t-1}, s_t, a)$) between the states, a **reward model** ($R(s, a)$) and the **discount factor** (γ).

State Space (S) The state space consists of all the possible discrete states the agent can be in.
Action Space (A) The action space consists of all the actions the agent may take.
Transition Probability ($P(s_{t-1}, s_t, a)$) The transition probability is the system model. It describes the likelihood of ending up in state s_t given the agent takes action a from state s_{t-1}.
Reward Model ($R(s, a)$) The reward model describes the reward, r_t, that the agent obtains when it transitions to a new state.
Discount Factor (γ) The discount factor is used to make the sum of rewards to infinity bounded for non-episodic tasks.

B. Return, Value and Policy

The sum of rewards the agent obtains is called the *return*. RL agents aim to maximize the return. The *value* ($V(s)$) of a state is given by the expectation of the discounted sum of rewards from that state to the terminal state (Eq. (1)). An RL agent learns better ways of performing a task by acting in its environment and estimating the value of each of the states. The value is an estimate of the return that may be obtained from a state while following a certain policy. Being in states of higher value implies the agent will be able to accumulate more rewards.

$$V^{\pi}(s_t) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1}$$

The policy ($\pi(s, a)$) expresses the probability of taking an action when in a specific state. The value of a state depends on the way the agent acts, thus the estimated values in RL depend on a specific policy ($V^{\pi}(s)$). The optimal policy (π^*) maps the action which leads to the maximum return to every state (i.e. the best action is selected 100% of the time). A goal of RL is to find the optimal policy (π^*). Choosing an action which leads to the state of highest value is referred to as acting "*greedily*".

$$\pi^*(s_{t-1}) = \arg\max_a \sum_{s_t} P(s_{t-1}, s_t, a) V^*(s_t) \tag{2}$$

C. Q-Learning

Temporal difference (TD) learning [21] is a RL method which can be used to find optimal policies by estimating the value of each state through interactions with the environment. TD learning works by adjusting the estimate of the value of a state based on the reward and the value of the next state.

$$V(s_{t-1}) = V(s_{t-1}) + \alpha(r_t + \gamma V(s_t) - V(s_{t-1})) \tag{3}$$

On the one hand, the agent must explore the environment to estimate the value of the states. On the other hand, it must use the estimated values to act optimally. These two contradictory requirements lead to the dilemma of exploration versus exploitation in RL.

An approach to deal with the dilemma is using a policy that encourages random actions at the start of learning and optimal actions at the end of learning. This study uses the ϵ-greedy policy which picks a random action with probability $1 - \epsilon$ and a greedy (or optimal) action with probability ϵ[a]

Through the use of such a policy, a value approximation relation of Eq. (3) and an optimal action definition of Eq. (2), it is possible to come up with progressively better policies by interacting with the environment. With sufficient exploration and improvement of the policy, the optimal policy will be found given the problem is a discrete time MDP [33].

[a]Many prefer to use another interpretation of ϵ, where a higher ϵ implies more random actions

However, without a model of the system one cannot find the optimal actions from the state values ($V(s)$) alone as the agent has no way of knowing how every action causes the state to transition. The model is needed to estimate the optimal action shown in Eq. (2). There are numerous solutions to this problem; the one used in this study is Q-Learning as proposed by Watkins [34]. In Q-learning state-action values (or Q-values) are used instead of state values to bypass the need for a model.

$$Q(s_{t-1}, a_{t-1}) = Q(s_{t-1}, a_{t-1}) + \alpha \left[r_t + \gamma \max_a Q^\pi(s_t, a) - Q(s_{t-1}, a_{t-1}) \right] \tag{4}$$

$$\pi^*(s_t) = \arg\max_a Q^*(s_t, a) \tag{5}$$

Estimating the Q-values for acting optimally allows the agent to select the best actions without a transition probability as it can pick the action with the maximum value from the state it is in (Eqs. (4), (5)). An added advantage of Q-learning is its off-policy learning capacity. This makes any policy that sufficiently explores the state-action space a viable option for approximation of the Q-values. This is the case as Q-learning always makes the Q-value updates based on the next optimal action (Eq. 4).

III. Tasks and Agent

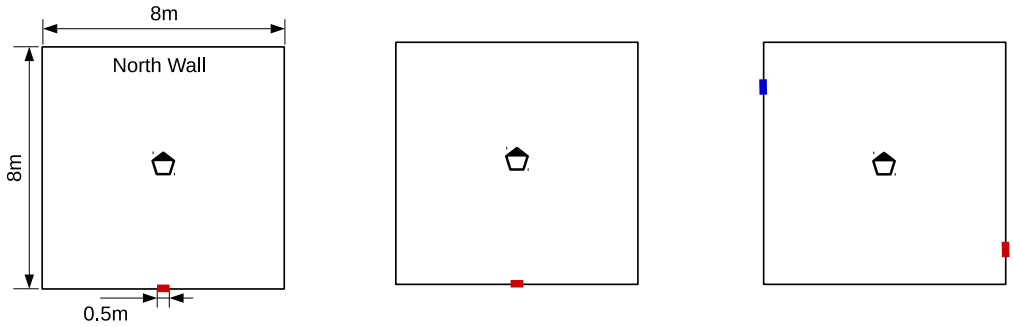
This section describes the guidance tasks, the designed RL agent and a rule based controller for performance comparison with the RL agent. The exact nature of the vision-based state and the reward scheme selected are described. Following this, the performed simulations and their results are discussed in Section IV.

A. Guidance Tasks

The guidance task consists of approaching fixed markers in an obstacle free 8m by 8m square room (Figure 3a). The goals are physically represented by colored rectangles of dimensions approximately 42 cm by 60 cm (the dimension of an A2 papers). Two guidance tasks are defined:

One goal One red goal is placed at the middle of the South wall. The quadcopter must turn and approach the goal until a threshold amount of red pixel is seen. With fixed initializations, the quadcopter starts facing 180° away from the goal at a distance of 4m from the goal (Figure 3b).

Two goal A blue goal is placed near the northern side of the East wall and a red goal is placed near the southern side of the West wall (Figure 3c). The quadcopter must approach both goals until the threshold amount of the specific goal is seen. With fixed initializations, the quadcopter starts in the middle of the room facing the North wall; one goal is in its front left and another in its rear right .



(a) Dimensions **(b) One goal task** **(c) Two goals task**

Figure 3: The dimensions of the environment, position of the goals for the two tasks and their initialization state

The room for the flight test is a 10m by 10m enclosure called the Cyber Zoo. It is equipped with a motion sensing system[b] which accommodates experiments with robots. The quadcopters environment is constricted to an

[b]Optitrack: Motive Tracker http://www.naturalpoint.com

8m by 8m region bounded by virtual walls inside the Cyber Zoo. For the simulation, an approximate 3D model of the 8m by 8m environment of the quadcopter in the Cyber Zoo is created. Snapshots from the real and the simulated environments are presented in Figure 4.

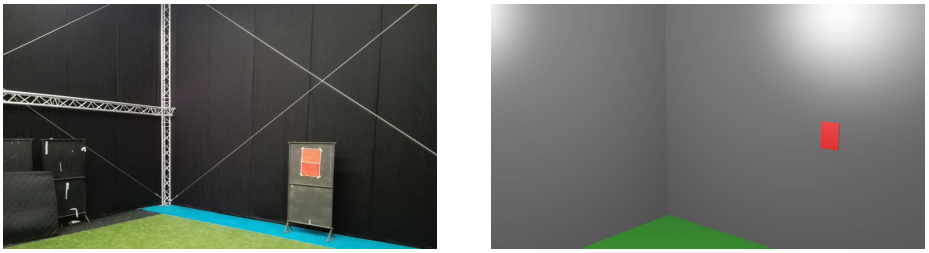


Figure 4: A picture of the real environment (left) and simulated environment (right).

The tasks are simulated with two sets of actions in order to look at the effect of incorporating expert knowledge into the RL agent through pre-programmed actions. Additionally, the effects of fixed and random initialization are studied.

B. Agent

This subsection discusses details about the agent: the vision-based state used by the agent is described, its learning and exploration scheme are discussed, the two action sets used in this study are presented and the reward scheme is characterized.

1. Vision-Based State

The separate components of the vision-based state is shown in Figure 5. The first component of the state consists of three integers which represent seeing the goal at different parts of the quadcopter's field of view. The second component represents the amount of pixels being seen by the quadcopter and provides a sense of distance to the goal. The third and fourth components are memory states; the third component represents which goals have been visited and the fourth component represents a collision with the wall during the previous action.

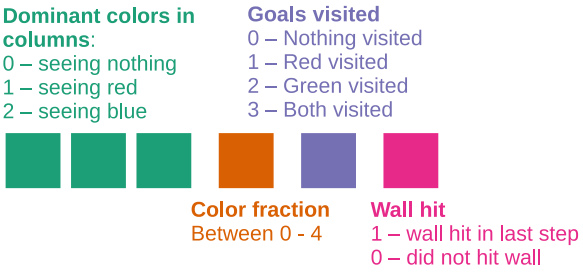


Figure 5: Representation of the vision-based state and its constituents

The first and second components of the state contain the information required to guide the agent towards the goal. The task requires the agent to approach colored markers (goals). Thus, the agent has to be able to distinguish the colored markers and get an idea of where it is relative to the markers from vision information. The first component of the state represents the information about the specific colored marker being seen and the relative lateral location of the marker with respect to the quadcopter.

The image frame is divided into three columns and the number of pixels above predefined thresholds are counted in each of columns. The three integers represent the detection (or no detection) of colored pixels above the threshold in the three columns. Some examples of what the quadcopter sees and the corresponding first component of the vision-based state are shown in Figure 6. The integers take a value of "0" if there are no pixels above the threshold, a value of "1" if there are pixels above the red threshold and a value of "2" if there pixels

above the blue threshold. Therefore the state "0,0,1" represents the presence of red pixels (above the threshold) in the right column.

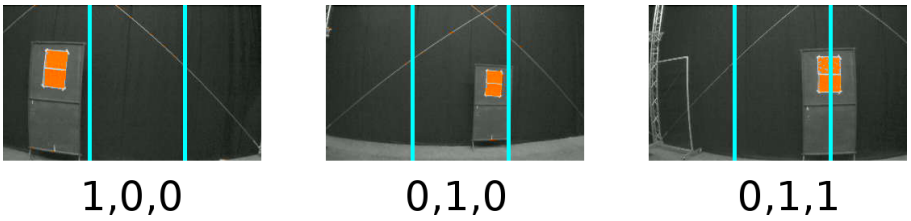


Figure 6: Three snapshots of the quadcopter's vision during simulation. The segmentation of the image into three columns and the consequent dominant columns part of the vision state have been depicted.

The detection of the red or the blue goal is based on the detection of pixels whose values are higher than a threshold. The *thresholding* for the detection is in RGB color space in the simulation and in YUV color space in the flight tests. In simulation, the RGB values of each pixel is summed and the percentage of red and blue calculated. A pixel is considered a red or a blue goal if more than 75% of its value is in the respective color. In flight tests, a pixel is considered red if its U components is less than 0.0 and its V component is greater than 0.13. The thresholds for detecting the blue are not defined for the flight tests as only the one goal task is performed in real life.

The second component (fourth integer) of the state represents how much of a goal is being seen by the quadcopter, and is a relative description of the distance to the goal. The sum of pixels above the threshold is divided by 5000 and the resulting number floored to obtain a discretized representation of the number of pixels above the threshold which are being seen. This integer is named the "Color Fraction" in this study. Its other use is deciding if a goal has been visited. In the simulations, the red goal is considered visited if the color fraction is greater than "3" and the blue goal is considered visited if the color fraction is greater than "2". The difference in the two thresholds is due to the lighting in the models of the environment which makes the blue goal less visible. Hence the range of values for this component of the state is between "0" and "3".

The third component (fifth integer) is a memory state which is relevant for the two goal task. This integer remembers which of the goals has been visited. It has a value of "0" when neither goals have been visited, a value of "1" if the red goal has been visited, a value of "2" if the blue goal has been visited and a value of "3" if both goals have been visited. For the one goal task this component can take one value; for the two goal task this state has three possible values.

The fourth component (sixth integer) of the state represents a collision with the wall caused by the previous action. This information is used to teach the agent to avoid walls . Its value is "1" if a forward movement in the previous action would have resulted in a collision with the wall. In such a case, the quadcopter does not make a forward movement[c]. For all other actions, its value is "0".

Such a state description leads to a total permutation of 64 states for the one goal task[d] and 748 states for the two goal task[e]. However, in reality the number of states are lower than this due to the definition of the environment. For example, it is not possible to see a disjoint goal [f] neither is it possible to see two goals at the same time[g].

2. *Learning scheme and parameters*

This study uses Q-learning with an ϵ-greedy policy. The values for the learning rate (α) and the discount factor (γ) are kept constant at 0.3 and 0.9 respectively. As the goal of this work is the real life application of a vision-based RL agent on a quadcopter, a sensitivity of the agent to the RL parameters is not performed, as long as there is learning and convergence with the used values.

The exploration-exploitation problem is handled by increasing the greediness of the agent sequentially as the training progresses. A number of starting exploration rates (i.e. ϵ_0) were tried out in simulation; the trials led to a starting value of 0.60. A high early exploration rate was found to be wasteful in terms of learning rate as the

[c] It is assumed that it has some kind of distance sensor to obtain this information
[d] $2^3 \times 4 \times 1 \times 2$
[e] $3^3 \times 4 \times 3 \times 2$
[f] the state "1,0,1" is not possible
[g] the state "1,2,0" is not possible

number of possible states in either of the tasks are not significantly high. The schedule of increasing the ϵ for the two simulated tasks and the flight test is presented in Figure 7.

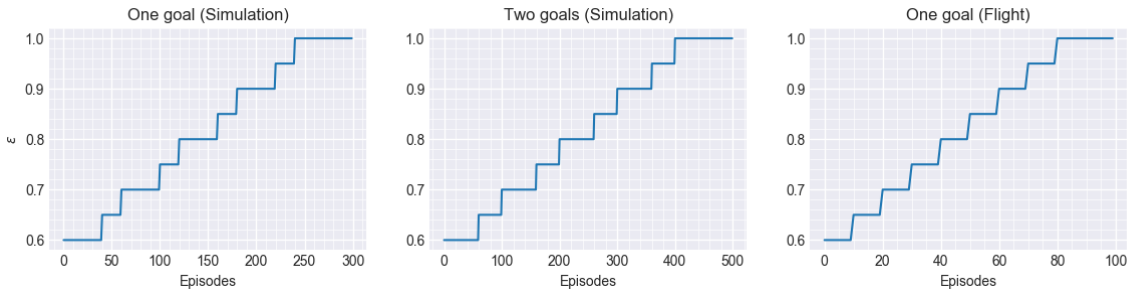


Figure 7: The ϵ schedule for training the RL agents in this study

3. *Actions*

Two sets of actions are defined for the agent. The *primitive action set* consists of going forward, turning left by 22.5° or turning right by 22.5°. The *extended action set* consists of the actions in the primitive set supplanted with an additional action. The extra action is a temporally extended action, called an *option* [35], which causes the agent to keep turning right until it sees a goal. The option is available to the agent only when it is seeing nothing (i.e. in the state 0,0,0;0;x;x). It is implemented with the intention of speeding up learning by incorporating external knowledge. The two sets of actions available to the agent are visualized in Figure 8.

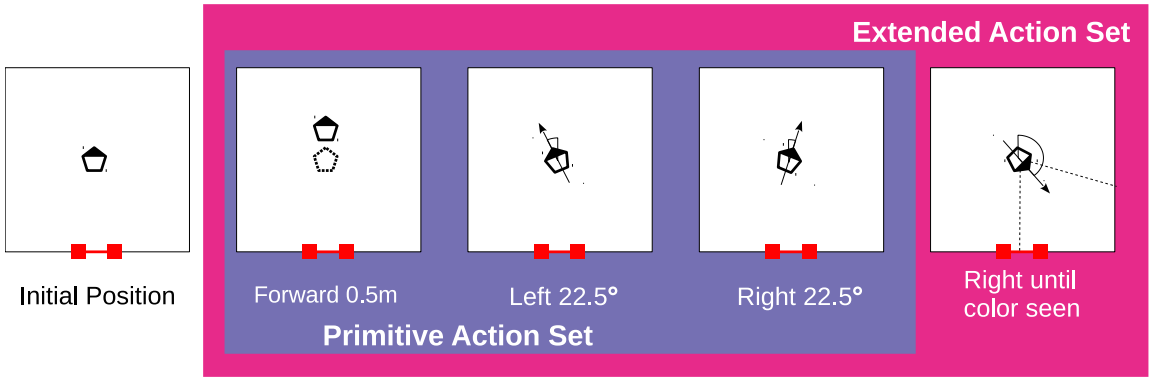


Figure 8: Depiction of the primitive actions and the extended action that are avalable to the agent.

4. *Reward scheme*

The reward scheme is defined with the objective of teaching the RL agent the task of guiding itself to (and between the) goals in the least number of steps. The designed reward accomplishes this objective by encouraging (or discouraging) three things: all movements are penalized to minimize the number of steps, seeing an unvisited goal is encouraged while seeing a visited goal is discouraged and finally visiting an unvisited goal is encouraged. A flow chart depicting the logic of the reward scheme is presented in Figure 9.

The logic for these three encouragements to the agent through the reward scheme are colorcoded in Figure 9. The green rhombuses are the initial penalty applied to every action as it is desired to minimize the number of steps required to finish the task. Simulations found that incorporating the knowledge of moving forward when seeing a goal and turning when not seeing anything in the reward scheme speeds up learning.

The orange rhombuses represent the reward for taking an action that ends up with the goal in sight. This reward is higher if more of the goal is seen and thus depends on the color fraction. For the two goal task, there is a penalty for having a visited goal in sight. This encourages the agent to visit the other goal. Finally, the agent is awarded a big positive reward for visiting an unvisited goal through the logic in the purple rhombus.

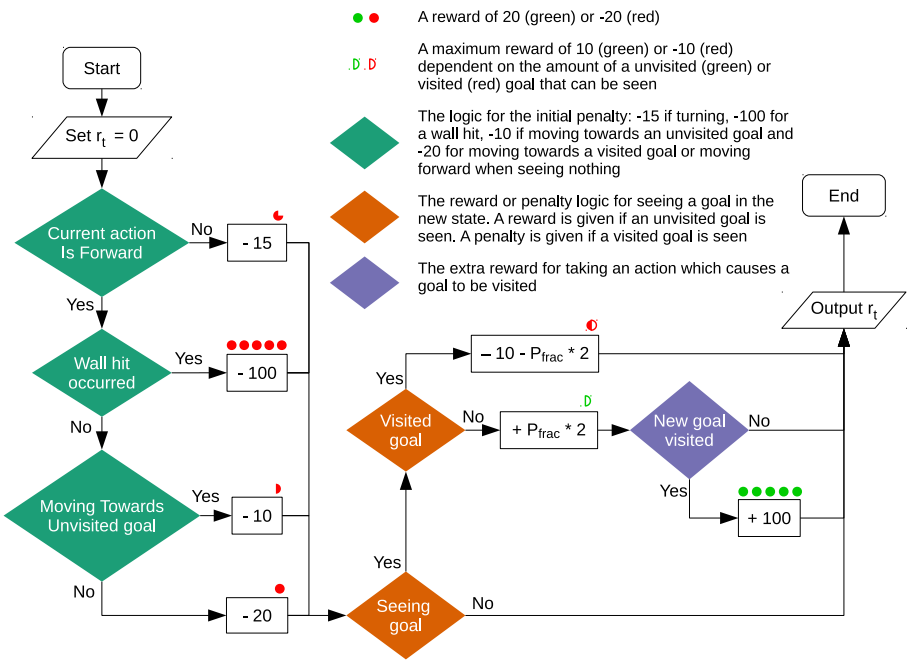


Figure 9: Flowchart depicting the selected reward scheme for the task. r_t represents the current reward and P_{frac} represents the count fraction. The reward for each step is initialized at 0 and modified according to the logic presented in this chart.

C. Rule based controller

The performance[h] of the trained RL agent is compared to a rule based controller using the same vision-based states as for the RL agent. The action selection logic of the rule based controller is presented below:

Action logic for approaching one goal

1. If not seeing goal → *Turn right*
2. Else if seeing goal:
   (a) If "Hitwall" = 0 (False):
      i. if "Color Fraction" <= 2
         A. If goal ONLY in middle column : → *Move forwards*
         B. Else : → *Turn towards goal*
      ii. Else if "Color Fraction" > 2 : → *Move forward*
   (b) Else if "Hitwall"= 1 (True): → *Turn away from goal*

Action logic for approaching two goals

1. If seeing visited goal → *Turn right*
2. Else → Use logic for approaching one goal

IV. Simulation

The goal of this work is the implementation of a vision-based RL guidance agent in an AR Drone 2. A simulation is performed before trying out a real life implementation. This section describes the setup and results of the simulations.

[h] in terms of the steps required to reach the goal

A. Setup

The software used, the training scheme it's scheme are described in this section.

1. *Software*

The simulations are carried out using a combination of the PaparazziUAV's [36] simulator and FlightGear [37]. PaparazziUAV uses JSBSim [38] to simulate the autopilot software that it generates. FlightGear is an open source flight simulator which generates the vision information required for simulation.

The two features which make FlightGear appropriate for vision simulation with PaparazziUAV are its capacity to obtain state information from an external software and its ability to run a local HTTP server on a specified port. The server can be used to obtain screen shots of FlightGear's rendering of the environment. PaparazziUAV has built in infrastructure to interface with FlightGear and send it the aircraft state information. FlightGear can use this state information to visualize the environment of the aircraft.

The software for the RL agent has been implemented as a module inside PaparazziUAV. The state and rewards of the RL agent are defined based on visual data. A submodule for vision simulation is created which downloads screenshots from a pre-programmed HTTP address. The flight dynamics of the agent is simulated using PaparazziUAV's JSBSim. This data is fed to FlightGear through a prescribed UDP port[i]. FlightGear renders the environment and uploads its current view to the screenshot path of the HTTP server upon receiving a HTTP request on that path (e.g. http://localhost:9723/screenshot). This is the pre-programmed address in the vision simulation submodule. The screenshot is then unencoded as a RGB bitmap from JPEG and used to generate the vision based state and the reward.

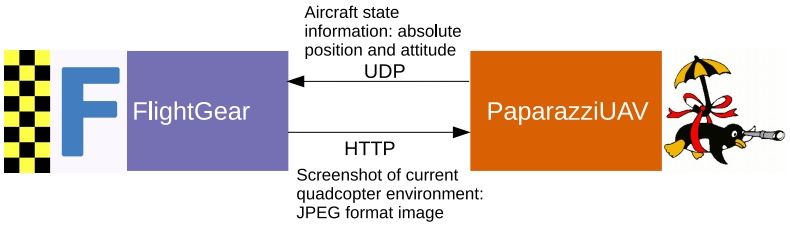


Figure 10: The interactions between the simulated autopilot software and FlightGear.

2. *Task Training*

The parameters and the training scheme are described in the following paragraphs. Both tasks are simulated with the fixed and random initializations; each initialization is simulated with the primitive and the extended action sets (Figure 8). This leads to a total of four factor levels for each of the tasks. First the learning in both the tasks are presented. The four cases trained are : (a) Fixed initialization; Primitive action set (b) Fixed initialization; Extended action set (c) Random initialization; Primitive action set (d) Random initialization; Extended action set. Furthermore, the performance of the trained RL agents are compared to a rule based autopilot (see Section III-C).

The one goal task is trained for 300 episodes using Q-learning. The training runs are repeated 50 times to obtain statistical estimates of the performance. The schedule for increasing the greediness (ϵ) is presented in Figure 7. The learning rate (α) is kept at 0.30 and the discount factor (γ) is kept at 0.90. The two goal task is trained for 500 episodes with 50 repetitions. The ϵ is increased according to Figure 7 and like the one goal task, the α and γ are kept at 0.30 and 0.90 respectively.

B. Results

This section discusses the learning performance in terms of the number of steps required to reach the goal. Note that to improve readability, the RL agent is referred to as the agent, and the rule based autopilot as the autopilot.

[i]The FlightGear and the simulation software need to be run with specific arguments for this to work

The learning of the agents for the one goal and the two goal task are summarized in Figures 11 and 12. In both figures, the left plot shows the change in steps, the middle plot shows the sum of changes to the Q-values and the right plot shows the sum of reward over the 50 runs. The sum of changes to the Q-values are calculated by summing the changes made to the Q-values at each step, over the whole episode.

Two means are taken in order to obtain the statistics of the run as presented in Figures 11 and 12. For both tasks the aforementioned quantities at each episode is averaged over the 50 repetitions of the training. Than the means over 50 repetitions are split in groups of 60 episodes for the one goal task and 100 episodes for the two goal task. The data points in Figures 11 and 12 represent the grouped run means. The reason for grouping 60 episodes for the one goal task and 100 episodes for the two goal task is based on the number of episodes over which the greediness (ϵ) is increased by 10%.

