Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics

Scheper, Kirk

DOI
10.4233/uuid:389f453e-f7ff-4fea-a353-32755cf9a9e1

Publication date
2019

Document Version
Final published version

Citation (APA)
Scheper, K. (2019). *Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:389f453e-f7ff-4fea-a353-32755cf9a9e1

Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics

Kirk Yannick Willehm Scheper

Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics

Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
dinsdag 10 september 2019 om 12:30 uur

door

Kirk Yannick Willehm Scheper

Ingenieur Luchtvaart- en Ruimtevaarttechniek
Technische Universiteit Delft, Nederland
geboren te Castries, St. Lucia

Dit proefschrift is goedgekeurd door de promotoren

Samenstelling promotiecommissie bestaat uit:

Rector Magnificus	voorzitter
Prof. dr. ir. M. Mulder	Technische Universiteit Delft, *promotor*
Dr. G.C.H.E. de Croon	Technische Universiteit Delft, *promotor*
Onafhankelijke leden:	
Dr. E. Tuci	Université de Namur, Belgium
Prof. dr. A.E. Eiben	Vrije Universiteit Amsterdam
Prof. dr. K.P. Tuyls	University of Liverpool, United Kingdom
Prof. dr. ir. M. Wisse	Technische Universiteit Delft
Prof. dr. D.G. Simons	Technische Universiteit Delft

Trefwoorden: Evolutionaire robotica, reality gap, abstractie, robuust gedrag, micro-luchtvoertuigen

Gedrukt door: Off Page, www.offpage.nl

Voor- en Achterkant: Kirk Y.W. Scheper (photo by Jaime L. Junell)

ISBN 978-94-6366-197-3

Een elektronische versie van dit proefschrift is beschikbaar op
http://repository.tudelft.nl/.

Logic is the beginning of wisdom, not the end.

- Spock

Star Trek VI: The Undiscovered Country

Contents

Summary

Automatically optimizing robotic behavior to solve complex tasks has been one of the main, long-standing goals of Evolutionary Robotics (ER). When successful, this approach will likely fundamentally change the rate of development and deployment of robots in everyday life. Performing this optimization on real robots can be risky and time consuming. As a result, much of the work in ER is done using simulations which can operate many times faster than realtime. The only downside of this, is that, due to the limited fidelity of the simulated environment, the optimized robotic behavior is typically different when transferred to a robot in the real world. This difference is referred to as the *reality gap*.

Of the several methods developed to find a bridge to the reality gap, one of the most effective is the *envelope of noise* by Jakobi. This work showed that if you apply appropriate levels of noise in simulation to the environment and robot, the optimized behavior is more robust to the differences between simulation and the real world. Adding noise in this way is a form of abstraction, separating the vehicle from reality. This work showed that abstracting away from the real world can increase the reliability of the optimized behavior. Despite the fact that this added noise limits the ability of the optimization to find the truly optimum behavior, the trade-off in favor of robustness is arguably worth it.

Abstraction is a tool often used in several fields to improve robustness of controlled systems. A good example of this is in control theory, where cascaded or nested control systems have several properties that help to improve robustness. The closed-loop control scheme helps to actively reduce errors caused by small differences in the vehicle manufacturing and assembly or environmental disturbances. This thesis investigates how abstraction can be used to improve the robustness of the robotic behavior to the reality gap, driven by the following problem statement:

How can abstraction be used to bridge the reality gap in evolutionary robotics?

In this thesis, we investigate three properties of closed-loop control systems that can help bridge the reality gap.

First, the cascaded or nested structure of the control system helps to segment the larger control problem into a set of smaller, simpler and more tractable problems. This improves the ability of the control designer to identify and target shortcomings in the control scheme. This is in stark contrast to typical ER approaches which use neural networks to control the robotic platform. These *black-box* controllers try to optimize a complex, global solution to the task at hand. Although powerful, this

approach often results in fragile behavior which quickly breaks down when moving to the real world. Abstracting away from these global controllers may improve robustness. This concept leads to the first research question:

RQ1: How can abstraction in the behavioral representation help in crossing the reality gap?

To answer this question, we investigated the use of a more intelligible behavioral representation for the robotic behavior. This was tested by optimizing behavior for the 20 g DelFly Explorer flapping-wing Micro Air Vehicle (MAV) using a *Behavior Tree* representational structure. The DelFly was tasked to fly around a small room, avoiding the walls while searching for, and eventually flying through, an open window.

After being trained in simulation with success rates above 85%, there was a clear reality gap when testing on the real vehicle which initially showed a 0% success rate. The intelligible nature of the Behavior Tree allowed the human operator to quickly understand the desired robotic behavior and identify the source of the reality gap, in this case due to biases in the sensing and actuation of the real world vehicle. The structure of the Behavior Tree also easily lends itself to simple modification. This allowed the user to update the behavior, actively reducing the reality gap and improving performance to 54%. Although lower than observed in simulation, this result was higher than the baseline user-designed behavior at 46%. The remaining difference in performance between the simulated and real flights is likely due to the unmodeled flight dynamics of the DelFly and the presence of unmodeled external disturbances such as gusts.

The Behavior Tree achieves this increased intelligibility at the cost of representational power as compared to the neural network. With its subsumption-like architecture, the behavior abstracts away from a global input-output mapping, segmenting the actions to a finite set of options. For the behavior to remain intelligible, the trees must remain small, somewhat limiting the ability of the evolutionary optimization to find the ideal solution to the problem. That said, without the intelligibility of this representational structure the optimized behavior would have been useless once transferred. This trade-off therefore seems worthwhile. Abstracting the behavioral representation can help to speed-up the re-optimization process by segmenting the behavior into simpler sub-behaviors.

Second, closed-loop control systems use feedback to continuously reduce any error between a desired state and the actual vehicle state. Most implementations of ER tend to use the simplest control method, directly driving the actuators of the robot with open loop control and directly feeding raw sensor information to the control system, with little to no preprocessing. This leads to an inherent sensitivity of the behavior to the performance of the sensors and actuators, leading to fragile behavior. The use of closed-loop control may be a powerful tool in rejecting environmental changes and disturbances, as well as biases in the vehicle control, facilitating to robust operations. This leads to the second research question:

RQ2 - How can abstraction in the control output be used to improve the robustness of a robot to differences between simulation and reality?

For this, we optimized two different neurocontrollers tasked to control a homogeneous swarm of three quadcopters, to form a predetermined pattern from a random initial distribution. The first neurocontroller's output directly set the rotor speeds of the quadcopter, as would typically be done in ER. Alternatively, the second neurocontroller's output was used to set the desired vehicle velocity to the onboard closed-loop controller, representing a significant abstraction in control. Although both controllers were able to robustly achieve the desired swarming formation in simulation, only the abstracted controller worked when transferred to the real world. The observed behavior was near identical to that seen in simulation.

The use of closed-loop controllers helps to increase the robustness of the optimized behavior, actively rejecting environmental disturbances and adjusting for any unique vehicle bias due to manufacturing and operational differences. Additionally, the abstraction allowed the simulation to be significantly sped-up. The abstracted neurocontroller only required a simple velocity model representing the response of the closed-loop controller, whilst the non-abstracted neurocontroller required a significantly higher fidelity aerodynamic model.

Lastly, unlike the tightly coupled sensory-motor coordinated systems typically optimized, closed-loop systems are generally sensor agnostic. These systems simply must generate an appropriate signal to identify and reduce the error of the controlled system. This increases robustness by ensuring that the controlled system is no longer tightly coupled to the sensory input. This is contrary to the typical ER approach where often basic sensors such as a set of infrared or light sensitive sensors are employed. This approach is not only difficult to scale to more complex sensors such as optical cameras, it also makes the optimized robotic behavior sensitive to any variation in the inputs. This leads us to the third and final research question:

RQ3 - How can abstraction on the sensory input be used to reduce the sensitivity of the robotic behavior to the reality gap?

This was investigated by optimizing high speed landing behavior for a quadcopter MAV using the divergence of the optical flow field from a visual camera, and its derivative, as the only input to the control system. Unlike typical applications of automated behavioral development, this abstracted input gives the user the freedom to use any sensor that can generate this signal, reducing the dependence of the robotic behavior to the input used. A reality gap was artificially induced by performing real world experiments using both a conventional CMOS camera and the novel event-based Dynamic Vision Sensor to generate the divergence input signal. These two systems not only produce different input signal responses than each other, but also significantly different than those used to optimize the robotic behavior in simulation.

Despite these clear differences in the input data, the resultant landing behavior of the real world vehicle was very similar to that seen in simulation. This demonstrates

how abstraction provides the system developer with tools to scale the robotic application to various vehicles with different sensor solutions. Additionally, it improves the robustness of the optimized behavior to environmental uncertainties.

In conclusion to the problem statement, abstraction can indeed be used as a tool to balance the implicit trade-off of optimization power and robust transfer from simulation to the real world. With this tool, the user is empowered to effectively control the reality gap. With limited reduction in the optimization power, robotic behavior can be developed in simulation and more effectively transferred to the real world.

Generally, the focus in the literature has been to either improve the individual simulator elements of the environment, robotic sensors or actuator systems to enhance simulation fidelity. Alternatively, some work has focused on some form of online re-optimization to find a new behavior in reality which achieves better performance. In this thesis we focus on ensuring that the robotic behavior can be robustly transferred from simulation to reality by tackling the behavioral reality gap. We show that if the simulation sufficiently represents the real world, the behavior in simulation will result in similar performance in reality. The problem of crossing the reality gap therefore changes from directly optimizing performance to increasing the similarity of the behavior in reality to that observed in simulation. In combination with differing forms of abstraction, the required simulation fidelity is often much lower than conventional approaches allowing for a notable reduction in optimization time. This simple change in perspective radically changes the goal of the transfer problem, with significant results.

This thesis is just the first step in fully understanding the impact of abstraction on the optimization of the robotic behavior. A significant area for future work is in better understanding the factors that affect the trade-off between optimization power and robustness to differences in the simulation and reality. How much abstraction is sufficient or excessive? Here, evaluation methods similar to controlability and observability analysis in control theory, would help making decisions on the level of abstraction appropriate for a given task.

This thesis was mainly concerned with the reality gap, but abstraction may be used as a tool to improve robustness for the wider transfer problem. Additional research is required to determine how this can be effectively used.

Aside from the reality gap, this thesis also showed that abstraction can help to segment the optimization task, reducing the time required to achieve reliable performance. This technique may be used to speed-up the online development of behavior in embodied evolutionary optimization, another promising area for future research.

Samenvatting

Het automatisch optimaliseren van robotgedrag om complexe taken op te lossen is één van de belangrijkste, al lang bestaande doelen van Evolutionary Robotics (ER). Wanneer deze aanpak succesvol is, zal dit waarschijnlijk de snelheid van ontwikkeling en inzet van robots in het dagelijks leven fundamenteel veranderen. Het uitvoeren van deze optimalisatie op echte robots kan riskant en tijdrovend zijn. Als gevolg hiervan wordt veel van het werk in ER gedaan met behulp van simulaties die vele malen sneller functioneren dan in het echt. Het enige nadeel hiervan is dat, vanwege de beperkte betrouwbaarheid van de gesimuleerde omgeving, het geoptimaliseerde robotgedrag vaak anders is wanneer het wordt overgebracht naar een echte robot. Dit verschil wordt de *reality gap* genoemd.

Van de verschillende methoden die zijn ontwikkeld om een brug te slaan naar de reality gap, is één van de meest effectieve *de omhullende ruis* van Jakobi. Dit werk toonde aan dat als je geschikte niveaus van ruis toepast in simulatie op de omgeving en robot, het geoptimaliseerde gedrag minder gevoelig is voor de verschillen tussen simulatie en de echte wereld. Ruis op deze manier toevoegen is een vorm van abstractie, die het voertuig van de werkelijkheid scheidt. Dit werk toonde aan dat abstraheren van de echte wereld de betrouwbaarheid van het geoptimaliseerde gedrag kan vergroten. Ondanks het feit dat deze toegevoegde ruis het vermogen van de optimalisatie beperkt om het werkelijk optimale gedrag te vinden, is de afruil ten gunste van de robuustheid aantoonbaar de moeite waard.

In verschillende vakgebieden is abstractie een tool die vaak gebruikt wordt om de robuustheid van gecontroleerde systemen te verbeteren. Een goed voorbeeld hiervan is de besturingstheorie, waar trapsgewijze- of geneste besturingssystemen verschillende eigenschappen hebben die bijdragen aan het verbeteren van de robuustheid. Het gesloten-lusbesturingsschema helpt om fouten die worden veroorzaakt door kleine verschillen in de voertuigfabricage en -assemblage of omgevingsstoringen actief te verminderen. Dit proefschrift onderzoekt hoe abstractie kan worden gebruikt om de robuustheid van het robotgedrag in de werkelijkheid te verbeteren, aangedreven door de volgende probleemstelling:

Hoe kan abstractie worden gebruikt om de reality gap in evolutionaire robotica te overbruggen?

In dit proefschrift onderzoeken we drie eigenschappen van gesloten-lusbesturingssystemen die kunnen helpen de reality gap te overbruggen.

Ten eerste helpt de trapsgewijze- of geneste structuur van het besturingssysteem het grotere besturingsprobleem te segmenteren in een reeks kleinere, eenvoudi-

gere en meer handelbare problemen. Dit verbetert het vermogen van de ontwerper van de besturing om tekortkomingen in het besturingsschema te identificeren en hierop te richten. Dit staat sterk in contrast met gebruikelijke ER-benaderingen die neurale netwerken gebruiken om het robotplatform te besturen. Deze *black-box*-controllers proberen een complexe, algemene oplossing voor de betreffende taak te optimaliseren. Hoewel dit succesvol is, resulteert deze aanpak vaak in fragiel gedrag dat geen standhoudt als het in de echte wereld wordt toegepast. Het abstraheren van deze globale controllers kan de robuustheid verbeteren. Dit concept leidt tot de eerste onderzoeksvraag:

RQ1: Hoe kan abstractie in de gedragsrepresentatie helpen om de reality gap te overbruggen?

Om deze vraag te beantwoorden, onderzochten we het gebruik van een meer begrijpelijke gedragsrepresentatie voor het robotgedrag. Dit werd getest door het gedrag te optimaliseren voor de 20 g DelFly Explorer flapping-wing Micro Air Vehicle (MAV) met behulp van een representatieve structuur van de *behaviour tree*. De DelFly kreeg de opdracht om door een kleine kamer te vliegen en de muren te ontwijken, terwijl ze op zoek waren naar een open raam, om hier uiteindelijk doorheen te kunnen vliegen.

Nadat in simulatie getraind was met succespercentages boven 85%, was er een duidelijke reality gap bij het testen op het echte voertuig, dat in eerste instantie een succespercentage van 0% liet zien. De begrijpelijke aard van de Behavior Tree liet de menselijke bestuurder toe om snel het gewenste robotgedrag te begrijpen en de bron van de reality gap te identificeren, in dit geval als gevolg van vooroordelen in de waarneming en activering van het echte voertuig. De structuur van de behavior tree leent zich ook gemakkelijk voor eenvoudige aanpassingen. Hierdoor kon de gebruiker het gedrag bijwerken, de reality gap actief verkleinen en de prestaties verbeteren tot 54%. Hoewel lager dan waargenomen in simulatie, was dit resultaat hoger dan het door de gebruiker ontworpen gedrag van 46%. Het resterende verschil in prestaties tussen de gesimuleerde en echte vluchten is waarschijnlijk te wijten aan de ongemodelleerde vluchtdynamiek van de DelFly en de aanwezigheid van ongemodelleerde externe verstoringen zoals windvlagen.

De behavior tree bereikt deze toegenomen begrijpelijkheid ten koste van het representatievermogen in vergelijking met het neurale netwerk. Met zijn subsumptieachtige architectuur, abstraheert hij het gedrag van een globale input-output mapping, waarbij de acties uiteindelijk worden gesegmenteerd tot een reeks aan opties. Om het gedrag begrijpelijk te houden, moeten de trees klein blijven, wat het vermogen van de evolutionaire optimalisatie om de ideale oplossing voor het probleem te vinden enigszins beperkt. Dat gezegd hebbende, zonder de begrijpelijkheid van deze representatiestructuur zou het geoptimaliseerde gedrag nutteloos zijn geweest na overdracht. Deze afweging lijkt daarom de moeite waard. Het samenvatten van de gedragsrepresentatie kan helpen het heroptimalisatieproces te versnellen door het gedrag te segmenteren in eenvoudigere subgedragingen.

Ten tweede gebruiken gesloten-lusbesturingssystemen terugkoppeling om continu een fout tussen de gewenste toestand en de actuele voertuigstatus te verminderen. De meeste implementaties van ER hebben de neiging om de eenvoudigste besturingsmethode te gebruiken. Hierbij worden de actuatoren van de robot direct bestuurd met open-lusbesturing en onbewerkte sensorinformatie direct aan het besturingssysteem toegevoerd, met weinig of geen voorverwerking. Dit leidt tot een inherente gevoeligheid van het gedrag voor de prestaties van de sensoren en actuatoren, wat leidt tot fragiel gedrag. Het gebruik van gesloten-lusbesturingssystemen kan een sterk hulpmiddel zijn bij het afwijzen van veranderingen in de omgeving en verstoringen, evenals vooroordelen in de voertuigbesturing, hetgeen robuuste operaties mogelijk maakt. Dit leidt tot de tweede onderzoeksvraag:

RQ2 - Hoe kan abstractie in de besturingsoutput worden gebruikt om de robuustheid van een robot te verbeteren voor verschillen tussen simulatie en realiteit?

Hiervoor hebben we twee verschillende neurocontrollers geoptimaliseerd om een homogene zwerm van drie quadcopters te besturen, en hiermee een vooraf bepaald patroon te vormen uit een willekeurige initiële verdeling. De output van de eerste neurocontroller stelde de rotorsnelheden van de quadcopter rechtstreeks in, zoals normaal in ER wordt gedaan. Als alternatief werd de uitvoer van de tweede neurocontroller gebruikt om de gewenste voertuigsnelheid in te stellen op de ingebouwde gesloten-lusbesturingssysteem, die een significante abstractie bij de besturing vertegenwoordigt. Hoewel beide besturingssystemen de gewenste zwermvorming in de simulatie robuust konden realiseren, werkte alleen de geabstraheerde besturingssysteem bij overdracht naar de echte wereld. Het waargenomen gedrag was vrijwel identiek aan dat wat werd gezien in de simulatie.

Het gebruik van gesloten-lusbesturingssystemen helpt om de robuustheid van het geoptimaliseerde gedrag te vergroten, actief omgevingsverstoringen af te wijzen en aan te passen voor elke unieke vooroordeel van het voertuig als gevolg van fabricage- en operationele verschillen. Bovendien maakte de abstractie het mogelijk dat de simulatie aanzienlijk versneld werd. De geabstraheerde neurocontroller vereiste slechts een eenvoudig snelheidsmodel dat de respons van de gesloten-lusbesturingssystem weergeeft, terwijl de niet-geabstraheerde neurocontroller een aërodynamisch model met aanzienlijk hogere betrouwbaarheid nodig had.

Ten slotte zijn, in tegenstelling tot de nauw gekoppelde sensor-motor gecoördineerde systemen, doorgaans geoptimaliseerde gesloten-lusbesturingssystem zijn in het algemeen sensor agnostisch. Deze systemen moeten eenvoudig een geschikt signaal genereren om de fout van het gecontroleerde systeem te identificeren en te verminderen. Dit verhoogt de robuustheid door ervoor te zorgen dat het gecontroleerde systeem niet langer strak is gekoppeld aan de sensorische input. Dit is in strijd met de typische ER-benadering waarbij vaak basissensoren zoals een reeks infrarood- of lichtgevoelige sensoren worden gebruikt. Deze benadering is niet alleen moeilijk te schalen naar meer complexe sensoren zoals optische camera's, maar maakt ook het geoptimaliseerde robotgedrag gevoelig voor elke variatie in de ingangen. Dit leidt ons naar de derde en laatste onderzoeksvraag:

RQ3 - Hoe kan abstractie van de sensorische input worden gebruikt om de gevoeligheid van het robotgedrag voor de werkelijkheid te verkleinen?