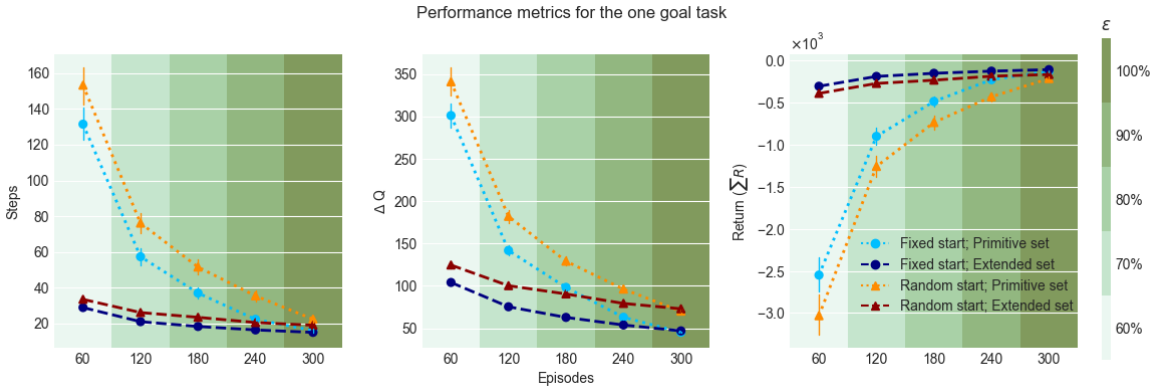


Figure 11: Learning of the one goal task for four conditions

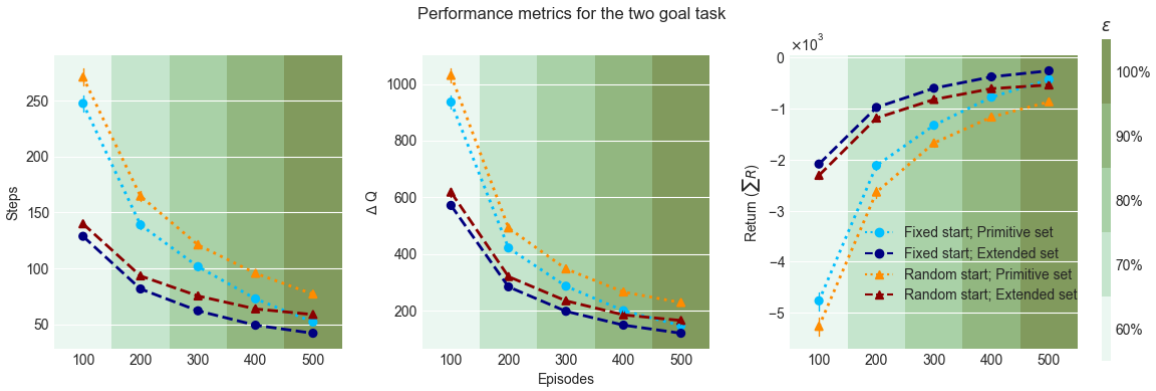


Figure 12: Learning of the two goal task for four conditions

The decreasing trend of the steps and the increasing trend of the returns in Figures 11 and 12 show that, for both tasks and all simulated factors, the agent manages to learn to perform the task better as it accumulates more experience. However, there are differences in the learning and final performance of the trained agents.

ONE GOAL TASK: Table 1 compares the performance of the trained agents and the autopilot. The table shows that the randomly initialized agents take more steps to learn and perform the task than the agents with fixed initializations. It also shows that the agents perform worse than the autopilot in all the simulated conditions. The agent performs nearly as good as the autopilot only for the one goal task with fixed initializations.

The exclusion of the episodes, where the autopilot gets stuck in infinite loops, is one of the reason for its lower mean steps to reach the goal with random initializations. The autopilot gets into infinite loops when it is initialized near the wall, while also seeing a goal. The inability of the vision-based state to remember a collision with the wall for more than one step causes this infinite loop. When the autopilot collides with a wall, while it is seeing a

goal, it first tries to turn away from the goal. On the very next step it turns towards the goal to put it in the center of its field of view. After having the goal in the center, it tries to move forward again. The forward movement results in a collision, restarting the cycle.

However, the agent does not necessarily get stuck forever in these kinds of loops. The value of the learned actions keep changing while it goes goes through the loop. At some point, the values of the actions causing the loop may decrease low enough that another action become more valuable in comparison to it. This causes the agent to pick the other action, breaking the loop.

Table 1: Performance metrics of the agent and the autopilot for the two tasks

Initialization	Fixed		Random	
Action set	Primitive	Extended	Primitive	Extended
		One goal task		
Sum of steps	15839.08	5960.18	20317.54	7336.20
Steps to goal (RL)	14.89	14.91	24.22	19.27
Steps to goal (Rule based)	14.56		15.26	
		Two goals task		
Sum of steps	61311.12	36419.14	72941.62	43149.54
Steps to goal (RL)	52.25	41.22	74.40	58.40
Steps to goal (Rule based)	35.04		33.57	

The extended action of turning until a goal is seen, decreases the steps required to learn the one goal task by a factor of approximately 2.5 (Table 1). This is on account of the way the tasks and the agents are formulated in this study. There is greatest ambiguity in the states when the agent does not see anything. Most of the real positions and headings the agent can be in, maps to seeing nothing. The positions for four different headings where the agent sees nothing is illustrated in Figure 13. Having the option to turn until a goal is seen allows the agent to circumvent part of this ambiguity; the agent can use the option to transition from the ambiguous state of not seeing any goals to a less ambiguous one of seeing a goal.

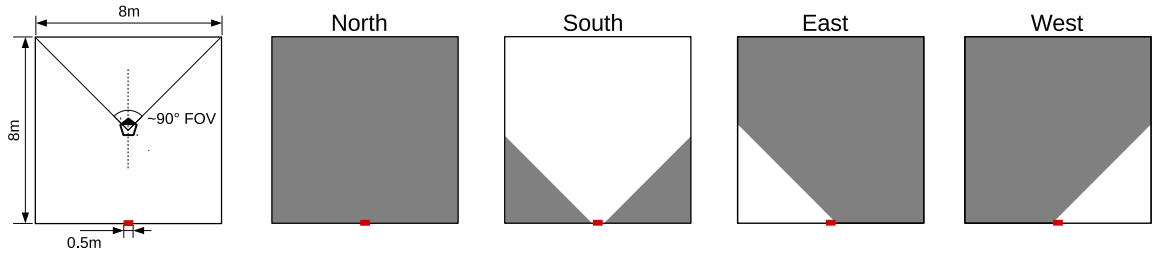


Figure 13: Figures illustrating the regions in the environment where the quadcopter cannot see anything for four different headings. Grey represents seeing nothing and white represents seeing a goal. The leftmost figure shows the dimension of the environment, the goal and the field of view of the quadcopter. The leftmost figure shows the quadcopter facing the reference North.

The agents using the extended action set have a better final performance than the ones using the primitive action set (see Table 1). Due to the ambiguity in the state description, the optimal policy is dependent on the initialization of the quadcopter. For example, of the environment and the task implies that for some initial headings the goal will be in the left region of the quadcopter, while for others it will be in it's right. The optimal actions for these different initial headings are left and right turns respectively. However, as both initializations of the heading have the same vision-based state (i.e. "0,0,0"), they can map to one action. The vision-based state does not encode any information about the relative heading of the quadcopter with respect to the goal. Thus the agent cannot learn optimal actions for this case, and performs sub-optimally.

TWO GOALS TASK: The use of the extended action decreases the steps required to learn the two goal task by a factor of approximately 1.5. The improvement in learning is less than in the one goal task because the option is less effective in reducing the ambiguity for the two goal task. Firstly, the option does not always transition the state of the agent from seeing nothing to seeing an unvisited goal. If, for example, the agent has already visited the blue goal and turns left a number of times resulting in a state where it sees nothing, choosing the option will transition the quadcopter back to seeing the blue goal again. Secondly, the agent has to spend more time in a region of the state-space where the option is not available. The agent cannot choose the option when it is seeing a goal. In the two goal task, the goal is in sight of the quadcopter for a bigger part of the environment and it has to spend more time learning the right values for each action while seeing either of the goal.

Unlike the one goal task, both initializations with the two goal task perform worse than the rule based autopilot (see Table 1). The two goal task is harder to learn; the state space is bigger and the agent has to learn more things. Furthermore, there are more points where the agent and the autopilot can get stuck near the wall. There are two goals in the environment, thus there are more points where the agent can be initialized next to a wall with the goal in front of it. As, with random initializations, the autopilot gets into infinite loops more in the two goal task than the one goal task. Ignoring the infinite loop episodes have a greater effect in decreasing the mean of the autopilot in the two goal task.

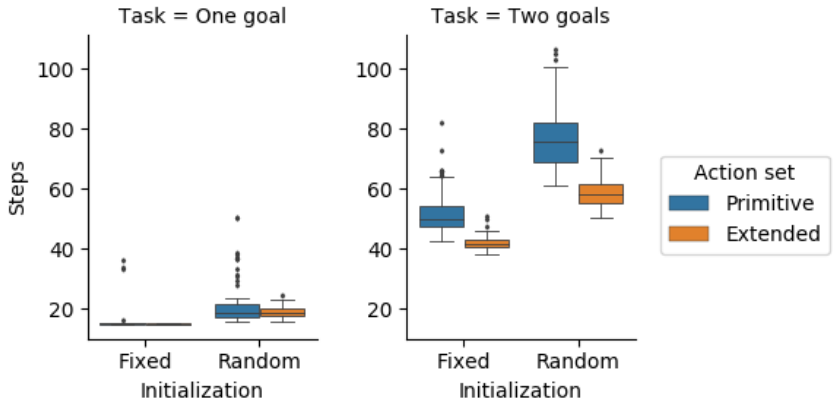


Figure 14: Box plots of the steps taken by the agent for the two tasks after training

The means and standard deviations of the final performance for both tasks with both initializations is presented in Figure 14. The box plots are created using the final fully greedy parts of the training for the two tasks[j]. Due to the small variance for the one goal task with fixed initialization, their boxes appear as lines in Figure 14. The smallness of the variance in the final performance of the one goal agent with fixed initialization is evidence of the convergence of the learning.

A difference is observed in the number of outliers with and without options. The number of outliers are higher for the primitive action set for both tasks, with both initializations. This indicates that with the primitive action set the agent may get lost, but with the option the agent performs consistently.

There is a bigger gap in the mean performance of the trained agent between the two action sets for the two goal task than for the one goal task. This can be explained by the difference in the "usefulness" of the option in performing the different tasks. As the one goal task comprises of turning to the red goal and approaching it, it is simpler and there is a smaller chance for the quadcopter to get lost. Hence near optimal performance is easier to obtain without the need for an action which helps bring the quadcopter back to the more relevant region of the state space (i.e. when it is seeing a goal). The two goal task on the other hand comprises of four steps: turning towards the closest goal, approaching it, turning towards the other goal and approaching it. There is a bigger chance of getting lost and following a trajectory which adversely influences the Q-value estimates this case. With the extended actions, the quadcopter stays in trajectories which reduce the variance in the value estimate that result from the state ambiguity. Therefore, the final performance is better and the variance in the performance is lower with the extended action set for the two goal task.

[j] i.e. the last 60 episodes for the one goal task and last 100 episodes for the two goal task

V. Flight test

The RL agent is implemented in an AR Drone 2 and trained to perform the one goal task in real life flights. The flight tests aim to show that a vision-based RL agent can be trained to perform a guidance task in real life. Q-values learned in simultion are tested on the AR Drone 2 to study the reality gap in the simulations.

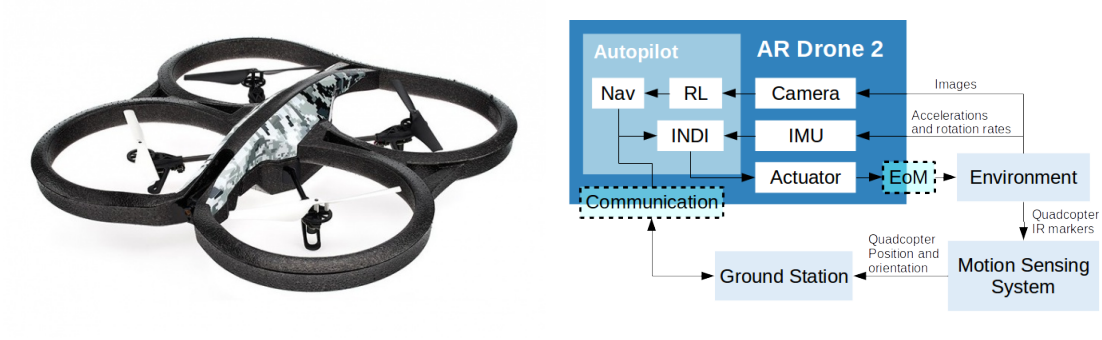
A. Setup

The setup of the one goal task has been described in Section III. This section discusses the other elements of the flight tests. First, the AR Drone 2 and its applicability to this study is outlined. After this, an overview of the software and hardware elements of the autopilot, and their interactions is provided. Lastly, differences between the simulated and the real life vision are discussed.

1. AR Drone 2

This study uses a Parrot AR Drone 2 Elite Edition (see Figure 15a) to perform flight test on the developed RL agent. The key features of the quadcopter which make it suitable for these flight tests are its onboard camera and its use of an open source operating system.

The specifications of the AR Drone 2 meet the requirement of this study. The forward facing camera is capable of producing 1280 by 720 pixel video at 30 FPS. The simulations are carried out at 800 by 600 resolution and the designed agent takes at least 1 second for each step. As it takes 1 second for each step, it requires a new estimate of the vision information at approximately 1 Hertz (i.e. it requires images to be processed at 1 FPS). Therefore both the resolution and the video frame rate of the AR Drone 2 are sufficient. It uses BusyBox, a distribution of Linux, as its operating system. This enables the use of open source autopilots such as PaparazziUAV for its control.



(a) Parrot AR Drone 2 Elite edition **(b) Controller architecture**

Figure 15: The AR Drone 2 and the controller architecture

A key limitation of small quadcopter, such as the AR Drone 2, is their short flight times. On full battery, the AR Drone 2 is capable of approximately 12 minutes of flight. This creates a challenge for RL which requires a thorough exploration of the state-space. The quadcopter is tethered to an external power supply to enable the prolonged flights required for RL.

2. Autopilot Elements

An overview of the interaction between the different elements of the system is presented in Figure 15b. Three parts of the PaparazziUAV autopilot are visualized: the implemented "RL" module, the position controlling navigation module ("Nav") and the rate controlling Incremental Non-linear Dynamic Inversion ("INDI") [39] module. The hardwares these modules interact with are the actuators, the communication system, the IMU and the camera.

PaparazziUAV's autopilot controls the internal rate, position and attitude loops using feedback from the quadcopters IMU and the motion sensing system of the CyberZoo (see Section III). The RL agent is a higher level

controller which decides between going forward or turning in either direction so that the quadcopter can reach the goal.

The movements of the quadcopter are also detected by the external motion sensing system of the Cyber Zoo. This information is passed to the autopilot through the ground station. Although the autopilot uses absolute position and attitude information for stabilization and position control, the RL agent makes higher level guidance choices based solely on the visual data. The inner loop control, although a challenging fields in its own regard, is out of the scope of this study.

3. *Differences with simulated vision*

The visual stream from the real runs have the following differences in comparison to the simulation runs:

- There is no noise in the vision data in the simulation but real life vision data from the quadcopter is noisy (see Figure 16).

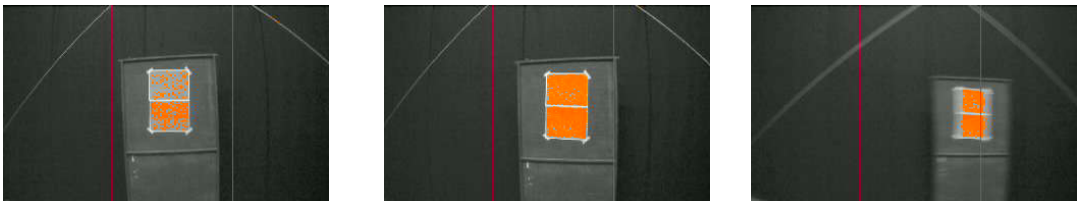


Figure 16: A noisy viewing (left), a less noisy viewing (middle) and a blurred viewing (right) of the red goal captured during flight tests. The red pixels are detections.

  The simulations are carried out with the time of day and the lighting conditions frozen at a specific point. Although the CyberZoo is enclosed in three directions, ambient light can still get in and change the nature of the visual data depending on the time of day. Further, blurring of the visual data (see right image in Figure 16) caused by the motion of the quadcopter is not present in the simulated vision as FlightGear does not support it.
- The resolution of the image, the field of view of the camera and the colorspace are different between the simulation and the real-life system. The real system is tuned such that its outputs resemble that of the vision module in simulation.
- The computer vision module operates at a frequency of about 1 Hz in the real-life implementation. The total time to perform one action is between 1 to 3 seconds. There is a chance for the agent to use an image frame from before an action is over to estimate its vision state after the action is taken. This can result in the RL agent sensing the wrong vision-state. Figure 17 illustrates the delay in perception. This does not happen in simulation as the RL agent grabs a frame from FlightGear only after performing an action. The problem of delayed perception can be mitigated by modifying the implementation; modifications to alleviate this problem are not attempted due to time constraints of the project.

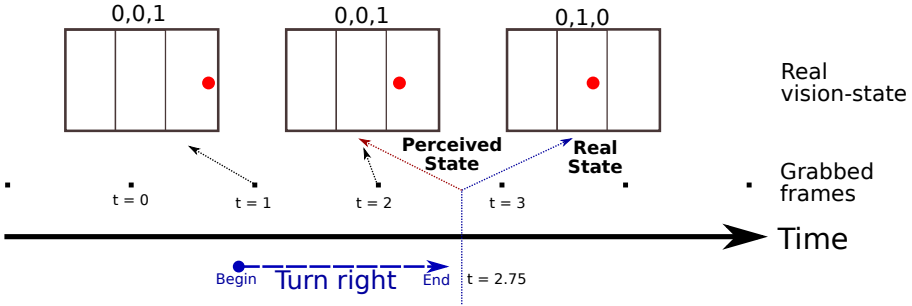


Figure 17: Illustration of the erroneous perception resulting from the fixed update rate of the real-life vision module.

- The camera distorts the visual data due to its
  - varying sensitivity to the different spectrums of visual light

– fixed focal length which blurs objects out of its depth of field
– imperfections (or flaws) in the lens

B. Results

Flight tests are performed to demonstrate the learning in real life and study the reality gap between the learning in simulation and in real life.

FLIGHT TESTS: The one goal task is trained for the shorter duration of 100 episodes in the flight tests. Simulations of the one goal task with 100 episodes of training showed that the agent can learn the task with 100 episodes of training. The leftmost and rightmost plots in Figure 19 visualizes the steps required by the simulated agent during and after training with 100 episodes.

The scheme for increasing the ϵ is justified in Section III and visualized in Figure 7. Figure 18 shows the mean trends in the learning over the 5 runs of the flight tests. The datapoints in Figure 18 represent means over 10 episodes which corresponds to a 5% increase of ϵ.

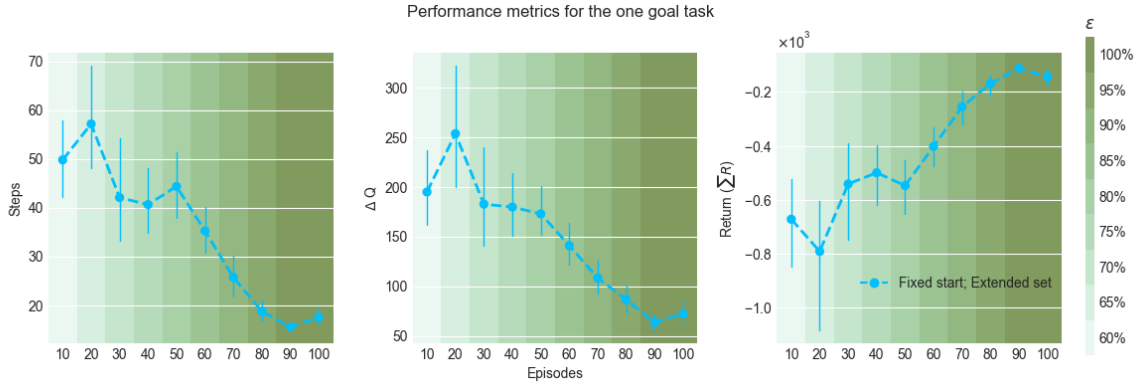


Figure 18: The learning metrics from 5 flight test of the one goal task

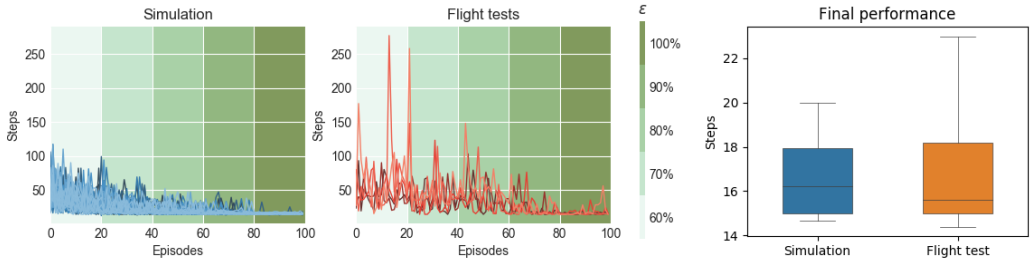


Figure 19: The two left figures show steps against episode for the 50 simulated runs and the 5 flight test runs. The rightmost figure shows the statistics of the fully trained performance.

Figure 18 shows a decreasing trend in the steps and the sum of changes to the Q-values along with an increasing trend in the returns over the training. These indicate that the agent manages to learn the task better as it explores the environment. The performance of the agent gets worse when learning with full greed; this is seen in the rightmost parts of the plots in Figure 18. The decrease in performance with fully greedy actions is on account of the ambiguity of the state description. Ambiguity in the state leads to an unstable reward function and consequently, to non-convergent learning.

Some of the differences between the real life learning and the simulated learning can be seen when comparing the left and right plots of Figure 19. The figure shows the steps required to reach the goal for the 5 flight test runs, and the 50 runs of the simulation with 100 episodes of training.

The trajectories taken by the real and the simulated quadcopter, at different episodes of the training, from a specific training run are visualized in Figure 20. The trajectories of the agent in the flight test are more erratic

because of the greater amount of noise present in real flights. Besides the noisy motion, there are visible loops in the quadcopter's trajectory during the flight tests. These occur when the quadcopter chooses the option and keeps yawing until it sees a goal. The inherently unstable yawing motion of quadcopters and the introduced forces by the power tether contributes to the drift of quadcopter while it yaws, which results in the aforementioned loops. The simulated quadcopter also has these loops, but their diameter is smaller.

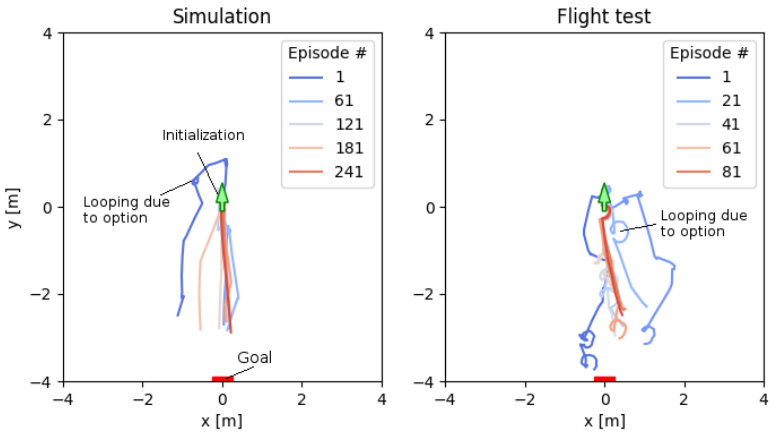


Figure 20: Trajectories taken by the simulated and the real quadcopter during one training

There is more variance in the steps required to reach the goal in flight tests in comparison to the simulations. The of noise in the setup of real-life flights and in the flights themselves cause this variance. The noise in the flight introduces variance in the result of the state transitions, which gets reflected in the Q-values. Noise in the vision information makes the quadcopter "see" erroneous things. Erroneous state perception leads to erroneous Q-value updates, leading to a greater variance in its estimate. Further, there are inconsistencies in the initialization of the quadcopter and on the accuracy of the motion sensing system. Small variations in the initialization and calibration of the motion sensing system make the motion of the real quadcopter different from the simulated quadcopter.

Another observable feature is the maximum number of steps required to reach the goal in simulation and in flight. Figure 19 shows two of the episodes in the flight test requiring more than 250 episodes to reach the goal. The maximum steps required to reach the goal in simulation out of all the episodes from all the runs is about 125 steps. In general the learning performance indicates that the performance in simulation is more consistent. This is expected as the simulation does not include many factors, such as the noise in the vision, which makes learning the task harder in reality. The fully trained quadcopter performs nearly as well as the quadcopter trained in simulation (rightmost plot in Figure 19) but with higher variance in the final performance.

SIMULATION VALIDATION: The reality gap in the transfer of policies learned in simulation to the real quadcopter is examined. This information is desirable because online learning with quadcopters is often challenging, and learning in simulation is the preferable way to attempt RL based solutions.

The Q-values learned by an agent over 300 episodes of training in simulation are uploaded and tested in the AR Drone 2. The Q-values at specific points in the training is uploaded to the quadcopter and 10 real life episodes performed with the trained Q-values. The performance of the agent in the flight test over this 10 episodes is compared to that of the simulated agent over the 10 episodes from the corresponding training region. The comparison is visualized using box plots in Figure 21.

There are two confounding factors in the validation study. Firstly, no repetitions are carried out for the episodes performed with the real quadcopter. Secondly, the seed for the random number generator in the simulation and the flight test are not the same; the random actions picked by the policy are different between the simulated agent and the real life agent.

The rightmost plot in Figure 21 shows the difference in the mean steps required to perform the task over 10 episodes for the five validation points. There is a difference of about 80 steps between the simulated and the real-life agent with the Q-values after 60 episodes of training. Q-values from consequent phases of the training have a difference of between 10 to 20 steps with the simulated agent.

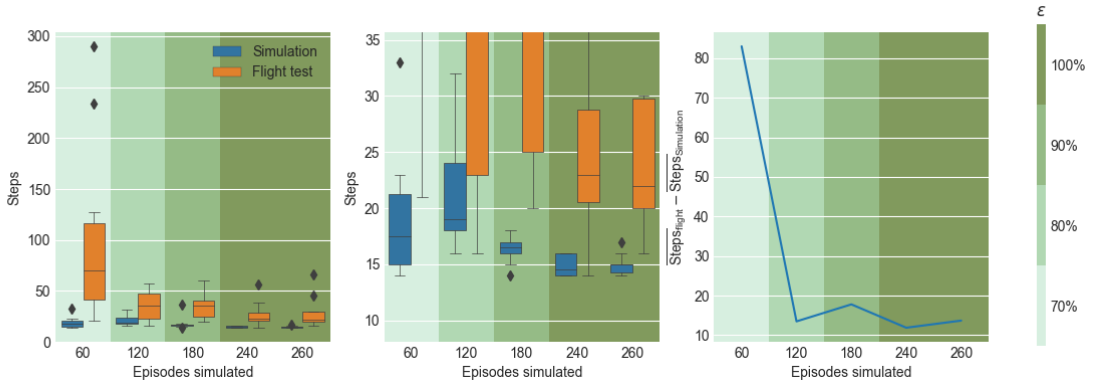


Figure 21: Statistics of the validation runs. The performance of the simulated agent and the real-life agent for 10 episodes at different points of the training is presented in the left plot; the middle plot is a zoomed view of the left plot. The right plot presents the difference between the mean steps over the 10 episodes at the different validation points.

Besides the noise, the delayed perception in the flight test (Figure 17) causes the performance of the agent trained in simulation to degrade when transferred to to real-life quadcopter. When the agent is trained in flight, this delay is incorporated into the learning of the agent. However, in the validation flights, the Q-values on which the real life agent acts are trained without the delayed perception of the real world. The underlying transition model between the states in the simulation and the flight test are different.

This validation study led to three findings: First, the real life agent is bad at incorporating Q-values from early stages of the training. Second, there is a near constant offset between the performance of the simulated agent and the real life agent for Q-values which are relatively stable. Third, the delayed perception in the flight tests is one of the causes for the difference in behavior.

VI. Conclusions

This work explores the possibility of a learning vision-based guidance controller for a quadcopter. Such systems may improve the flight of Unmanned Aerial Vehicles (UAVs) in dynamic and Global Positioning Systems (GPS) denied environments. The possibility for such a guidance controller is investigated by developing a vision-based Reinforcement Learning (RL) controller for an AR Drone 2 and training it to perform simple guidance tasks first in simulation and later in real flights.

The descriptions of the state, reward and terminal condition for the tasks are derived from vision data. The training of one task, consisting of turning to a red marker, is performed in simulation and in flight test. A more complex task of approaching two markers (one red and one blue) is trained in simulation. The effect of incorporating knowledge into the agent is studied by expanding the actions it can take to include a multi-step action, called an *option*, designed to decrease the ambiguity in the state perception of the agent. The robustness of the learning is examined by simulating it with fixed and random initializations.

The simulation results show that the agent performs[k] the tasks better with increasing exploration of the environment. The option of turning until a goal is seen improves the learning rate. Further, if the task is relatively simple and the *options* are made to consist of a sequence of primitive actions, both the set of primitive actions and the set of extended actions can converge to the optimal performance. As the complexity of the task is increased, for example by using a random starting position, the final performance of the agent with the extended set of action is better than the agent with the primitive set of actions. This is on account of the increased ambiguity in the state perception with random initializations which the *option* is better able to mitigate.

The agent in the flight test also manages to learn the task, albeit with worse learning and final performance than in simulation. The noise in the visual perception and motion of the quadcopter in flight tests lead to greater variance in the real-life learning in comparison to the simulated learning. Other factors that contribute to the

[k]Note that "performance" is used refer to the steps required to perform the task after full training and "learning performance" is used to refer to the steps required to learn the task.

reality gap are the delay in perception of the vision-based state in the real-life agent and the greater drift of the quadcopter position while yawing in real life.

The simulations and flight tests are carried out without any tuning or sensitivity analysis on the hyperparameters in order to meet time constraints. This does not have consequences for the final performance of the simulated agent performing the one goal task with fixed initialization. The observed final performance of that agent is close to the performance of the rule based autopilot. This is not true for any of the other simulations or the flight tests. Hence, the final performance for the other agents and the learning characteristics of all the trained agents in this study can potentially be improved by tuning the learning rate (α), the discount factor (γ) and trying out other policies. Namely a higher discount factor should speed up learning as more accurate information of the value of a state-action gets propagated backwards.

The development of a learning and vision-based guidance system for UAVs will broaden their domain of operation and thus increase their demand. This technology will enable the development of generic Unmanned Aerial Systems (UAS) that can be targeted towards specific markets. For example, a generic learning and vision-based UAS that is designed for checking the inventory will be usable in warehouses, supermarkets, workshops etc. Using learning and vision-based systems for guidance will remove the need to setup a local positioning system or programming the UAS autopilot for the environment on an ad-hoc basis. Furthermore, as one software architecture can be used to learn different tasks, there are potential savings in terms of software development for the UAV manufacturers. If vision-based learning can be made generic, it can be marketed to mass consumers who train their UAVs for personalized applications they desire. As the trained agent takes over the task of guiding the UAV, operating them will require less man power.

New regulatory bodies will need to be created to certify the systems and operators of learning based UAVs like the ones presented in this study. Social norms and outlooks will need to adapt to the emergence of mobile robots that are not defined by their programming, but have the capacity to improve on their designed roles over time. Before the mass marketing of learning based UAS, the manufacturers will have to ensure the safety and security of their owners and the societies where they are being marketed. Typical questions pertaining to the use of artificial intelligent systems in daily life will need to be answered. Questions such as who will be held accountable for damages caused by the autonomous operation of these systems and how to ensure the livelihood of people whose jobs are going to be replaced by such systems will need to be answered.

Perhaps the biggest factor which prevents the use of RL based controllers in current UAS is the time they take to learn. The agent designed in this study takes about two hours of flight to learn the relatively simple task of turning and approaching a goal. The learning time increases as the dimensionality of the state space is increased to enable more complex tasks. The results show that incorporating programmed behaviors can speed up learning. However, it also increases complexity of the design and implies the encoding of expert knowledge into the system. The time required for an agent to learn forces most applications to carry out the learning offline and then implement the learned policies in the UAS. Besides the slow learning in RL, vision-based applications need to deal with the problem of making sense of the large volume data that are contained in images. Generalized RL structures to map pixel based vision data to actions exist [40], although these methods are still expensive in terms of memory and computations.

References

[1] Nister, D., Naroditsky, O., and Bergen, J., "Visual odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Vol. 1, June 2004, pp. I–652–I–659 Vol.1.

[2] Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., Grixa, I. L., Ruess, F., Suppa, M., and Burschka, D., "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue," *IEEE Robotics Automation Magazine*, Vol. 19, No. 3, Sept. 2012, pp. 46–56.

[3] Kelly, J., Saripalli, S., and Sukhatme, G. S., "Combined visual and inertial navigation for an unmanned aerial vehicle," *Field and Service Robotics*, Springer, 2008, pp. 255–264.

[4] Krajnk, T., Nitsche, M., Pedre, S., Peuil, L., and Mejail, M. E., "A simple visual navigation system for an UAV," *International Multi-Conference on Systems, Sygnals Devices*, March 2012, pp. 1–6, 00034.

[5] Kendoul, F., "Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems," *Journal of Field Robotics*, Vol. 29, No. 2, March 2012, pp. 315–378.

[6] Fischer, C. and Gellersen, H., "Location and Navigation Support for Emergency Responders: A Survey," *IEEE Pervasive Computing*, Vol. 9, No. 1, Jan. 2010, pp. 38–47.

[7] Durrant-Whyte, H. and Bailey, T., "Simultaneous Localization and Mapping: Part I," *IEEE Robotics Automation Magazine*, Vol. 13, No. 2, June 2006, pp. 99–110.

[8] Bachrach, A., Prentice, S., He, R., and Roy, N., "RANGE–Robust Autonomous Navigation in GPS-Denied Environments," *Journal of Field Robotics*, Vol. 28, No. 5, Sept. 2011, pp. 644–666.

[9] Weiss, S., Scaramuzza, D., and Siegwart, R., "Monocular-SLAM–based Navigation for Autonomous Micro Helicopters in GPS-Denied Environments," *Journal of Field Robotics*, Vol. 28, No. 6, Nov. 2011, pp. 854–874.

[10] Grzonka, S., Grisetti, G., and Burgard, W., "A Fully Autonomous Indoor Quadrotor," *IEEE Transactions on Robotics*, Vol. 28, No. 1, Feb. 2012, pp. 90–100.

[11] Campoy, P., Correa, J. F., Mondragn, I., Martnez, C., Olivares, M., Mejas, L., and Artieda, J., "Computer Vision Onboard UAVs for Civilian Tasks," *Journal of Intelligent and Robotic Systems*, Vol. 54, No. 1-3, Aug. 2008, pp. 105.

[12] Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N., "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera," *Robotics Research*, edited by H. I. Christensen and O. Khatib, No. 100 in Springer Tracts in Advanced Robotics, Springer International Publishing, 2017, pp. 235–252, DOI: 10.1007/978-3-319-29363-9_14.

[13] Bonin-Font, F., Ortiz, A., and Oliver, G., "Visual Navigation for Mobile Robots: A Survey," *Journal of Intelligent and Robotic Systems*, Vol. 53, No. 3, May 2008, pp. 263.

[14] Bipin, K., Duggal, V., and Krishna, K. M., "Autonomous navigation of generic monocular quadcopter in natural environment," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 1063–1070.

[15] Shen, S., Michael, N., and Kumar, V., "Obtaining Liftoff Indoors: Autonomous Navigation in Confined Indoor Environments," Vol. 20, No. 4, 12 2013.

[16] Cesetti, A., Frontoni, E., Mancini, A., Zingaretti, P., and Longhi, S., "A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks," *Journal of Intelligent and Robotic Systems*, Vol. 57, No. 1-4, Oct. 2009, pp. 233.

[17] Izzo, D. and Croon, G. D., "Landing with Time-to-Contact and Ventral Optic Flow Estimates," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 4, 2012, pp. 1362–1367, 00022.

[18] Kendoul, F., Fantoni, I., and Nonami, K., "Optic Flow-Based Vision System for Autonomous 3D Localization and Control of Small Aerial Vehicles," *Robotics and Autonomous Systems*, Vol. 57, No. 6–7, June 2009, pp. 591–602.

[19] Hrabar, S., Sukhatme, G. S., Corke, P., Usher, K., and Roberts, J., "Combined optic-flow and stereo-based navigation of urban canyons for a UAV," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug. 2005, pp. 3309–3316.

[20] Horn, B. and Schunck, B., "Determining optical flow," *Artificial Intelligence*, Vol. 17, No. 1-3, 1981, pp. 185–203, 11476.

[21] Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*, Vol. 1, MIT press Cambridge, 1998.

[22] Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y., "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," *Advances in neural information processing systems*, Vol. 19, 2007, pp. 1.

[23] Valasek, J., Doebbler, J., Tandale, M. D., and Meade, A. J., "Improved Adaptive-Reinforcement Learning Control for Morphing Unmanned Air Vehicles," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 38, No. 4, 8 2008, pp. 1014–1020.

[24] Valasek, J., Kirkpatrick, K., May, J., and Harris, J., "Intelligent Motion Video Guidance for Unmanned Air System Ground Target Surveillance," *Journal of Aerospace Information Systems*, Vol. 13, No. 1, 2016, pp. 10–26.

[25] Bou-Ammar, H., Voos, H., and Ertel, W., "Controller Design for Quadrotor UAVs Using Reinforcement Learning," *2010 IEEE International Conference on Control Applications*, IEEE, 2010, pp. 2130–2135.

[26] Sharma, R., "Fuzzy Q Learning Based UAV Autopilot," *2014 Innovative Applications of Computational Intelligence on Power, Energy and Controls with Their Impact on Humanity (Cipech)*, 2014, pp. 29–33.

[27] Junell, J., Mannucci, T., Zhou, Y., and van Kampen, E.-J., "Self-Tuning Gains of a Quadrotor Using a Simple Model for Policy Gradient Reinforcement Learning," *Policy*, Vol. 9, 2016, p. 10.

[28] Junell, J., Van Kampen, E.-J., de Visser, C., and Chu, Q., "Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight," *AIAA Guidance, Navigation, and Control Conference*, 2015, p. 1990.

[29] Yijing, Z., Zheng, Z., Xiaoyi, Z., and Yang, L., "Q learning algorithm based UAV path learning and obstacle avoidence approach," *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 3397–3402, 00000.

[30] Zhang, B., Mao, Z., Liu, W., and Liu, J., "Geometric Reinforcement Learning for Path Planning of UAVs," *Journal of Intelligent & Robotic Systems*, Vol. 77, No. 2, Feb. 2015, pp. 391–409.

[31] Kober, J., Bagnell, J., and Peters, J., "Reinforcement Learning in Robotics: A Survey," *International Journal of Robotics Research*, Vol. 32, No. 11, 2013, pp. 1238–1274.

[32] Mannucci, T., Kampen, E. J. v., Visser, C. d., and Chu, Q., "Safe Exploration Algorithms for Reinforcement Learning Controllers," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. PP, No. 99, 2018, pp. 1–13, 00001.

[33] Bertsekas, D. P. and Castanon, D. A., "Adaptive aggregation methods for infinite horizon dynamic programming," *IEEE Transactions on Automatic Control*, Vol. 34, No. 6, June 1989, pp. 589–598.

[34] Watkins, C. J. C. H., *Learning from delayed rewards*, Ph.D. thesis, University of Cambridge England, 1989, 05115.

[35] Sutton, R. S., Precup, D., and Singh, S., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, Vol. 112, No. 1-2, Aug. 1999, pp. 181–211.

[36] Hattenberger, G., Bronz, M., and Gorraz, M., "Using the paparazzi uav system for scientific research," *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, 2014, pp. pp–247, 00012.

[37] Perry, A. R., "The flightgear flight simulator," *Proceedings of the USENIX Annual Technical Conference*, 2004.

[38] Berndt, J., "JSBSim: An Open Source Flight Dynamics Model in C++," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Guidance, Navigation, and Control and Co-located Conferences, American Institute of Aeronautics and Astronautics, Aug. 2004, DOI: 10.2514/6.2004-4923.

[39] Sieberling, S., Chu, Q. P., and Mulder, J. A., "Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 6, 2010, pp. 1732–1742.

[40] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and others, "Human-Level Control through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.

3
Background

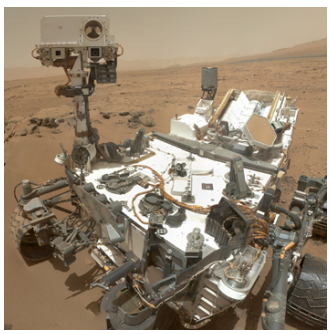
This chapter provides background on the concepts and methods used in this project. Section 3.1 defines guidance, navigation and autonomy within the context of this project. The following section explains the working principle of RL. The final section discusses some computer vision method which can be used for the autonomous guidance and navigation of UAVs. **RQ1** " What is the current state of the art for RL in UAVs and vision-based RL for robotics applications?" is answered in Section 3.2.

3.1. Autonomous Guidance and Navigation of UAVs

Within the context of this project, guidance is defined as "*the planning and decision-making functions to achieve assigned missions or goals*" and navigation is defined as "*the process of monitoring and controlling the movement of a vehicle from one place to another*" (Kendoul, 2012). Usually, guidance and navigation is performed using an Integrated Localization Systems such as GPS or through the use of constructed beacons. However, these are not always available (Salichs & Moreno, 2000; Kendoul, 2012) and for increased autonomy, a robot requires localization systems which are self-contained. This project ignores methods of navigation based on Integrated Localization Systems and focuses on more self-contained methods.

The autonomous guidance and navigation of robots has become an increasingly relevant field both in industry and academia (Kendoul, 2012). There are four major drivers for the improvement of autonomous navigation.

1. A desire for a more efficient alternative to human based navigation.
2. The need to control robots which are hard to communicate with, as in with probes sent around the solar system.
3. The replacement of mundane or dangerous tasks such as vacuuming or exploring burning buildings.
4. The limits of human controllers. For example, using an individual human controller to control a swarm of UAVs will in most cases be worse off than optimized swarm control algorithms.



(a) The Curiosity rover on Mars

(b) The Roomba vacuuming a floor

(c) The Kiva robot moving goods

Figure 3.1: Some autonomous robots[1]

A formalism for the problem of autonomous guidance and navigation of robots in unknown environments has been proposed by Salichs and Moreno (2000). They divide the problem into following five subproblems: Motion Control problem, World Modeling problem, Localization problem, Planning problem and Architecture problem. This project intends to provide a solution to the World Modeling, Localization and Planning problems by introducing a vision-based RL control system. An alternate means to model the world is proposed by the use of a vision-based state which the quadcopter will use to obtain a sense of where it is (i.e. localize itself); the planning problem will be solved by the learning of the task.

Kendoul (2012) classifies navigation systems in the three following categories: systems which are based on conventional IMU/GPS systems, vision-based systems and range sensor based systems. GPS based systems are outside the scope of this project; purely IMU based navigation systems (i.e., odometry) are ignored because of their inherent bias. Simultaneous Localization and Mapping (SLAM) (Durrant-Whyte & Bailey, 2006) uses estimates of the distance to elements in the environment and/or vision information for operation. Vision systems mainly focus on visual information to make judgments about their state and the environment. The rest of this chapter discusses SLAM and vision-based methods of autonomy.

SLAM (Durrant-Whyte & Bailey, 2006) is currently the state of the art for environmental modeling and mapping. The technology can be used to map a robot's environment and localize itself at the same time. It has been used for UAVs without a distance sensors (Weiss, Scaramuzza, & Siegwart, 2011; Kim & Sukkarieh, 2004), although most applications rely on the distance information (usually from a laser rangefinder) such as in (Bachrach et al., 2011; Grzonka et al., 2012). SLAM can be problematic for real time applications due to it's computational and memory requirements. This problem is mitigated by the use of alternate versions of SLAM which require less computation and memory.

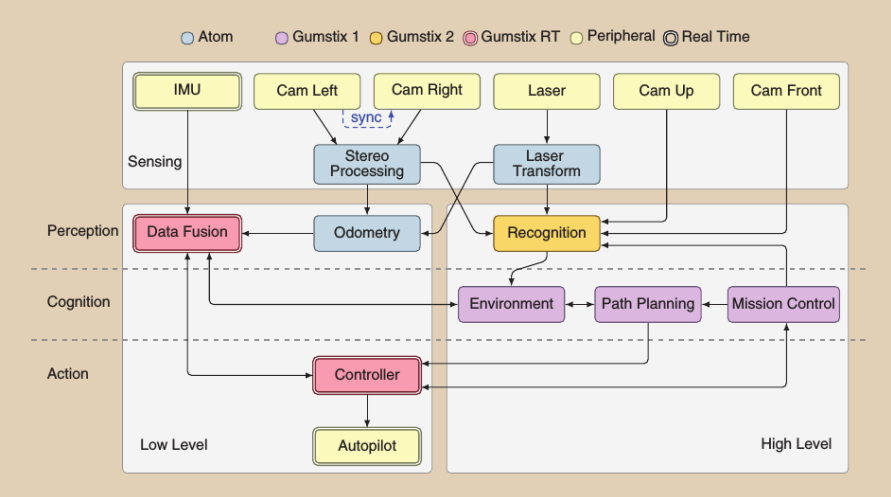


Figure 3.2: Architecture of the GNC system from (Tomic et al., 2012)

Vision contains a lot of information which can be used for the autonomous guidance and navigation of robots(Bonin-Font et al., 2008). Optic flow has been used in (Kendoul, Fantoni, & Nonami, 2009) to control the position and velocity of a quadcopter. Recognizing objects and estimating distances to them can be used as a landmark based means of navigation. Tomic et al. (2012) came up with a fully on-board implementation which uses optic flow to aid the odometry and object recognition for vision-based navigation. An architecture of their multi-layered guidance navigation and control scheme is visualized in Figure 3.2. An application of vision-based guidance using supervised learning has been described in (Bipin, Duggal, & Madhava, 2015). They use supervised learning to estimate the distance to objects from labeled images. The knowledge of the distance to objects is than used to plan an obstacle free path.

[1] `http://mars.nasa.gov/multimedia/images/?ImageID=4845&s=2`; Last accessed: 25/01/2017
`https://www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx`; Last accessed: 25/01/2017
`http://www.marketwatch.com/story/amazon-deploys-fleet-of-robotic-elves-2014-12-01`; Last accessed: 25/01/2017

3.2. Reinforcement Learning

A brief introduction to RL is provided in Chapter 2; this section presents a more detailed description of RL. RL is based on models of animal learning; numerous similarities have been found between the mechanism of animal learning and RL (Wörgötter & Porr, 2005).

As an example, an experience in the life of a hippopotamus is first described using natural language and than using RL terminology. The situation consists of a hippopotamus who wants to go from being on the shore to being inside water. She had experiences of pleasure when she went into the water on a hot day, so she knows that getting into the water will make her feel good. Equivalently, we can say, she (the **agent**) wants to **act** in her **environment** to change her **state** because she wants to increase the amount of total **reward** she can accumulate. Here the reward is proportional to the relative comfort she enjoys.

In RL the agent is a software entity which carries out a desired task. The agent can be a robot that acts in a physical environment or a software acting in a digital environment. The agent takes **actions**, and like the hippopotamus, learns desired or undesired **actions** based on a scalar of feedback. In RL, the feedback of the "desirability" of an action is called the **reward**. The goal of the agent is to maximize the sum of reward it receives over its lifetime.

The next section defines the framework of RL. Following this, the elements used to define the RL problem are explained. The next two parts deal with explaining the representation of the delayed reward in RL and using the perception of the agent and the reward information to learn ways of taking actions that maximize the long term sum of rewards. Finally the state of the art in RL and its application to robotics are discussed, answering **RQ1**.

3.2.1. Framework of RL

The RL problem consists of an agent that acts in an environment in order to accomplish a defined goal.

Agent The agent is the entity that acts in the environment and tries to maximize its long-term sum of rewards. Firstly, it needs to be able to manipulate the environment which is often manipulating itself within the environment. Secondly, it needs a way to perceive the effects of its manipulations. Thirdly, it needs to use the obtained reward to approximate the expected sum of rewards that can be obtained from the different states.

Environment The environment is the space where the agent lives, acts and performs its task. The environment is assumed to contain everything pertinent to the problem. The agent picks an action. The environment transitions the agent to the next state based on its current state, the action it took and the transition probabilities. It also gives the agent the reward based on the transition.

3.2.2. Representation of the RL Problem

The agent needs to take actions and change its state to accomplish a predefined goal. The state transitions are based on the transition model and each state transition is accompanied with a scalar reward. The reward is used by the agent to learn better actions. The *Markov Decision Process* is used to formally describe a generic RL problem.

State (s) The state is a representation of the information required to perform the RL task. It is a group of numbers[2] which encodes information from the environment, that is used to make a decisions. For an autonomous car agent, the state can be represented by the kinematic quantities of the car. For an agent that controls investments, it can be the current value of investments, the available capital, current projected profits etc.

In order to satisfy the Markov property, the state should contain enough information to be able to determine the probability distribution of consequent possible states given the current state and a chosen action.

Action (a) The agent takes actions to perform the task. It is the goal of the agent to discover "good" actions by exploring the state and action space.

Transition model This is a probabilistic model that maps the probability of ending up in state s' given the agent is at s and takes action a. It is expressed by $\mathcal{P}^a_{ss'}$.

Reward (r) The reward is designed such that maximizing its sum leads to the best performance of the task. The **reward model** is represented by $\mathcal{R}^a_{ss'}$ which expresses the expected reward at the next time step given action a, current state s and expected next state s'[3].

Markov decision process A Markov decision process (MDP) consists of a set of states, actions, the transition model, the reward model and a discount factor (these terms are described in Sections 3.2.2 and 3.2.3). A MDP must satisfy the Markov property which states that the transition probability of the environment is fully defined by the current state and action. That is, the probability distribution over the next possible states depends only the current state and action. Temporal difference based RL methods (see Section 3.2.5) assumes an underlying MDP when estimating values.

3.2.3. Representation of Delayed Rewards

A problem of learning optimal ways of acting is the fact that actions taken now influences the possibility of future (unknown) rewards that can be accumulated. This section introduces returns, policies, values and action-values, ideas which are used to represent the total reward the agent can accumulate from its current situation.

Return (R_t) The sum of the reward an agent receives over its lifetime is called the return. The agent tries to select actions that lead to the highest return. The RL algorithms describe ways to estimate the return from each state. If the return from each state can be estimated, best actions can be selected by the selection of actions that lead to states of higher return.

There are three main types of return modeling (Kaelbling, Littman, & Moore, 1996). The flat reward model is used for finite-horizon cases and is expressed by the following expression:

$$R_t = \sum_{t=0}^{h} r_t$$

The discounted reward scheme is used for infinite-horizon cases. Here a discount factor, γ ($|\gamma| < 1$), is used to make the infinite sum of the reward bounded.

$$R_t = \sum_{t=0}^{\infty} \gamma^t r_t$$

[2] It is theoretically possible for the state to be non-numeric. However, almost always its representation in an RL problem is numeric for two reasons: (1) dynamics are already described using numbers (2) computers only understand numbers

[3] Note this can also be read in a way where s' is the current state, a and s were the previous states and actions respectively, while $\mathcal{R}^a_{ss'}$ is the reward obtained from the previous transition

The average reward scheme estimates the long-term return by averaging over the long-run rewards.

$$R_t = \lim_{h\to\infty} \frac{1}{h} \sum_{t=0}^{h} r_t$$

Policy (π) The agent uses the policy to decide what it should do. Mathematically, it is a mapping from state to action. It can be stochastic, deterministic or a combination of both. $\pi(s,a)$ represents the probability of selecting action a given state s following policy π. The designer has to select a policy that trades off the amount the agent explores the environment against how much obtained knowledge is exploited to maximize rewards, a dilemma called the "exploration-exploitation" problem in RL.

A way to explore the environment is to randomly pick different actions out of the ones available. But if an agent always behaves randomly it is almost always non-optimal. The ϵ-greedy policy takes a greedy action [4] with probability ϵ and a random action with probability $1-\epsilon$. Increasing the value of ϵ is an approach to tackle the exploration-exploitation problem.

Value ($V(s)$) The expected return of a state is its value. The value function ($V^\pi(s)$) represents the expected return given current state is s and the agent follows policy π from s onwards.

$$V^\pi(s) = E_\pi \{R_t | s_t = s\}$$

The techniques of RL deal with estimating V and the corresponding policy (or policies) (π) that lead to maximum returns. The optimal value function is represented by V^*.

Note that one still requires a transition model to pick optimal actions after estimating the values. Although the agent knows the value of each state, it still needs to have a way to find which actions lead to states of higher values.

Action-value (Q) function Action-value functions represent the value of state-action pairs instead of just states. These functions represent the expected return given an initial state s and an action a. Including the action into the estimation of the return gives the agent information of the values associated with each of the actions. This allows it to directly pick actions which maximuze the long-term sum of rewards. In small and discrete cases one can make a table with the states along the rows and the actions along the column. With such a table, acting optimally becomes a matter of picking the column (i.e. the action) with the maximum value for the current row (i.e., state).

$$V^\pi(s,a) = E_\pi \{R_t | s_t = s, a_t = a\}$$

3.2.4. Learning Optimal Actions

The values of states are estimated using the Bellman equations. RL methods use the agents experiences (samples) to learn better actions. The idea of progressively improving the estimate of the value by using previous estimates (bootstrapping) is important for temporal difference based RL methods.

Bellman equations The *Bellman Expectation Equation* (Equation 3.1) expresses the value of a state before an action, in terms of the reward received after the state transition and the value of the next state. This equation can be used to calculate the expected value of states under a policy π. The state values can than be used with the model to learn optimal actions.