Dit werd onderzocht door het landingsgedrag bij hoge snelheden van een quadcopter MAV te optimaliseren met behulp van de divergentie van het optische stroomveld van een visuele camera en de afgeleide daarvan als de enige input voor het besturingssysteem. In tegenstelling tot typische toepassingen van geautomatiseerde gedragsontwikkeling geeft deze geabstraheerde input de gebruiker de vrijheid om elke sensor te gebruiken die dit signaal kan genereren, waardoor de afhankelijkheid van het robotgedrag ten opzichte van de gebruikte input wordt verminderd. Een reality gap werd kunstmatig veroorzaakt door het uitvoeren van experimenten uit de echte wereld met behulp van zowel een conventionele CMOS-camera als de nieuwe op gebeurtenissen gebaseerde Dynamic Vision Sensor om het divergentie-inputsignaal te genereren. Deze twee systemen produceren niet alleen verschillende reacties van het inputsignaal ten opzichte van elkaar, maar ook significant verschillend van die worden gebruikt om het robotgedrag in de simulatie te optimaliseren.

Ondanks deze duidelijke verschillen in de inputgegevens, was het resulterende landingsgedrag van het echte voertuig in de praktijk erg vergelijkbaar met wat werd gezien in de simulatie. Dit toont aan hoe abstractie de systeemontwikkelaar tools biedt om de robottoepassing te schalen naar verschillende voertuigen met verschillende sensoroplossingen. Bovendien verbetert het de robuustheid van het geoptimaliseerde gedrag naar omgevingsonzekerheden.

Als conclusie op de probleemstelling kan abstractie inderdaad worden gebruikt als een hulpmiddel om de impliciete afweging van optimaliseringsvermogen en de robuuste overdracht van simulatie naar de echte wereld in evenwicht te brengen. Met deze tool is de gebruiker bevoegd om de reality gap effectief te beheersen. Met een beperkte vermindering van de optimalisatiekracht kan robotgedrag worden ontwikkeld in simulatie en effectiever worden overgedragen naar een robot de echte wereld.

Over het algemeen lag de nadruk in de literatuur op het verbeteren van de individuele simulatorelementen van de omgeving, robotsensoren of actuatorsystemen om de simulatienauwkeurigheid te verbeteren. Als alternatief heeft een deel van het werk zich gericht op een vorm van online heroptimalisatie om een nieuw gedrag in de werkelijkheid te vinden dat betere prestaties levert. In dit proefschrift richten we ons erop dat het robotgedrag goed kan worden overgedragen van simulatie naar realiteit door de reality gap in gedragsweergaven aan te pakken. We laten zien dat als de simulatie voldoende representatief is voor de echte wereld, het gedrag in simulatie in werkelijkheid tot vergelijkbare prestaties zal leiden. Het probleem van het overschrijden van de reality gap verandert daarom van het direct optimaliseren van de prestaties tot het vergroten van de gelijkenis van het gedrag in werkelijkheid met dat wat wordt waargenomen in de simulatie. In combinatie met verschillende vormen van abstractie is de vereiste simulatienauwkeurigheid vaak veel lager dan bij

conventionele benaderingen, wat een opmerkelijke vermindering van de optimalisatietijd mogelijk maakt. Deze eenvoudige verandering in perspectief verandert het doel van het overdrachtsprobleem radicaal, met significante resultaten.

Dit proefschrift is slechts de eerste stap in het volledig begrijpen van de impact van abstractie op de optimalisatie van het robotgedrag. Een belangrijk gebied voor toekomstig werk is het beter begrijpen van de factoren die van invloed zijn op de afweging tussen optimaliseringsvermogen en robuustheid voor verschillen in de simulatie en de realiteit. Hoeveel abstractie is voldoende of in overmaat? Hier zouden evaluatiemethoden die vergelijkbaar zijn met beheersbaarheid en waarneembaarheidsanalyse in de besturingstheorie, helpen beslissingen te nemen op het niveau van abstractie dat geschikt is voor een gegeven opdracht.

Dit proefschrift heeft voornamelijk betrekking op de reality gap, maar abstractie kan worden gebruikt als een hulpmiddel om de robuustheid voor het bredere overdrachtsprobleem te verbeteren. Aanvullend onderzoek is nodig om te bepalen hoe dit effectief kan worden gebruikt.

Afgezien van de reality gap, toonde dit proefschrift ook aan dat abstractie kan helpen om de optimalisatietaak te segmenteren, waardoor de tijd die nodig is om betrouwbare prestaties te bereiken wordt verkort. Deze techniek kan worden gebruikt om de online ontwikkeling van gedrag in belichaamde evolutionaire optimalisatie te versnellen, een ander veelbelovend gebied voor toekomstig onderzoek.

1

Introduction

Since before the word *robot* was first introduced by Karel Čapek in 1920 [25], the dream of automated synthetic machines has fascinated humanity. This fantasy has been immortalized in countless pop culture movie classics from *Metropolis* to *Terminator*™and even *Wall-E*™. Driven in part by this dream, there has been a concerted effort to bring about a future populated by robots.

Through the end of the 20th century, robots were mainly used to perform repetitive tasks in sterile industrial environments [163]. These types of tasks are predictable and relatively easy to automate. Additionally, the areas in which the robots operate are typically strictly controlled, such that the operating conditions are unchanging. Such 'dull, dirty and dangerous' jobs are well suited to simple robotic systems.

Although the scale of the automation is impressive, these types of robots are still far afield from those conceived by dreamers over a century ago. Moving from the sterile industrial environment to our everyday world has proven a greater challenge than many imagined. Various research groups and companies have attempted to tackle this problem, with varying levels of success.

One method that shows great promise in integrating useful robots into everyday life, is the automatic development of autonomous robotic behavior. Many optimization techniques are used to achieve this, with Evolutionary Robotics (ER) [129] and Reinforcement Learning (RL) [161] being the most commonly used. Both of these approaches leverage recent advances in simulation technology to optimize robotic behavior in a virtual world rather than the real world. In this simulated environment, the robot cannot damage itself or its surroundings allowing for learning in a safe space reducing the resources required. Additionally, optimization time can be reduced by running the simulation at faster than real-time speed. This type of automated robotic development promises to fundamentally change the way we design and use robots, significantly increasing their rate of development and deployment in novel environments.

1

Looking around us today, we can unfortunately see that the reality of this promise has not yet been met. There are still several open challenges that must be completed before this dream can be achieved. Chief among them is bridging the *reality gap* [51, 92, 132]. In short, the reality gap is the difference between the simulated environment and the real world. These differences typically lead to a significant degradation in task performance limiting the use of this type of optimization.

This thesis investigates methods to improve the robustness of the optimized robotic behavior to environmental differences, bridging the reality gap. We will focus our application to Micro Air Vehicles (MAVs), where the reality gap can be significant. Typically weighing less than 500 g, these vehicles are very difficult to accurately simulate [5, 24]. At these small scales, the computational power needed to effectively simulate the environment would slow the simulation, negating the time advantage of using a simulator [131]. To date, there have been few demonstrations of behavior optimized for an MAV being successfully transferred to a real-world vehicle to complete a complex task.

To limit the scope of the work presented, we will focus on the Evolutionary Optimization (EO) techniques used in ER to develop the behavior of our robotic agents. This global heuristic optimization methodology provides the designer with a powerful and varied toolset to solve a wide set of problems [71].

The following sections will dive deeper into what the sources of the reality gap are and common methods to deal with it, the optimization technique used in this work and eventually to the research questions.

1.1. REALITY GAP

When a robotic agent is automatically developed, the agent is required to learn a particular mapping from its perception of the world, generated from sensor inputs, to some action to take in the world to achieve a desired goal. This input-output mapping is referred to as a *policy*. The robotic *behavior* is the result of the interaction between the physical robot, its policy, and the world. The *performance* of the behavior is a measure of how well the robot is achieving the given goal. These elements and their interaction are shown in Fig. 1.1. This figure highlights that the development of a policy is directly coupled to the embodied robot and the environment in which it operates.

Now, when the policy is developed in simulation, the real world and all of the interactions therein must be modeled and virtually evaluated. Due to limits in available computation and our limited understanding of our physical world, every element of this virtual world is subject to modeling error. The degree to which certain elements of the simulation represent reality is referred to as *fidelity*, the closer the accuracy, the higher the fidelity. Conversely, the difference between simulation and reality is typically referred to as the *reality gap*.

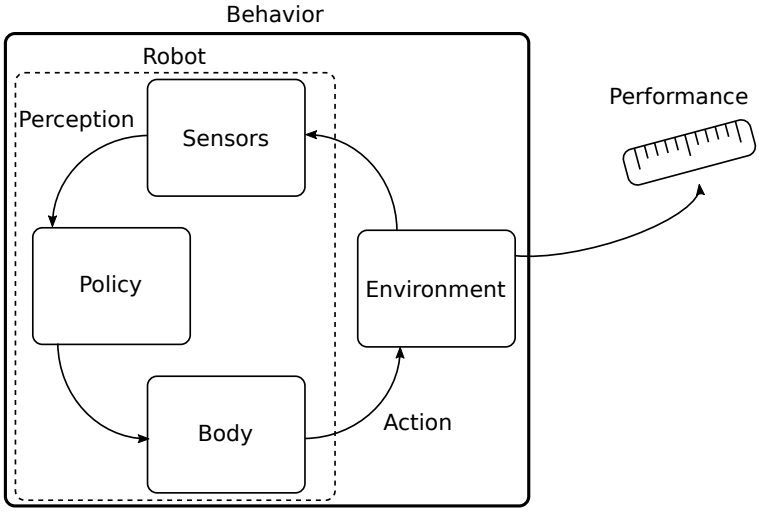


Figure 1.1: Schematic representation of an embodied agent highlighting the components of the robot and their interaction.

An interesting analogy can be found in research investigating how accurate simulator environments should be to allow human operators to learn the right set of skills in preparation for real world flights. Simulation fidelity is an important issue when training pilots. Here, four different measures of fidelity are typically defined [145].

- Objective Fidelity - The degree to which the simulation replicates all aspects of the real world aircraft [6].
- Perceptual Fidelity - The degree to which a pilot cannot distinguish the difference between the simulated and real world input stimuli.
- Behavioral Fidelity - The degree to which the behavior of a skilled pilot in simulation differs to that in reality [80].
- Error Fidelity - The degree to which the aircraft response represents that in the real world.

We can appropriate these definitions to the use of modeling autonomous robotic systems as shown in Fig. 1.2. Here the pilot is replaced with the robot and the above definitions can be adjusted to suit.

There have been several approaches to tackle the reality gap with most attention placed on the environmental, sensory and vehicular gaps. These approaches can be broadly separated into three main categories [19].

- **Adjusting objective and perceptual simulation fidelity** Increasing the simulation fidelity is one way to reduce the reality gap but this comes at the cost of a slower optimization process. Some researchers have used machine learning techniques [118] or novel noise models [116] to better model sensors pro-

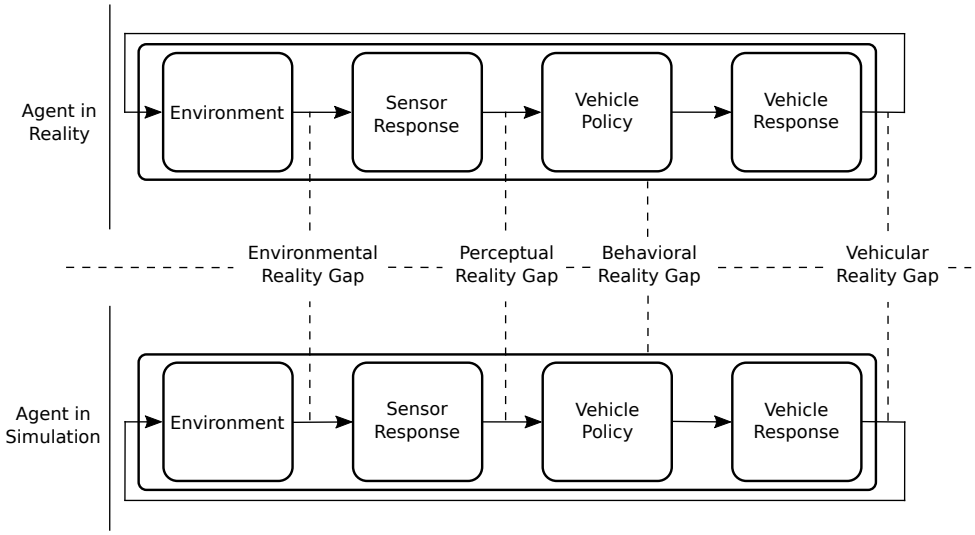


Figure 1.2: Schematic representation of the different measures of reality gap (adapted from [145]).

viding agents with more realistic input data. Similar work has been done to model the actuators [164]. Some work has perturbed the simulated robot [166] or changed the environment [144, 164] while others have explicitly simulated the reality gap by evolving agents in multiple simulated environments [18, 108], all with the goal of developing robust policies.

These approaches have all shown promise but none truly solve the problem. Additionally, they often result in slower simulations, increasing optimization times. An alternative approach which has had significant impact is based on the idea that rather than accurately modeling the real world, it is better to apply an appropriate amount of uncertainty to the simulated environment [92]. If this uncertainty is sufficient, the optimized policy will have to become robust to the eventual uncertainty it will encounter in the real world. This technique is the so called *envelope of noise* [90]. This solution does have its drawbacks however, from a purely optimization perspective, the added noise can cause the agent to settle for a (potentially highly) sub-optimal solution.

- **Mixed reality optimization** These approaches perform some optimization both in simulation and reality. Some work has been done to iteratively update the simulation model by testing the optimized policies in the real world, reducing the reality gap by artificially augmenting the simulated world [20, 177]. Other work has been done by evaluating the policies in the real world and including a measure of their transferability in the optimization process [102]. Some have also alternated between simulated and real data to encourage the agent to be robust to the differences and optimize a general policy [21]. This

effectively tries to tackle the perceptual and vehicular reality gaps.

- **Online Adaptation** If the reality gap is small, the policies can be updated after the transfer to the real world [73]. With this approach, the general macro policy can be safely developed quickly in simulations and then fine-tuned in the real world.

  Rather than just optimizing the robotic behavior in simulation, some methods have the optimization also develop a learning framework that can quickly learn to solve the task in the real world [64, 133, 168]. This leads to a reduction in the optimization time and actively reduces the reality gap on the real robot. This approach has also been shown on relatively complex applications making it a promising approach moving forward [40].

  With this approach, the behavior of the vehicle is not important, only the objective performance is considered. The behavior optimized in simulation is simply a starting point from which the optimum behavior in reality can be quickly achieved. One downside of this approach is that the final behavior of the vehicle is not predetermined or predictable. This may limit where this approach can be used.

In this thesis we will focus on the little discussed behavioral reality gap. The hypothesis is as follows:

Proposition

Given a sufficiently accurate model of the world, the behavior optimized in simulation should result in similar performance in reality. As such, goal of the transfer should be to reduce the behavioral reality gap, ensuring that the real world behavior is the same as the simulated behavior, rather than directly optimizing real world performance.

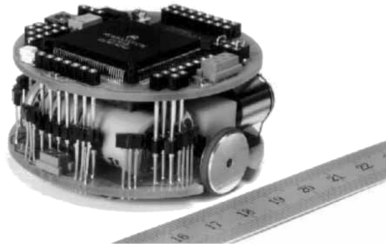
This changes the problem from optimizing performance in reality, as done with online learning, to simply reducing the difference in the behavior observed in simulation and that in reality. To achieve this, we must ensure that the robot can have a robust combination of perceptual and actuation systems that operate similarly in both environments.

1.2. Evolutionary Robotics

There are many types of optimization techniques used to develop robotic behavior but one of particular interest is that used in the field of Evolutionary Robotics. This optimization methodology is built around the concept of automatically developing a robot's policy, and possibly its physical body, using EO techniques [19, 131]. This global optimization technique often results in unconventional methods which exploit

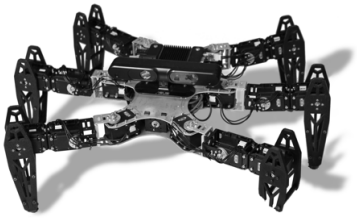
sensorimotor coordination of the embodied agent and its environment to achieve complex tasks [129].

The first applications of ER were done directly on real robotic platforms but this process is very time consuming [61, 132, 160]. Although there has been a resurgence in performing evolutionary optimization on real robotic platforms, accelerated by using swarms of agents [53–55], a significant part of the research community uses simulation based optimization with transfer to the real world. Ever improving computing technologies has facilitated a significant improvement in simulation techniques.



(a) Khepera wheeled robot [167]

(b) Quadruped robot [20]



(c) Hexapod robot [40]

Figure 1.3: Some robots commonly used for Evolutionary Robotics

Early work focused on applications such as obstacle avoidance [92], homing [61, 74], T-maze [91, 134], foraging [26, 127] and walking [91, 123, 177]. Many of these tasks were attempted using the simple Khepera wheeled robot shown in Fig. 1.3a. The field has since moved to more complex platforms like the quadruped robot in Fig. 1.3b [20] and hexapod robot in Fig. 1.3c [40]. Other research has been done on MAVs with tasks including autonomous navigation [152], control [162] and even odor source localization [42].

To truly optimize robotic behavior, it may sometimes be necessary to not only optimize robotic behavior but also the physical body of the robot. This has been the focus of several research groups, some with impressive results [20, 86, 138, 159].

EVOLUTIONARY OPTIMIZATION

All of these advances have been achieved with the use of evolutionary optimization. This powerful tool is a heuristic global policy search technique modeled on the theory of evolution as postulated by Charles Darwin in the mid 19^{th} century [71]. Genetic Algorithms (GA) form the digital implementation of the Darwinian principle of natural selection to artificial systems.

The process of the GA can be summarized in the following steps:

1. **Initialization** - Start with an initial population of feasible solutions.
2. **Evaluation** - In each iteration the performance or *fitness* of each individual of the population is determined.
3. **Selection and Procreation** - Parents of the next generation are selected based on some heuristic fitness function. Parents are paired and create children though crossover where genes are combined in a random manner. Mutation is then applied to the children based on a mutation rate.

Steps 2 and 3 are iterated until some stop condition is met. The GA is characterized by 5 parameters:

- **Population size** - Total number of individuals in the GA. Each individual represents a policy on a multi-dimensional fitness-behavior landscape. Larger populations can generally more effectively explore this landscape but this comes at the cost of more computation as each individual must be evaluated.
- **Selection of Parents** - The heuristic used to determine which individuals of the current generation will become parents. This is an important part of the process as the *selection pressure* generated, drives the population to generally improve. Too high selection pressure causes the optimization to converge prematurely, too low results in a slow process. Some popular implementations are Roulette Wheel, Rank Selection, Truncation Selection, Tournament Selection [71, 120, 131].
- **Genetic Variation** - This typically includes some combinatorial operator which takes attributes from two or more parents to generate members of the new generation. This is typically referred to as a reproduction or crossover operator. In addition, there is also an operator to make random changes to an individual, either to change it in-place or to generate a new individual. This is referred to as a mutation operator [57].
- **Genetic Operator Rates** - The rate at which the genetic operators will affect the population. Like the selection method, this determines how aggressive the changes made to the population are. Too high rates result in an unstable optimization whilst too low results in a very slow process.
- **Stopping Rule** - The GA is stopped based on some rule such as a maximum number of iterations, a maximum CPU time or a maximum number of iterations

without an improvement of the best individual fitness value.

The definition of the fitness function has significant impact on the resultant optimized behavior and the performance of the EO [126]. It is very difficult to define a measure of the robot's performance without injecting undue amounts of user bias to the solution. This problem is compounded once you consider that many tasks actually require the behavior to *balance* multiple goals.

To tackle this, a popular method to evolve agents is with the use of a multi-objective optimization [47, 176]. This allows the user to define relatively simple fitness functions that can help the agents complete the given task, but still allows the optimization the flexibility to trade-off one fitness for another to find new solutions. If we consider all the different possible robotic behaviors as a multi-dimensional surface, the multi-objective optimization is trying to have the population of agents spread out to find the optimum in all directions.

Alternatively, some use the so-called *novelty search* method, where solutions are not evaluated based on performance on completing the task, but rather on how novel they are compared to previous solutions [72, 106]. Using the same multi-dimensional analogy as before, novelty search is trying to spread the population of agents evenly over the surface, covering as many different behaviors as possible.

1.3. MICRO AIR VEHICLES

Recently, the development and use of Unmanned Air Vehicle have exploded. From being used by enthusiasts as a hobby a few years ago, they are now being used to secure our skies, observe our plants, inspect bridges and even deliver packages. This has all been driven by the rapid reduction in cost of micro electronics that make up the computation and sensing of these light weight flying robots [65, 156].

An important subset of these vehicles is the Micro Air Vehicle. Often defined as a small flying vehicle weighing less than 500 g, these vehicles are potentially safe in close proximity around humans, opening up a plethora of potential use cases. Due to the comparable size and flight dynamics of these vehicles, they also provide a flexible platform to perform research into biological systems [42, 95].

Due to their small size, their aerodynamics are not well understood and their associated flight dynamics are not well modeled [5, 24]. Additionally, the small parts that constitute their body are fragile and susceptible to change or damage over time. This is accompanied by relatively low-quality actuators and sensors. Although these properties make it difficult to routinely and reliably operate these vehicles, it makes them ideal platforms to investigate the reality gap and methods to bridge the gap. Throughout this thesis, we will utilize MAVs to test different approaches to improving the robustness of robotic behavior to the reality gap.

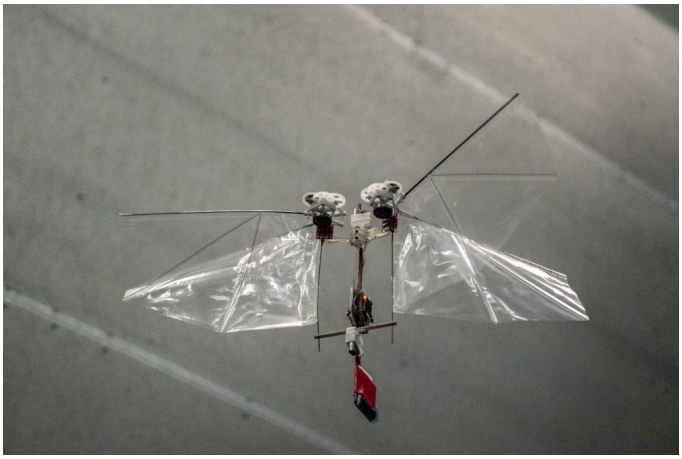


Figure 1.4: DelFly Nimble flapping wing MAV [95]

1.4. Research Questions

From the previous work done to cross the reality gap in ER, the work that most stands out is that of the envelope of noise by Jakobi. At the cost of some optimization power, we can automatically develop autonomous robots in simulation that actually work when transferred to the real world. This suggests that there is some power in separating the robot behavior from the world by abstracting away from some aspects of reality. This leads us to the problem statement of this thesis.

Problem Statement

How can abstraction be used to bridge the reality gap in evolutionary robotics?

If the reality gap is large enough, it may be necessary to adapt the robotic behavior onboard the real robot after transfer to the real world. Most applications of ER use Artificial Neural Networks (ANNs) to represent the policy [131]. Although analysis of the ANNs is possible, this black-box framework does not lend itself well to knowledge-based adaptation hence requiring complex machine learning techniques to retrain the policy. It may be possible that by using a more abstracted behavioral representation framework, the intentions of the robot can be made more transparent, allowing the robot operator to understand the source of the reality gap and actively reduce it. This concept leads us to our first research question.

1

RQ1: Research Question 1

How can abstraction in the behavioral representation help in crossing the reality gap?

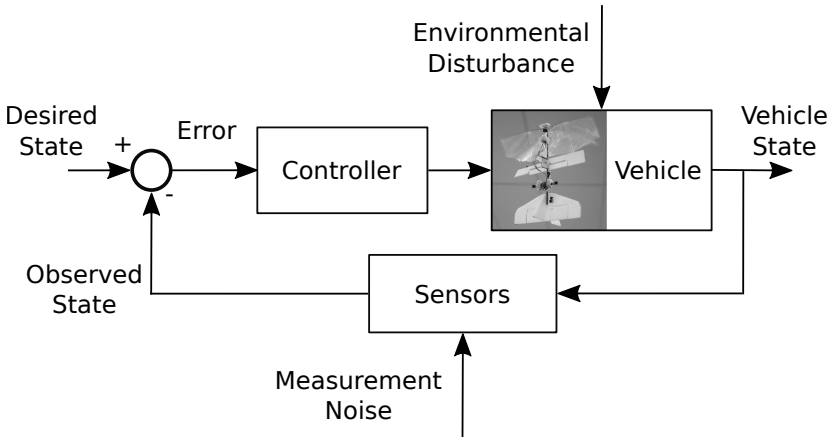


Figure 1.5: Closed loop control scheme, minimizing the error between the desired and measured parameters, this scheme promotes robust flight control performance in the presence of unknown environmental disturbances and sensor noise.

In control theory, abstraction is a powerful tool to improve the robustness of a control system (shown in Fig. 1.5) [115]. Most complex systems are controlled by a closed-loop controller which tries to minimize the error between some desired state and the observed vehicle state [158]. These controllers can also be nested or cascaded such that one layer generates the set-point for a lower layer generating a form of hierarchical control scheme. This is typically done to address the uncertainty in the vehicle and in the environment. This concept leads us to our second research question.

RQ2: Research Question 2

How can abstraction in the control output be used to improve the robustness of a robot to differences between simulation and reality?

When we look to more complex tasks for autonomous robots, we often must consider more complex methods to sense our environment. Many robots use cameras to observe the world around them. Converting raw data to some understanding of the world around us and then to making decisions about what to do is no easy task. Splitting this into two parts, perception and action, may speed up the behavioral development process and result in a more robust solution. This concept leads us to our final research question.

RQ3: Research Question 3

How can abstraction on the sensory input be used to reduce the sensitivity of the robotic behavior to the reality gap?

1.5. Scope and Limitations

To isolate the impact of abstraction on the reality gap we investigate it in the absence of other common methodologies used to bridge the reality gap. As such, in this work, we do not apply online machine learning or iterative model fidelity improvement techniques. Although these techniques show significant promise in aiding the transfer from simulation to reality, they fall outside the scope of this work.

There is a growing field of research in ER performing evolution directly on real world robots, effectively circumventing the reality gap. If successful, this will change the way we develop autonomous robots but until some fundamental problems, such as optimization time, are addressed, bridging the reality gap is still important. It is even possible that lessons learned here may help to speed-up developments in this new field as well.

1.6. Outline

The following chapters will answer these three research questions, eventually shedding light onto the full problem statement. Each chapter is comprised of work either published or currently submitted for review in independent peer-reviewed journals and conference proceedings. Each has an added introduction to tie it into the larger body of this thesis.

Chapter 2 addresses **RQ1** and describes the result of abstracting behavioral representation into a form that is comprehensible to a human and the inherent benefit therein. Here, Behavior Trees (BTs) were used in combination with ER to optimize an exploration task where an MAV must autonomously find a window in a room and fly through it. This is the first time BTs were used within an ER context.

RQ2 is considered in Chapter 3, which presents an investigation into abstraction of the robotic control. To achieve this, two ANNs, using different levels of action abstraction, were compared to each other. The ANNs were optimized to perform coordinated formation flight of a swarm of three MAVs. One controller was optimized with a high level of abstraction controlling the speed of the MAV and the other was tasked to control the rotor speeds of the MAV. We show how well these controllers were able to cross the reality gap.

Before we can truly consider the impact of abstraction of sensors with **RQ3**, we must first define an alternative to the conventional sensor processing pipeline to

test the effect of the abstraction. This is presented in Chapter 4 where we describe an efficient method to extract optic flow from the novel event-based camera. We demonstrate the efficacy of this method on a quadrotor MAV performing high-speed landings, the fastest demonstrated in literature to date (2019).

With this, we can effectively answer **RQ3** in Chapter 5, and evaluate the effect of abstracting robotic perception. We selected the same task as in Chapter 4, this time, having the solution automatically optimized rather than user-defined.

Finally, some conclusions will be made from the presented results and the primary research question will be answered. We will also reflect on the lessons learned and what this means for the research community at large.

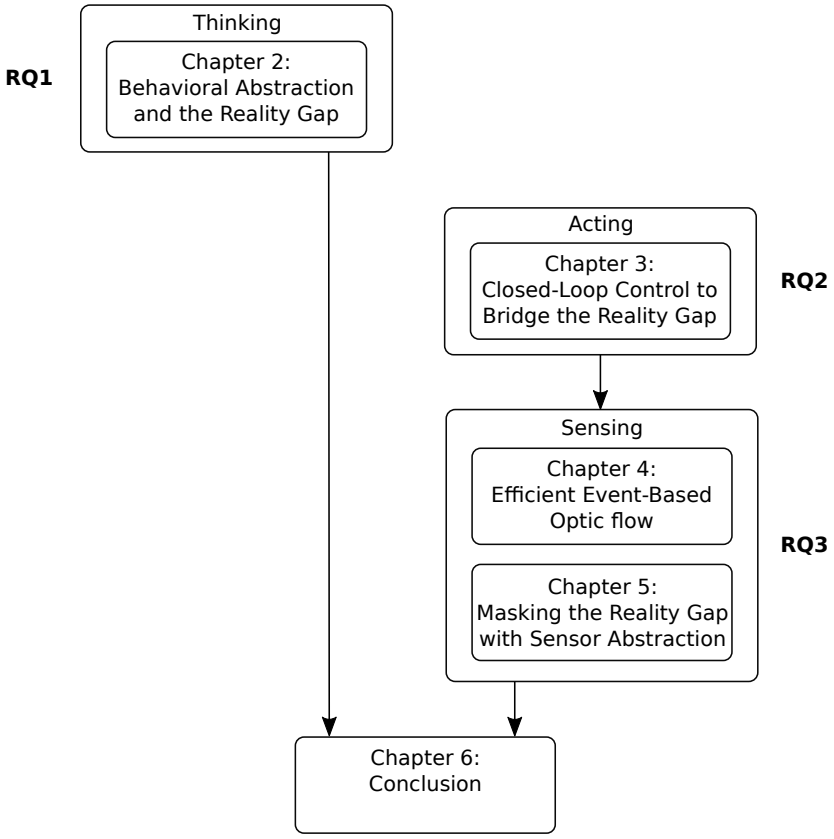


Figure 1.6: Outline of the thesis chapters.

2

Behavioral Abstraction and the Reality Gap

There is a way out of every box; a solution to every puzzle. It's just a matter of finding it

- Jean-Luc Picard

Star Trek: The Next Generation, Season 7 Episode 8

The contents of this chapter have been published as:

Title	Behavior Trees for Evolutionary Robotics
Journal	Artificial Life, 22(1):23–48, 2016
Authors	K.Y.W. Scheper, S. Tijmons, C.C. de Visser and G.C.H.E. de Croon

---

There are several sources that lead to the eventual reality gap when transferring robotic behavior optimized in simulation, to the real world. One significant source is in how the behavior is expressed. Many representation structures used to capture robotic behavior are difficult for a human user to interpret. This makes it a challenge to understand the source of the reality gap. A more intelligible structure should aid in the transfer of behavior.

This chapter describes the process and result of automatically developing robotic behavior in an abstracted, human legible structure. We use the Behavior Tree to represent the behavior of the DelFly Explorer flapping wing MAV to find and fly through an open window. This behavior is made in a virtual environment using Evolutionary Optimization techniques. The following sections describe this task, optimization implementation, results of the behavioral transfer, and the effect of the representation structure.

2.1. Introduction

Small robots with limited computational and sensory capabilities are becoming more commonplace. Designing effective behavior for these small robotic platforms to complete complex tasks is a major challenge. This design problem becomes even more complex when the robots are expected to collaboratively achieve a task as a swarm. A promising methodology to address this problem is found in Evolutionary Robotics (ER), in which a robot's controller, and possibly its body, is optimized using Evolutionary Algorithms (EAs) [19, 89, 131]. This approach satisfies given computational constraints, while often resulting in unexpected solutions which exploit sensory-motor coordination to achieve complex tasks [129].

Early investigations into ER used on-line EAs, in which behaviors generated by evolution were evaluated on real robotic hardware. However, this process can be time consuming [61, 132] and is not widely used, although notably there has been renewed interest in online evolution with swarms of small robots [56]. With the ever improving computing technologies, off-line EAs based on simulation have become the predominant method used in ER. However, this method has some drawbacks of its own. Simulated environments always differ to some degree from reality. The resultant artifacts from the simulation are sometimes exploited by the evolved solution strategy [61]. As a result the behavior seen in simulation can often not be reproduced on a real robotic platform. This problem has been termed the *reality gap* [92, 132].

Many methods have been investigated to reduce this reality gap, which can be separated into three main approaches [19]. The first approach investigates the influence of simulation fidelity on the EA, with investigation focusing on the influence of adding differing levels of noise to the robot's inputs and outputs [92, 116, 118]. It was shown that sufficient noise can deter the EA from exploiting artifacts in the simulation but that this approach is generally not scalable as more simulation runs are needed to distinguish between noise and true features. A notable exception to this is the work of Jakobi who discusses the idea of combining limited but varying noise with differing levels of simulation fidelity in what he calls *Minimal Simulations* [90]. This approach requires the designer to make choices as to which aspects of the environment the robot will use before evolution even begins, limiting the solution space of the EA. Additionally, selecting the type and magnitude of the noise applied requires some foreknowledge of the environmental model mismatch which is not always the available.

The second approach focuses on co-evolution, this approach simultaneously develops a robotic controller which is evaluated in simulation while the simulation model is updated using the performance error with a real world robotic platform [20, 177]. Alternatively, the error between the simulation and real world environment can be used to estimate the suitability of a learned behavior on the real robot. A multi-objective function is used to trade off simulated robotic performance and the transferability of the behavior [102].

The third approach performs adaptation of the real robot behavior after first being optimized by the EA. This can be achieved using many methods which are differentiated by their level of supervision and how the fitness of the behavior is determined. One approach involves the use of unsupervised learning where the neural structure and ontogenetic learning rules are optimized using evolution and are used to generate a population of adaptive individuals [62, 130, 133]. Alternatively, semi-supervised methods such as Reinforcement Learning can be used to retrain the neural nets after evolution [60]. This work shows that systems which adapt to their environments are typically more robust to the reality gap. A typical downside of this approach, however, is that the time needed for the on-line learning to converge may be significant. This is especially problematic for robotic platforms performing complex tasks and operating in an unforgiving environment.

One factor adding to the reality gap problem is that typically Artificial Neural Networks (ANNs) are used to encode the robot behavior [131]. Although analysis of the evolved ANNs is possible, they do not lend themselves well to manual adaptation hence requiring retraining algorithms to bridge the gap. Encoding the optimized behavior in a more intelligible framework would aid a user in understanding the solution strategy. It would also help to reduce the reality gap by facilitating manual parameter adaptation when moving to the real robotic platform.

Traditionally, user-defined autonomous behaviors are described using Finite State Machine (FSM) which has also been successfully used within ER [67, 100, 137, 143]. FSMs are very useful for simple action sequences but quickly become illegible as the tasks get more complex due to *state explosion* [121, 169]. This complexity makes it difficult for developers to modify and maintain the behavior of the autonomous agents.

A more recently developed method to describe behavior is the Behavior Tree (BT). Initially developed as a method to formally define system design requirements, the BT framework was adapted by the computer gaming industry to control non-player characters [27, 52]. BTs do not consider states and transitions the way FSMs do, but rather they consider a hierarchical network of actions and conditions [27, 79]. The rooted tree structure of the BT makes the encapsulated behavior readily intelligible for users.

Previous work on evolving BTs has been applied to computer game environments where the state is fully known to the BT and actions have deterministic outcomes [109, 136]. The evolution of BTs has not yet been applied to a real world robotic task. Operating in the real world introduces complicating factors such as state and action uncertainty, delays, and other properties of a non-deterministic and not fully known environment. There is a growing body of research into proving the operation of BTs through logic and statistical analysis which goes a long way to improving the safety of using BTs on real vehicles [35, 99].

In this chapter, we perform the first investigation into the use of Behavior Trees in Evolutionary Robotics. Section 2.2 will describe the *DelFly Explorer* [46], the flapping wing robotic platform selected to demonstrate our approach as well as the

fly-through-window task it had to perform. This is followed by a detailed description of the BT framework used in Section 2.3. Section 2.4 goes on to describe how offline EAs techniques are used to automatically develop BTs. The results of the optimization are presented in Section 2.5. Additionally, the performance of the best individual from the EA is compared to a human user designed BT to show the efficacy of this automatically generated behavior. The implementation of both behaviors on the real world DelFly Explorer is described in Section 2.6. The method used to bridge the reality gap is described in Section 2.7. This is followed by the real world test results in Section 2.8. Finally we provide a short discussion of the results and talk about how this technique can be scaled to more complex systems and applied to other applications in Section 2.9.

2.2. DelFly Fly-Through-Window

The limited computational and sensory capabilities of the DelFly Explorer make it a challenge to design even the most simple behavior. As a result, the DelFly Explorer is an ideal candidate for the implementation of ER. We will give a brief description of this platform and its capabilities.

2.2.1. DelFly Explorer

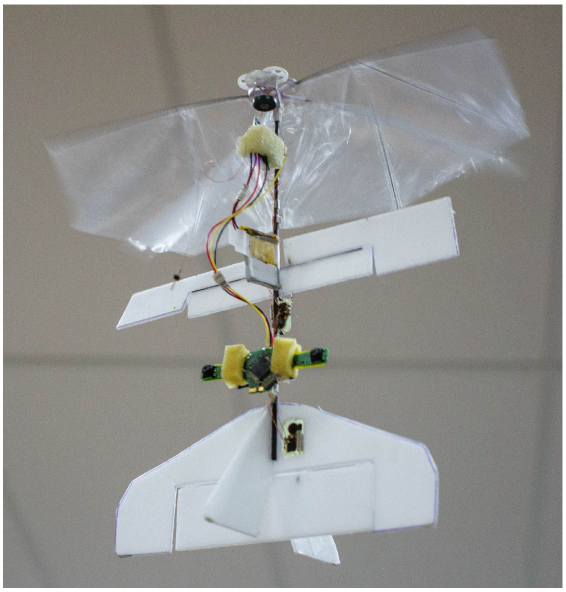


Figure 2.1: DelFly Explorer 20-gram flapping wing MAV in flight with 4-gram dual camera payload. An onboard stereo vision algorithm generates a depth map of the environment which is used for autonomous navigation.

The DelFly is a bio-inspired flapping-wing Micro Air Vehicle (MAV) developed at the Delft University of Technology (TU Delft). The main feature of its design is its biplane-wing configuration which flap in anti-phase [43]. The DelFly Explorer is a recent iteration of this micro ornithopter design [46]. In its typical configuration, the DelFly Explorer is 20 g and has a wing span of 28 cm. In addition to its 9 minute flight time, the DelFly Explorer has a large flight envelope ranging from maximum forward flight speed of 7 m/s, hover, and a maximum backward flight speed of 1 m/s. A photo of the DelFly Explorer can be seen in Fig. 2.1.

The main payload of the DelFly Explorer is a pair of light weight cameras used to perform onboard vision based navigation as shown in Fig. 2.1. Each camera is set to a resolution of 128 $\times$ 96 pixels with a field of view of 60° $\times$ 45° respectively. The cameras are spaced 6 cm apart facilitating stereo-optic vision. Using computer vision techniques these images can be used to generate depth perception with a method called Stereo Vision [153]. This makes the DelFly Explorer the first flapping wing MAV that can perform active obstacle avoidance using onboard sensors facilitating fully autonomous flight in unknown environments [46].

2.2.2. FLY-THROUGH-WINDOW TASK

In this chapter, the DelFly Explorer is tasked to navigate a square room in search for an open window which it must fly through using onboard systems only. This is the most complex autonomous task yet attempted with such a light-weight flapping wing platform. Due to the complexity of finding and flying through a window, we currently limit the task to directional control: height control can be added in future work.

Other work on the fly-through-window task include the H^2Bird 13 g flapping wing MAV [94]. Unlike the DelFly Explorer, the H^2Bird used a ground based camera and off-board image processing to generate heading set-points. In this work the DelFly must perform all tasks using only onboard computation and sensing making the task much more complex than that of the H^2Bird.