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \tag{3.1}$$

The innermost term, $\mathcal{R}^a_{ss'} + \gamma V^\pi(s')$, represents an estimate of the value of the initial state s given the agent took action a and ended up in state s'. A weighted sum of the values relating to the possible states s' that a may result in, is performed in the next layer. The weight is the transition probability $\mathcal{P}^a_{ss'}$. This accounts for the stochasticity of ending up in different states after taking action a. The last

[4] Actions which maximize the return based on the current knowledge of the problem are called greedy actions.

layer is a weighted sum of the of the probability of picking different actions a, given the current state s. The *Bellman Optimality Equation*, as shown in Equation 3.2, is used to find the optimal value function.

$$V^*(s) = \max_{a \epsilon \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^*(s')] \tag{3.2}$$

The corresponding action-value functions for Equations 3.1 and 3.2 are Equations 3.3 and 3.4 respectively.

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a')] \tag{3.3}$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')] \tag{3.4}$$

Sampling Sampling is the idea that the state (or state-action) values can be estimated by taking samples of experiences from the RL problem in question. The Monte Carlo and the Temporal difference methods use sampling for value estimation.

In large multi-dimensional state spaces one is faced with the problem of dealing with an exponentially growing number of states with increasing dimensions of the state space. This cannot be managed by Dynamic Programming (DP) (see Section 3.2.5) methods since they find the best actions by searching through all possible trajectorie the agent may take.

Sampling sidesteps this problem by storing experiences (i.e. states, actions and rewards) which are later used to obtain multiple realizations of the return from a state. The state value can be estimated by averaging the stored returns from a state.

Bootstraping Bootstrapping is the idea of updating an estimate based on an older estimate(Sutton & Barto, 1998). Dynamic programming methods use this during policy and value iterations. Namely, in order to improve the current estimate of the value of a state($V^\pi_k(s)$), its previous estimate ($V^\pi_{k-1}(s)$) is used.

3.2.5. Solution Methods

In this section some of the basic solution methods of RL are described. These explanations are summarized accounts of the explanations from (Sutton & Barto, 1998) and from the lectures on RL by David Silver that are available on the internet [5].

Dynamic programming (DP) DP methods iteratively estimate the state values for a policy (**policy evaluation**), and improve the policy based on new estimates of the value (**policy improvement**). They are model based methods which can be used to find optimal policies for known MDPs.

The *Bellman Estimation Equation* (Equation 3.1) is used to estimate the value of the states. Correspondingly, greedy actions[6] are selected to improve the policy. Some generic DP methods are described below:

Policy iteration (PI) A step of policy evaluation followed by a step policy improvement.

Value iteration (VI) One need not wait for the convergence of the value in policy evaluation. One iteration of policy evaluation is followed by policy improvement. Its mathematical form is obtained by converting the *Bellman Optimality Equation* (Equation 3.2) into an iterative form as shown in Equation 3.5.

$$V_{k+1}(s) = \max_{a \epsilon \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')] \tag{3.5}$$

Asynchornous DP Heuristics can be used to select states whose values and actions should be updated based on defined criteria. Such heuristics can speed up DP by bypassing redundant calculations.

[5] UCL course on RL (COMPM050/COMPGI13) by David Silver (2015); Course website: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`; Lecture videos: `https://www.youtube.com/playlist?list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT`; Last Accessed: 12-04-2017

[6] Actions that lead to states of highest value based on its current estimate.

Monte Carlo (MC) methods In MC methods the agent acts in the environment to obtain samples of the MDP describing the RL problem. The agent stores the state, action and reward while it performs a task and the at the end, uses the stored experience to calculate the return from each state visited in that episode[7]. As it experiences more episodes in the environment it gets more measurements of the return from each state. These measurements can be averaged to obtain a statistical estimate of the value of the states under a specified policy.

In contrast to DP, MC samples experiences from the environment and does not bootstrap. Not bootstrapping makes MC estimates of the value bias free, as the sampled experiences are averaged and no underlying MDP is assumed. However, this causes the estimate of the value to fluctuate leading to a higher variance. As MC methods do not assume an underlying MDP, they suffer less when the state is non-Markov.

Temporal difference (TD) learning In TD methods the agent acts in the environment to sample the value of state-action pairs, while also using a previous estimate to make new estimates (bootstrapping). The algorithm initializes with a policy that allows sufficient exploration and an arbitrary action-value function. The agent takes action a transitioning from s to s'. This transition is accompanied with the reward $\mathcal{R}^a_{ss'}$ (or r in Equation 3.1). The value of the initial state , $V(s)$, is updated towards its new estimate after taking action a, using Equation 3.6.

The reward is being sampled and the previous estimate of the value ($V_k(s)$) is being bootstrapped to obtain the new estimate (i.e. $V_{k+1}(s)$). The α in Equation 3.6 is the learning rate and it determines how much the previous estimate of the value is moved towards the new estimate.

$$V_{k+1}(s) = V_k(s) + \alpha[r + \gamma V(s') - V_k(s)] \tag{3.6}$$

The learning is slow in TD as only one state is being updated at each step. The idea that all the previous states the agent was in has something to do with the current reward it receives is incorporated into TD learning using the idea of "Eligibility traces". At each step, the value of all previous states are updated by a factor called their eligibility. An array of eligibility traces is maintained for all available state-action pairs. This eligibility is decayed each episode for all state-actions except for the current one, whose eligibility is increased (see Figure 3.3). In a way this assigns updates to states based on an idea of how much that state is responsible for the current rewards; i.e., recent states are updated more than states visited long time ago.

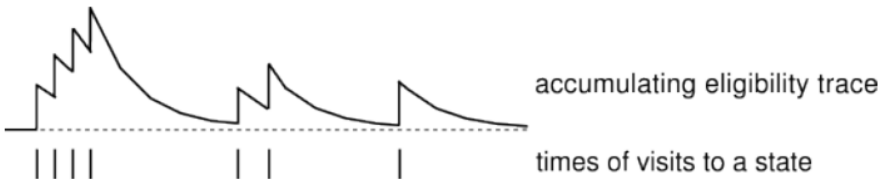


Figure 3.3: The *eligibility* of a state, decays with time unless it is visited, in which case its eligibility is incremented (Sutton & Barto, 1998).

If an optimal policy is desired without model knowledge, one must use the action-value function. This extension results in the TD algorithm called SARSA (State Action Rewards nextState nextAction) for value estimation and Q-learning for optimal value estimation. The SARSA the Q-learning updates are given by Equations 3.7 and Equation 3.8.

Bootstrapping has two consequences in TD learning in comparison to MC learning. Firstly, the state in the problem needs to be "more Markov" for TD learning than it needs to be for MC learning as bootstrapping works based on the assumption of an underlying MDP. Secondly, the value estimations are biased and have lower variance than in MC methods.

[7] In RL, performing a task (i.e. going from the intial to the terminal state) is called an episode. Non-episodic taks do not have an episode.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{3.7}$$

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)] \tag{3.8}$$

3.2.6. State of the Art

Some of the challenges of applying RL to robots are its inability to deal with multidimensional state spaces, partial observability, noise, safe exploration, model inaccuracies and difficulties in reward definition (Kober et al., 2013); these challenges are discussed in this section. As the use of vision-based RL for the control of UAVs has been discussed in the article (Chapter 2), some applications of vision-based RL to land robots are mentioned.

The problem of large state spaces can be made tractable by using function approximators to describe the state space. Xu, Zhao and Huang (2014) reviewed the use of function approximators in policy search, value function and actor-critic methods. They find many applications of value-based RL methods and claim that their popularity is on account of their simplicity.

RL cannot be used to solve problems of large state space as the number of computations required to come up with solutions scales exponentially with the dimensions of the state and action spaces. a phenomena called the "Curse of Dimensionality"(Sutton & Barto, 1998). A way to deal with the "Curse of Dimensionality" is to divide the task into subtasks using Hierarchical Reinforcement Learning (HRL). The three main approaches to it in HRL are through a relative description of the state, division of the state space into subdomains and division of the action space into subdomains(Xu et al., 2014).

When the state is not fully observable, the problem can be framed as a Partially Observable MDP (POMDP) (Wiering & Van Otterlo, 2012, pg. 387). Hutter (2009) proposes a different framework for the POMDP, called ΦMDP. In ΦMDP a mapping of the past observation-reward-action history to a state is used to convert a POMDP into a MDP.

A method called Safety Handling Exploration with Risk Perception Algorithm (SHERPA) has been proposed by Manucci et al. (2018). In the algorithm, the state space is divided into Fatal State Space (FSS) and Safe State Space (SSS). SHERPA assumes the agent can perceive risk and that it starts from a known SSS. It extends the SSS as it explores more of the environment. While exploring the environment it always keeps a backup of a sequence of control action that transitions it to a state in the SSS. When it perceives risk, it uses the backup to return to a safe state.

The reward for the agent can be defined based on demonstrations of good (or ideal) actions to the agent, using Inverse RL (IRL). IRL is a type of RL where a value function is extracted from expert demonstrations and optimal policies found based on the extracted value function. Abbeel, Coates, Quigley and Ng (2007) demonstrated numerous aerobatic maneuvers with helicopter in autonomous flight, using Inverse RL.

In (Mnih et al., 2015) a deep Convolutional Neural network (CNN) is used to train an RL agent to play various Atari 2600 games based on pixel data from the screen and score information as a reward signal. The agent manages to learn to play a diverse set of Atari 2600 games without any modifications to the CNN configuration and outperform human testers for more than half of the games they tried it on.

Vision based RL is used to control a land robot by Gaskett, Fletcher and Zelinsky (2000). They use a Neural network based, continuous state-action implementation of RL. The camera view is gridded and correlation between pixel values of the grids is used as state information. Vision-based RL schemes that use supervised learning to interpret visual information and use the information for tasks learned using RL are presented in (Michels, Saxena, & Ng, 2005; Cicirelli, D'Orazio, & Distante, 2005; Choi, Lee, & Won, 2011). Michels et al. (2005) uses the vision information to teach a remote control car obstacle avoidance, Cicirelli et al. (2005) uses it to learn how to approach a door and Choi et al. (2011) uses it to learn line tracking and obstacle avoidance using a land robot.

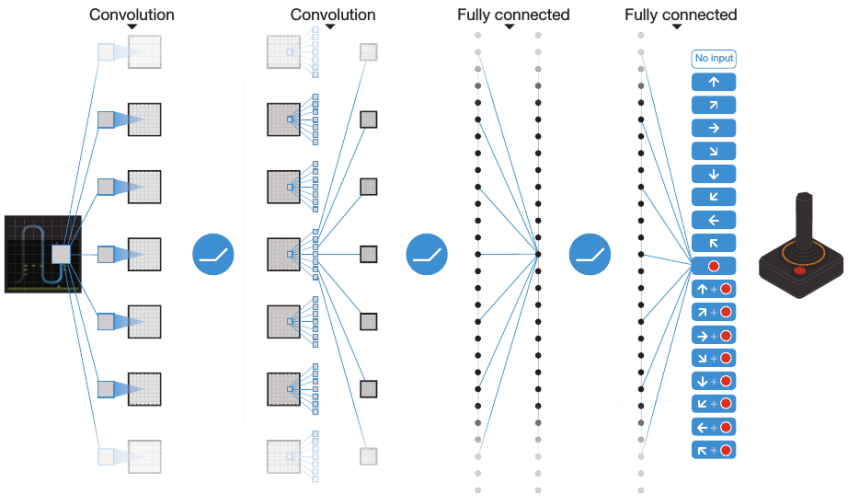


Figure 3.4: A schematic representation of the NN used to train the RL agent in(Mnih et al., 2015).

3.3. Computer Vision

In this section, prospective Computer Vision (CV) algorithms that maybe relevant to this research are explored. CV is in itself a very broad topic. Since the primary aim of this project is contributing to the autonomy of UAVs, a detailed study of computer vision is not performed. Rather some of the basic techniques which are considered relevant for this project have been outlined. Pixel-based methods, distance estimation techniques, object recognition, optic flow and vision-based guidance and navigation are reviewed.

Pixel-based methods In this study pixel-based methods are used to refer to CV methods which directly use the pixel value to interpret higher-level information from the image. Numerous examples of such systems are provided in (Jimenez, Ceres, & Pons, 2000). Due to the simplicity of pixel based methods of detection, such a technique can be implemented easily. This makes them a very good method for this project.

Distance estimation One of the techniques to estimate distance in CV is by using two photographs of the same scene from different positions and/or orientations. The position information of the photographing points along with the two photographs can be used to estimate the distance of points in the real world from the observation point. Points in one image are matched to the other image and the pixel distance between them is used to estimate the real life distance of the point.

Object recognition The Scale Invariant Feature Transform (SIFT) (Lowe, 1999) and Speeded Up Robust Features (Bay, Ess, Tuytelaars, & Van Gool, 2008) are reviewed in order to look at techniques that use generalized features for object detection. The SIFT algorithm describes points (using the SIFT descriptor) in images which can be matched to rotated and scaled instance of that point in another image. This makes SIFT suitable for detecting possible landmarks. SURF is another algorithm that uses the working principle of SIFT but trades off robustness for computational efficiency. The upright variant of SURF (U-SURF, (Bay et al., 2008)) maybe of specific interest to this project as it opens up the possibility of using simple designed landmarks which can aid guidance and navigation. Examples of a designed landmark can be a printed barcode patterns as in (Lin & Chen, 2011; Briggs, Scharstein, Braziunas, Dima, & Wall, 2000).

Viola and Jones (2004) came up with a fast method for face detection. They calculated *integral images* based on the input image (Figure 3.5) and defined feature windows (Figure 3.6) which are used to filter out regions of an image that are not faces.

The current state of the art for object recognition is convolutional Neural-network based feature extraction and deep-learning for training the networks.

Optic Flow Optic flow measures the change in pixel location from frame to frame in order to come up with an estimate of the motion of the camera and/or the external environment. The inability to distinguish between the nature of the origin of the motion is often solved by defining a boundary conditions.

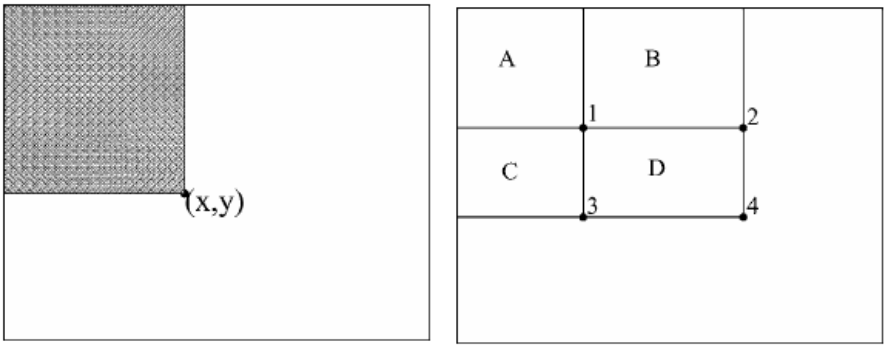


Figure 3.5: The left image shows how each pixel in the integral image takes the sum of a region of pixels. The right image shows how having the integral image allows the calculations of the intensity of region D by accessing four memory locations. The sum of intensity of the D region is equivalent to $I_4 - (I_2 + I_3) + I_1$ where I_i represents the value at the specific pixel in the integral image. (Viola & Jones, 2004).

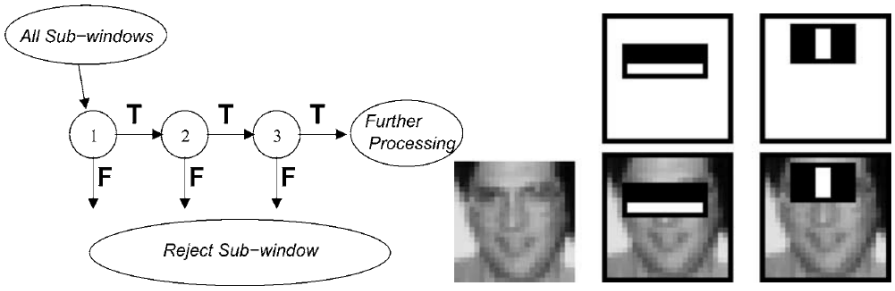


Figure 3.6: The left figure visualizes the elimination of sub-windows from the image and the right figure shows the feature windows used for the filtering in (Viola & Jones, 2004).

Optic flow can be used to estimate the linear and angular velocities of a vehicle. These velocities can be integrated in time to obtain estimates of the vehicles position, which is called Visual odometry. It can also be used for obstacle avoidance and tracking. The pattern of optic flow that is created when moving inside different environments (e.g. corridors, rooms, stairs, fields etc.) may be usable to define a state which is able to distinguish those environments.

Simultaneous Localization and Mapping: Namely graph based SLAM (Grisetti, Kummerle, Stachniss, & Burgard, 2010) and the Parallel Tracking and Mapping (PTAM) (Klein & Murray, 2007) are considered interesting for this study. The graph based approach is appealing for this study as it has lower computation and memory requirements than dense SLAM implementations.

In (Weiss et al., 2011) a vision-based SLAM for the navigation of a quadcopter is developed. Weiss et al. (2011) uses a monocular camera and off-board SLAM to map and navigate an environment. Consequently (Shen et al., 2013) presented a fully onboard system that use an IMU and a laser scanner directed in the up-down direction for pose estimation. The pose information is combined with vision information and passed to a simplified SLAM algorithm for onboard mapping. Figure 3.7 shows some of the 3D maps generated by (Shen et al., 2013).

Vision-based navigation Although vision-based navigation is not a CV method, it is mentioned here due to its relevance for the goal of this project. Vision information can be encoded for guidance and navigation purposes in many different ways(Bonin-Font et al., 2008). For example the processed pixel values, object detection values or situational awareness values can be binned into a histogram to represent the general state of the robot. The raw image can be used as an identifier for a location. Landmarks can be placed in the robots environment or it can be given the ability to identify unique elements in its environment as landmarks dynamically. The interpreted information can be used to generate direct motor commands or plan paths.

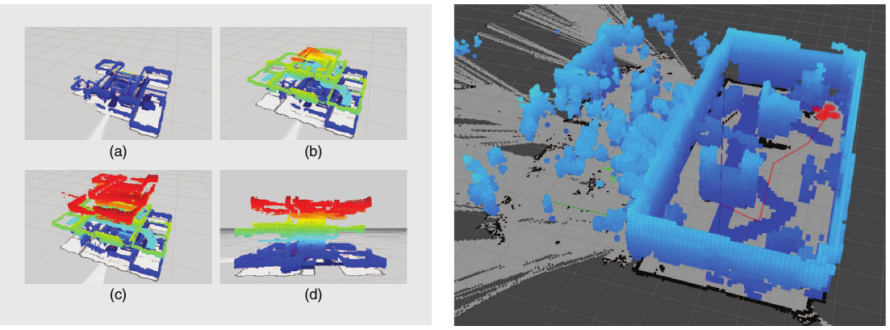


Figure 3.7: The left and right images show the 3D maps of a building generated by (Shen et al., 2013). The left figures shows different views of a three-story building that was mapped. The right figures shows the UAV mapping a room.

3.4. Summary

This chapter discussed some background required for the work described in this report. First, a discussion about the autonomous flight of UAV was presented. After this, the RL problem and some methods of learning *values* in the RL problem was discussed. Through a review of current RL methods in Section 3.2.6 **RQ1** "What is the current state of the art in RL in UAVs and vision-based RL for robotics applications?", has been answered. Lastly, some CV methods are reviewed; the knowledge from this review will be used to come up with proposals for a vision-based state to answer **RQ2.1** in Section 5.1. Before coming up with concepts for the vision-based RL agent to be designed in this project, a simplified gridworld study is carried out in Chapter 4.

4

Gridworld Simulation Study

A simpler simulation of a vision-based guidance and navigation task is created in order to get more insight into the problem. The nature of the simplified vision-based state and the influence of RL parameters on the performance of the algorithm is studied. The first part of this chapter provides a description of the simulation task and the implementation of the simulation tool. Than the findings from the simulations are discussed.

4.1. Simulated Reinforcement Learning Task

A task is defined where an agent has to learn optimal ways to travel to two goals in a gridworld. This task is performed with absolute position information and simulated vision information. The vision information is simulated by defining a state which consists of information about the contents of the gridworld in front of the agent. Details about the task, the environment and the learning are provided below.

Guidance Task The task is to find an optimal route between two static goals (grids) in a gridworld while visiting each of them at least once. Here, optimal means using the least number of actions.

Gridworld A 9 by 9 gridworld is used for most of the simulations. Only the study into the effect of environment size uses 3 other dimensions. Namely the performance in a 20 by 20, a 40 by 40 and a 60 by 60 gridworlds are compared. Note that none of the goal locations are changed for the bigger gridworlds.

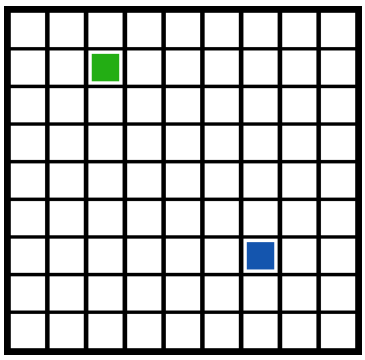


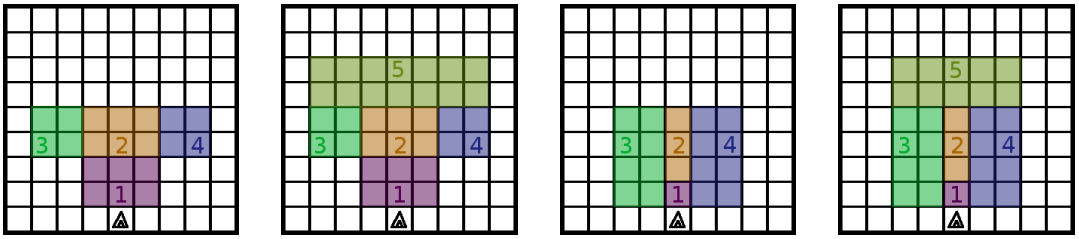
Figure 4.1: The 9 by 9 gridworld used in the simulations. The green square is goal 1 and the blue square is goal 2.

Goals There are two goals in the gridworld. The first goal is at the 2nd row and 3rd column [1]. The second goal at the 7th row and 7th column [2]. Each of the goals are assigned a "vision signature". Which lets the agent distinguish between objects when using vision-based information. Further elaboration about vision-based states and the vision signatures are presented in the following paragraph.

Vision-based States These are defined by creating numerous grids in front of the agent to simulate its line of sight and assigning values to those grids based on their contents in the gridworld. This loosely reflects

[1] i.e., (2,3) using a (row,column) convention for co-ordinates in the gridworld and starting the count from 1
[2] i.e., (7,7)

gridding the vision stream on a real life agent and assigning values to the grids based on their contents. The exact configuration of two vision states used in the simulations are visualized in Figure 4.2.



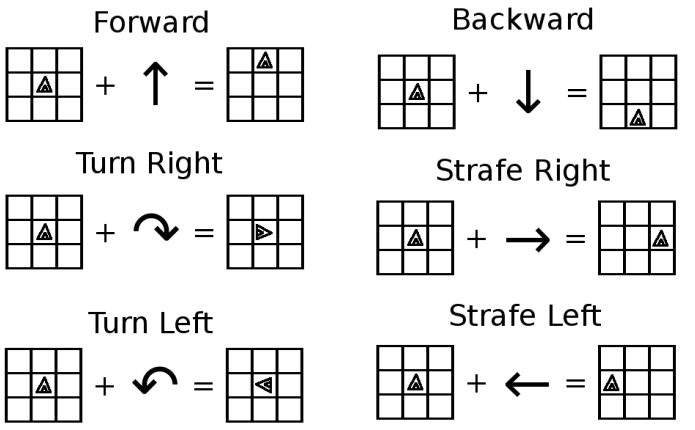
(a) *vis0* (b) *vis01* (c) *vis1* (d) *vis11*

Figure 4.2: The vision grids used in the simulations. The reward for having a goal in the grids labeled 1 through 5 are 0.75, 0.5, 0.25, 0.25 and 0.1 respectively.

The grids take the value of the object inside of them. The set value for objects of interest in the gridworld are defined as the "vision signature" of the object in this report. In general, the vision grid takes a value of 0 if nothing is in it, a value of 1 if the base grid is in it, a value of 2 if goal 1 is in it and a value of 3 if goal 2 is in it. For some of the simulations other recognizable non-goal objects are put into the gridworld to study the effect of more detectable features. The extra objects have vision signature 4 and 5. The conflict between seeing two goals in one vision-cell is resolved by always prioritizing goal 1 i.e., selecting goal 1 in case both goals 1 and 2 are seen in one vision-cell. Lastly, some simulations test the effects of incorporating wall detection. If a vision-cell is fully outside of the gridworld, it is labeled as a wall cell and given a vision signature of 8.

For most of the experiments the only visually recognizable elements in the gridworld are goals 1 and 2; even for simulations where the agent only has to go to goal 1. For numerous other experiments the agent is given the ability to detect walls. Extra identifiable objects (i.e., ones with vision signatures) are added for simulation where the effect of having more features are studied.

Actions The baseline set of actions consist of turn left, turn right and go forward. The turning action is considered a requirement for the simulation as there is need for the agent to be able to change its heading so that it can see different things around it. An extended set of action is defined which extends the baseline set by adding 3 more actions. The extended set adds the actions go backward, strafe left and strafe right. The actions are visualized in Figure 4.3.



(a) Baseline set of actions (b) Additions in the extended set

Figure 4.3: The two action sets in the gridworld simulations.

State representation Two types of states are used in these experiment: absolute states and vision-based states. Both of the state descriptions are appended with the compass heading of the agent. As the agent can only see what is infront of it, the heading is a relevant piece of information for performing

the guidance task. Further, the states are appended with information about goals the agent already visited as that information is required for the task.

The absolute state space is described using a 5 character long string. The first character is a letter representing the compass heading. The next two characters represent the row location, and the next two the column location. The vision-based state is also represented as a string, but its length depends on the number of vision grids. Numerous vision grids are used in the simulations, all with 4 or more grids. The first character in the vision-state encodes the compass heading of the agent. The following 4 or more numbers represents the contents of the vision grids.