2.2.3. VISION SYSTEMS

In the light of the task, the following vision algorithms will be running onboard the DelFly Explorer:

LONGSEQ STEREO VISION

The DelFly Explorer uses a Stereo Vision algorithm called *LongSeq* to extract depth information of the environment from its two onboard optical cameras [46]. The main principle in artificial stereo vision is to determine which pixel corresponds to the same physical object in two or more images. The apparent shift in location of the pixels is referred to as the disparity. This can be applied to entire features, groups

of pixels or to individual pixels. The stereo vision algorithm produces a disparity map of all pixels in the images [153].

LongSeq is a localised line based search stereo vision algorithm. This is one candidate resulting from the trade-off between computational complexity and image performance made by all image processing algorithms. The relatively low computational and memory requirements of LongSeq makes it a good candidate for application on the limited computational hardware onboard the DelFly Explorer.

Window Detection

An Integral Image window detection algorithm is used to aid the MAV in the fly-through-window task. Integral image detection is a high speed pattern recognition algorithm which can be used to identify features in a pixel intensity map [39, 93]. The integral image ($II(x, y)$) is computed as

$$II(x,y) = \sum_{x' \leq x, y' \leq y} I(x', y') \tag{2.1}$$

where x and y are pixel locations in the image I. As each point of the integral image is a summation of all pixels above and to the left of it, the sum of any rectangular subsection is simplified to the following computation

$$\begin{aligned} rect(x,y,w,h) =& II(x+w, y+h) + II(x,y) \\ & - II(x+w, h) - II(x, y+h) \end{aligned} \tag{2.2}$$

This method has been previously used to identify a dark window in a light environment by using cascaded classifiers [45]. That algorithm was designed specifically to operate when approaching a building in the daytime on a light day. Naturally, a more generalized method is to apply the same technique described above to the disparity map rather than the original camera images. The disparity map would show a window as an area of low disparity (dark) in an environment of higher disparity (light).

2.2.4. SmartUAV Simulation Platform

SmartUAV is a Flight Control Software (FCS) and simulation platform developed in-house at the TU Delft [2]. It is used primarily with small and micro sized aerial vehicles and it notably includes a detailed 3D representation of the simulation environment which is used to test vision based algorithms. It can be used as a ground station to control and monitor a single MAV or swarms of many MAVs. As SmartUAV is developed in-house, designers have freedom to adapt or change the operating computer code at will, making it very suitable for use in research projects.

SmartUAV contains a large visual simulation suite which actively renders the 3D environment around the vehicle. OpenGL libraries are used to generate images on

the PC's GPU increasing SmartUAV's simulation fidelity without significant computational complexity. In this chapter we will only utilize the simulation capabilities.

The BT will be placed in series following the LongSec disparity map generation and the window detection algorithm. In terms of the larger SmartUAV simulation, the vision based calculations are the most computationally intensive portion making it the limiting factor for the speed of operation of the wider decision process. The higher the decision loop frequency relative to the flight dynamics the longer a single simulation will take. This must be balanced by the frequency at which the DelFly is given control instructions, where generally higher is better. Considering this trade-off, the decision loop was set to run at 10Hz. This is a conservative estimate of the actual performance of the vision systems onboard the real DelFly Explorer.

2.2.5. SIMPLIFIED DELFLY MODEL

The modeling of flapping wing MAV dynamics is an active research area driven by the largely unknown micro scale aerodynamic effects [4, 24, 43]. Due to the lack of accurate models, an existing model of the DelFly II previously implemented based on the intuition of the DelFly designers will be used in this work. This model is not an accurate representation of the true DelFly II dynamics but was sufficient for most vision based simulations previously carried out.

The DelFly II has three control inputs, namely: Elevator (δ_e), Rudder (δ_r) and Thrust (δ_t). The elevator and rudder simply set the control surface deflection and the thrust sets the flapping speed. The actuator dynamics of the DelFly rudder actuator is implemented using a low pass filter with a rise time of 2.2 s and a settling time of 3.9 s. The elevator deflection and flapping speed have no simulated dynamics and are directly set to the set-point.

For the simulated flights in this chapter, the throttle setting and elevator deflection were held constant at a trim position resulting in a flight speed of 0.5 m/s and no vertical speed. Additionally, the rudder deflection was limited to a resultant maximum turn rate of 0.4 rad/s resulting in a minimum turn radius of 1.25 m. The simulated dynamics had no coupling in the flight modes of the simulated DelFly which is a significant simplification of real world flight.

Now, there are some notable differences between the DelFly II and DelFly Explorer. Firstly the Explorer replaces the rudder with a pair of ailerons which allows the DelFly Explorer to turn without the camera rotating around the view axis. Additionally, the DelFly Explorer is 4 g heavier and has a slightly higher wing flapping frequency. It is expected that the DelFly model mismatch will exaggerate the resultant reality gap.

2.3. Behavior Tree Implementation

BTs are depth-first, ordered Directed Acyclic Graphs (DAGs) used to represent a decision process [14]. DAGs are composed of a number of nodes with directed edges. Each edge connects one node to another such that starting at the root there is no way to follow a sequence of edges to return to the root. Unlike FSMs, BTs consider achieving a goal by recursively simplifying the goal into subtasks similar to that seen in the Hierarchical Task Network (HTN) [59]. This hierarchy and recursive action make the BT a powerful way to describe complex behavior.

2.3.1. Syntax and Semantics

A BT is syntactically represented as a rooted tree structure, constructed from a variety of nodes. Each node has its individual internal function whilst all nodes have the same external interface making the structure very modular. When evaluated, each node in a BT has a return status which dictates how the tree will be traversed. In its simplest form, the return statuses are either *Success* or *Failure*. As the terms suggest, Success is returned on the successful evaluation of the node and Failure when unsuccessful. As this does not provide much information as to the condition under which the node failed, some implementations have augmented states such as *Exception* or *Error* to provide this information.

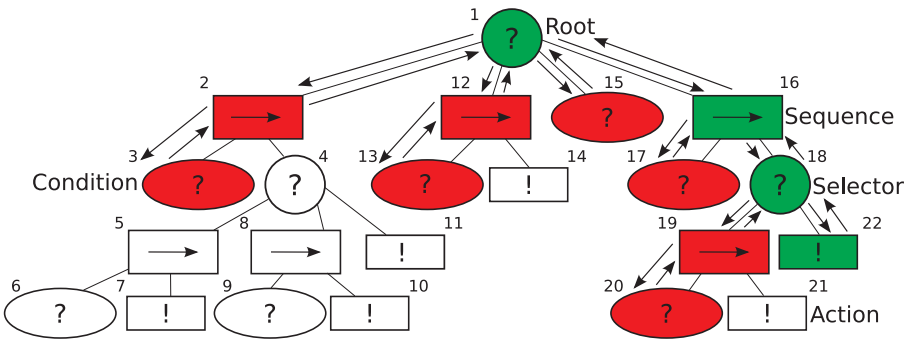


Figure 2.2: Typical representation of the Behavior Tree showing the basic node types and execution flow. The leaf nodes of the tree are composed of Action and Condition nodes whilst the branches are referred to as Composites. All nodes return either Success or Failure. There are two types of Composite nodes used: Selectors and Sequences. Selectors return Success if one of their children is successful and Failure if they all fail. Conversely, Sequences return Failure if one of their children fail and Success is they all succeed. In this example, Condition nodes 3, 13, 15, 17 and 20 return Failure in the given time step or tick. The lightly shaded nodes return Success and the dark nodes evaluate Failure. The nodes with no shading are not evaluated in this tick. The arrows show the propagation of evaluations in the tree.

Fig. 2.2 shows a typical BT and node types used in this chapter. Basic BTs are

made up of three kinds of nodes: *Conditions*, *Actions* and *Composites* [27]. Conditions test some property of the environment whilst Actions allow the agent to act on its environment. Conditions and Actions make up the leaf nodes of the BT whilst the branches consist of Composite nodes. Naturally, leaf nodes are developed for specific robotic platforms dependent on the available sensors and actuators.

Composite nodes however are not platform dependent and can be reused in any BT. Each node requires no information about its location in the tree. Only Composite nodes need to know who its children are in order to direct the flow of execution down the tree. This structure makes BTs inherently modular and reusable.

The tree execution can also be seen in Fig. 2.2. This demonstrates how the Composite nodes determine the execution path of the tree dependent on the return value of their children. To understand this flow structure we must first describe the Composite node in more detail. Although many different types of Composite nodes exist, we will only consider the most basic nodes in this chapter: *Selectors* and *Sequences*.

Composites evaluate their children in a fixed order, graphically represented from left to right. Selectors will break execution and return Success when one of its children return Success, or Failure when all of its children return Failure. Conversely, Sequences will break execution and return Failure when one of its children fails, or Success if all of its children return Success. The first node in the tree is called the *Root* node, which is typically a Selector with no parent. The execution of the behavior tree is referred to as a *tick*.

This execution framework means that not all nodes are evaluated in every tick. The left most nodes are evaluated first and determine the flow through the tree implementing a sort of prioritized execution.

2.3.2. DELFLY IMPLEMENTATION

Aside from the generic Sequence and Selector Composite nodes, two condition nodes and one action node were developed for the DelFly, namely: *greater_than*, *less_than* and *set_rudder*. These behavior nodes are accompanied by a *Blackboard* which was developed to share information with the BT.

The Blackboard architecture implemented for the DelFly contains five entries: *window x location* (x), *window response* (σ), *sum of disparity* (Σ), *horizontal disparity difference* (Δ) and *rudder deflection* (r). The first four are condition variables and the last item is used to set the BT action output. The condition variables are set before the BT is ticked and the outputs are passed to the DelFly FCS after the tick is complete. Note that this implementation of a BT has no explicit concept of memory or time.

The Condition nodes check if some environmental variable is greater than or less than a given threshold. This means that each Condition node has two internal settings: the environmental parameter to be checked and the threshold. The Action

node *set_rudder* sets the DelFly rudder input and therefore only has one internal setting. Actions were defined to always return Success.

2.3.3. User Designed Behavior Tree

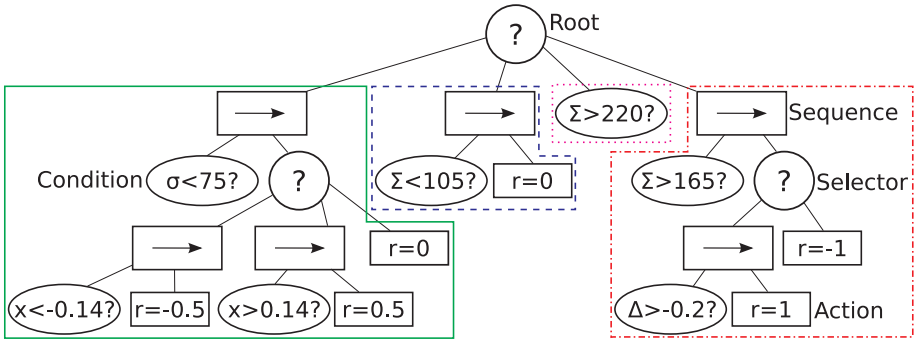


Figure 2.3: Graphical depiction of user-defined BT for the fly-through-window task. Different sub-behaviors of the flight are encapsulated in boxes. x is the position of the center of the window in frame, σ is window response value, Σ is sum of disparity, Δ is the horizontal difference in disparity and r is the rudder deflection setting for the simulated DelFly II.

A human designed behavior was used as a baseline to judge the performance of the genetically optimized solution. The designed tree has 22 nodes and the structure of the BT as shown in Fig. 2.3. The behavior is made up of four main sub-behaviors:

— window tracking based on window response and location in frame - try to keep the window in the center of the frame

- - go straight when disparity very low - default action, also helps when looking directly through window into next room

-.- wall avoidance when high disparity - bidirectional turns to avoid collisions with walls, also helps to search for window

... action hold when disparity very high - ensures the chosen action is not changed when already evading a wall

After validation of this BT, it was observed that for 250 random initializations in the simulated environment, 82% of flights where successful. This behavior is good but suffers from one main flaw which was observed during the validation. Unwittingly, the bidirectional wall avoidance in a square room can result in the DelFly getting caught in and crashing into corners. There are available methods to correct for this behavior [165, 178] but as this is a conceptual error typical for human designed systems, we will keep this behavior as is. Fig. 2.4 shows the path of successful and failed flight realizations of DelFly with the user-defined behavior.

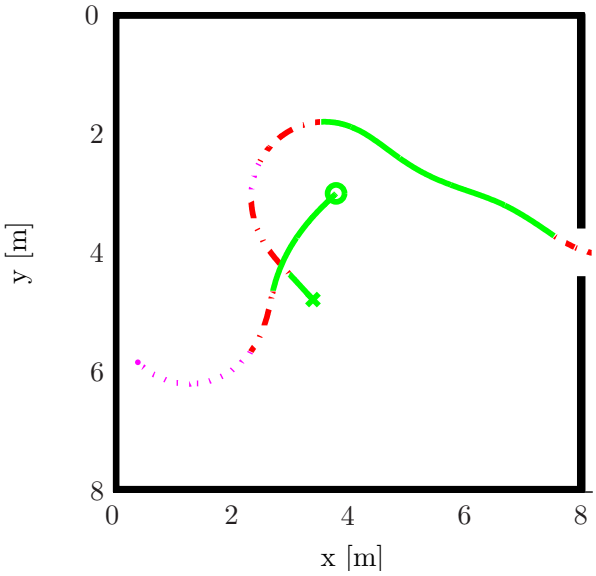


Figure 2.4: Path of successful (x) and unsuccessful flight (o) initializations of DelFly with the user-defined behavior (top-down view). Line types denote different decision modes: Solid - window tracking; Dash - default action in low disparity; Dot Dash - wall avoidance; Dot - action hold

2.4. EVOLUTIONARY ALGORITHM

Evolutionary Algorithms are a population based metaheuristic global optimization method inspired by Darwinian evolution [71, 83]. A population of feasible solutions for a particular problem are made up of a number of individuals. The fitness of each individual is measured by some user-defined, problem specific, objective function. The fitness of the individuals is evaluated each generation. Successful individuals are selected to generate the next generation using the genetic recombination method *crossover*. Each generated child may also be subject to *mutation* where individual parts of their genes are altered. These operations allow the EA to effectively explore and exploit the available search space [117].

There are many implementations of EAs, each with a different method to encode the genetic material in the individuals [60, 71, 103]. In this chapter we will use an EA to optimize the behavior for a task using the BT framework. The custom EA for BTs used in this work is described in the following sections.

2.4.1. GENETIC OPERATORS

Initialization The initial population of M individuals is generated using the *grow* method [103]. Nodes are selected at random to fill the tree with Composite, Action and Condition nodes with equal probability. Once a Composite node is selected, there is equal probability for a Sequence or Selector. This was done as more leaf nodes are typically needed in trees than branch nodes.

The grow method results in variable length trees where every Composite node is initialized with its maximum number of children and the tree is limited by some maximum tree depth. This provides an initial population of very different tree shapes with diverse genetic material to improve the chance of a good EA search.

Selection A custom implementation of Tournament Selection is used in this chapter [120]. This is implemented by first randomly selecting a subgroup of s individuals from the population. This subgroup is then sorted in order of their fitness. If two individuals have the same fitness they are then ranked based on tree size, where smaller is better. The best individual is typically returned unless the second individual is smaller, in which case the second individual is returned. This was done to introduce a constant pressure on reducing the size of the BTs.

Crossover Crossover is an operation where the composition of two or more parents is recombined to produce offspring. In this chapter we use two-parent crossover to produce two children. Each parent is selected from a different tournament selection. The percentage of the new population formed by Crossover is defined by the Crossover Rate P_c. The point in the BT used to recombine the parents is selected at random. This selection is independent of its type or its location in the tree. Crossover can be applied to any node location till the maximum tree depth after which nodes are ignored. Figures 2.5 and 2.6 graphically show this process.

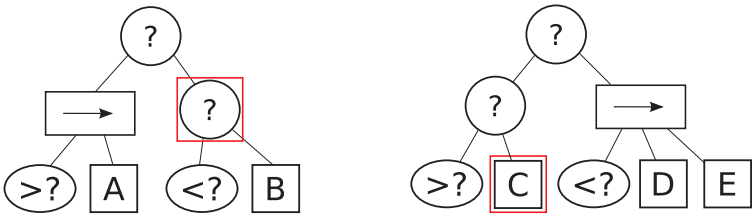


Figure 2.5: Sample parent trees with selected nodes for crossover highlighted. Two-parent, single point Crossover is used for evolution.

Mutation Mutation is implemented using two methods, namely: micro-mutation and macro-mutation (also referred to as *Headless Chicken Crossover* [3]). Micro-mutation only affects leaf nodes and is implemented as a reinitialization of the node with new operating parameters. Macro-mutation is implemented by replacing a selected node by a randomly generated tree which is limited in depth by the maximum tree depth. This is functionally identical to crossover with a randomly generated BT.

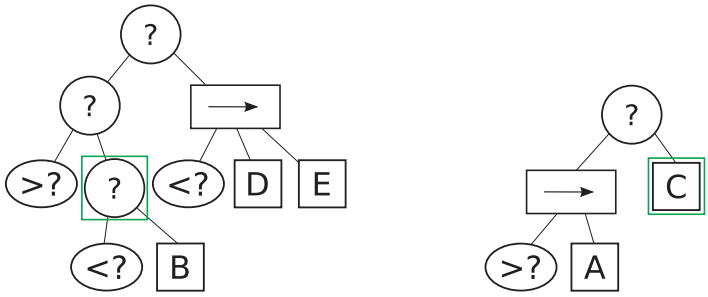


Figure 2.6: Children of crossover of parents in Fig. 2.5.

The probability that mutation is applied to a node is given by the mutation rate P_m. Once a node has been selected for mutation the probability that macro-mutation will be applied rather than micro-mutation is given by the Headless-Chicken Crossover Rate P_{hcc}.

Stopping Rule Like many optimization methods, EAs can be affected by over-fitting. As a result an important parameter in EA is when to stop the evolutionary process. Additionally, due to the large number of simulations required to evaluate the performance of the population of individuals, placing a limit on the maximum number of generations can help avoid unnecessarily long computational time.

For these reasons, the genetic optimization has a maximum number of generations (G) at which the optimization will be stopped. Additionally, when the trees are sufficiently small to be intelligible, the process can be stopped by the user.

2.4.2. FITNESS FUNCTION

The two main performance metrics used to evaluate the DelFly in the fly-through-window task are: Success Rate and Tree Size. The fitness function was chosen to encourage the EA to converge on a population that flies through the window as often as possible. After trying several different forms of fitness functions a discontinuous function was chosen such that a maximum score is received if the MAV flies through the window and a score inversely proportional to its distance to the window if not successful. The fitness F is defined as:

$$F = \begin{cases} 1 & if\ success \\ \frac{1}{1+3|\mathbf{e}|} & else \end{cases} \tag{2.3}$$

where success is defined as flying through the window and **e** is the vector from the center of the window to the location of the MAV at the end of the simulation. This particular form of fitness function was selected to encourage the DelFly to try to get close to the window with a maximum score if it flies through. The values selected are not very sensitive and were chosen at the discretion of the designer. Changing the gain of the error term effects the selection pressure of the EA.

Although not incorporated in the fitness function, we will also analyze some secondary parameters that are not vital to the performance of the DelFly. These define the suitability of its behavior from a user point of view and define the characteristics of a given fly-through-window behavior. These parameters are defined as: Angle of Window Entry, Time to Success and Distance from center of Window at Fly-Through.

2.5. DelFly Task optimization

2.5.1. Simulated 3D Environment

The environment chosen to evaluate the DelFly in simulation was an $8\times8\times3$ m room with textured walls, floor and ceiling. A 0.8×0.8 m window was placed in the center of one wall. Another identical room was placed on the other side of the windowed wall to ensure the stereo algorithm had sufficient texture to generate matches for the disparity map when looking through the window.

As it is not the purpose of this research to focus on the vision systems, the environment was rather abundantly textured. A multi-colored stone texture pattern was used for the walls, a wood pattern was used for the floor and a concrete pattern used for the ceiling as shown in Fig. 2.7. The identically textured walls ensure that the behavior must identify the window and not any other features to aid in its task.

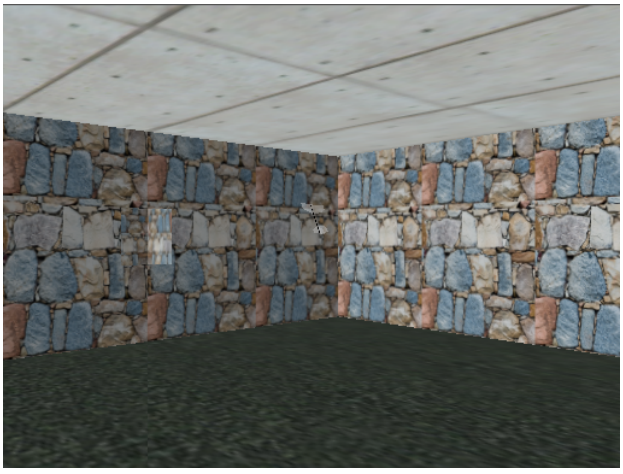


Figure 2.7: Virtual $8\times8\times3$ m room used to evaluate DelFly fly-through-window task showing: virtual DelFly Explorer, textured walls used for stereo vision and target 0.8×0.8 m window.

2.5.2. EXPERIMENTAL SET-UP

The evolved DelFly behavior should be robust and therefore must fly through the window as often as possible. To evaluate this, each individual behavior must be simulated multiple times in each generation defined by parameter k. Each run is characterized by a randomly initiated location in the room and a random initial heading.

Initially, it was observed that by randomly changing the initializations in every generation made it difficult for evolution to determine if the behavior in subsequent generations improved due to its behavior or due to the initialization. To remedy this initial conditions are held over multiple generations until the elite members of the population (characterized by P_e) are all successful. Once all the elite members are successful in a given initialization run, the initial condition in question is replaced by a new random initialization. Each simulation run is terminated when the DelFly crashes, flies through the window or exceeds a maximum simulation time of 100 s.

For the EA to converge to a near-optimum solution the Crossover rate must be high enough to push the optimization to exploit the local maxima. Additionally, the mutation rate must be high enough to explore the state space while not too high to prematurely exit current solutions. The characteristic parameters for optimization shown in this chapter are shown in Table 2.1. The parameter combination selected is naturally only one realization of many possibilities. The relatively large number of runs per individual selected should promote the development of robust flight behavior. This however increases the total simulation time needed to evaluate each generation hence affecting the choice of population size.

Table 2.1: Parameter values for the Evolutionary Computation

Parameter	Value
Max Number of Generations (G)	150
Population size (M)	100
Tournament selection size (s)	6%
Elitism rate (P_e)	4%
Crossover rate (P_c)	80%
Mutation rate (P_m)	20%
Headless-Chicken Crossover rate (P_{hcc})	20%
Maximum tree depth (D_d)	6
Maximum children (D_c)	6
No. of simulation runs per generation (k)	6

The maximum tree depth is measured with the root node as depth 0. The maximum tree size can be determined by $maxchildren^{maxdepth}$. So a tree depth of 6 with at most 6 children per Composite was used resulting in an upper limiting tree size of over 46000 nodes. This is however not likely as the node type selected in the trees is chosen at random over Composite, Condition and Action.

2.5.3. Optimization Results

The main parameter which dictates the progress of the genetic optimization is the mean fitness of the population. Fig. 2.8 shows the population mean fitness as well as the mean fitness of the best individual in each generation. It can be seen in Fig. 2.8 that at least one member of the population is quickly bred to fly through the window quite often. Additionally, as the generations progress and new initializations are introduced the trees have to adjust their behavior to be more generalized. The mean fitness also improves initially and then settles out at around the 0.4 mark. The fact that this value doesn't continue to increase suggests that the genetic diversity in the pool is sufficient to avoid premature conversion of the EA.

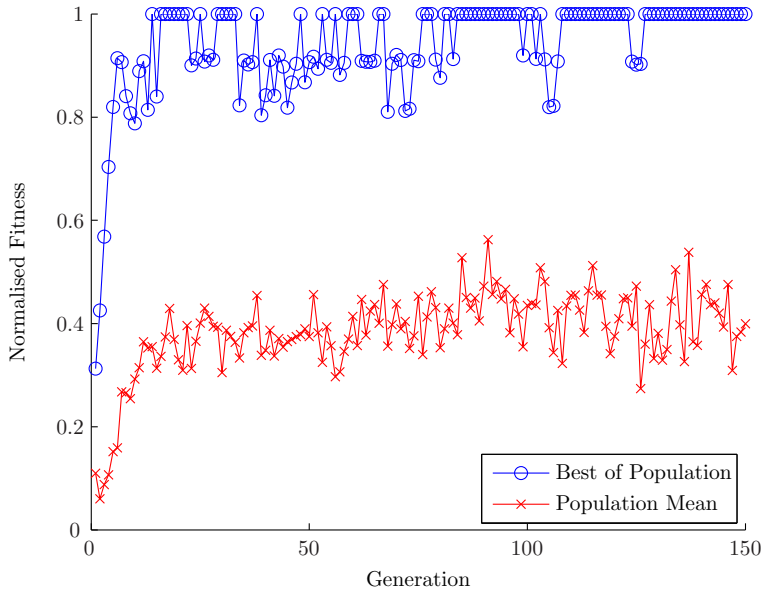


Figure 2.8: Progression of the fitness score of the best individual and the mean of the population throughout the genetic optimization. The fitness value is the mean of the k simulation runs from each generation.

The other main parameter which defines the proficiency of the BTs is the tree size. The mean tree size of the population as well as the tree size of the best individual from each generation is shown in Fig. 2.9. This figure shows that the average tree size began at about 5000 nodes and initially increases to 7000 before steadily dropping to around 1000 nodes at generation 50. The trees then slowly continue to reduce in size and eventually drop below 150 nodes. The best individual in the population oscillated around this mean value. The best individual after 150 generations had 32 nodes. Pruning this final BT, removing redundant nodes that have no effect on the final behavior, resulted in a tree with 8 nodes. The structure of the tree can be seen graphically in Fig. 2.10.

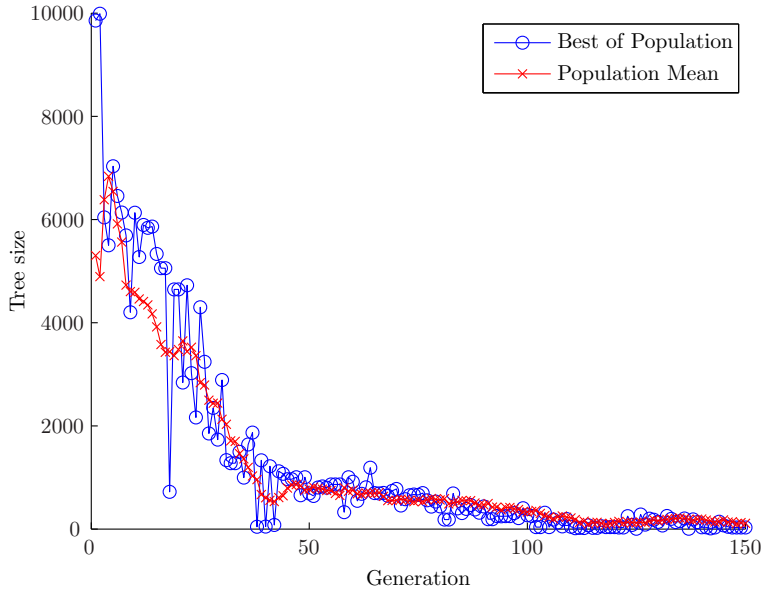


Figure 2.9: Progression of the number of nodes in the best individual and the mean of the population.

Fig. 2.11 shows the progression of the validation success rate for the best individual of each generation. It can be seen that the score quickly increases and oscillates around about 80% success. In early generations the variation of success rate from one generation to the next is larger than later generations.

Figures 2.9 and 2.11 suggest that the population quickly converges to a viable solution and then continues to rearrange the tree structure to result in ever smaller trees. The fact that the best individual of each population does not improve much above the 80% mark possibly indicates that the selected initial conditions used for training are in-fact not representative for the full set of initial conditions. One method to make the initial conditions more *difficult* is to adapt the environment to actively challenge the EA in a sort of *predator-prey* optimization. Alternatively, the fact that the behavior does not continue to improve over the 80% mark may indicate that the sensory inputs used by the DelFly are not sufficient.

The optimized BT was put through the same validation set as used with the user-defined behavior resulting in a success rate of 88%. The performance characteristics of the best individual from the optimization as compared to those from the user-defined BT is summarized in Table 2.2. The optimized BT has slightly higher success rate than the user-defined BT but with significantly less nodes. The results of the secondary parameters suggest that the genetically optimized behavior typically has a sharper window entry angle and enters the window closer to the edge than the user-defined behavior. It also has a longer time to window fly-through as

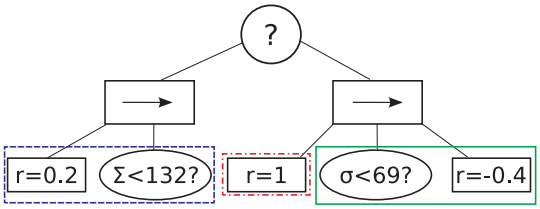


Figure 2.10: Graphical depiction of genetically optimized BT. Different sub-behaviors of the flight encapsulated by boxes. x is the position of the center of the window in frame, σ is window response value, Σ is sum of disparity, Δ is the horizontal difference in disparity and r is the rudder deflection setting for the simulated DelFly II.

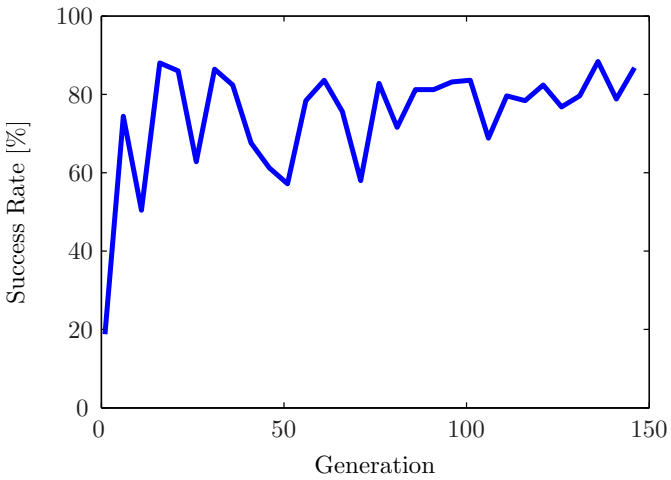


Figure 2.11: Progression of the validation score of the best individual of each generation subjected to the same set of 250 spacial initializations in the simulated room.

it circles the room more often than the user-defined behavior. This result highlights the fact that EAs typically only optimize the task explicitly described in the fitness function, sometimes at the cost of what the user might think is beneficial.

The successful flight shown in Fig. 2.12 shows that the behavior correctly avoids collision with the wall, makes its way to the center of the room and then tracks into the window. Analyzing the BT from Fig. 2.10, the logic to fly through the window can be separated into three sub-behaviors:

- - slight right turn default action when disparity low
- -.- max right turn to evade walls if disparity high (unidirectional avoidance)
- — if window detected make a moderate left turn

Although this very simple behavior seems to be very successful, Fig. 2.12 also

Table 2.2: Summary of validation results

Parameter	User-defined	Genetically Optimized
Success Rate	82%	88%
Tree size	26	8
Mean flight time [s]	32	40
Mean approach angle [°]	21	34
Mean distance to center [m]	0.08	0.15

highlights one pitfall of this solution. As the behavior does not use the location of the window in the frame for its guidance it is possible to drift off center and lose the window in frame and enter a wall avoidance turn quite close to the wall resulting in a collision.

These results show that based on the given fitness function and optimization parameters the genetic optimization was very successful. The resultant BT was both smaller and better performing than the user-defined tree.

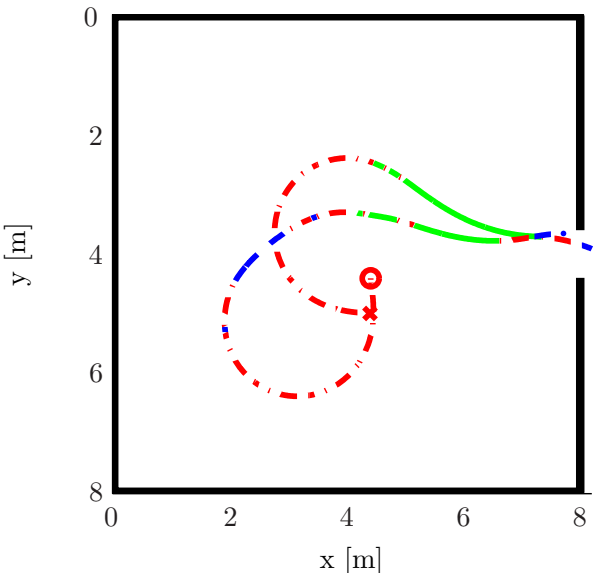


Figure 2.12: Path of successful (x) and unsuccessful (o) flight initializations of DelFly with the genetically optimized behavior (top-down view). Line styles denote different decision modes: Solid - window tracking; Dash - default action in low disparity; Dash Dot - wall avoidance.

2.6. DelFly Onboard Flight Testing

The BT was implemented on the camera module of the DelFly Explorer which is equipped with a $STM32F405$ processor operating at 168 MHz with 192 kB RAM. The BT is placed in series with the stereo vision and window detection algorithms as was done in simulation and was found to run at $\sim$12 Hz. The commands were sent from the camera module to the DelFly Explorer flight control computer using serial communication. The DelFly flight control computer implements these commands in a control system operating at 100 Hz.

2.6.1. Test 3D Environment

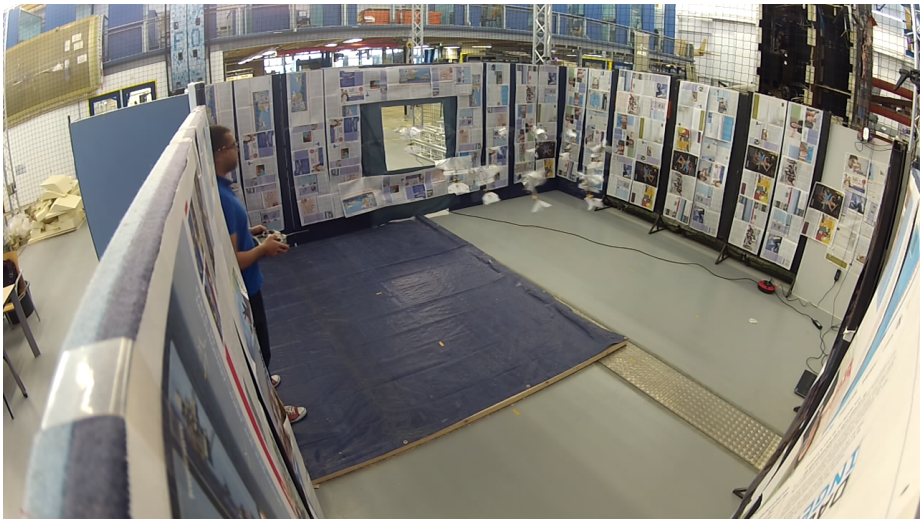


Figure 2.13: Photograph showing the room environment used to test the DelFly Explorer for the fly-through-window task. Inset is collage of DelFly as it approaches and flies through window.

The environment designed to test the MAV was a 5$\times$5$\times$2 m room with textured walls. A 0.8$\times$0.8 m window was placed in the center of one wall. The area behind the window was a regular textured area. Artificial texture was added to the environment to ensure we had good stereo images from the DelFly Explorer onboard systems. This texture was in the form of newspapers draped over the walls at random intervals. A sample photograph of the room can be seen below in Fig. 2.13.

2.6.2. Experiment Set-up

At the beginning of each run, the DelFly was initially flown manually and correctly trimmed for flight. It was then flown to a random initial position and pointing direc-

tion in the room. At this point the DelFly was set to autonomous mode where the DelFly flight computer implements the commands received from the BT. The flight continued until the DelFly either succeeded in flying through the window, crashed or the test took longer than 60 s. As the BT controls the horizontal dynamics only, the altitude was actively controlled by the user during flight which was maintained around the height of the center of the window.

All flights were recorded by video camera as well as an Optitrack vision based motion tracking system. Optitrack was used to track the DelFly as it approached and flew through the window to determine some of the same metrics of performance that were used in simulation. As a result, information on the success rate, flight time, angle of approach and offset to the center of the window can be determined.

2.7. CROSSING THE REALITY GAP

The flight speed of the DelFly was set to ~0.5 m/s, the same as was used in simulation. However, there were significant differences observed between the system simulated in SmartUAV and that in the flight tests. The most significant observations are summarized in Table 2.3. In short, the turn radius was smaller and the actuator response was faster and asymmetric. Additionally, aileron actuation would result in a reduction in thrust meaning that active altitude control was required from the user throughout all flights. It was also observed that there were light wind drafts around the window which affected the DelFly's flight path. These drafts would typically slow down the DelFly's forward speed and push it to one side of the window.

Table 2.3: Summary of the reality gap

Parameter	Simulated	Reality
Flight Speed [m/s]	0.5	0.5
Minimum Turn Radius [m]	1.25	0.5
Actuator Response Time [s]	2.2	<1
Decision Loop Speed [Hz]	10	12
Actuator Deflection	Symmetric	Asymmetric
Environmental	No Disturbances	Drafts

With these significant differences between the model used to train the BTs and the real DelFly there was a clear reality gap present. Initially both behaviors were not successful in flying through the window. To adjust the behavior to improve the performance we first considered the definition of success as defined by Jakobi [90]. In his chapter he suggested that the performance of the robotic system should be judged on a subjective measure of how reliably the robot performs the task in reality with no consideration to how the behavior achieves the task objective. In the case of this chapter, that would simply be defined as how often the DelFly flies through the window.

We initially tried to directly adjust the behavior in reality without comparing it to the

behavior seen in simulation. To improve the fly-through-window performance we mainly considered the final portion of the flight but this proved ineffective. This results from the fact that the embodied agent's success is tightly coupled with interaction of the robot's sub-behaviors during the entire flight. For example, the way the DelFly wall avoidance sub-behavior performed defined its approach to the window in such a way that the window approach sub-behavior would be successful. This suggests then that to achieve a task reliably in reality the robot must behave similarly to that observed in simulation for all sub-behaviors.

The insight into what parameters to change and how, comes from the user's understanding of the BT. From this the user can identify individual sub-behaviors. The technique of grouping nodes into sub-behaviors can be seen in Figures 2.3 and 2.10. This segmentation of the behavior helps to identify individual gaps simplifying the behavior update process.

To demonstrate this let us first look at the evolved behavior tree shown in Fig. 2.10 which can be considered as made up of three sub-behaviors. Let us first look at the window detection sub-behavior. We flew the DelFly around our test room and observed the window response value was never achieved with the certainty value of 69 (a lower value represents higher certainty that a window is in the frame). We increased the threshold of node 7 till the node would be activated by the window but false positives from other locations would not be likely.

Let us now investigate the wall avoidance sub-behavior. This mode is entered when the total disparity is larger than a threshold set by node 3. Observing the behavior in Fig. 2.12, the DelFly tries to circle in around the center of the room entering the wall avoidance mode at $\sim$4m from the wall in the 8 $\times$ 8 m room. This would suggest that the real DelFly should enter this mode at $\sim$2.5 m in the real 5 $\times$ 5 m room so the threshold in node 3 should be changed accordingly.

It should be noted that it appears that evolution has optimized the DelFly behavior to fly through windows in square rooms. The approach of avoiding walls at a fixed distance to line the DelFly up for the window entry would be more difficult if the window was not in the center of the wall or if the room size changed. This reiterates the strong coupling between optimized behavior and the environment that is characteristic of ER. It is therefore essential to vary the environment sufficiently to encourage the EA to converge to solutions robust to changes in the environment.

Last but not least, applying this to the wall avoidance action, the simulated DelFly had a minimum turn radius of 1.25 m which was much smaller in reality. A scaling factor was applied to increase the turn radius to that seen in simulation.

Using this approach, tuning these parameters took about 3 flights of about 3 minutes each to result in behavior similar to that seen in simulation. The updated behavior can be seen in Fig. 2.14.