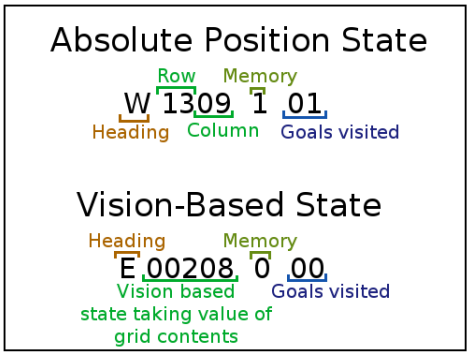


Figure 4.4: The two types of states used in the simulations

When the task consists of going to one goal, just the absolute position or the vision grid information is sufficient. However, for the task of travelling to two goals, the state needs to be appended with information about goals visited. Hence, for the task of optimally going to the two different goals, the state is appended with a binary number (represented as a string). A memory state is defined, but this is not used for anything in the simulations[3]. The goals visited part of the state is used to define the terminal state. The state is represented by an array of three objects: a string consisting of the absolute or the vision-based state, an integer representing the memory state, and a string of a binary number representing which of the two goals have been visited.

Policies Two types of policies are tried out. The random policy picks an action in random and the ϵ-greedy policy picks a greedy action with probability ϵ(i.e., higher ϵ is used to signify higher greediness). The simulations are carried out with a random policy to benchmark the ϵ-greedy policies. The absolute and vision-based state simulations are performed using an ϵ-greedy policy to study the effects of the type of state on the performance of RL.

Rewards Two reward schemes are defined. The absolute position based reward scheme awards the agent a reward of 1 when it is on the cell of an unvisited goal. The vision-based reward scheme, rewards the agent when an unvisited goal is in one of the vision grids, or when it is on top of an unvisited goal. The reward for reaching a goal is set to 1 and the reward for seeing a goal is based on the vision-cell the agent sees the goal in. If it sees multiple goals in its grids (i.e., seeing goal 1 in cell 1 and goal 2 in cell 3) the rewards from the grids are summed. The reward for seeing the goals in different grids are 0.75, 0.5, 0.25, 0.25 and 0.1 for each of the different grids. The grids are organized such that the ones closer to the agent have higher rewards for having the goal in them. Both the reward schemes have a penalty of -1 for hitting a wall and of -0.1 for every action. Penalizing every action causes the agent to minimize the number of steps.

State-value functions The state-value function is in the form of a Q-table. The row indices are the states and the column headings are the different actions. Each cell in the table represents the Q-value for taking the action on the column while being in the state on the row. The values are initialized at 0. As all actions have a penalty, such an initialization encourages the exploration of unvisited states.

[3]It was included to perform tasks as described in (Junell, Van Kampen, de Visser, & Chu, 2015)

Transitions The state transitions are deterministic. The agent can take the turning actions at all states. The translational actions (i.e., forwards, backwards, strafe left and strafe right) are restricted when it causes collision with the wall.

Terminal conditions The episodes are terminated either when a fixed number of maximum steps[4] are reached, or when the agent visits both goals.

4.2. Findings

The gridworld simulations and their findings are described in this section.

4.2.1. Learning with Vision-Based States

The agent is simulated with the absolute position based state and the "*vis0*" gridded vision-based state in a 9 by 9 gridworld. The agent has to travel to the two defined goals. The task is also performed with fully random actions for comparison with the RL agents. The learning is carried out using a constant ϵ of 0.6, an α of 0.3 and a γ of 0.95. Both simulations are run for 500 episodes.

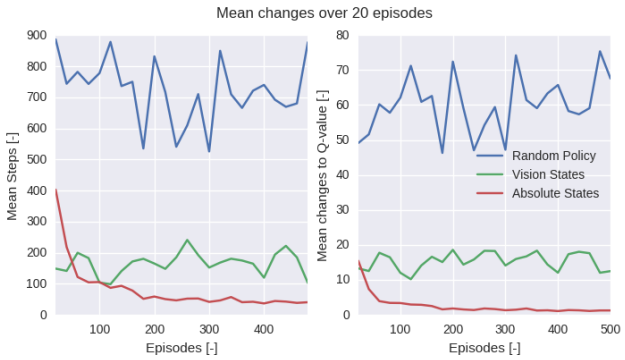


Figure 4.5: Mean of the steps and changes to Q-values over 20 episodes for the whole run.

Figure 4.5 shows the steps in each episode over 500 episodes for a random policy, absolute position states and vision-based states. From the figure we can conclude that vision-based state information as an effect on the performance as the agent performs better than with the random policy. However, there is no learning with the vision-based states. In contrast, the agent using absolute states learns to perform the task better as it gains more experience.

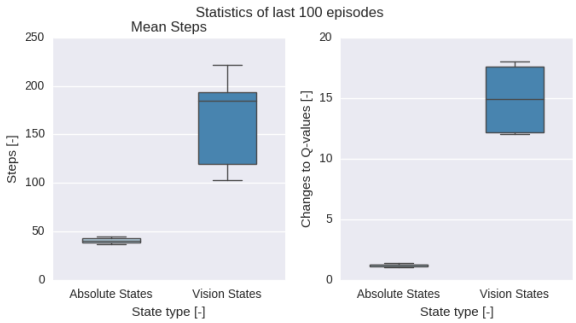


Figure 4.6: Statistics from the last 100 episodes for the absolute position and the vision-based state.

The box plots (Figure 4.6) show the performance while using absolute position based and vision-based states for the last 100 episodes. They show that the vision-based learning makes more changes to its value function compared to the absolute position based learning. This is on account of the ambiguity in the vision state information.

The agent cannot distinguish its real life absolute position based on the vision information. Many absolute grid locations (cells) have the same vision state as shown in Figure 4.7. Since the return (sum of the

[4] 5000 steps for most simulations

cumulative reward) from the different cells is different, but the agent is unable to distinguish between them, the estimated value of the vision state which maps to many real life states keeps fluctuating around a locally optimal policy as shown in Figures 4.8c and 4.8d.

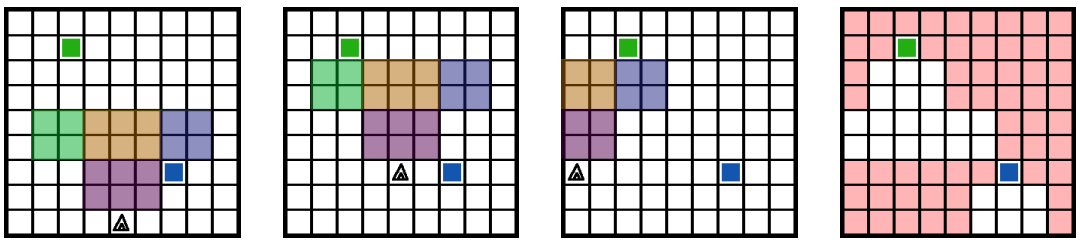
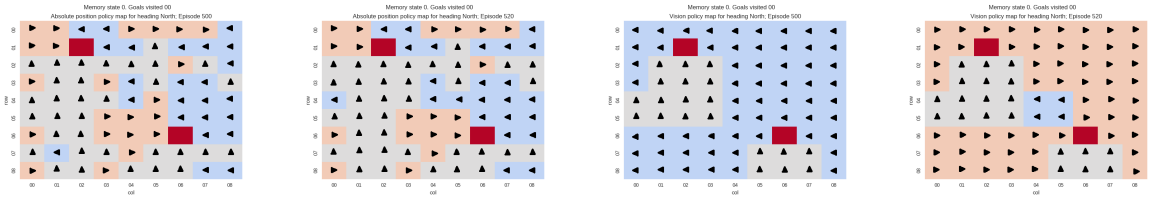


Figure 4.7: The figures shows all the cells where the vision-based states (for a "*vis0*" gridding) in the North heading maps to the value "N0000". The three figures on the left show example situations in those locations with the agent and its vision grid superimposed. The right most figure shows all the cells where the vision-based state is "N0000" by tinting them in a red hue.



(a) Absolute; episode 500 (b) Absolute; episode 520 (c) Vision; episode 500 (d) Vision; episode 520

Figure 4.8: Change to greedy policy for the North heading over 20 episodes for absolute position based states (left) and vision-based states (right). Forward, left and right are represented by triangles facing North (towards the top of the page), East and West repsectively. The dark red grids represents the two goal states.

4.2.2. Sensitivity Study

Figures 4.9 and 4.10 show how the performance of the agent changes with ϵ, α and γ for the absolute position based and the vision-based agent. While varying one of the parameter the other parameters are kept at a value of 0.5. The agent has to learn to travel over two goals in a 9 by 9 gridworld.

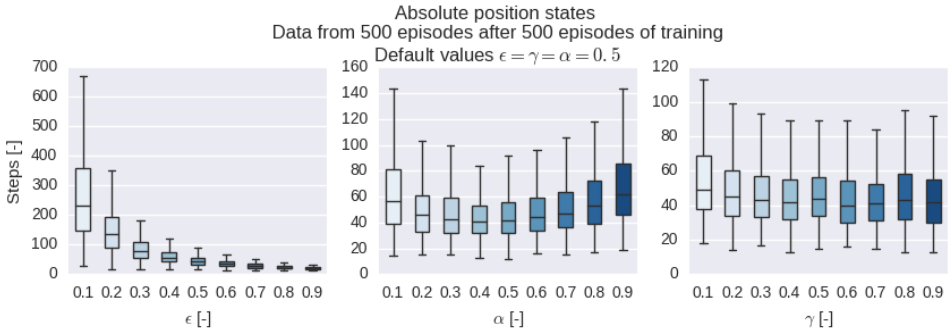


Figure 4.9: The sensitivity of the RL parameters for a absolute position based state.

When using absolute position (Figure 4.9) the most influential parameters on the performance are ϵ and α. Very low discount factors (γ) lead to worse performance. On account of the lack of propagation of the reward from states further backwards in time. The behavior of ϵ is as expected. The quadratic shape of the learning rate (α) is because both low and high learning rates lead to unconverged policies; the former due to lack of adaptation and the latter due to too much adaptation to newly observed values.

The effect of these parameters on the performance of the vision-based agent are different than their effects on the absolute position based agent. Figure 4.10 shows how the performance varies (nearly) quadratically with the greediness (ϵ) of the policy and (nearly) linearly with learning rate (α). A low ϵ is bad as the agent does not act greedily; however a high ϵ is worse as the ambiguity in the state information makes it impossible

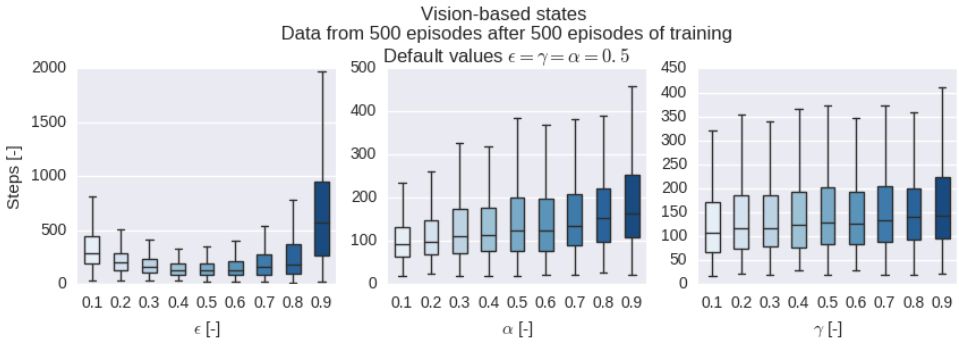


Figure 4.10: The sensitivity of the RL parameters for a vision-based state.

to obtain a mapping from state to an optimal action. The decrease in performance with increasing α can also be explained by the state ambiguity. Since states are ambiguous, adapting them less to newly experienced rewards and values lead to a better overall performance.

(a) (b)

Figure 4.11: Heatmap depicting the performance on the navigation task for different values of α and ϵ. The mean steps from 500 episodes after 500 episodes of training for the absolute position based and the vision-based state have been plotted.

Based on the findings of the parameter analysis, a realtionship between varying two of the parameters and the performance is simulated and plotted in Figure 4.11b. The γ value is kept at 0.5 for this study. The performance of the absolute position based state improves linearly with increasing ϵ and changes quadratically with α. The quadratic relation decreases with increasing ϵ. The performance of the vision-based agent gets linearly worse with increasing α and varies quadratically with ϵ. The heatmap shows the best performance can be obtained with an ϵ between 0.5 and 0.6 while keeping α lower than 0.1.

A limitation of this sensitivity study is the use of the number of steps required as a measure of performance. These parameters also influence how fast the agent learns and how good it adapts to changes among other things.

4.2.3. Effect of a Larger Gridworld

Simulations are run in a bigger gridworld to observe the effect of the size of the environment on the learning, memory requirements and end performance. For this investigation the simpler task of reaching one goal is used with a constant ϵ of 0.6, an α of 0.3 and a γ of 0.95. Gridworlds of dimension 20 by 20, 40 by 40 and 60 by 60 are simulated to study the effect of environment size. The vision grid "*vis0*" is used for the vision-based states. The simulations are run for 500 episodes in the 20 by 20 and the 40 by 40 gridworlds. Since the agent takes much longer to learn in the 60 by 60 world, that simulation is run for 3500 episodes and visualized separately.

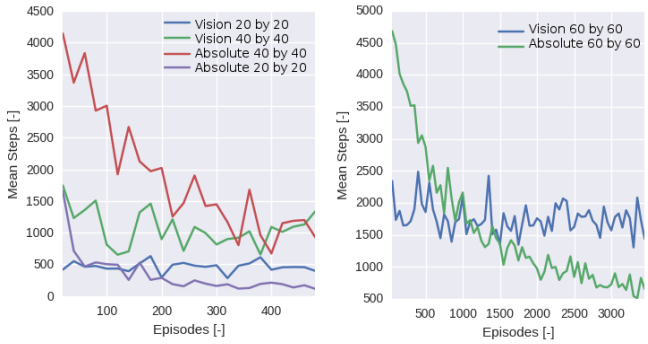


Figure 4.12: The mean steps during learning for the 3 gridworlds (Left: 20 by 20 and 40 by 40; Right: 60 by 60). Note that about 3000 episodes are required to train the absolute position based controller in the 60 by 60 gridworld; thus only for that gridworld 3500 episodes are run. The mean is taken every 50 episodes for the 60 by 60 gridworld case.

The left plot in Figure 4.12 shows the steps and sum of changes to the Q-values averaged over 20 episodes for the 20 by 20 and the 40 by 40 gridworlds. The vision-based agent starts off performing better due to its ability to learn faster, but it does not learn as much as the absolute position based agent. With more learning, the absolute position based agent starts outperforming the vision-based agent. The vision-based agent's perception does not have enough granularity to distinguish states whose optimal actions are different.

These simulations show how the time required to learn can increase with increasing number of states. Approximately 163,000 steps are required by the absolute position based agent in the 20 by 20 gridworld to achieve its final performance. In contrast, the agent in the 60 by 60 gridworld needs approximately 5,500,000 steps to reach its final performance. For a 9 times growth of the state space, the required computations to approach convergence grows by a factor of 30 .

The mean steps per episodes for the last 100 episodes and the number of unique states in the Q-table for the 3 gridworlds with two types of state are presented in Table 4.1. The number of unique states for the absolute position based state in comparison to the vision-based states shows an advantage of using vision-based states; 14400 unique states for the absolute position based agent compared to 29 for the vision-based agent.

The performance of the vision-based agent is worse off. It takes about 3 times the steps compared to the absolute position based agent, to finish the task in the 20 by 20 and the 60 by 60 gridworlds. The trend in the learning of the agents in the 40 by 40 gridworld indicate that the absolute position based agent will out perform the vision-based state with more episodes of training.

Table 4.1: Mean steps in the last 100 episodes and the number of unique states for the two state types in the different gridworlds

State type	Mean steps in last 100 episode			Number of unique states		
	20 by 20	40 by 40	60 by 60	20 by 20	40 by 40	60 by 60
Absolute	145	1065	668	1600	6400	14400
Vision-based	485	1148	1502	29	29	29
Random	-	-	4628	-	-	-

4.2.4. Effect of Increasing Features for Vision-Based State

All the previous simulations have two identifiable objects, the two goal grids. In this study, four cases are simulated to study the effect of more visually distinguishable features for the vision-based agent. The addition of an extra identifiable object, the addition of two extra identifiable objects, wall detection but no extra identifiable objects and lastly wall detection with the addition of one identifiable object are simulated.

All the simulations are run in a 9 by 9 gridworld with a constant ϵ of 0.6, an α of 0.3 and a γ of 0.95. The "*vis0*" vision grid is used. Five hundred episodes are simulated for each of the cases. The first extra identifiable object is added at (4,5) and second one is added at (7,2) as shown in Figure 4.13.

There is an improvement in the performance of the vision-based agent as the number of identifiable

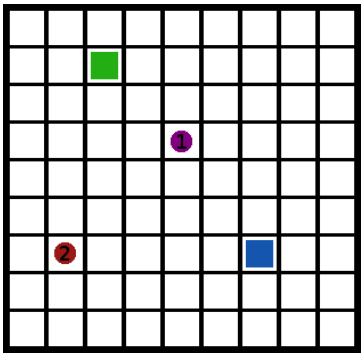


Figure 4.13: New features introduced to the environment

features are increased. This can be seen in the left plot of Figure 4.14. The improvement is bigger with wall detection.

This is believed to be due to the negative reward associated with hitting a wall and the fact that walls exist all around the gridworld making them good features for the state. Since the addition of features only improves the richness of the state space without providing a task based advantage, it does not lead to similar improvements. This is true notwithstanding the fact that the two feature case and the wall detection both have similar number of unique states; 168 for the 2 feature case and 164 for the wall detection case.

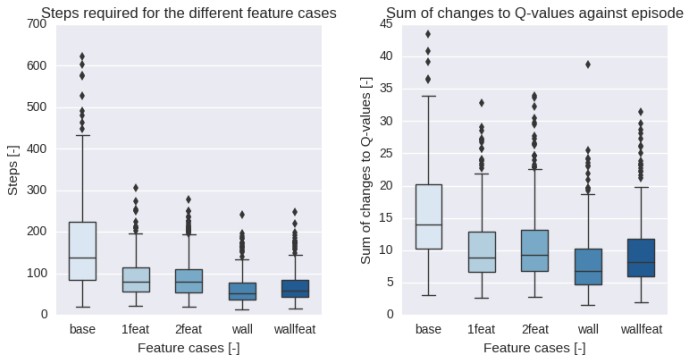


Figure 4.14: Statistics of the steps required and the sum of changes made to the Q-values for the last 100 episodes.

These simulations show how the performance of the vision-based agent is dependent on recognizable features in the environment. This can be used to mitigate the drawback from large state spaces as described in Section 4.2.3. The requirements for large memories to store big Q-tables can be reduced by designing feature based states tailored to the environment and the task.

4.2.5. Effect of Changing Action Space

The effect of improving the acting potential of the agent is analyzed by extending the actions available to it. The absolute position based agent and the vision-based agent are trained the task of travelling to two goals using the baseline action set and the extended action set. The simulations are performed in a 9 by 9 gridworld, with an ϵ, α and γ of 0.6, 0.3 and 0.95 respectively. The "*vis0*" vision grid is used for vision-based states.

A simulation of 2000 episodes are run for each of the above cases and the statistics of the steps per episode for the last 500 episodes are plotted in Figure 4.15. Figure 4.15 shows that extending the action set improves the performance for the absolute position based agent. However, it has a negative effect for the vision-based agent; it requires about 100 more steps on average with the extended set of actions than with the baseline set of actions.

In order to explain the difference in the performance of the vision-based agent with the extended actions, its trajectory in an episode is analyzed. It is found that the extended action set leads to local regions of optimality in the Q-function when using vision-based states. The ability to translate backwards, leftwards and rightwards makes sets of actions available to the vision-based agent, where it can keep getting rewards if it keeps going back and forth The reward scheme for the vision-based agent rewards it for having the goal in the

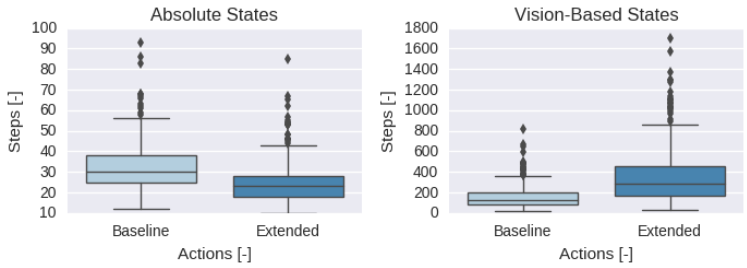


Figure 4.15: The performance with the baseline and extended set of actions for the absolute state case (left) and the vision-based case (right). The last 500 episodes are used to make the plots.

its "line of sight", which encourages this back and forth movements when seeing the goal. This idea is tested by running simulations with the vision-based agent for the two action sets with two different reward schemes.

Four simulations are run, all using vision-based states with the "*vis0*" vision grid. The vision-based reward is simulated with the two different actions sets (i.e. the baseline set and the extended set). Each action set is simulated with the absolute position based and the vision-based reward schemes. The usual simulations conditions are used: 9 by 9 gridworld and a constant ϵ,α and γ, of 0.6, 0.3 and 0.95 for the task of travelling to both goals.

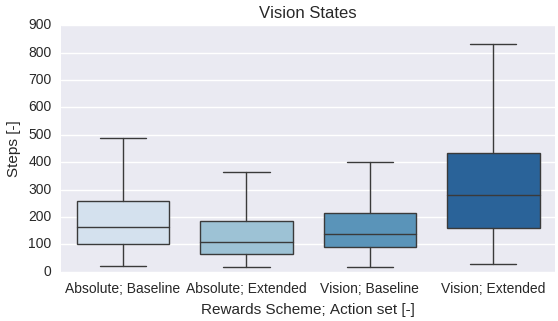


Figure 4.16: Statistics of the steps required from the last 500 episodes of the simulation for the different sets of rewards schemes and action sets.

A 1000 episode simulation is performed and the statistics of steps in each episode for the last 500 episode are presented in Figure 4.16. It is found that the agent with the absolute-position based reward and the extended actions performs the best among four the cases simulated.

The vision-based agent with the vision-based rewards and extended actions performs poorly for the reasons discussed before. Namely the agent finds states where it can keep getting rewards from seeing the goal, thus it tends to jump between those states. The increased mobility and the removal of vision-based rewards, for the agent with the extended action set and position based rewards, are the reasons for its improved performance. When the agent is no longer rewarded for having the goal in its line of sight, it does not get stuck in local rewarding loops and learns to use it increased mobility to get to goal faster than the agent with the baseline action set.

4.3. Summary

The findings for the design of a vision-based RL agent from the gridworld simulation studies are summarized in this section.

Lack of learning with vision-based states: The simulated vision-based agent does not learn the task; its performance does not improve over time. This is on account of the non-Markovness of the vision-based states.

Need for stochasticity with ambiguity in state descriptions: Due to the inherent ambiguity in generalized state descriptions, the vision-based RL problem is partially observable. Thus, policies with a random component or the application of Partially Observable MDP models may be required to ensure good performance.

Effect of action space: It is shown that the selected action space is related to the quality with which a vision-based agent can perform a task. There are relations between the defined rewards and the available actions. The vision-based agent with the extended action set performs worse as the interaction between the rewards and the actions lead to local minima in the value function.

The design of an RL agent should consider the interactions between the defined actions and reward scheme, and how they cause the agent to behave. A reward scheme and an action space should be selected which enables the agent to perform the task. The action scheme shall allow the agent as much freedom as possible without causing it to lose performance, by for instance, introducing local minima in the value function.

With some ideas about the behavior of a RL agent using vision-based state, concept for the vision-based state, the guidance task and the agent to be implemented in this study are presented in Chapter 5.

5

Concepts for Vision-based States, Guidance Task and Agent

This chapter presents the conceptual design of the vision-based state, the guidance task, and the agent. Following these discussions, an initial starting point for the design of a vision-based RL agent is described. Lastly, the differences between the implemented agent and the initial concept are considered. The sections in this chapter provide answers to the sub-questions of **RQ2**: "What will be the task and the architecture of the controller simulated and tested in this study?". Further, the work presented in Chapter 2 and the discussion about the implemented agent in this chapter are an answer to **RQ3**: "Which learning schemes and vision-based states are simulatable and testable within the resources of this project?".

5.1. Vision-Based State

This section discusses possible concepts for vision-based states that can used in this study. First, absolute and relative states are compared. Than three ideas for vision-based states are presented. This will answer **RQ2.1**: "What are some options for vision-based states that can be used by RL for learning a guidance and navigation task?".

5.1.1. Absolute or Relative States

The goal of this project is to use vision-based RL for the guidance of a quadcopter, therefore, the position information has to be obtained from visual sources. Two ways of obtaining position from vision are considered. One way is to attempt to create a map of the environment using visual resources, and using the map to locate oneself in absolute terms. The other way is to use some kind of relative position information.

Note that the difference between an absolute description of the state and a relative description of the state is not based on their frame of reference. Both absolute and relative descriptions of the state will have their own form of a "frame of reference". The absolute description of the state is based on the usual description of position and attitude as has been the basis in guidance and navigation. Relative descriptions of the state have some information, but they are not as clear or crisp as the absolute information.

For example an absolute description of a quadcopters state can be (with respect to a local reference): "4.57m North, 2.45m East, 035° Heading". A relative description can be: "5±1m away from base; base in South West quadrant relative to *self*[1]". For this project, some form of representation of the visual information that can be used by the agent to perform the guidance task will be a relative description of the state.

The pros and cons of absolute and relative positioning are examined in Table 5.1. They are partly derived from the simulation study discussed in Chapter 4. Due to the relative simplicity of vision-based relative states over vision-based absolute states and due to the lack of experience of the author on SLAM based vision methods, it is decided to use vision-based relative states for this project.

[1] Self is the quadcopter in this example

Table 5.1: A comparison of using vision-based absolute or relative states

	Absolute states	Relative states
1	Has no state ambiguity	State ambiguity is likely
2	Suffers strongly from the Curse of dimensionality	Suffers less from the Curse of dimensionality
3	Knowledge is focused on specific implementation	Function focused knowledge
4	No need to come up with representation since the absolute states are sufficient	A representation of the states which is sufficient for the task needs to be created

5.1.2. Concepts

Three concepts for vision-based relative states are proposed and a selection made for the first iterations of task and agent designs.

VisSta1: Landmark The agent can look for and remember unique objects from its environment and estimate its current location based on the relative size and position of the object in the image. The landmark does not need to be an object, it can be a blob of color or a barcode. Once an object is identified, its size in pixels is estimated. Seeing the landmarks from different places will make it appear bigger or smaller in relation to its initial viewing. The ratio of the current dimension in comparison to the initially observed dimension can be used as part of the state to signify proximity to the object. The relative heading with respect to the quadcopter at which the identified object is seen can be used to get an idea of quadcopter's heading.