This same approach was used with the user-defined BT with significantly more nodes and took a total of 8 flights of about 3 minutes each to tune the parameters to mimic the behavior observed in simulation. The updated behavior can be

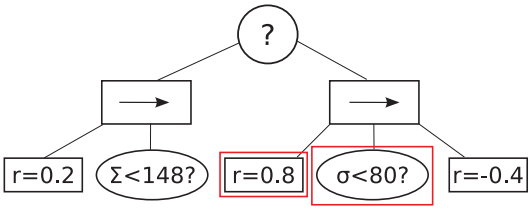


Figure 2.14: Graphical depiction of genetically optimized BT after modification for real world flight. Encapsulating boxes highlight updated nodes. x is the position of the center of the window in frame, σ is window response value, Σ is sum of disparity, Δ is the horizontal difference in disparity and r is the aileron deflection setting for the DelFly Explorer.

seen in Fig. 2.15.

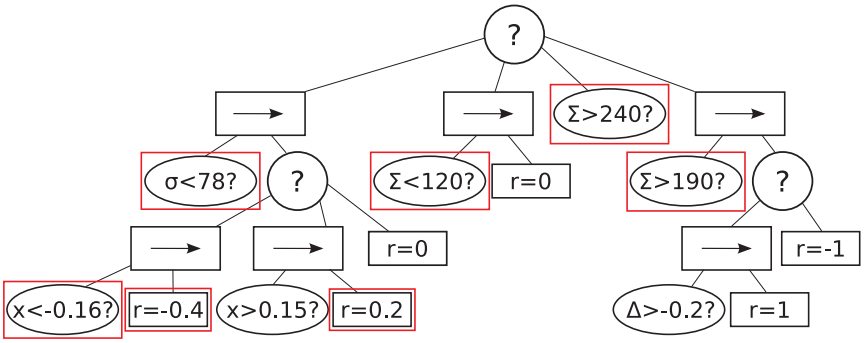


Figure 2.15: Graphical depiction of user-defined BT after modification for real world flight. Encapsulating boxes highlight updated nodes. x is the position of the center of the window in frame, σ is window response value, Σ is sum of disparity, Δ is the horizontal difference in disparity and r is the aileron deflection setting for the DelFly Explorer.

2.8. FLIGHT TEST RESULTS

26 test flights were conducted for both the user-defined behavior as well as the genetically optimized BT[1]. The results of the tests are summarized in Table 2.4. It can be seen that the success rate of both behaviors is reduced for both behaviors but notably, the relative difference of the two behaviors is maintained. Additionally, the other performance parameters which are the characteristic behavior descriptors are similar to that seen in simulation. This suggests that the user adaptation of the real behavior to emulate the simulated behavior was successful. The relative performance of the behaviors is also similar to that seen in simulation. The mean flight time of the behaviors was reduced but notably the relative flight times of the

[1]An accompanying video with some of the test flights can be found at: https://www.youtube.com/watch?v=CBJOJO2tHf4&feature=youtu.be

behaviors is the same as seen in simulation. The reduction in the time to success can be explained by the reduced room size.

Table 2.4: Summary of flight test results

Parameter	user-defined	genetically optimized
Success Rate	46%	54%
Mean flight time [s]	12	16
Mean approach angle [°]	16	37
Mean distance to window center [m]	0.12	0.12

The mean distance to the center of the window was higher for the user-defined behavior than observed in simulation. This can be the result of the drafts around the window pushing the DelFly to the edges of the window. This draft would also push the approaching DelFly into the window edge on some approaches. The time to success was lower for both behaviors as compared to the values observed in simulation. This is mainly due to the smaller room size used in reality.

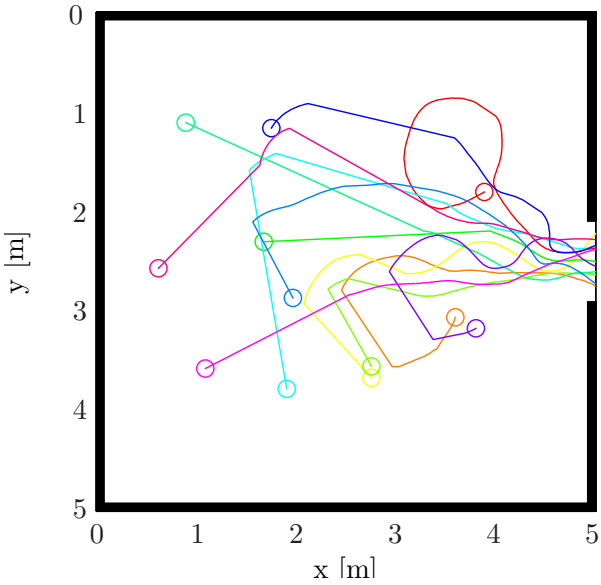


Figure 2.16: Flight path tracks of the last 7 seconds of all successful flights for the user-defined behavior. o represents start location of each flight.

Notably, the user-defined behavior showed the characteristic failure of being caught in corners. This happened 4/26 flights for the user-defined behavior but not once in the genetically optimized behavior. This is representative of the observations of the behavior in simulation, a fundamental deficiency of the bi-directional wall avoidance in a room with corners. This observation additionally suggests that the behavior seen in simulation is effectively transferred to the real DelFly. Figures 2.16 and

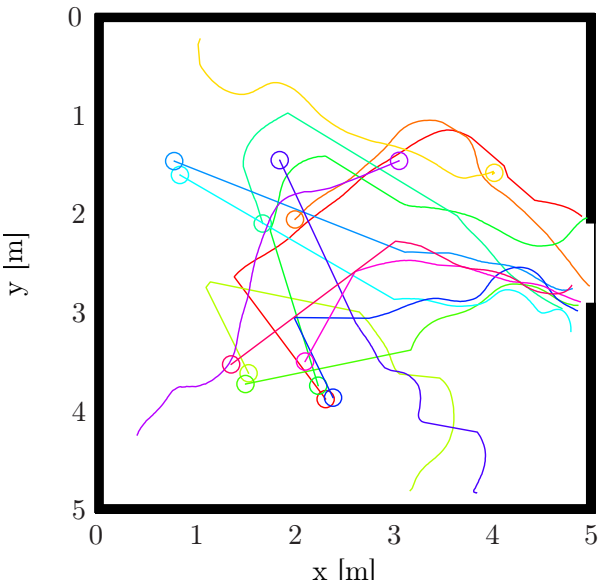


Figure 2.17: Flight path tracks of the last 7 seconds of all unsuccessful flights for the user-defined behavior. o represents start location of each flight.

2.17 show the last 7 seconds of the user-defined behavior for all flights grouped in successful and unsuccessful tests respectively. The Optitrack flight tracking system did not successfully track the DelFly in all portions of the room resulting in some dead areas but did accurately capture the final segment of the window approach.

These plots show that the DelFly tried to approach and fly through the window from various areas of the room at various approach angles. Approaches from areas of high approach angle typically resulted in a failed flight as the DelFly would hit the edge of the window. Additionally, the crashes in the wall due to being caught in corners can also be seen. Fig. 2.18 shows one full successful and unsuccessful flight of the DelFly user-defined behavior.

Similarly, Figures 2.19 and 2.20 show the successful and unsuccessful flights of the genetically optimized behavior as captured from the Optitrack system. In these figures it can be seen that the flight tracks of genetically optimized behavior are tightly grouped with the same behavior repeated over multiple flights. The DelFly always approaches from about the center of the room with a coordinated left-right turn described earlier. It can be seen that some of the unsuccessful flights occur when the DelFly makes an approach from farther way than normal so the coordination of the left-right turning is out of sync causing the DelFly to drift off course and hit the window edge. Fig. 2.21 shows one entire successful and unsuccessful flight of the genetically optimized behavior in more detail. The typical failure mode was turning into the edge of the window in the final phase of the flight. This is likely

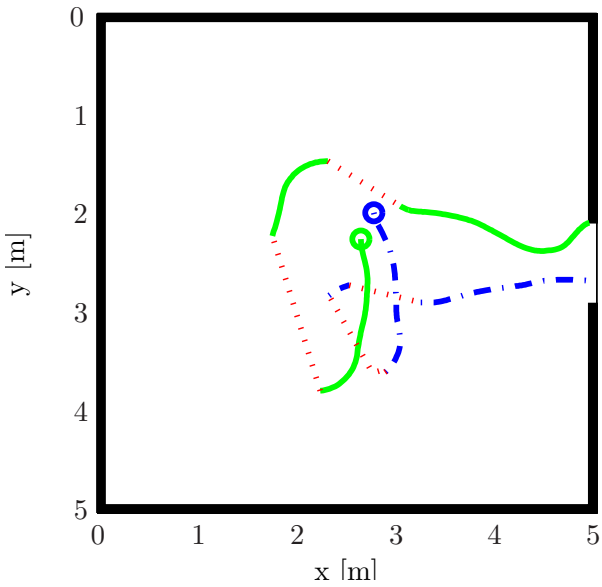


Figure 2.18: Flight path tracks showing one complete successful (dash dot) and unsuccessful (solid) flight for the user-defined behavior. o represents start location of test. Doted path shows where tracking system lost lock of the DelFly.

mainly due to the drafts around the window. Additionally, the faster decision rate of the BT in reality combined with the faster dynamics of the vehicle may play a role here as well.

The fact that the real world test was conducted in a different sized room than tested in simulation would have an effect on the success rate. In the future it would be interesting to observe the converged behavior if the simulated room were not kept constant during evolution. It is expected that this would result in behavior more robust to changes in the environment.

The failure mode of hitting into the window edge for both behaviors can be in part the result of the drafts observed around the window or in part due to the lack of detailed texture around the window. These external factors would affect the two behaviors equally so would not affect the comparison of behaviors.

The fact that both the user-defined and genetically optimized behaviors were initially not able to fly through the window but after user adaptation were able to fly through more than 50% of the time shows that the reality gap was actively reduced by the user. These results show that it is feasible to automatically evolve behavior on a robotic platform in simulation using the BT description language. This method gives the user a high level of understanding of the underlying behavior and the tools to adapt the behavior to improve performance and reduce the reality gap. Using this technique an automated behavior was shown to be at least as effective as, if not

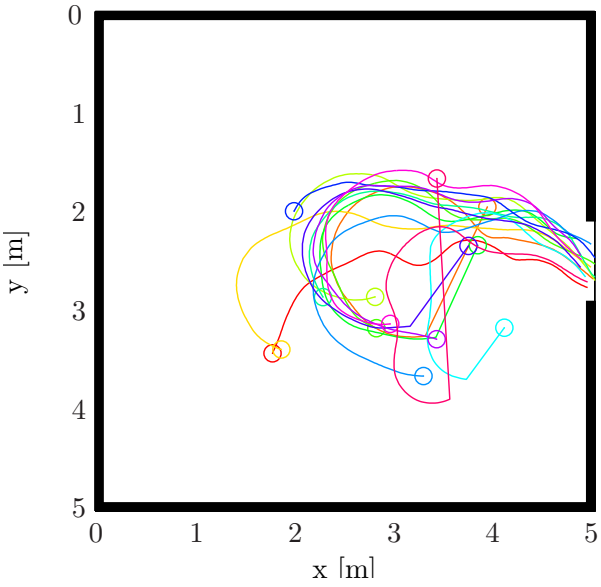


Figure 2.19: Flight path tracks of the last 7 seconds of all successful flights for the genetically optimized behavior. o represents start location of each flight.

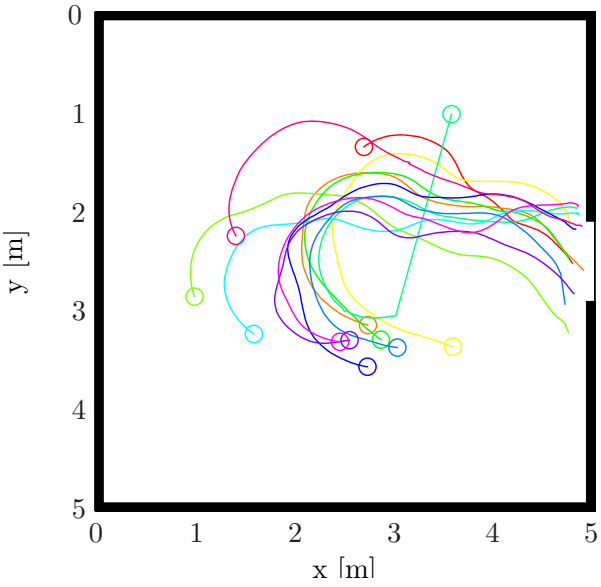


Figure 2.20: Flight path tracks of the last 7 seconds of all unsuccessful flights for the genetically optimized behavior. o represents start location of each flight.

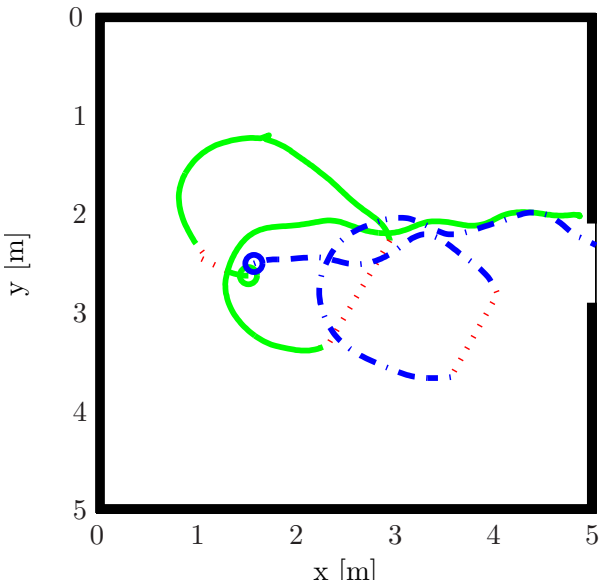


Figure 2.21: Flight path tracks showing one complete successful (dash dot) and unsuccessful (solid) flight for the genetically optimized behavior. o represents start location of test. Doted path shows where tracking system lost lock of the DelFly.

better than, a user-defined system in simulation with similar performance on a real world test platform.

2.9. Discussion

2.9.1. Behavior I/O Abstraction

In this chapter we use the descriptive and user legible framework of the BT to improve the user's understanding of the solution strategy optimized through evolution. With this insight the user can identify and reduce the resultant reality gap when moving from simulation to reality. This approach therefore necessitates that the elements of the tree are also conceptually tangible for the user, as such a higher abstract level was used for the sensory inputs. Unlike standard approaches which use ANNs where *raw* sensor data is used as input, we first preprocess the data into a form that a user can understand. The only question is then, how do we determine what is the best set of inputs to the robotic platform that will facilitate a robust and effective solution to be optimized by evolution.

Now, compared to typical ER approaches, preprocessing the inputs may affect the level of emergence of the EA such as hat seen in Harvey et al. [76]. That chapter demonstrated a robot completing an object detection task which was simplified by

an EA to the correlation of just a few image pixels. This level of optimization may not be possible if the inputs are preprocessed. However, preprocessing the input data typically reduces its dimensionality, thereby reducing the search space of the EA. This reduction in search space is crucial as we implement this technique on even more complex task and robotic platforms.

Robotic actions are typically not robust as they are susceptible to unmodeled simulator dynamics and changes in the operating environment. For example, in this chapter we set the output of the BT to be the rudder deflection which in hindsight is not a very robust parameter to control. It may have been more effective to have controlled the turn rate and have a closed loop control system controlling the actuator deflection. The closed loop controller would reduce the BT's reliance on the flight model in simulation. This would make the behavior more robust on the real robot inherently reducing the reality gap. The concept of using nested loops to bound control systems in order to improve robustness is a concept long used in control theory.

Considering the reality gap, recent work suggests that by limiting the EA to a set of predefined modules can actually improve the optimized behavior to the eventual reality gap [67]. In this work, Francesca et al. compare an optimized FSM using a limited set of predefined modules to a traditional system using an ANN. The two systems performed similarly in simulation but the ANN performed significantly worse in reality whilst the FSM maintained its performance. Francesca et al. present their work in the context of the *bias-variance* trade-off where they suggest that the introduction of the appropriate amount of bias will reduce the variance of the optimized system thereby improving its generality. Bias can be introduced to an optimization problem by limiting the representational power of the system, which in this case is achieved by limiting the options of the optimization to a limited input-output state space [50]. This idea can also be considered in this chapter where the limitation of the state space is not a hindrance or a limitation of the system but is in fact a benefit of this approach.

The abstraction of the behavior from the low level sensor inputs and actuator output importantly not only introduces a bias but additionally shields the behavior from the simulation mismatch causing the reality gap. The improved intelligibility in combination with the improved generalizability and robustness to the reality gap should ultimately make the approach presented in this chapter more suitable for extensive use in real robots attempting complex tasks than conventional ER approaches.

2.9.2. SCALABILITY

The task completed in this chapter is more complex than other ER tasks typically quoted in literature. Yet in the larger scale of autonomous navigation this task is only just a start. To facilitate this growing task complexity we will recommend some points for future research. Firstly, it is interesting to investigate the implementation of memory and time to the BT. Memory could be implemented as elements of the

BlackBoard that are not outputs of the BT to the platform but rather just internal variables. Time could be added by including a Running state to the nodes where they would hold till the action is completed. Alternatively, an explicit *Timer* node could be added that would run for a given number of ticks.

Another point worth consideration is the addition of a *Link* node to the BT framework. This node creates a symbolic link to a static BT branch outside of the tree. Evolution could select branches of its own behavior which could be linked and reused in other parts of the tree. This should help the optimization to reuse already developed behavior effectively throughout the tree. This would provide the EA with not only the raw materials to build the behavior but the ability to save combinations of these raw materials in a blueprint which can be reused at will.

With that said, the technique described in this chapter is dependent on the user's understanding of the underlying robotic behavior, so how does this change with the growing task complexity? We showed in this chapter that the BT can be broken down into sub-behaviors which helps the user to understand the global behavior. The prioritized selection of behaviors based on their location in the tree creates an inherent hierarchical structure. This structure will automatically group the nodes of a sub-behavior spatially in the tree. This makes the identification of the sub-behaviors straight forward. Tuning of the sub-behaviors would be accomplished using a divide and conquer approach, one sub-behavior at a time.

2.9.3. EVOLUTION OF BEHAVIOR TREES FOR BEHAVIORAL MODELING

The BT framework could also be used to model existing behavior or cognition of robots or animals [75]. This would be in a similar vein as a recent ER study on odor source identification, in which the insight into the evolved neural controller's strategy was verified by constructing an equivalent FSM controller [42]. Instead of manually designing such a controller, EAs could be used to optimize a BT to best approximate the behavior of a robot or animal. The BT, optimized to mimic reality, would give researchers increased insight into the underlying system dynamics. To mention a few examples, this approach can be applied to: self-organisation, swarming, emergence and predator-prey interaction.

2.10. CONCLUSION

We conclude that the increased intelligibility of the Behavior Tree framework does give a designer increased understanding of the automatically developed behavior. The low computational requirements of evaluating the Behavior Tree framework makes it suitable to operate onboard platforms with limited capabilities as it was demonstrated on the 20 g DelFly Explorer flapping wing MAV. It was also demonstrated that the Behavior Tree framework provides a designer with the tools to iden-

tify and adapt the learned behavior on a real platform to reduce the reality gap when moving from simulation to reality.

Future work will also investigate further into optimizing the parameters of the evolutionary algorithm used in this chapter. Multi-objective fitness functions and adaptive simulated environments are possible avenues to improve the generality of the optimized behavior. Additionally, work will be done on investigating how Behavior Trees scale within Evolutionary optimization, both in terms of behavior node types but also in task complexity. Regarding the DelFly, the most immediate improvement would be extending the automated control to include height facilitating extended fully autonomous flight.

3

Closed-Loop Control to Bridge the Reality Gap

Improve a mechanical device and you may double productivity, but improve man, you gain a thousandfold.

- Khan Noonien Singh

Star Trek: The Original Searies, Season 1 Episode 24

The contents of this chapter have been published as:

Title	Abstraction as a Mechanism to Cross the Reality Gap in Evolutionary Robotics
Conference	From Animals to Animats 14: Proceedings of the 14th International Conference on Simulation of Adaptive Behavior (SAB2016), Aberystwyth University, Wales, UK: Springer International Publishing, 280–292, 2016
Authors	K.Y.W. Scheper and G.C.H.E. de Croon

Title	Abstraction, Sensory-Motor Coordination and the Reality Gap in Evolutionary Robotics
Journal	Artificial Life, 23(2):124–141, 2017
Authors	K.Y.W. Scheper and G.C.H.E. de Croon

Aside from the behavioral representation, the previous chapter highlighted that a significant reality gap is present at the lowest level of the control of a robot. Due to differences in the actuator effectiveness or poorly simulated dynamics, the learned motion primitives in reality are often different than those in simulation. This portion of the reality gap can be tackled by abstracting away from the low level control and adding a closed-loop mechanism to minimize any actuation errors on the real robot.