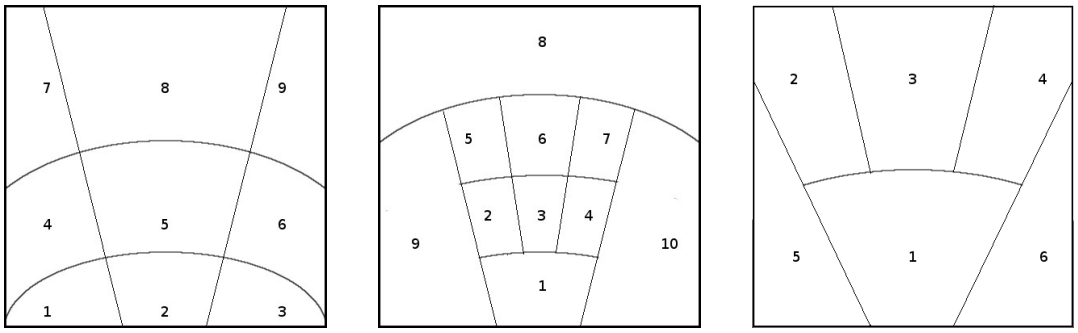
There are issues of perspective, skewing and noise among other things, when trying to estimate dimensions of objects that are detected using computer vision. SIFT features provide some invariance to scale and rotation. A method of tracking the different views of a landmark can be considered at the cost of further increasing the implementation challenges. Monocular distance estimation techniques can be used to come up with estimates of the distance to the landmark. This can be used to generate a coarse map of the quadcopter's environment. Using these techniques in the design will increase the difficulty of implementing such a system.

Figure 5.1: The mug is selected as a landmark. Its approximate relative dimensions and lateral position are shown from three different point of views.

An example state vector can be one number representing the index of the landmark being tracked, another expressing its relative size in comparison to the first observation and a third number expressing the relative heading of the landmark with respect to the heading of the quadcopter. In order to get rid of ambiguity, the absolute heading of the quadcopter will also need to be included. This is required because the relative size and lateral position of an object will be the same when the quadcopter is facing it from opposite sides.

Assuming a quadcopter heading and relative landmark heading representation in discrete steps of 45°, leads to 8×8 possible permutations for the headings and landmark headings. Using a relative size range of 0.3 to 2.4 in steps of 0.3 leads to 8 different values for the relative size. This leads to 512 possible combinations for the position. If there are 4 trackable landmark, there will be 2048 possible states. This is a big number of states to learn in real life flights. If such a method is selected, a significantly coarser representation of the state space will have to be used.

VisSta2: Visual grids The visual feed can be divided in grids and the grids assigned values based on their contents. Easily identifiable objects, such as plain blobs of color, are good options for visual content as they will simplify the CV implementation. A one dimensional array, the size of the number of grids, will hold the vision information. Each of the possible visual elements will be represented by a number, thus each cell in the array will take the numerical value of the contents of the corresponding grid. This array can be used to represent a relative vision state. An early inspiration of such a state is obtained from (Gaskett et al., 2000).



(a) 9 grids (b) 10 grids (c) 6 grids

Figure 5.2: Possible ways of dividing the vision stream into grids

A simple implementation of this can be based on a color filter and a few goal locations marked by solid colored objects, such as colored cubes. If there are two goal locations and the vision is divided into 9 cells, the state can be represented using an array with 9 integers where each integer takes a value of 0 if it sees nothing and a value of 1 or 2 if it sees a goal object in the grid. The quadcopter can see this in four possible headings, which is assumed to be part of the state. If we make the assumption that no two goal objects can appear on the same grid, the total number of possible states sum to 404[2]. This is a relatively small number and manageable by RL algorithms.

If only seeing nothing or seeing goal objects are considered in a k grid (i.e., cells in the state array) vision state with n possible goal objects, the number of visual grid states is given by Equation 5.1. P and C are the permutation and combination functions respectively. There are ${}^{k}P_{r}$ possible permutations for seeing r goals in k grids. If there are n goals, the combination of ways r goals are seen at one time is given by ${}^{n}C_{r}$. The product of these two numbers are the number of possible states for a k gridded vision state with r of the n goals in its line of sight. The sum of the aforementioned product over all values of r is the number of possible states. If we want to consider walls as another visual entity that is recognized, properly estimating the total number of states becomes more complicating. In such a case estimating an upper bound is more tractable.

$$\sum_{r=0}^{n} {}^{k}P_{r}\,{}^{n}C_{r} \tag{5.1}$$

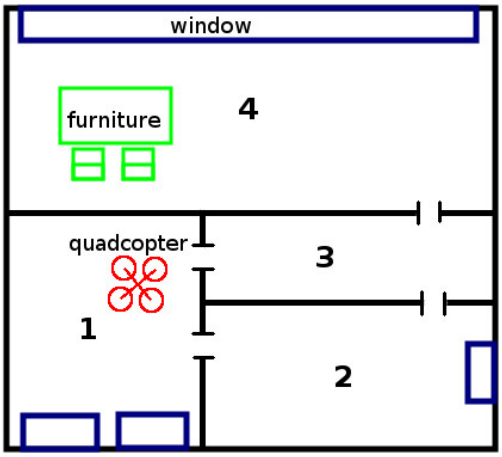
This example state also highlights a problem of such a state definition. Although there are 404 states, only four states represents seeing nothing. There can be many different real locations where the agent sees nothing. Thus seeing nothing is not sufficient to tell what the following state will be, given an action. This makes such a state description non-Markov and difficult for TD learning methods.

The problem of state ambiguity can be alleviated by introducing more goal objects for the task, and trying to minimize the numerous real states which maps to the “seeing nothing” visual state. This puts us against one of the typical trade-off in representation: increase in the detail of the feature that is interpreted by the system (which for this state description means using more grids), is accompanied with decreasing computational efficiency due to the growing state-space.

[2]Note the goal states 1 and 2 cannot appear more than once in a cell. 9×8 possible states where the first and second goals are seen together. 9×2 possible states where either the first or the second goal is seen. 1 possible state where nothing is seen. It can see these things in 4 possible directions

VisSta3: Location tagging This concept for a vision-based state uses visual information to represent a coarse description of the quadcopters location inside a structured environment. Features such as specific trees observed by the quadcopter, patters on a wall, relative position of doors and windows in a room etc. can be used by it to localize itself coarsely.

Such a state can be defined by an array of numbers, each representing specific features. For example there can be number representing the wall color, the number of doors observed, the number of windows observed etc. If there is sufficient detail and disambiguation in the state definition, it can be used to identify the different rooms in a building. This information can be used by the quadcopter to guide itself to different rooms in a building. For such information, the quadcopter will have to perform scans of its environment and the CV algorithms will need to perform object recognition and matching to generate the state.



Room	Doors	Windows	Furniture
1	2	2	0
2	2	1	0
3	3	0	0
4	1	1	1

Figure 5.3: The left part of the image shows the ground plan of a building with four rooms. The location has been tagged based on the number of doors, windows and the presence or absence of furniture in each room. The right part of the image shows the corresponding location tag states for this ground plan.

The problems of this state definition are the facts that it cannot be used for precise positioning of the quadcopter and that high level interpretation of the visual data is required in order to localize itself. As it is incapable of fine localization, it may be used to guide the quadcopter on a higher level, such as finding paths in a graph based representation of an apartment. This system is also more challenging to implement as object recognition systems will need to be developed.

Selected configuration: At this stage it is decided to use the visual grid states (**VisSta2**). **VisSta2** is selected due to the ease of implementing such a system in comparison to a landmark (**VisSta1**) or a feature based (**VisSta3**) system. Moreover, if a feature based state is used, the guidance task has to be over places with different external features, which will be hard to do in the Cyber Zoo.

From the three types of vision states described above, the visual-grids and landmark based method are suitable for fine positioning while the location tagging method is suitable for coarse positioning. This leads to the idea of a possible system which uses feature based information for coarse navigation and than switches to either grid based or landmark based positioning for finer guidance to its goal. However, such a compound system is out of the scope of this study as it is too complex.

Goals are considered to be boxes of a specific color. If extra detail is desired, barcodes can be used in which case vision-based barcode reading algorithms will be required.

5.2. Guidance and Navigation Task

This section presents possible guidance and navigation task for the RL agent. First, the various competing requirements that the defined task needs to satisfy are considered. Than, concepts for guidance tasks for this project are described. This section attempts to answer **RQ2.2** by defining a task that will be used for this study.

5.2.1. Requirements for Guidance and Navigation Task

The following requirements are defined for the guidance task:

TaskReq1: The task should use vision-based information and no external localization system (such as GPS) in order to localize itself.

TaskReq2: The task must be performable by an RL agent. It must be attempted to define a task such that the benefits of using an RL agent over other autonomous means of control are highlighted.

TaskReq3: The task should be simulatable with the resources available to this project.

TaskReq4: The visual information required for the task must be attainable within the limits of this project. The task will be performed by an AR Drone 2, thus the vision information shall be obtainable and processable within the limitations of its onboard camera and CPU.

TaskReq5: The task should be performable in the Cyber Zoo of TU Delft.

TaskReq6: The task should represent a challenge to the field of autonomous navigation of UAVs. It must attempt to contribute something that can be related to a real application for autonomous UAVs.

TaskReq7: The defined task needs to be simple enough so that flight tests can be performed.

The first two requirements are higher level requirements that are derived from the core goal of this project. The next three requirements are related to defining a task that can be useful to accomplish the defined goals of the project. The last two requirements are related to the quality and feasibility of the guidance task.

5.2.2. Outline of Possible Tasks

Four guidance and navigation task are defined based on the assumption of using some form of the visual grids state (**VisSta2**). One task will be chosen for the design of the RL agent.

RLTask1: Photograph points in a 2D space There are four points in a room the size of the Cyber Zoo that have to be photographed once by the quadcopter. This means there is no reward for photographing the same point twice. The quadcopter has a memory of carrying two images after which it must return to base. The RL agent has to find a path to photograph two points, return the images to base and photograph the two remaining points.

On visiting the base, the memory is returned to zero and the quadcopter is reinitialized. One way to reinitialize is to place the quadcopter in a random position of the Cyber Zoo. Another way is to reinitialize it at a specific location around base. For this task, a state that keeps an account of the photographed sites will need to be appended to the state.

RLTask2: Photograph points in an indoor situation Four points are placed within four separate rooms, each of which have a passage into two of their neighbors. One of the rooms represents the base station. As in the previous task, the quadcopter has to photograph the four points with the added difficulty of having to navigate through the passageways after photographing a point. This will be difficult to test in the Cyber Zoo because it will be hard to create the 4 separate rooms. This task can be performed outside the Cyber Zoo, in a place with two or more interconnected rooms. However, the lack of stabilization feedback of the motion sensing system will add other challenges.

RLTask3: Run a race course Running a race course is another task. Points will be visually marked to be checkpoints. The quadcopter has to travel to these points in the Cyber Zoo and hover over them for a set duration, before the next checkpoint is revealed. The quadcopter has to fly over the checkpoints as fast as possible.

The reward can be given to the agent based on reaching and hovering over the current checkpoint. The RL agent will control the heading and the velocity of the quadcopter.

RLTask4: Follow a target Learning to follow another quadcopter, while keeping it in the center of its vision, could be a task for the RL agent. This task is inspired by the work of Valasek et al. (2016).

If the image of the tracked object becomes bigger, the quadcopter needs to slow down and vice versa. The quadcopter can choose to yaw or move laterally to track the target. Different transformation of the visual representation of the tracked object will be mapped to different motions by the RL agent.

This project will use **RLTask1: photographing points in a flat space**(Figure 5.4) as the starting point for the design of the RL agent. This task is considered to be sufficiently representative of a real life task a quadcopter may need to perform.

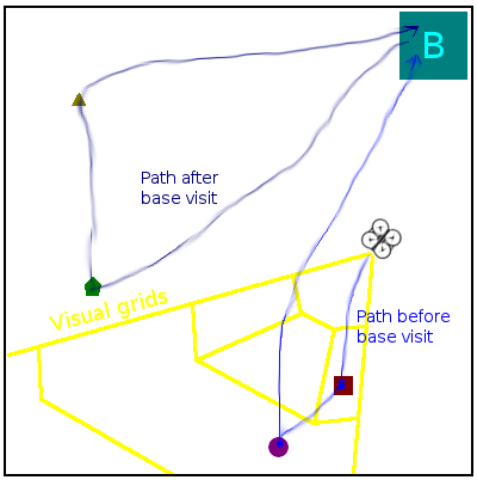


Figure 5.4: A representation of **RLTask1: photographing points in a 2D space**. An example path the quadcopter can take to act near optimally is shown. The four goal locations are marked by the small filled square, circle, pentagon and triangle. The yellow grid shows a possible visual grid.

5.3. Initially Proposed Agent

With an idea of the state representation and the task the agent needs to perform, other details about the agent can be defined. Here, a learning scheme is selected, a more detailed definition of the state and actions is created and lastly a reward scheme is decided upon. Selecting a learning scheme will lead to an answer of **RQ2.3**.

5.3.1. Learning Scheme

At this point, SARSA is considered to be the most appropriate learning method for this project. Q-learning is potentially problematic as the results with the gridworld study indicates (Chapter 4) that learning with full greed leads to suboptimal solutions, when the state description is non-Markov. DP methods are undesirable as the goal of this project is to eventually carry out learning with a real life quadcopter and real time learning is simpler with TD methods. Also there is no transition model as is required by DP methods. Due to the non-Markov nature of the state and transitions, MC methods maybe superior to DP methods. However, MC require more iterations for convergence in comparison to TD methods(Sutton & Barto, 1998) and it updates the values after each episode, which ares undesirable for real-life learning.

For the time being, other more complicated methods, such as the use of a continuous description of the state space or the use of RL methods such as LSPI or Deep Q-Learning are avoided due to inherent complexities of the project goal. The training of a vision-based RL controller for an AR Drone 2 presents a significant enough challenge.

5.3.2. State and Actions

State: The visual grid (**VisSta2**) state will be used for the task. The value of each grid will be based on the contents of that grid. The goals will be represented by cubes of a specific color. Decisions have to be made regarding the use of the forward facing and the bottom cameras. Further, the exact division of the visual feed into grids will need to be fixed.

One goal object may end up occupying multiple grids. In such a case, the grid that contains the bigger portion of the detected goal will represent the value of the grid. Multiple goal objects can end up occupying the same grid too. In such a case two measures will be taken. First, if there is a previously visited goal in the grid, it will be ignored. If there are still multiple goals, the grid's state will be defined by the bigger goal object.

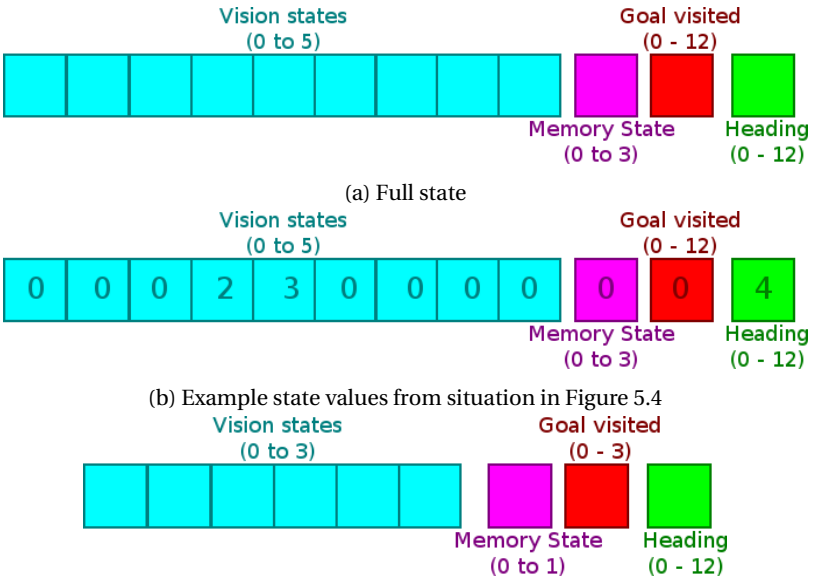
For the initial implementation, it is decided to consider a hover over goal when the goal object is brought to the grid closest to the quadcopter's forward facing camera. This ignores the requirement of having to hover exactly over the target. It simplifies the task as hovering over a goal would either enlarge the state space or require the use of complicating methods such as two different high level controller or the use of Hierarchical RL.

A 3 by 3 gridding of the vision stream is selected for the initial design. The state will consist of a vector

containing twelve numbers. Nine of these numbers will represent the vision information from the nine vision grids. One number will represent the memory state, another will represent the goals that have already been visited and one number will represent the heading of the quadcopter.

The vision states will take a value 0 if nothing is seen, a value of 1 if the base is seen and a value between 2 and 5 if one of the four goals are seen. This leads to a total of $\sum_{r=0}^{4} {}^{4}C_r {}^{9}P_r = 39806$ possible vision states. Such a large number of vision states makes this problem definition intractable. Even without considering the other parts of the state, it is evident that the task needs further simplifications.

Hence for first iterations of the guidance task, many simplifications are made. The number of goals will be decreased from four to two, the memory is decreased from two to one and the visual grid is simplified to 6 grids (Figure 5.2c). This brings the possible vision states down to 229 and the total number of states to 24,000 states. This number is still big and further simplifications will be made during implementations. Wall detection is not included into the state description as the state space is already big.



(a) Full state

(b) Example state values from situation in Figure 5.4

(c) Simplified state

Figure 5.5: A visual representation of the full and simplified state vectors

Action: Three actions are defined for the agent. The actions are move forward, turn left 30° and turn right 30°. For the actions to require similar amounts of time, their magnitude (the real distance moved or angle yawed) will be defined based on simulation results. An extra action of setting the quadcopter heading towards a target may be included. This action will require other supporting logic for picking targets.

5.3.3. Rewards

The reward scheme must cause the agent to perform the guidance task optimally. The reward scheme is defined for the full guidance task. It can be adjusted for simplified task as defined in Subsection 5.3.2. We want the agent to find an optimal path of photographing the two closest goals, returning to base, and than photographing the remaining two before returning to base again. An example of an algorithm that always goes to the closest goal has been presented below:

Go to closest goal algorithm

1. Start in random location on the map
2. Make a scan to find closest goal object
3. Head towards closest goal object and hover over it; remove current object from list of goals
4. If memory not full go to step 2, else go to next step
5. Scan for base; head for base
6. Check if this is the first time the quadcopter came to base; if yes than go to step 2, else end episode

There should be some kind of intermediate reward guiding the quadcopter towards the goals. This can be a small reward when the quadcopter sees an unvisited goal in one of the farthest grids which slowly increases as the goal object moves to rows closer to the quadcopter. This shaping reward needs to be implemented carefully as the preliminary simulation found that one can cause the performance to degrade by giving the agent rewards for having the goal in its line of sight (Subsection 4.2.5); i.e., if the agent can transition between states where it can keep the goal in its line of sight and keep getting rewards, it will be encouraged to keep repeating those states.

The reward for seeing an unvisited goal in a vision grid should be awarded only once. After the quadcopter reaches the goal it should receive a big reward for having accomplished an objective. After visiting a goal, all rewards associated with it should be removed to encourage the quadcopter to visit other goals.

The reward for the base should be 0 when the memory is empty while the reward of goals are left changed. The reward for the goals should be progressively reduced to 0 and the base to a high value as the memory gets filled. A small negative reward should be awarded at each step which would encourage the agent to learn actions which perform the whole task faster.

Due to the influence of the state and the reward scheme on the performance of the RL algorithm, these are two most likely candidates for tweaks and modification. By the end of the project, numerous changes and modifications may lead to significantly different representations of the state and reward scheme than discussed above.

5.4. Comparison with Final Task and Agent

This section discusses the differences between the implemented agent as described in Chapter 2 and the initial concept. Further, some characteristics of the designed controller, not elaborated upon in Chapter 2, are also included.

Task: The implemented agent has been described in Chapter 2. The implemented task is simpler than the initial concept. The tasks consists of approaching markers on the wall of a square room. Three task are simulated, all of which are significantly less complex than **RLTask1**. The simulated tasks are:

1. approaching a red goal; only task performed in flight tests
2. approaching a red goal and a blue goal
3. travelling across a corridor to approach a red goal.

Simpler tasks are more desirable for this project as its goal is the implementation of the agent on a real-life system.

State: The final form of the vision state consists of 3 numbers instead of the predicted 6(see Chapter 2). The divisions of the image is made using straight lines instead of curved lines as initially considered. No memory states or headings are used. Instead a state to record wall hits and another to represent the amount of goal seen (the Color Fraction) is used. Learning is faster in smaller state space, thus this simplification of the state space is beneficial for real life tests.

Actions: The action of going forward remains the same. For the turning actions, an angle of 22.5 ° instead of the 30 ° is used. Further, an option of turning until a goal is seen is introduced.

Although the idea of the *option* is taken from (Sutton, Precup, & Singh, 1999), the exact way it updates the value has been implemented in a differently. In (Sutton et al., 1999), the value of an *option* is updated with the sum of the discounted reward that was obtained while performing the option. The update to the value is made after the option has been fully carried out.

It was found in this study that updating the *option* every step, as if it was a primitive action, leads to faster learning than updating it in the end. Since the *option* is available in primarily one state[3], update to the value of the option every step would be an iterative update to itself which ensures the correct state-action is being updated.

Learning: Q-Learning is used instead of SARSA in the final agent. The initial idea was to use SARSA as the gridworld simulations (Chapter 4) indicated a degree of randomness in the action is desirable when the there is ambiguity in the state descriptions. However, the actions in the implemented agent implies a more predictable transition model than in the gridworld simulation. The results from the simulation and flight tests with the designed agent show that the Markovness in the state description for the one goal task is sufficient to allow fully greedy actions without divergence.

5.5. Summary

The initial concepts for the vision-based states, the guidance task and the agent are described. Following this, the differences with the implemented agent are highlighted. In general, the initial concept was optimistic of what can be accomplished. The requirement to focus on a RL problem which can be implemented and learned in real life led to a simpler task and state description in the implemented system.

This chapter has answered **RQ2**: "What will be the task and the architecture of the controller simulated and tested in this study?". Furthermore, Chapter 2 has answered **RQ3**: "Which learning schemes and vision-based states are simulatable and testable within the resources of this project?". Remaining questions regarding the performance of the agent are answered in Chapter 6.

[3] It is also found in the less common seeing nothing, wall hit state (0,0,0;0;x;1)

6
Additional Results

The findings from the implemented controller, not described in Chapter 2, are presented here. The comparison of the RL agent with the rule based controller in Chapter 2 and with an absolute position based autopilot in Section 6.1.4 answers **RQ4.2**. The agent developed for the two goal task is trained to perform the corridor task in Section 6.1.6 answering **RQ4.3**.

6.1. Results

Simulations are performed to see how many episodes are required to learn the task. The sensitivity of the learning to γ and α is studied. However, as only one repetition is carried out for the different values of the parameters, the results from the sensitivity study are not statistically significant. The effect of changing the reward function by incorporating less knowledge is simulated. Incorporating more knowledge in the reward function improves the learning of the task. The performance of the agent is compared to a programmed flight plan which uses absolute position information. The absolute information based autopilot does the task twice as fast as the trained agent. The effect of fully random exploration is compared with ϵ-greedy exploration and the benefit of using an ϵ-greedy policy demonstrated. The agent for the two goal task is trained a corridor task to demonstrate the generalized applicability of the designed agent.

6.1.1. Number of Episodes Required for Training

The number of episodes required to train the one goal task (with fixed initialization) is studied by training it for a range of different durations. The ϵ is increased from 60% (greedy) to 100% (greedy) in all the runs. Each simulation is repeated 50 times. The agent is trained for:

- 30 episodes (ϵ increased every 3 episodes)
- 50 episodes (ϵ increased every 5 episodes)
- 100 episodes (ϵ increased every 10 episodes)
- 200 episodes (ϵ increased every 20 episodes)
- 300 episodes (ϵ increased every 30 episodes)

The number of steps required to reach the goal in the last episode of every run from every training duration is visualized using a box plot in Figure 6.1. The figure shows that the performance for the learning of the one goal task degrades only for the 30 episode training case. Four outliers are present from the runs with 30 episodes of training, each requiring more than 20 steps to do the task. There are no outliers (of above 20 steps) when the agent is trained for 100 or more episodes. Further, the spread in the final performance, i.e. the variance in final steps required to perform the task, decreases with more episodes of training. These results indicate that the task can be trained satisfactorily with 100 episodes of training.

6.1.2. Sensitivity to γ and α

The sensitivity of the training to varying γ and α are shown for **1 run** in Figures 6.2 and 6.3. As these results contain data from only one run, they are not statistically significant. Results from the only run indicate that a higher γ and a higher α can speed up the learning of the one goal task.

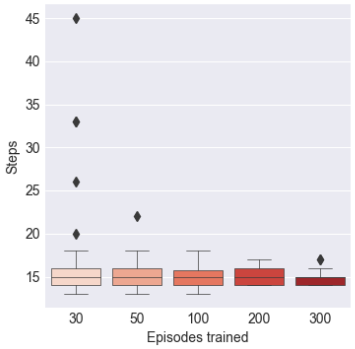


Figure 6.1: Box plots of the steps required in the last episode of the 50 repetitions of each experiment factor (i.e. for each training duration).

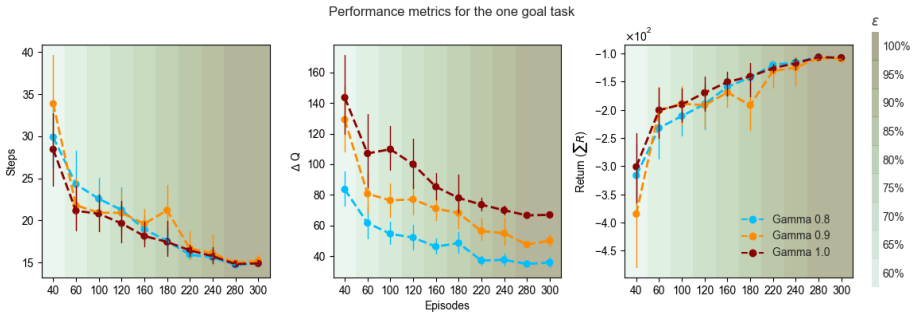


Figure 6.2: Varying γ

6.1.3. Effect of Changing the Reward Scheme

The reward scheme used in the study encodes an extra reward if the quadcopter is moving towards a visited goal compared to an unvisited goal. Simulations are performed without this extra element in the reward scheme to study its effects. The change to the reward scheme, visualized in Figure 6.4, is studied (full reward scheme presented in Chapter 2). It can be seen in Figure 6.5 that encoding the knowledge of going forward when seeing color speeds up learning (**RewScheme1**). The lower curve of the steps required with the vision-based penalty (orange line in the leftmost plot of Figure 6.5) indicates that the agent trains with fewer steps in total.

6.1.4. Comparison with Absolute Position Based Autopilot

The time to perform the task by the RL agent is compared to a pre-programmed autopilot. This is done as time is a more realistic metric of performance than steps for guidance and navigation systems.

The one goal task is carried out approximately 15 seconds faster by the pre-programmed autopilot than by the fully trained agent (see Figure 6.6). Further, the pre-programmed autopilot does not need any training or the design of an abstracted state. Although performance of the RL agent is worse in terms of time, is has some features which the autopilot does not. The autopilot is not independent of an absolute positioning system and it is incapable of learning or adaptation.

6.1.5. Fully Random and ϵ-greedy Exploration

The effect of learning the Q-values for the one goal task with a fully random exploration and an ϵ-greedy exploration are compared. The one goal task is trained 50 times, and the final learned policies are visualized to look at the stability of the learning.

Figure 6.7 visualizes the number of times a specific action is selected as the final action from the 50 runs. The one goal task is performed with a fully random exploration and an ϵ-greedy exploration. The dominant color of the columns and the color fraction part of the state are expressed along the x-axis of Figure 6.7. The number of times the different actions are learned as the final policy are expressed by the lengths of the bars of specific colors for each state.

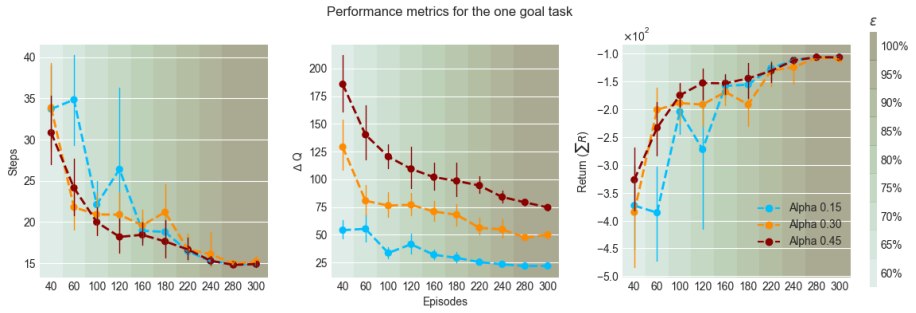


Figure 6.3: Varying α

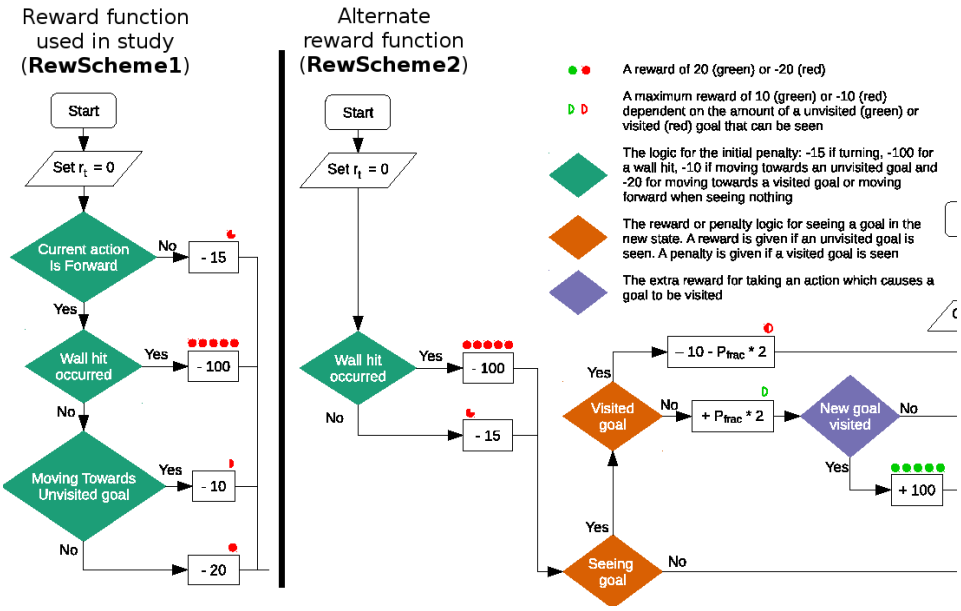


Figure 6.4: Change to reward scheme

The ambiguity in the state perception is demonstrated in Figure 6.7. Most of the runs should converge to the same action for states which are less ambiguous and their bars in the figure should be mainly of one color. States which are ambiguous or where optimal actions are difficult to find, will converge to different actions for the different runs; thus their bars in Figure 6.7 will consists of 2 or more colors. If the state is more Markov, than a bigger fraction of the runs will converge to one action and there will be a big difference between the lengths of the two colors in the bar of the state. Conversely, when the state is less Markov, the lengths of the two (or more) colors in the bar of a state will be similar.

In general, Figure 6.7 shows that the learning with ϵ-greedy exploration converges to one action more frequently than with random exploration. For example with the goal in the middle (0,1,0) the learning with the ϵ-greedy exploration converges to the forward action for almost all of the runs. However, with random exploration, the agent converges to the left or right actions more often. The non-Markovness of the states makes Q-learning with fully random exploration less convergent than with ϵ-greedy exploration.

The ϵ-greedy exploration, although more convergent than random exploration, also has numerous non-convergent states. For example with the state "0,1,1;0" the agent trained with the ϵ-greedy policy converges to the left action for a number of the runs (top plot of Figure 6.7), although it would be better for the agent to turn right as the goal is towards the right and relatively far.

A possible reason for this is the skewing of the goal when viewed from different angles (see Figure 6.8). When the goal is viewed obliquely, it gets skewed, and the area it takes on the visual stream changes due to perspective effects. Perhaps in some of the runs taking a left action increased the area of the goal, leading to higher rewards from turning left in the "0,1,1;0" state.

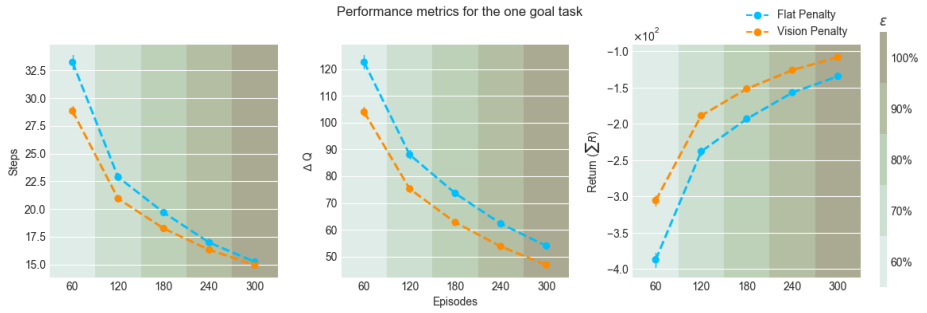


Figure 6.5: Effect on learning with the two reward schemes

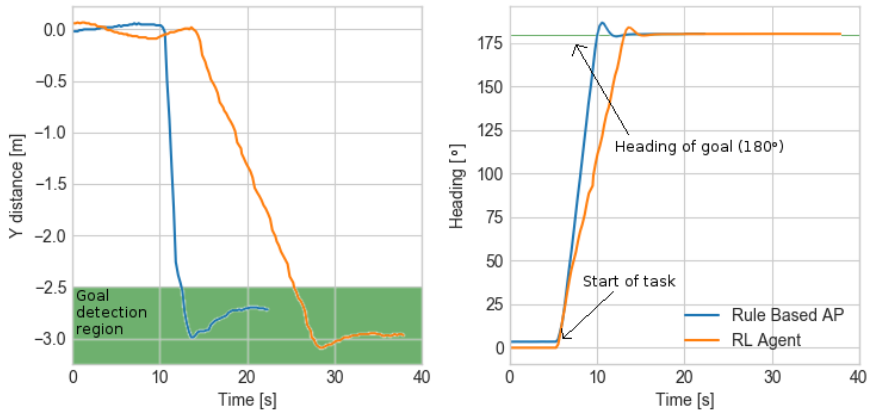


Figure 6.6: State (The heading and Y positions (with origin being 0) of the RL agent and the rule based agent performing the one goal task

6.1.6. Corridor Task

The initial goal of this thesis was to have a representative guidance and navigation task to demonstrate the use of RL for vision-based guidance (Section 5.2.1, **TaskReq6**). However the final tasks performed are rather simple (see Chapter 2). In order to increase task complexity a corridor task is designed. In this task the agent initializes at the end of a corridor and has to learn to turn past a corridor to reach its goal (see Figure 6.9).

The two goal agent is used for the training of the corridor task with one exception: there is no reward for visiting the blue goal. The blue goal acts as an intermediate beacon for guiding the agent to the red goal. The state description and action space are the same as the two goal task agent. The learning scheme is also the same as the two goal task: 500 episodes of training with $\gamma = 0.9$, $\alpha = 0.3$ and ϵ going from 0.6 to 1.0 in steps of 0.05 every 50 episodes. The steps to the goal and the trajectories taken by the agent have been visualized in Figure 6.10. It can be seen that with training, the agent manages to learn to perform the corridor task better.

6.1.7. RGB versus YUV in Flight Tests

Initially the computer vision (CV) module converted the YUV data (that is natively available from the vision module of PaparazziUAV) to RGB. This was done because he simulation is implemented in the RGB colorspace. Converting the YUV information to RGB allowed the direct application of the vision modules developed in simulation to the real-life agent. This worked out fine until the first quadcopter crashed beyond repair and a replacement had to be used. The replacement quadcopter had higher noise in the vision data; tests with YUV revealed that it was less noisy. Thus, YUV was used for the final flight tests and for drawing conclusions in this study.

Figure 6.11 shows the steps against episodes for the runs using the RGB based and YUV based CV modules. The right most plot in Figure 6.11 visualizes the mean trends over the runs. The plot shows that the final performance is better with the YUV based agent. However, the YUV based agent has a higher variance in the steps required during the learning phase.

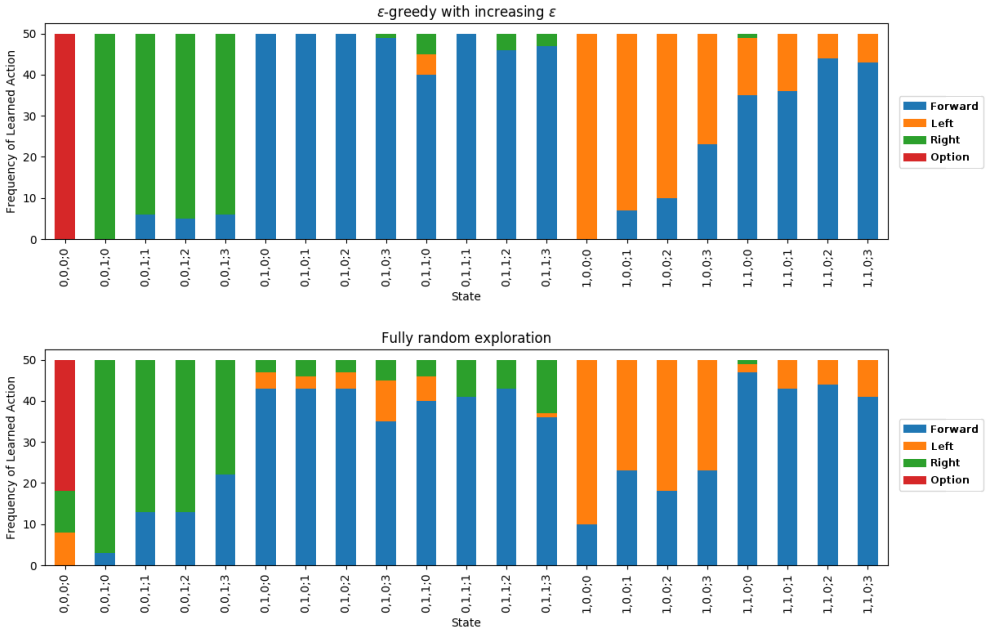


Figure 6.7: Stacked bar charts of the frequency of selection of specific actions for two types of exploration: ϵ-greedy with increasing ϵ (top) and fully random exploration (bottom).

(a) 22.5° left; 17k (b) Initial; 15k (c) 22.5° right; 19k

Figure 6.8: The skewing increases the number of pixels of the goal detected. The relative heading and the thousands of red pixels detected are mentioned in the captions.

The two left plots in Figure 6.11 visualize the statistical insignificance of the study. For the RGB based agent, two runs diverge from the optimal solution in the last phase of the training. In comparison, only one run diverges in the final phase for the YUV based agent. If, for example, 5 more repetitions are carried out with the YUV based agent and the RGB based agent, the means of the performance over the runs may turn out different.

External factors such as the time of day and internal factors such as the differences in the tuning of the YUV detection in comparison to the RBG detection influences the performance. Differences between the tuning of the RGB and the YUV modules could be one of the possible causes of the higher variance in the training with the YUV based agent. Such sources of errors in the flight tests and the differences between the performance with the RBG and the YUV based CV modules are not studied in greater detail due to the time constraints of this project.

6.2. Summary

The additional results from the simulations are summarized here.

Number of episodes to train (6.1.1): 100 episodes are enough to learn the one goal task.

Sensitivity test (6.1.2): As enough repetitions are not performed the sensitivity study is not statistically significant. Data from the one run at the different factor levels indicate that a higher γ and α can speed up learning.

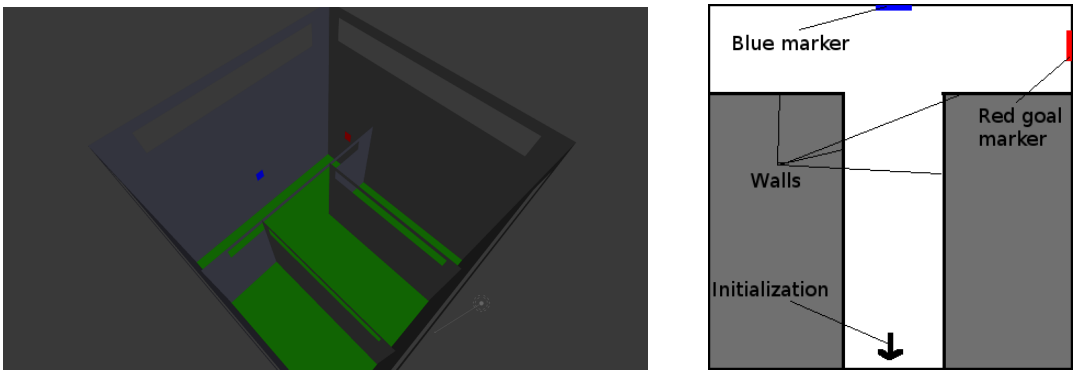
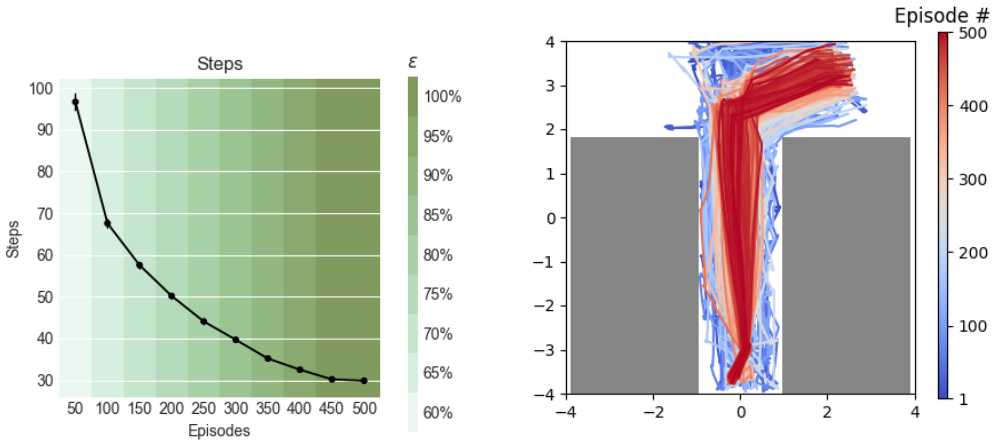


Figure 6.9: The corridor task. Left: Rendering of the environment; Right: Floor plan



(a) Steps to goal

(b) Trajectories

Figure 6.10: Results from the corridor task

Flat penalty versus vision based penalty (6.1.3): It is shown that the agent learns faster with a vision-based penalty.

Comparison of Performance (6.1.4): A rule based autopilot is designed (based on intuition) to estimate the steps required to perform the two tasks with random initializations. The simulations reveal that the state description is insufficient to learn the task. The memory of hitting a wall only lasts for one step which is insufficient information to avoid a wall, and track a goal at the same time.

The mean number of steps required to reach the goal by the rule based autopilot with random initializations is found to be near the optimal number with fixed initialization. This mean is calculated by excluding the episodes where rule based the autopilot got stuck on the wall due to the aforementioned lacking in the state description. The exclusion of these episodes in the calculation of its mean performance makes its performance appear better than it really is.

The comparison with the absolute position based autopilot reveals that it can perform the one goal task 15 seconds faster than the trained agent. This is expected as the RL agent takes discrete steps towards the goal. The autopilot, based on absolute position information and the use of waypoints, uses a PID controller to guide itself smoothly and continuously to the goal.

Random vs ϵ-greedy exploration (6.1.5): The study found ϵ-greedy exploration with increasing ϵ to be better for the convergence of the learning.

Corridor task (6.1.6): A corridor traversal task is trained with the designed vision-based RL agent. The agent manages to learn the task with the defined agent in simulation.

RGB and YUV based computer vision (6.1.7): There are effects in the flight test which are not fully analyzed

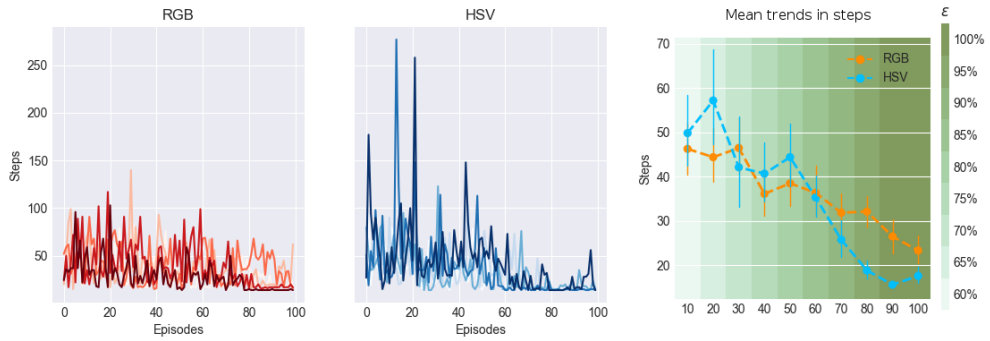


Figure 6.11: Learning in flight tests using RGB and YUV color spaces for processing vision.

due time constraints of the project. The limited amount of tests show that the YUV based system has a higher variance in the steps to reach the goal during learning than the RGB based system. The results also indicate that the final performance of the YUV based agent is better. However, as only five repetitions are performed, the data is not statistically significant.

The comparison of the trained agent with the rule based controller (in Chapter 2) and absolute information based controllers are answers to **RQ4.1** and **RQ4.2**. A task where the agent has to learn to move forward and turn past a corridor is performed with the RL agent designed for the two goal task, answering **RQ4.3**. The results of experiments about the transfer of knowledge between simulation and real life (**RQ4.4**) is presented in Chapter 2. With the answers to these sub-questions, **RQ4** "What is the performance of the designed controller" has been answered.

All the research questions posed in this project have been answered. Chapter 7 sums up the findings from this project and makes recommendations about future work on the topic.

7
Conclusion

The design and implementation of a vision-based Reinforcement Learning (RL) controller for the guidance and navigation of a AR Drone 2 has been described. The developed controller is a contribution to the field of autonomous flight of Unmanned Aerial Vehicles (UAVs) as it does not use absolute position information and is capable of learning.

The method proposed in this project uses pixel values for the detection of goals and Q-learning to find ways to approach the goal. The state of the art for vision-based guidance and navigation in UAVs is Simultaneous Localization and Mapping (SLAM). SLAM is computationally expensive and often requires a laser range finder. The computations in the method developed in this study are fast. However, the developed method is not a generalized solution like SLAM which has been shown to be applicable for the guidance and navigation in generic in-door environments(Bachrach et al., 2011; Grzonka et al., 2012). The state used by the RL agent is designed for the specific task of travelling towards (or between) colored goals on the walls in an obstacle free room [1].

The goal of this project was to **"design a reinforcement learning guidance controller for an AR Drone 2 that uses vision sensed features for state estimation and reward generation"**. Five research questions were posed to organize the project and structure the process of achieving the aforementioned goal. The questions are regarding the state of the art in vision-based RL for UAVs (**RQ1**), conceptual design of the state, task and agent for this study (**RQ2**), the implementation (**RQ3**) and testing (**RQ4**) of the RL agent.

RQ1 has been answered in Chapter 3 through the discussion on recent work in RL, computer vision and vision-based RL in robots. Following this a preliminary study in the effect of vision-based states are carried out by simulations in gridworld; the results from the gridworld study are presented in Chapter 4. Concepts for the vision-based state and the guidance task, along with design of the RL agent are presented in Chapter 5; these descriptions are an answer to **RQ2**. The implementation of the vision-based RL agent in a simulation and later in flight tests as described in Chapter 2 answering **RQ3**. Finally, the performance of the RL agent is compared to hardcoded controllers and its ability to perform another similar task studied in Chapter 6. The discussions in Chapter 6 along with the study to see how policies learned in simulation transfer to real life in Chapter 2 answers **RQ4**. Through the answers to the research questions, the goal of the design of vision-based RL guidance controller for an AR Drone 2 has been achieved.

7.1. Findings

A gridworld task is defined which consists of the agent travelling to two goal grids in a 9 by 9 gridworld. The results from this project are discussed in the following paragraphs. The vision-based agent in the gridworld simulation does not learn the task better with more training. The ambiguity in the state description for the gridworld task is too high to allow the learning of an optimal policy. A ϵ-greedy policy with 50% random actions is found to be optimal in terms of the steps required by the agent to travel to two goals in the gridworld. This also means Q-learning is not applicable for the gridworld task with the vision-based agent as fully greedy action causes divergence in the learning with the vision-based states. Further, interactions between the state description, the action space and the reward function can lead to situations where non-optimal sequences

[1] And the room is considered to have no other object with the color of the goals

of actions result in higher sum of rewards than optimal sequences of actions. Thus, the agent will learn non-optimal action.

Three vision-based guidance and navigation tasks are defined. The defined tasks consist of travelling to a red goal in a room (one goal task), travelling to a red goal and a blue goal in a room (two goal task) and turning past a corridor to travel to a red goal (corridor task). The simulation of the defined guidance task is carried out by using PaparazziUAV and FlightGear. Q-Learning is used to train the agent as the task and the state definition allows the learning of an optimal policy. Simulations are carried out to show that the learning is better with an ϵ-greedy exploration than with random exploration.

The incorporation of external knowledge improves learning. An *option*[2] is designed to decrease the ambiguity in the state and a reward scheme is used which encourages going forward when seeing the goal through the reward function. Both of these improve the learning. The performance of the trained agent is compared in simulation to a rule based controller and an autopilot using absolute position information. The trained agent performs[3] as good as the rule based controller for the one goal task and nearly as good as the rule based controller for the two goal task. The absolute position based agent is found to perform the one goal task over twice as fast as the trained agent; 30 seconds for the RL agent and 15 seconds for the absolute position based autopilot.

The agent is trained in flight tests to demonstrate the learning in real life. The agent is trained the task of travelling to one goal for 100 episodes and the training is repeated 5 times. The real life agent learns the task but performs worse than the simulated agent. The differences between the performance in simulation and the performance in real life are as a consequence of the following factors:

- the effect of the tether on the quadcopter dynamics
- the noise in the dynamics of the real life quadcopter
- the noise in the vision perception
- discrepancies between the tuning of the simulated and real life vision modules
- the variance in the initialization of the flight tests
- fixed update rate of the vision state perception in real life implementation

When Q-values learned in simulation are tested in fight tests, the real life agent shows a nearly constant offset in performance with the simulated agent for higher values of ϵ (i.e. for more greedy actions). There is a delay in the state perception of the real life agent which is as a result of the fixed update rate of vision-based state in the AR Drone 2. This delay is believed to be the cause of the offset in performance when transferring Q-values learned in simulation to real life.

7.2. Impact

There are numerous application for systems which can autonomously learn to guide and navigate themselves in GPS denied environments. The development of a learning and vision-based guidance system for UAVs will broaden their domain of operation and consequently increase their demand. This technology will enable the development of generic Unmanned Aerial Systems (UAS) that can be targeted towards specific markets.

For example, a generic learning and vision-based UAS that is designed for checking the inventory will be usable in warehouses, supermarkets, workshops etc. Using learning and vision-based systems will remove the need to setup a local positioning system or programming the UAS autopilot for the environment on an ad-hoc basis. Furthermore, as one software architecture can be used to learn different tasks, there are potential savings in terms of software development for the UAV manufacturers. If vision-based learning can be made generic, it can potentially be marketed to mass consumers who train their UAVs for specific applications. As the trained agent takes over the task of guiding the UAV, operating them will require thus less man power.

Moreover, the developments of such systems will have effects on society which are hard to predict. New regulatory bodies will need to be developed to certify the systems and operators of learning based UAVs as the one presented in this study. Social norms and outlooks will need to adapt to the emergence of mobile robots that are not strictly defined by their programming, but have the capacity to improve on their designed roles over time, by themselves. Before the mass marketing of learning based UAS, the manufacturers will have to

[2] a action that lasts over multiple steps(Sutton et al., 1999)

[3] The performance metrics are steps to reach the goal for comparison with the rule based autopilot and time to finish the task for comparison with the absolute position based autopilot.

ensure the safety and security of their owners and the societies where they are being marketed. Typical questions pertaining to the use of artificial intelligent systems in daily life will need to be answered. Questions such as, who will be held accountable for damages caused by the autonomous operation of these systems and how to ensure the livelihood of people whose jobs are going to be replaced by such systems, will need to be answered.