This chapter presents an investigation into abstracting away from low level actions and how this effects the reality gap and the optimization power of the automatic behavior generator. Here, two controllers with different levels of abstraction are optimized to form an asymmetric triangle with a homogeneous swarm of Micro Air Vehicles. These two controllers are transferred into real robots and the reality gap in both instances is reported.

3.1. Introduction

ER is a robotic design methodology centered around the concept of evolving a robot's body and mind. Since its origins in the 1990's, it was thought that the best way to facilitate the evolution of intelligent behavior was to provide the robot with raw sensor readings and control the motor output such that the resultant behavior would solve the given task [128]. We refer to individual inputs and outputs accessible to the evolutionary process as a *primitive*. Having evolution utilize all available primitives gives the optimization the greatest freedom whilst removing user bias to the final solution as primitive pre-selection is not necessary.

This design method was initially met with a series of quick successes with reactive agents solving type-2 problems[1]: obstacle avoidance [61], object characterization [15, 129], mapless spatial localization [70], just to name a few. These tasks were however relatively simple when compared to other autonomous robotic applications. Additionally, the robotic platforms used were rudimentary with relatively few sensors and actuators as well as slow dynamics. Extending ER to more complex systems has proven difficult.

One of the major problems limiting the study of ER on more complex robotic systems and tasks is the *reality gap*. This gap is defined by the differences between the relatively low fidelity simulated environments where the evolution takes place and the real world. Evolutionary optimization often tends to exploit the intrinsic characteristics of the sensory inputs, motor actions, and feedback between these two. Due to differences between simulation and reality, these solutions don't transfer well to the real robot.

The tight coupling between the actions taken by the robot and the sensory inputs received is referred to as Sensory-Motor Coordination (SMC). Utilizing SMC, an agent can partially determine the sensory input patterns it receives by acting in a particular way [129]. It has been suggested that this is the main reason that the simple agents evolved have been able to solve many non-trivial type-2 problems.

With this in mind it may not be so beneficial to use low-level, unprocessed sensory inputs and motor outputs as these tend to be environment and robot specific. The combination of this reliance on a tight coupling between sensing and action and the level of sensor and action modeling fidelity tends to exacerbate the reality gap, limiting progress in the ER research.

Additionally, as we move to more complex tasks, the use of low-level primitives makes it more difficult for evolution to find good solutions early on in the evolutionary process. This so called *bootstrap problem* is caused by the fact that the fitness function rewards performance on the complex, high-level task while the robot has to master low-level sensor processing and control. It is very difficult to design fitness

[1] Type-1 problems denote data sets where a direct input-output mapping can be found through statistical means. Type-2 problems do not have a direct mapping [34].

functions that guide evolution in the correct direction whilst not introducing too much designer bias to the final solution.

The main contribution of this chapter is to show that abstracting away from the conventionally accepted use of low-level primitives can make the optimized behavior more robust to the reality gap. To do this, we optimized two controllers to solve a decentralized formation task with a homogeneous swarm of three quadrotor MAVs. One controller, which we denote as the *low-level controller* controls the rotor rpm of the quadrotor whilst the second controller will command a velocity which is sent to a closed loop control system which controls the vehicle. Additionally, we investigate the effect of using a closed loop control system and high-level primitives has on the ability of evolution to develop SMC.

The following sections will describe work done to achieve our goal beginning with Section 3.2 where we discuss SMC in more detail. Section 3.3 discusses the task we optimized followed by Section 3.4 with a description of the real vehicle used in our tests as well as the kinematic model implemented for the evolutionary optimization. Section 3.6 details the implementation of the evolutionary optimizer used in this chapter and the results of the optimization. The real world flight test results are presented in Section 3.7 and a discussion of the reality gap is presented in Section 3.8.

3.2. SENSORY-MOTOR COORDINATION

The main differentiator of autonomous robotic systems and many other forms of Artificial Intelligence (AI) is the core principle that robots must physically interact with a real world environment. Robots are embodied agents which must perceive and understand the world around them in order to take rational actions to achieve their goals. This fact makes it imperative to understand embodied cognition and how it relates to rational behavior. Early work in this field had trouble understanding the cognition process until John Dewey first proposed the concept of a *sense-think-act* cycle [49]. He theorized that instead of an open loop cognitive process where a perceived state prompts and action, humans in fact exhibited a closed loop cognitive process by which the actions we take alter our perceived state. Thus creating a feedback loop with no clear initial point. He coined this tight coupling between perception and action *Sensory-Motor Coordination*. These ideas in human cognition are most notably further developed by Jean Piaget where he investigated the development of SMC in infants [141] and James Gibson's work in perception in aircraft pilots [68]. This concept can also be generalized to embodied agents within an AI perspective [28, 33, 139].

Pfeifer et al. state that SMC serves 5 main purposes: (1.) physical control over objects (i.e. interaction with the world); (2.) perception (i.e. understanding the world); (3.) induced correlations of the world which reduces the dimensionality of the sensor-motor space (i.e. simplifying the world); (4.) integration of several sensory

modalities (i.e. sensor fusion); and (5.) for sensor-motor coordination itself [139]. They go on to say that not only does SMC make correlations it helps to filter out sensory inputs which are not correlated to the action required to achieve a task [138]. SMC can be seen as a structuring of the sensor inputs of our complex world to facilitate the development of intelligent behavior [114].

If we consider our agent from a dynamical systems perspective, SMC can potentially be seen as a lower dimensional projection of the higher dimensional interaction with the world. If so, then optimizing behavior actually entails placing behavioral attractors in this system which will solve the task at hand [122].

In ER, SMC has also been used in connection with self-organization and emergence. Nolfi suggests that in embodied cognition, behavior cannot be considered based on internal mechanisms only but in-fact emerges from a strong coupling with the agent's interaction with the environment [128]. Where emergence can be explained in three ways: (1.) an agent has a behavior that is surprising and not fully understood; (2.) emergence refers to a property that is not constrained to any of its parts (i.e. self-organization); and (3.) behavior resulting from the agent-environment interaction whenever that behavior is not preprogrammed [140].

The general view in ER is to give evolution a bag of low-level primitives which it has to sift through using SMC to find relevant correlations needed to achieve the task. This adds little restriction or user biased direction to the task [114, 140]. This however has the unwanted side-effect of introducing a large bootstrapping problem for the optimization making the optimization of complex behavior difficult. Additionally, the low-level primitives are typically raw sensor inputs and raw motor outputs, which do not generalize well from one robot to another, nor from a simulated to a real robot. The use of low-level of the primitives is not conclusive to the development of complex robotic behavior within the ER framework.

Given these difficulties it seems beneficial to use higher level primitives when evolving behavior for a complex task but an unanswered question is: Can evolution still exploit SMC when using high-level primitives with an underlying closed loop controller? The remainder of this chapter will try to answer this.

3.3. Task

In this chapter, the evolutionary optimization is tasked with developing the behavior of a homogeneous swarm of quadrotor MAVs to form a given asymmetric formation. This task is based on that presented in [89] where a swarm of 3 SHERES spacecraft were optimized in simulation only. Before [89], methods had been developed to autonomously form symmetric formations with the asymmetric case proving difficult [88]. The design of asymmetric formations using a distributed control system without explicit roles in the formation is a non-trival task for most human designers making it an ideal task for automatic optimization.

Unlike [89], we use a simplified two dimensional formation task to make analysis of the resultant behavior more straight forward. The goal of the swarm is to achieve an asymmetric triangular formation with sides of length: 0.7 m, 0.9 m and 1.3 m. The MAVs can observe the relative position to the other members in the formation as well as relevant ego-motion primitives.

An ANN with one hidden layer was selected to perform the robotic control. A *tanh* activation function was used in the neurons and network weights were constrained to the range [-1,1]. Additionally, bias nodes were added to the input and hidden layer with the exiting network weights were constrained to the range [-5,5].

To sense the relative position to the other members of the swarm, the main input to the ANN is the sum of the Cartesian components of the relative positions of the other members of the formation ($\mathbf{r}$) defined in (3.1). Summing the components of all aircraft relative positions instead of inputting individual vehicle distances ensures the ANN cannot associate a specific input with a unique vehicle. Note that $\mathbf{r}$ is mathematically equivalent to triple the distance from the ownship to the centroid of the formation ($\mathbf{c}$). An additional and redundant input is the sum of absolute distances (d) as given by (3.2), this was added to aid the ANN achieve accurate positioning.

$$\mathbf{r} = \sum_{n=2}^{k} (\mathbf{p}_n - \mathbf{p}_1) \tag{3.1}$$

$$d = \sum_{n=2}^{k} |(\mathbf{p}_n - \mathbf{p}_1)| \tag{3.2}$$

Where $\mathbf{p}$ is the position vector of a vehicle and k is the total number of vehicles in the swarm. These inputs are computed for each vehicle where $\mathbf{p_1}$ is the ownship location. This formulation removes the possibility of inadvertently assigning vehicles a role which may occur if the relative position to each other vehicle was given a dedicated input to the ANN. Fig. 3.1 illustrates a possible solution to the formation problem and a sample computation of the inputs to one of the vehicles.

3.4. QUADROTOR KINEMATIC MODEL

The Parrot ARDrone 2 quadrotor MAV was used for all real world experiments shown in this chapter. This 420 g vehicle is equipped with a 1 GHz 32 bit ARM Cortex A8 processor running a Linux operating system [2]. The default flight software provided by Parrot was overwritten by custom flight software implemented using Paparazzi UAS, an open-source flight control software [77, 148].

The ARDrone2 has a typical X configuration widely used with quadrotors. These vehicles have two clockwise and two counter-clockwise rotors. Controlling pairs of

[2]Parrot ARDrone 2 technical specifications: https://www.parrot.com/eu/drones/parrot-ardrone-20-elite-edition

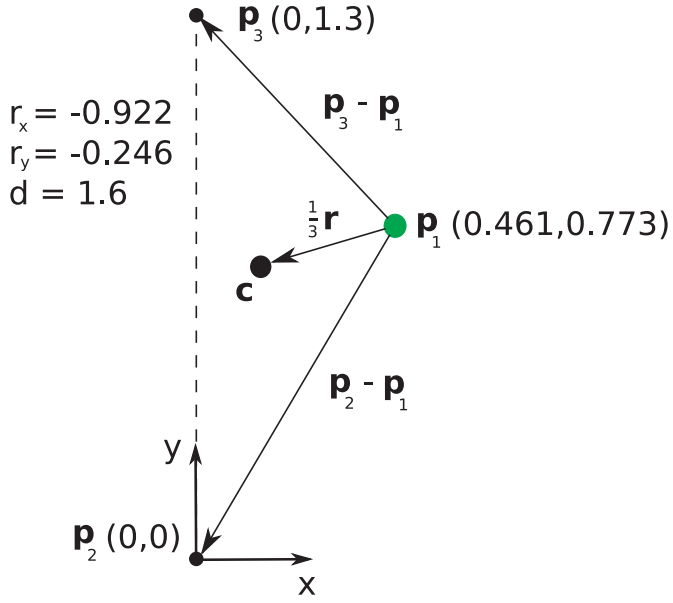
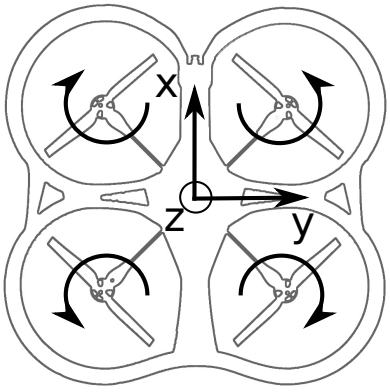
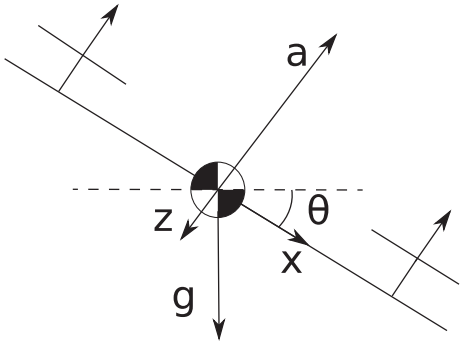


Figure 3.1: Illustration of a possible formation and a sample computation of the spatial parameters for the highlighted vehicle position (green). Notice that the combination of the three spatial inputs ($\mathbf{r}$ and d) represent a rotationally unique formation.

these rotors can create moments around the center of mass causing the vehicle to rotate with 6 DOF. Controlling all motors collectively creates a net linear acceleration in the direction of the rotors aligned with the vehicle's z axis. Fig. 3.2 shows the rotor directions and axis system for the ARDrone 2 quadrotor.



(a) Outline of the ARDrone 2 showing rotor directions and axis system.



(b) Resultant linear acceleration when assuming constant altitude and vehicle has an angle w.r.t. to ground.

Figure 3.2: ARDrone 2 quadrotor

In this chapter we will only investigate rotations around the x and y directions or roll and pitch respectively. Rotations around the z axis or yaw will be fixed for all flight tests. Additionally, all flights are limited to the xy plane and a closed loop controller is used to maintain a fixed altitude of the vehicle. To ensure the vehicle can maintain constant altitude the maximum allowable pitch and roll attitude of the vehicle will be conservatively constrained to $\pm 20^{\circ}$.

In this chapter the rotations around the x and y axises are generated by commanding a change in the rotational velocity of a pair of rotors. This will result in a moment around the respective axis causing an eventual angular acceleration of the vehicle. Given an actuator pair setting (ω), the angular acceleration ($\dot{\Omega}$) can be determined using equation (3.3).

$$\dot{\Omega} = \mathbf{G}\omega \tag{3.3}$$

Where Ω is the angular rate of the vehicle and $\mathbf{G}$ is the actuator effectiveness matrix. The values of the parameters in G have been identified through experimentation with a real ARDrone 2 [157]. Additionally, we model the rotors of the vehicle with a first order response with the time constant $\tau_{\omega} = 0.0306$.

Ignoring aerodynamic damping on the vehicle rotation, we can determine the vehicle attitude angles around the x axis (roll or ϕ) and y axis (pitch or θ) by simply integrating the angular rates. As all rotors have their thrust vector along in the z axis of the vehicle, when the vehicle is rotated w.r.t. earth, the rotors will have to produce more thrust to counteract gravity and maintain altitude. The acceleration due to the force created by the rotors can be computed as $a = g/(cos\theta cos\phi)$, where g is acceleration due to gravity.

Now, the tilted thrust vector of the rotors will also induce a linear acceleration in the direction of the vector. As the vehicle is not operating in a vacuum, as it picks up speed it will encounter atmospheric drag. In practice this means that the vehicle will not have a constant linear acceleration as suggested above but rather the vehicle will achieve a fixed velocity given a particular attitude. To model this, we have added two drag terms: profile or rotor drag ($D_r \propto v$) caused by the rotors moving through the air and parasitic drag ($D_p \propto v^2$) which is mainly caused by the non-lifting structure moving through the air [9]. The velocity kinematics are described in (3.4).

$$\dot{\mathbf{v}} = \begin{bmatrix} -sin\theta cos\phi \\ sin\phi \end{bmatrix} a - D_r|\mathbf{v}|\mathbf{v} - D_p\mathbf{v} \tag{3.4}$$

This velocity can then be directly integrated to update the vehicle position. The kinematics are summarized in (3.5).

$$\mathbf{x} = \begin{bmatrix} -\frac{1}{\tau_\omega+h} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{\tau_\omega+h} & 0 & 0 & 0 & 0 & 0 & 0 \\ G_{\omega_x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & G_{\omega_x} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -g \cdot tan & 0 & D_{x_1}|v_x| + D_{x_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{g \cdot tan}{cos\theta} & 0 & D_{y_1}|v_y| + D_{y_2} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \frac{u_x}{\tau_\omega+h} \\ \frac{u_y}{\tau_\omega+h} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \omega_{max} \tag{3.5}$$

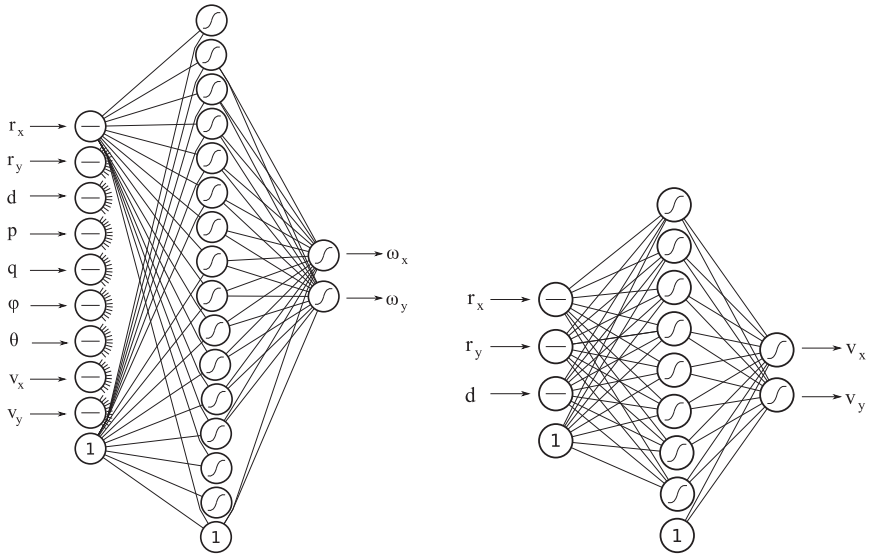
where $\mathbf{x} = [\omega_x, \omega_y, p, q, \phi, \theta, v_x, v_y]^T$, $\mathbf{u}$ is the output of the neural network, h is the time-step of the simulation and ω_{max} is the maximum rotor setting. Now, the ARDrone, like many other MAVs is equipped with a sensor suite to provide feedback on the real world performance of the vehicle. It is important to accurately model the sensor response of the vehicle along with the eventual sensor noise [90]. For this chapter we use the observation matrix y defined in (3.6) as identified from real world flight testing.

$$\mathbf{y} = \begin{bmatrix} R_p \sim \mathcal{N}(p, \sigma_p) \\ R_q \sim \mathcal{N}(q, \sigma_q) \\ \phi + \phi_0 \\ \theta + \theta_0 \\ v_x \\ v_y \end{bmatrix} \tag{3.6}$$

where σ_p and σ_q is the variance of p and q respectively and is set to 0.002 rad/s for this work. ϕ_0 and θ_0 are the roll and pitch bias of the Inertial Navigation System (INS), this is randomly generated each simulation run selected in the range $\pm 3^\circ$. In this chapter, we perform all experiments in a motion capture arena which is used to give position and velocity feedback to the vehicle, when fused with the on-board INS, the resultant velocity estimate is quite accurate. As a result, we will not add significant noise to this sensor reading as there is unlikely to be much reality gap here. In addition to the sensor inputs, the simulation constants $\mathbf{G}$, tau_ω, $\mathbf{D_1}$ and $\mathbf{D_2}$ were all randomly perturbed $\pm 5\%$ from their ideal values at the start of each simulation and for each vehicle.

3.5. CONTROL SCHEMES

To evaluate the effect of abstraction on SMC we will have evolution optimize two different solutions to the same problem applying control to different levels in the vehicle dynamics scheme described in Section 3.4. These two control schemes will be referred to as *low-level controller* and *high-level controller*.



(a) *low-level:* Tasked with controlling vehicle rotor pairs.

(b) *high-level:* Tasked with controlling velocity.

Figure 3.3: Inputs, outputs and structure of the neural controllers used in this chapter. A single hidden layer, fully connected ANN is used with a *tanh* activation function and an additional bias node at each layer. Shared inputs are the summed Cartesian components of the relative positions to the other vehicles in the formation ($\mathbf{r}$) along with the summed absolute distances (d). Additional inputs are roll rate (p), pitch rate (q), roll angle (ϕ), pitch angle (θ) and velocity ($\mathbf{v}$).

3.5.1. LOW-LEVEL CONTROLLER

For the low-level controller, the ANN is tasked with commanding the moments generated by the rotors. The output of the ANN is scaled to 15% of the total range of the rotor speeds to limit the maximum rotational rate of the vehicle.