7.3. Limitations

Some of the limitations of this study and the developed controller are discussed in the following paragraphs. The differences between the real life environment and the simulated environments are not studied in detail. Further, the hyperparameters of the designed RL agent are not tuned. Therefore, the agent can potentially learn faster with better tuning of the parameters. The guidance and navigation task is framed such that a RL agent will be able to learn it with Q-learning. The implemented RL agent is simple and more complex implementations may outperform the designed agent. The simulation has many differences with real life (see Section 7.1) which makes the simulations findings less representative of real life and decreases its external validity.

Additionally, the simulation cannot be time accelerated past a limit due to its high computation costs. The images are obtained by taking screenshots of FlightGear's rendering of the environment. The rendering of images is computationally expensive. If the simulation is accelerated past a certain time factor, the rendering cannot keep up with the simulation and the agent gets erroneous state information. As a consequence, the time acceleration has to be capped at a point[4].

The developed system lacks many elements which are desired from autonomous UAS. First and foremost, the real life training is carried out with a tethered power supply; a UAV loses most of its functionality if it has to be tethered to a power supply. Secondly, the tasks in this study are simple and not highly representative of real world situations. Thirdly, the agent is incapable of adapting to new information and lastly, the solution implemented is specific to the defined tasks.

The performance of the agent is compared to a rule based controller and an absolute position based controller. In terms of the steps required to perform the task, the trained agent is as good as the rule based autopilot for the one goal task but worse for the two goal task. In terms of time to perform the task, the trained agent takes over twice as much time as the absolute position based autopilot for the one goal task. Thus, the performance of the developed controller is satisfactory.

7.4. Further Work

The work carried out in this project can be continued by:

Improvements to state description Not using colors for goal object detection will relax the requirements on the environment. All objects of specific colors will not have to be removed from the environment. The goal object can be simple, such as shapes or bar codes. But the recognition of more complex indoor objects will allow the design of states which will be usable for performing more realistic tasks.

The ability to recognize stoves, beds, couches and toilets will allow a vision-based quadcopter agent to distinguish between different rooms in a house. Similarly, being able to interpret numbers will allow the UAV to learn paths of travel between labeled racks in a warehouse, for instance.

The raw image data can be used as the state if a Deep Q-learning (DQL)(Mnih et al., 2015) framework is used. However, a MSc. thesis into the applicability of DQL to quadcopter guidance(Kisantal, 2018) revealed many challenges and complexities of such an approach.

A simple addition to the state description can be a better wall and/or obstacle detection. This will allow the agent to learn to avoid walls or obstacles while performing guidance and navigation tasks.

Improvements to the action space The work in this project showed that the incorporation of external knowledge into the actions can improve the agent's performance. The action space can be extended further, improving the capacity of the agent to act optimally and increasing the range of things that can be done

[4]During this study the limit was found to be around a time factor of 5; a higher time factor can be obtained with better computational hardware

by it. For example, with a continuous action space the agent will have the ability to perform the tasks faster. The agent can be allowed to make translational movement (i.e. surge and sway).

Training more complex tasks The initial concept of the guidance task (see Section 5.2, **RLTask1**) was significantly simplified in this project in order to enable a real-life implementation. Now that a real-life implementation has been carried out, studies can be performed with more complex tasks.

Exploration of other RL frameworks This project focused on a simple RL method: table based Q-learning. One extension to that is using Monte Carlo (MC) methods for learning. The non-Markovness of the state degrades the performance of the agent. MC learning methods suffer less from the non-Markovness of the state.

The incorporation of continuous states and actions implies changes to the RL framework. With continuous states and/or actions some variant of the Actor-Critic method will have to be used with function approximators to represent the values.

The application of model based RL methods are interesting as having a model of the transitions enables some new applications. For example, an agent can be designed that uses a vision-based state description to locate itself in rooms of a house (like **VisSta3** in Section 5.1.2). If the agent learns the transition model between the different rooms, it can later use a different reward function and dynamic programming to find paths between different points in the house.

Safe exploration and adaptability are not addressed in this study. The work on safe exploration in (Mannucci et al., 2018) can be used to incorporate safe exploration into the developed agent.

Lastly, Hierarchical RL can be used with vision-based descriptions of the state to potentially tackle complex guidance and navigational task. The challenge in terms of the time required for learning of tasks such as **RLTask1** can be mitigated by dividing the state and action space into subdomains and applying Hierarchical RL to learn the task.

Performing the task without position feedback The implemented controller uses position feedback from the motion sensing system for the control of the quadcopter. The information is used for stabilization and for navigation[5]. The position feedback from the motion sensing system removes the drift in the heading and position that will be experienced by Attitude and Heading Reference System (AHRS) without absolute position information. Optic flow can be used for stabilization, but if the environment does not have many edges and/or visually distinct elements, optic flow methods will not work. Monocular depth perception techniques with feature detection can potentially be used for closing position and heading loops. Like optic flow, this will require visually identifiable elements in the environment along with a relatively complex computer vision implementation.

Implementation in fixed wing UAV Guidance tasks for fixed winged UAVs have been learned in simulation using vision information(Valasek et al., 2016). Such controllers can be implemented in PaparazziUAV and flight tested on fixed wing UAVs.

Improvements to the simulation framework The simulation grabs screenshots of FlightGear. This is wasteful as the image data can be directly taken from FlightGear, skipping this computation heavy rendering step. This is not a contribution to the autonomous flight of UAVs, but it can be part of a work which aims to use vision-based state descriptions and needs a faster simulation environment.

Design of "Battery replacement and recharging station" The AR Drone 2 has a stated flight time of approximately 12 minutes; the learning is found to take nearly 2 hours. The real life training of the RL agent is only possible as it uses a tethered power supply. As the real quadcopter is incapable for such a prolonged flight, this system is inapplicable to nearly all applications where a GPS free navigation and guidance is required. A design of a "Battery replacement and recharging station" would be a good contribution to the real life autonomy of quadcopters.

Identification of the visual noise in the Cyber Zoo The Cyber Zoo is used for numerous vision-based tests of UAVs. Most of these tests are influenced the effect of noise in the vision data which results from the Cyber Zoo itself, objects around it and the time of day. Studies can be carried out to quantify the noise so that its effect on vision-based experiments can be better understood and ways of mitigating the noise suggested.

[5]The RL agent only uses the vision information to make its decisions.

This chapter ends the report documenting the work carried out for the development of a vision-based RL guidance controller for an AR Drone 2.

A

Gridworld Implementation Details

The gridworld simulations are implemented in Python 2.7.6. Four external libraries are used in the implementation. The *Pandas* library is used to record the Q-tables and the *Numpy* library is used for computations. *Matplotlib* and *Seaborn* are used for visualization. The built-in libraries *copy*, *itertools*, *sys* and *pickle* are used to incorporate other functionalities into the simulation tool. A tool to visualize the episodes is created using the library *pygame*. This "episode visualizer" is used to debug the code and observe agent behavior.

An object oriented approach is used in the implementation of the simulation. Two major classes are used to create the simulation; the *environment* class and the *exp_template* class. The *environment* class contains the gridworld of the environment and methods to transition the agent. It contains an array that represents the locations of the agent and the objects in the environment.

The rest of the simulation is in the *exp_template* class. It controls both the agent and the experiment (i.e., the learning, recording of data, visualization of data). The *exp_template* contains a *reward* object which handles the rewards during learning. It may also contain a *vision* object which is responsible for the simulated vision of the agent; an instance of this object is only created when a vision-based state is used. The *vision* class consists of numerous *grid* objects based on how the exact vision state is defined.

Note that since the program is in development, the code is disorganized and volatile. For any questions or queries, do not hesitate to get in touch with the author.

B

PaparazziUAV Simulation Implementation

The simulation is implemented using PaparazziUAV and FlightGear. The reasons for using PaparazziUAV are its built-in flight simulator, its ability to interface with FlightGear and the fact that it is used to compile the autopilot software for tests on the R Drone 2. Likewise, FlightGear is used since it can be used to send screenshots of the 3D world simulated to PaparazziUAV. The developed program has been tested in FlightGear version 2017.2.1 and PaparazziUAV version 5.13.

B.1. PaparazziUAV

PaparazziUAV is an "open-source drone hardware and software project" [1]. A module named "my_visrl" is created which is used for both the simulation and the real life agent. A core set of RL functions are implemented in the *visrl.c* file Vision modules are implemented separately for the simulation and the real life autopilot in the files *vis_nps* and *vis_ap* respectively. The module consists of six files, namely: visrl, simsoft, mynavfuncs mydict, vis_ap and vis_nps.

visrl: All of the RL elements are implemented in this file; the state, actions, and Q-value updates are the agent functionalities that are implemented. Besides the agent, there are functions which parses the vision data into states and numerous other helper functions. The functions which implement the RL have been tabulated in Table B.1.

Table B.1: Table of functions in *visrl.c*

Name	Description
pick_action	picks action based on an ϵ-greedy policy
get_state_ext	Runs the computer vision function from manan_test and uses it to format the state for the RL agent. Also implements the logic for remembering which goals have been visited.
rl_init_ep	Initializes an episode for starting the RL. Resets numerous counters and booleans and calls the rl_set_nxt function.
rl_set_cur	Sets the "cur_sta" and "cur_act" to the values of "nxt_sta" and "nxt_act".
rl_get_reward	Uses the vision information and wall hit information to come up with a reward for the current state-action pair.
rl_set_nxt	Sets the "nxt_sta" and "nxt_act" variables. The "nxt_sta" is set by a call to get_state_ext and the "nxt_act" is set by a call to pick_action
rl_take_cur_action	Looks at the "cur_act" variable to decide if to go forward, turn left or turn right. In case an action causes a hit with the wall it sets the value of the "hitwall" variable to 1, otherwise it is set to 0.
rl_update_qdict	Looks up the value of the current and next state-action pairs in the Q-table and makes a SARSA update based on the reward and the values.
rl_check_terminal	Checks if the terminal condition for the episode are met; at this stage checks if both goals are visited which is indicated by one of the numbers in the state.

[1] `http://wiki.paparazziuav.org/wiki/Main_Page`

simsoft: Functions for setting up the simulation environment, logging the data and carrying out multiple runs of the training are implemented in this file.

mynavfuncs: There are 6 navigation functions in the "mynavfuncs.c" file. The functions have been tabulated in Table B.2.

Table B.2: Table of navigation functions in *mynavfuncs.c*

Name	Description
set_nav_heading	Sets the heading to the provided input in radians.
increase_nav_heading	Increases the heading by the input value in radians.
moveWaypointForwards	Moves the given waypoint forward by the given distance.
moveWaypointLeftwards	Moves the given waypoint left by the given distance.
setHeadingNorth	Smoothly sets the heading to north. Due to internal limitations of PaparazziUAV it cannot handle big heading changes; this limitation led to this function.
setAltToWp	Sets the altitude of one waypoint to that of anothers.

mydict: This file implements a linked list data structure and the corresponding functions required to add, remove and search through the list. This is required to hold the Q-table for the RL task. Functions required to print the Q-table to text file and read it from a text file are also implemented in this file.

vis_nps: The required functions to obtain screenshots from FlightGear and process them to generate outputs required for the vision state are implemented in this file. These functions are described in Table B.3.

Table B.3: Functions in *vis_nps.c* file

Name	Description
WriteMemoryCallback	Helper function for curl2mem. Handles the writing of the data from the source HTTP into the memory. Needs to follow format provided by libcurl.
curl2mem	Uses libcurl functions to load the screenshot from the FlightGear HTTP server into the programs memory.
get_bmp	Decompresses the jpeg image obtained from curl2mem into a bitmap containing the pixel data.
get_pointertopix	Helper function that returns the pointer to a pixel given its row and column. Assumes that each pixel contains 3 bytes of information.
colmax	Takes the 3 by 3 array of the RBG counts of each grid as input and writes a 3 element array which represents the color with the maximum number of pixels above threshold for each grid.
count_pixels_in_three_grids	Counts the RBG pixels above internally defined thresholds for each of the 3 grids. It also calculates the sum of RGB pixels above the threshold in the whole image and the color of the most number of pixels above threshold for each of the grid. This information is made available through the arrays count_arr, sumcount_arr and domcol_arr.
cv_3grids	This function implements the whole computer vision function by downloading an image to memory using curl2mem, than decompressing it using get_bmp and finally estimating the pixels above threshold in 3 grids using count_pixels_in_three_grids

vis_ap: The required functions to process the vision information from the UAV cameras are defined in this file. This is needed for the real life implementation of the computer vision. The functions in the file are described in Table B.4.

Table B.4: Function in the *vis_ap.c* file

Name	Description
colmax	Same as colmax in Table B.3.
my_image_yuv422_colorcounter	Same as count_pixels_in_three_grids in Table B.3 with the exception that HSV is used for pixel detection instead of RGB.
visrl_cv_func	This function packages the my_image_yuv422_colorcounter function so it can be added to the computer vision thread of PaparaazziUAV.
vis_ap_init	This function is called when visrl module is initialized and it adds visrl_cv_func to the computer vision thread of the autopilot.
cv_3grids	This is dummy function that does not do anything; it is included in the vis_ap to remove errors that were showing up with the compilation.

Flight plan All the elements of the RL agent that are defined in the module are implemented through the flight plan. A visual overview of the elements and logic of the flight plan is presented in Figure B.1.

B.2. FlightGear

FlightGear can take the state data from PaparazziUAV in order to render of the UAV's environment. A model of the CyberZoo is placed at a specific longitude and latitude inside FlightGear's database of 3d objects[2]. The origin of the flight plan in PaparazziUAV is made to coincide with the longitude and latitude of where the room was placed in FlightGear.

B.3. Running the Simulation

FlightGear and the PaparazziUAV simulator need to be run with specific arguments for the simulation to work. It is better to manually start the PaparazziUAV simulator from a console[3] as that allows the user to increase the simulation upto a certain limit. Due to certain limitations of the implementation **the FlightGear needs to be started first**; after that the flight simulator from PaparazziUAV should be started. The commands for starting them are included below.

Command to start FlightGear:

```
fgfs --fdm=null --native-gui=socket,in,30,,5501,udp --aircraft=ufo
    --httpd=1234 --config=$HOME/.fgfs/Export/ufo-model-export.xml --com2=121.9
    --timeofday=morning --enable-clock-freeze --geometry=800x600
    --prop:/sim/menubar/visibility=false
```

Command to start PaparazziUAV simulator:

```
$PAPARAZZI_HOME/sw/simulator/pprzsim-launch -a ardrone2 -t nps --fg_host 127.0.0.1
```

After both the simulators have started, one can use run the ground control station and the PaparazziUAV server and datalink from the PaparazziUAV application (i.e. the Paparazzi Center [4]).

[2]Object is placed by following the guidelines in a FlighGear Wiki page: `http://wiki.flightgear.org/Howto:Place_3D_objects_with_the_UFO`

[3]Documentation about how to start the simulator from the console can be found at "Running the Simulation" section of the website `https://wiki.paparazziuav.org/wiki/NPS`

[4]`https://wiki.paparazziuav.org/wiki/Paparazzi_Center`

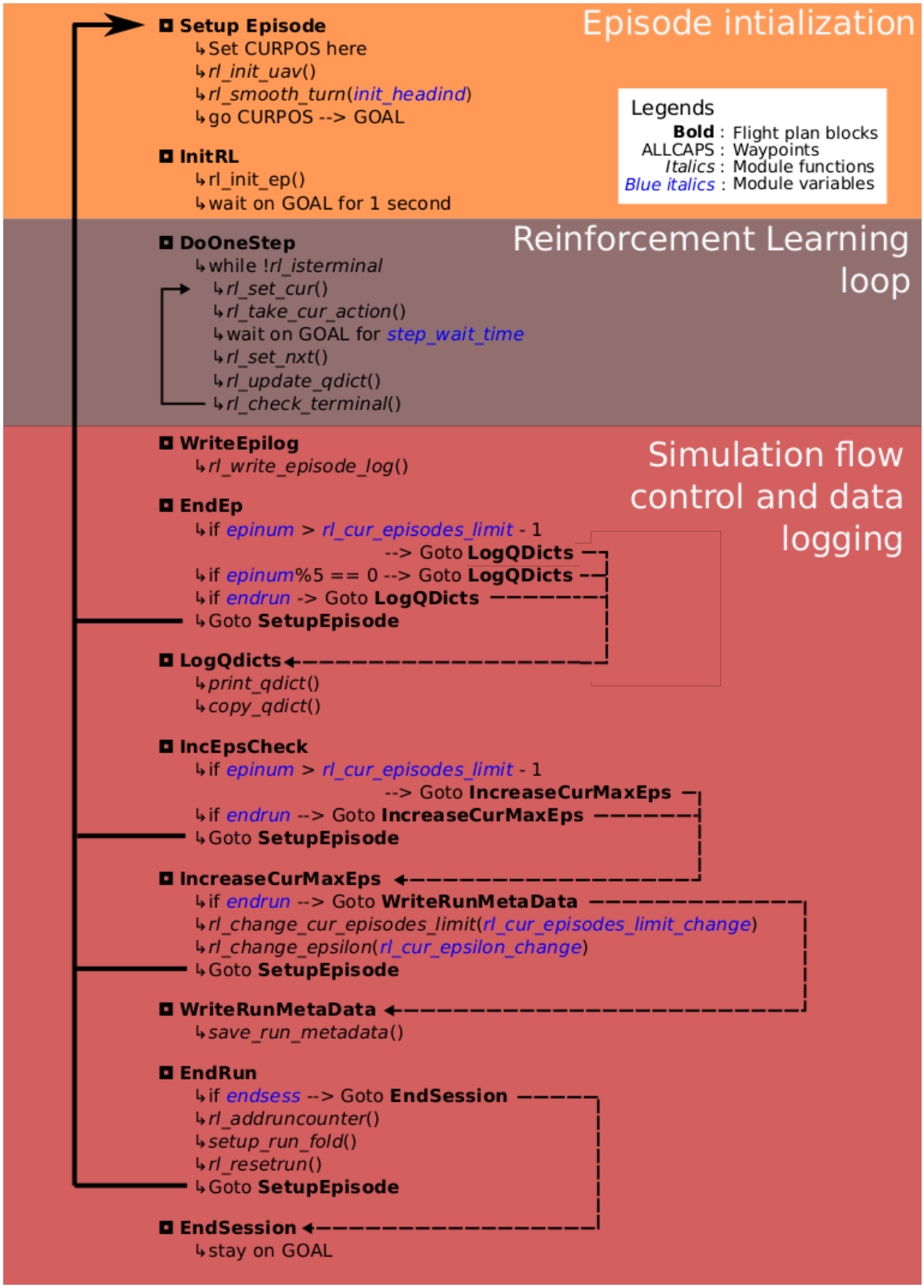


Figure B.1: The general flight plan for the RL agent

References

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, *19*, 1.

Bachrach, A., Prentice, S., He, R., & Roy, N. (2011, September). RANGE–Robust autonomous navigation in GPS-denied environments. *Journal of Field Robotics*, *28*(5), 644–666. doi: 10.1002/rob.20400

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, *110*(3), 346–359. doi: 10.1016/j.cviu.2007.09.014

Bipin, K., Duggal, V., & Madhava, K. (2015). Autonomous navigation of generic monocular quadcopter in natural environment. In (Vol. 2015-June, pp. 1063–1070). doi: 10.1109/ICRA.2015.7139308

Bonin-Font, F., Ortiz, A., & Oliver, G. (2008, May). Visual Navigation for Mobile Robots: A Survey. *Journal of Intelligent and Robotic Systems*, *53*(3), 263. doi: 10.1007/s10846-008-9235-4

Briggs, A. J., Scharstein, D., Braziunas, D., Dima, C., & Wall, P. (2000). Mobile robot navigation using self-similar landmarks. In *IEEE International Conference on Robotics and Automation* (Vol. 2, pp. 1428–1434). IEEE; 1999. Retrieved 2017-01-14, from `https://pdfs.semanticscholar.org/4d2b/2ab5292af0530d7f36b466c03b3bbcc6ce3a.pdf`

Choi, J.-M., Lee, S.-J., & Won, M. (2011). Self-learning navigation algorithm for vision-based mobile robots using machine learning algorithms. *Journal of Mechanical Science and Technology*, *25*(1), 247–254. doi: 10.1007/s12206-010-1023-y

Cicirelli, G., D'Orazio, T., & Distante, A. (2005). Different learning methodologies for vision-based navigation behaviors. *International Journal of Pattern Recognition and Artificial Intelligence*, *19*(8), 949–975. doi: 10.1142/S021800140500440X

Davison, A., Reid, I., Molton, N., & Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(6), 1052–1067. doi: 10.1109/TPAMI.2007.1049

Durrant-Whyte, H., & Bailey, T. (2006, June). Simultaneous localization and mapping: Part I. *IEEE Robotics Automation Magazine*, *13*(2), 99–110. doi: 10.1109/MRA.2006.1638022

Gaskett, C., Fletcher, L., & Zelinsky, A. (2000). Reinforcement learning for a vision based mobile robot. In *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings* (Vol. 1, pp. 403–409 vol.1). doi: 10.1109/IROS.2000.894638

Grisetti, G., Kummerle, R., Stachniss, C., & Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, *2*(4), 31–43. Retrieved 2017-01-14, from `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5681215`

Grzonka, S., Grisetti, G., & Burgard, W. (2012, February). A Fully Autonomous Indoor Quadrotor. *IEEE Transactions on Robotics*, *28*(1), 90–100. doi: 10.1109/TRO.2011.2162999

Hutter, M. (2009). Feature reinforcement learning: Part I. unstructured MDPs. *Journal of Artificial General Intelligence*, *1*(1), 3–24.

Jimenez, A. R., Ceres, R., & Pons, J. L. (2000). A survey of computer vision methods for locating fruit on trees. *Transactions of the ASAE*, *43*(6), 1911.

Junell, J., Van Kampen, E.-J., de Visser, C., & Chu, Q. (2015). Reinforcement learning applied to a quadrotor guidance law in autonomous flight. In *Aiaa guidance, navigation, and control conference* (p. 1990).

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237–285.

Kendoul, F. (2012, March). Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, *29*(2), 315–378. doi: 10.1002/rob.20414

Kendoul, F., Fantoni, I., & Nonami, K. (2009, June). Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, *57*(6–7), 591–602. doi: 10.1016/j.robot.2009.02.001

Kim, J., & Sukkarieh, S. (2004, July). Autonomous airborne navigation in unknown terrain environments. *IEEE Transactions on Aerospace and Electronic Systems*, *40*(3), 1031–1045. doi: 10.1109/TAES.2004.1337472

Kisantal, M. (2018). *Deep Reinforcement Learning for Goal-directed Visual Navigation.* MSc. Thesis. Retrieved 2018-03-22, from http://resolver.tudelft.nl/uuid:07bc64ba-42e3-4aa7-ba9b-ac0ac4e0e7a1

Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on* (pp. 225–234). IEEE. Retrieved 2017-01-14, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4538852

Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, *32*(11), 1238–1274. doi: 10.1177/0278364913495721

Lin, G. Y., & Chen, X. (2011). Robot indoor navigation method based on 2D barcode landmark. In *Applied Mechanics and Materials* (Vol. 44, pp. 1279–1284). Trans Tech Publ. Retrieved 2017-01-14, from http://www.scientific.net/AMM.44-47.1279

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* (Vol. 2, pp. 1150–1157). Ieee.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91–110. Retrieved 2016-12-07, from http://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94

Mannucci, T., Kampen, E. J. v., Visser, C. d., & Chu, Q. (2018). Safe Exploration Algorithms for Reinforcement Learning Controllers. *IEEE Transactions on Neural Networks and Learning Systems*, *PP*(99), 1–13. doi: 10.1109/TNNLS.2017.2654539

Michels, J., Saxena, A., & Ng, A. Y. (2005). High Speed Obstacle Avoidance Using Monocular Vision and Reinforcement Learning. In *Proceedings of the 22nd International Conference on Machine Learning* (pp. 593–600). New York, NY, USA: ACM. doi: 10.1145/1102351.1102426

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Salichs, M. A., & Moreno, L. (2000, May). Navigation of mobile robots: Open questions. *Robotica*.

Shen, S., Michael, N., & Kumar, V. (2013, 12). Obtaining Liftoff Indoors: Autonomous Navigation in Confined Indoor Environments. *IEEE Robotics Automation Magazine*, *20*(4). doi: 10.1109/MRA.2013.2253172

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT press Cambridge.

Sutton, R. S., Precup, D., & Singh, S. (1999, August). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1-2), 181–211. doi: 10.1016/S0004-3702(99)00052-1

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58–68.

Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., ... Burschka, D. (2012, September). Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue. *IEEE Robotics Automation Magazine*, *19*(3), 46–56. doi: 10.1109/MRA.2012.2206473

Valasek, J., Kirkpatrick, K., May, J., & Harris, J. (2016). Intelligent Motion Video Guidance for Unmanned Air System Ground Target Surveillance. *Journal of Aerospace Information Systems*, *13*(1), 10–26. doi: 10.2514/1.I010198

Valavanis, K. P. (2008). *Advances in unmanned aerial vehicles: state of the art and the road to autonomy* (Vol. 33). Springer Science & Business Media. Retrieved 2016-10-31, from https://books.google.nl/books?hl=en&lr=&id=EsjPyblwMdQC&oi=fnd&pg=PR11&dq=+Advances+in+unmanned+aerial+vehicles:+state+of+the+art+and+the+road+to+autonomy&ots=EPqQW5AFpC&sig=raqsTrGopIsDXpbRiB3u9N1_XDo (00347)

Viola, P., & Jones, M. J. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision*, *57*(2), 137–154. doi: 10.1023/B:VISI.0000013087.49260.fb

Weiss, S., Scaramuzza, D., & Siegwart, R. (2011, November). Monocular-SLAM–based navigation for autonomous micro helicopters in GPS-denied environments. *Journal of Field Robotics*, *28*(6), 854–874. doi: 10.1002/rob.20412

Wiering, M., & Van Otterlo, M. (2012). *Reinforcement learning* (Vol. 12). Springer.

Wörgötter, F., & Porr, B. (2005). Temporal sequence learning, prediction, and control: A review of different models and their relation to biological mechanisms. *Neural Computation*, *17*(2), 245–319. Retrieved 2017-05-14, from http://www.mitpressjournals.org/doi/abs/10.1162/0899766053011555

Xu, X., Zuo, L., & Huang, Z. (2014). Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences, 261*, 1–31. doi: 10.1016/j.ins.2013.08.037

Zhang, Z. (2000, 11). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(11), 1330–1334. doi: 10.1109/34.888718