As the controller must achieve a particular formation by controlling the moments of the quadcopter, the controller must have a feedback mechanism to achieve stable flight. As such, the inputs to the ANN controller are: roll rate (p), pitch rate (q), roll angle (ϕ), pitch angle (θ), velocity in the x axis (v_x), velocity in the y axis (v_y) and the formation inputs p_x, p_y and d. The structure of the ANN for this controller showing

the inputs and outputs can be seen in Fig. 3.3a. We chose for a fixed structure, single hidden layer network with 15 hidden neurons.

3.5.2. HIGH-LEVEL CONTROLLER

To demonstrate a high-level of abstraction, the evolved behavior will be tasked with controlling the velocity set-point of the quadrotor. The output of the ANN is linearly scaled to the velocity limits of the vehicle, which in the case of this chapter is set as v_{max} =0.5 m/s. This set-point is used as the input to closed loop control architecture which achieves the given velocity by controlling the individual motors for the quadrotor. This abstraction separates the optimization of the control system of the quadrotor the optimization of the high-level robotic behavior. Designing control systems is a long investigated area of research with many available methods to effectively control a wide range of systems. In its most simple form, a controller simply tries to reduce the error between a required state and the current state of a system. Much work has been done to make the control performance robust to unmodeled dynamics and environmental disturbances [112]. By definition, this makes the system more robust to the eventual reality gap between the dynamics of the simulated system and that of the real vehicle.

As a closed-loop controller will ensure performance of the vehicle dynamics, we can simplify the model described in 3.4 to a first order system with time constant $\tau_v = 0.3636$ as shown in (3.7). As before, the value of tau_v used in the simulation is randomly perturbed by $\pm$5% at the start of each simulation and is different for each vehicle.

$$\mathbf{v} = \begin{bmatrix} -\frac{1}{\tau_v+h} & 0 \\ 0 & -\frac{1}{\tau_v+h} \end{bmatrix} \mathbf{v} + \begin{bmatrix} \frac{u_x}{\tau_v+h} \\ \frac{u_y}{\tau_v+h} \end{bmatrix} v_{max} \tag{3.7}$$

This assumption greatly reduces the computational requirements of the simulator. As the time constant of the velocity system is relatively large we can use a large time step in the simulation as compared to the low-level controller which requires a very small time step. Additionally, the number of inputs to the ANN is also reduced to only the formation inputs p_x, p_y and d as shown in Fig. 3.3b as the closed-loop controller will account for all closed loop control.

3.6. EVOLUTIONARY ALGORITHM

All EAs share a common set of features, namely: Initialization, Evaluation, Selection and Recombination [58]. As the ANNs used in this chapter have a fixed structure, the EA is tasked with optimizing the neural weights interconnecting the neurons of the network. A population of 100 individuals was generated with neural weights initialized with a random number in the applicable range. Once initialized all individuals were simulated using a kinematic simulator with model structure as described in Section 3.4. The three vehicles were initialized at hover in a 4 $\times$ 4 m area with

a minimum inter-vehicular separation of 1.5 m. The centroid of this initial formation was set at the origin of the test area. All simulations were constrained to a maximum simulation time of 50 s. Simulations are cut short if any two vehicles came within 30 cm which would constitute a real world collision or any vehicle exceeded the maximum allowable bank angle of $\pm 20^{\circ}$. The resultant behavior was evaluated using the set of fitness functions to be minimized as described in (3.8).

$$\begin{aligned} f_1 &= 1 - \tfrac{t_{sim}}{t_{max}} \\ f_2 &= \sum_{n=1}^{k} |L_n - l_n| \\ f_3 &= \sum_{n=1}^{k} |\mathbf{v}_n|^2 \\ f_4 &= \begin{cases} 0, & |\mathbf{c}| < 3 \\ |\mathbf{c}|, & else \end{cases} \end{aligned} \tag{3.8}$$

Where L is the required distance, l is the distance between the vehicles at the end of the simulation, both L and l are sorted in ascending order. $\mathbf{v}$ is the velocity vector of the MAV at the end of the simulation and $\mathbf{c}$ is the location of the centroid of the triangle. t_{sim} is the simulation time at the end of the simulation and t_{max} is the maximum allowable simulation time. All fitness functions are limited to the range [0,100].

These equations attempt to promote the EA to optimize individuals that complete the desired formation (f_2) with zero resultant speed (f_3) with stable behavior indicated by long flights (f_1). f_4 tries to impose a restriction on the behavior for the formation to form within the 3 $\times$ 3 m box around the center of the simulation environment. This is to facilitate the real world test which must occur in a flight arena which is 10 $\times$ 10 m. The final fitness function provides a negative reinforcement when the simulation is prematurely terminated either due to a collision or due to exceeding the vehicle's maximum bank angle. f_2 and f_3 are similar to the function used in [89].

Once evaluated, the population is ranked using the widely used multi-objective Non-dominated Sorting Genetic Algorithm II (NSGA-II) with crowding sorting applied in the fitness domain [47]. Selection is achieved using a *tournament* of eight randomly selected individuals from the sorted population returning the highest ranked individual.

Mutation was the only evolutionary operator used in this chapter as some works have shown that mutation only evolution to be effective [175]. Each weight in the ANN was considered for mutation with a probability of 10%. Mutation consisted of a random perturbation of the previous value using roulette selection, small perturbations have a higher probability than larger ones. All weights were keep in the range

[-1,1] except for the weights from the bias nodes which were keep in the range [-5,5].

3.6.1. LOW-LEVEL CONTROLLER OPTIMIZATION

Fig. 3.4 shows the progression of the evolution for the low-level controller. Here the best individual is the individual with the largest euclidean distance from the origin in the 4-dimensional fitness space. Notice that around 500 generations were needed before basic flight capability was evolved after which a slow but steady improvement in the performance with the total formation error around 3 cm after 2500 generations. The total simulation time was about 18 hrs on a standard desktop PC with a simulation time step of 0.01 s which is necessary to simulate the fast dynamics of the actuators.

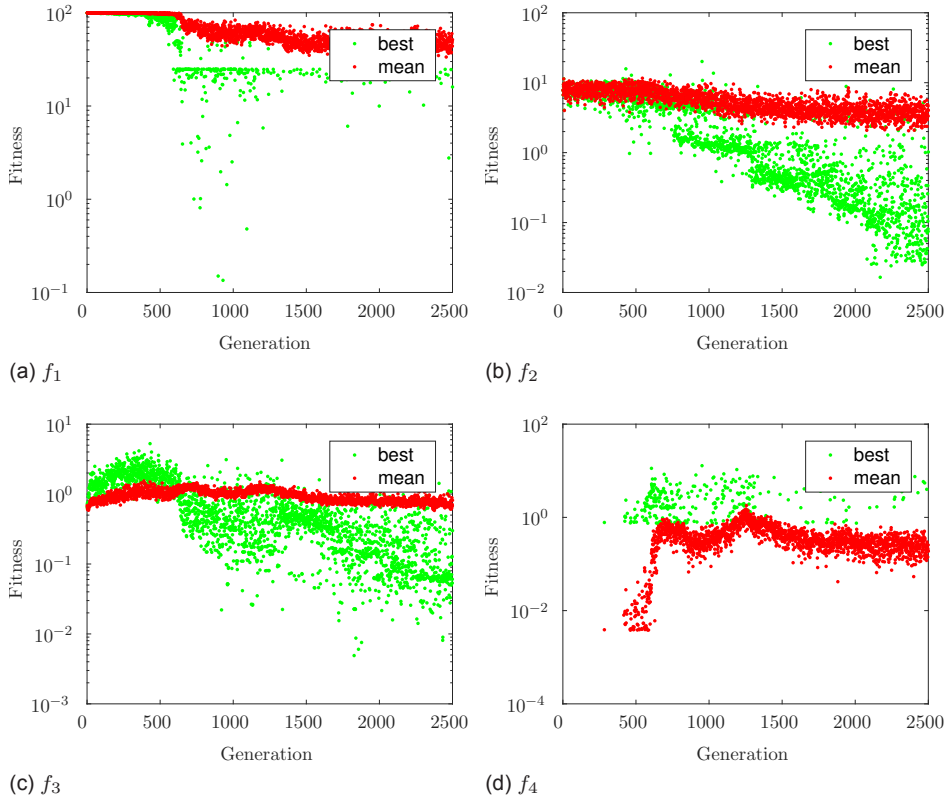
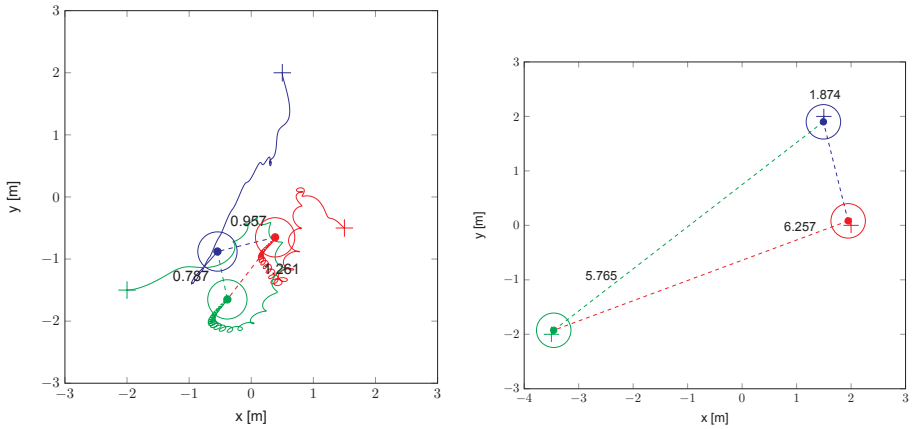


Figure 3.4: Progression of the performance of the best individual and the mean of the population during evolution for the low-level controller.

The individual with the lowest average fitness was evaluated in more detail by a validation run of 250 different initial conditions. During the validation run, the simu-

lation was not cut short if a collision occurred. A formation is considered accurate when the summed error of the lengths is below 0.15 m or 5 cm average error over the final 2 seconds of the flight. The results show that 79% of runs resulted in a successful triangle formation within 50 s, if we ignore crashes this increases to 90%. Of these successful runs, the mean error was 0.0189 m with a standard deviation of 0.01. Of the runs would not have completed the formation even when not considering crashes, the main failure case was that the vehicles exceeded the maximum attitude limits. Fig. 3.5 shows one of the successful runs of the formation behavior and one case where a collision would have occurred.



(a) Successful (b) Unsuccesful

Figure 3.5: Ground track of one successful and one unsuccessful flight of the low-level controller. The length of each side is shown in text, + marks the start location and the circle with a dot in the center marks the end location with the diameter of the vehicle to scale.

Interestingly, it was observed that the formation formed was always oriented with the same rotation to the axis system. This suggests that the behavior is trying to change the output of the network which would result in a unique set of formation inputs ($\mathbf{r}$ and d) rather than a rotationally invariant formation.

3.6.2. HIGH-LEVEL CONTROLLER OPTIMIZATION

Fig. 3.6 shows the performance of the best individual from each generation of the evolutionary optimization for this problem. This figure shows that evolution gradually reduces the error in the final vehicle distances and the final velocity. This figure also shows that the behavior does not guarantee a collision free flight for all initial conditions. After 1000 generations the average error of each length of the formation is about 2 cm. This was significantly faster than the low-level controller. Additionally, the total optimization time was 30 minutes on a standard desktop PC with a simulation time step of 0.2 s facilitated by the slower dynamics of the velocity controller response.

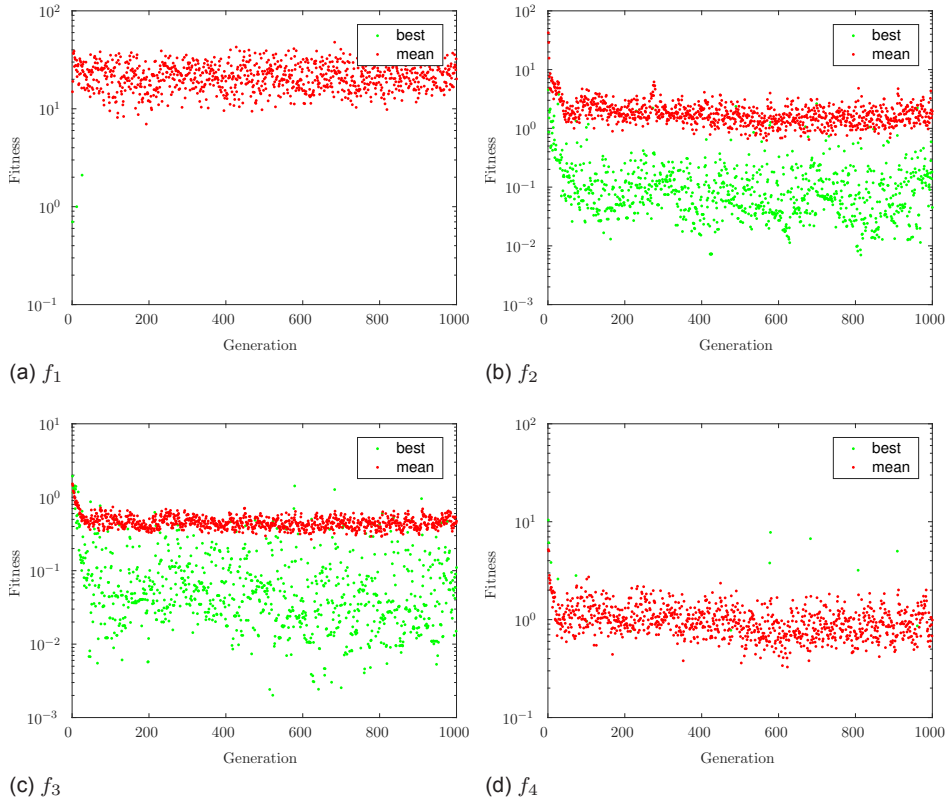
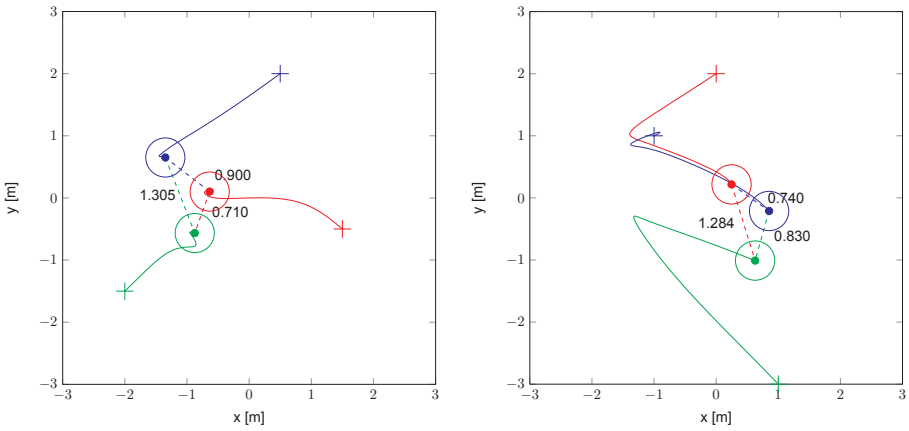


Figure 3.6: Progression of the performance of the best individual and the mean of the population during evolution for the high-level controller.

As before, the behavior for the best controller was evaluated by a validation run of 250 different initial conditions. The results show that 84% of runs resulted in a successful triangle formation within 50 s, again, if we ignore crashes this value increases to 98%. Of these successful runs, the mean error was 0.0222 m with a standard deviation of 0.0262 m. In the remaining 2% of the runs, the triangle was not formed after the given 50s but was in-fact formed later, if we extend the 50 s simulation to 100 s and ignore collisions we find a 100% success rate. These results are summarized in Table 3.1. Fig. 3.7 shows one of the successful runs of the formation behavior and one case where a collision would have occurred.

An interesting observation was that the formation created was observed to be in the same orientation to the axis system, a similar solution strategy to that seen in the low-level optimization. The orientation of the formation for the two controllers is different and in fact, each time the behavior was re-evolved a different orientation was optimized suggesting that there is no special geometric optimum but rather the output is being regulated to maintain a fixed combination of the inputs p_x, p_y and d.



(a) Collision free (b) Collision at (-1.25,1)

Figure 3.7: Ground tracks of a collision free flight and a flight that would have ended in a collision of 3 ARDrones performing the asymmetric formation task. The length of each side is shown in text, + marks the start location and the circle with the dot in the center marks the end location with the diameter of the vehicle to scale.

Table 3.1: Success rate of the formation task.

	Low-level	High-level
With All Termination Conditions	79%	84%
Ignore Collision	90%	98%
Ignore Collision and t_{max} = 100 s	91%	100%
Average Formation error	0.0189 m	0.0222 m

3.7. FLIGHT TESTS RESULTS

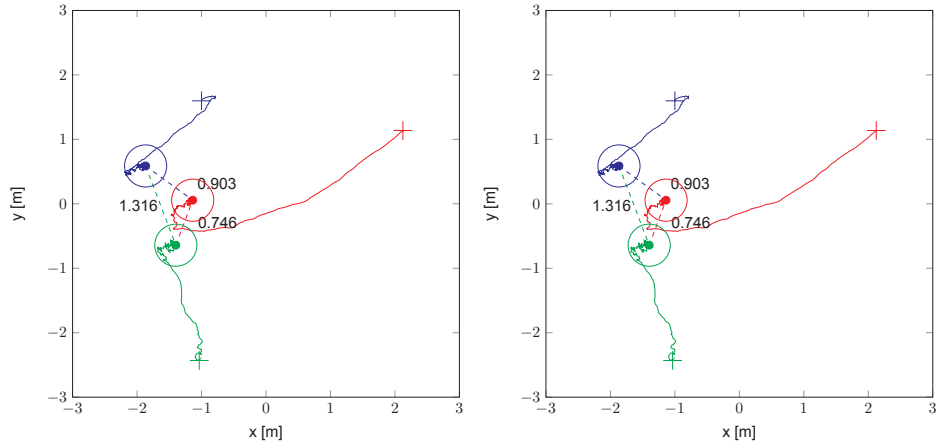
Moving from the simulated world to the real, the behaviors shown above were implemented on a swarm of 3 ARDrones. Flights were performed in an 10 $\times$ 10 m flight arena and the flight path of the vehicles was captured using an Optitrack motion camera system which was used to determine the relative location inputs to the neural network [3]. For the first set of tests, as in simulation, the 3 vehicles were initialized at random in a 4 $\times$ 4 m area in the flight arena with the centroid of the initial formation at the origin of the arena just like in simulation.

When testing the low-level behavior it quickly became apparent that there was a significant disparity map. When switching from hover into autonomous mode, the vehicles would quickly loose stability and exceed their 20° attitude limits ending the trail. The root cause of the gap was not immediately clear and was quite difficult to

[3] A video of some of the flight tests can be found at the following url: https://www.youtube.com/playlist?list=PL_KSX9GOn2P8ru70sZZj0H6K2R7lWCoDx

pin point given the multiple inputs and the fast dynamics of the vehicle.

The high-level controller did however successfully cross the reality gap with the flight path of one flight shown in Fig. 3.8 compared to a simulated flight with the same initial position.



(a) Real world (b) Simulation

Figure 3.8: Ground track of the real world flight test and the simulated flight for the same initial positions. The length of each side is shown in text, + marks the start location and the circle with a dot in the center marks the end location with the diameter of the vehicle to scale.

The flight path observed in reality was very similar to that predicted in simulation however, deviations can be seen when the vehicles come into close proximity of each other. This was due to aerodynamic interactions of the rotor downwash, something that was not modeled. The closed-loop controller was however able to cope keeping the vehicles close to the desired formation. Fig. 3.9 shows the commanded velocity of the ANN and the true vehicle velocity along with the result of the simulation.

In contrast to the simulation, the real-world quadrotors have clear velocity tracking errors and oscillations, in part due to the aerodynamic interactions. These tracking errors represent a significant reality gap. Despite this, the observed high-level behavior is very similar to that seen in simulation with the correct formation achieved with an average formation error of 0.034 m, only 10 cm deviation from that in simulation. The resultant stable formation in the face of these unmodelled external disturbances highlights the robustness of the optimized solution.

Looking more closely at Fig. 3.9 we can see some overshoot and oscillation in the real world velocity controller that was not seen in simulation. Unlike the first order system used to model the velocity response of vehicle in simulation, the real world controller resembles more a second order system with a short rise time. This

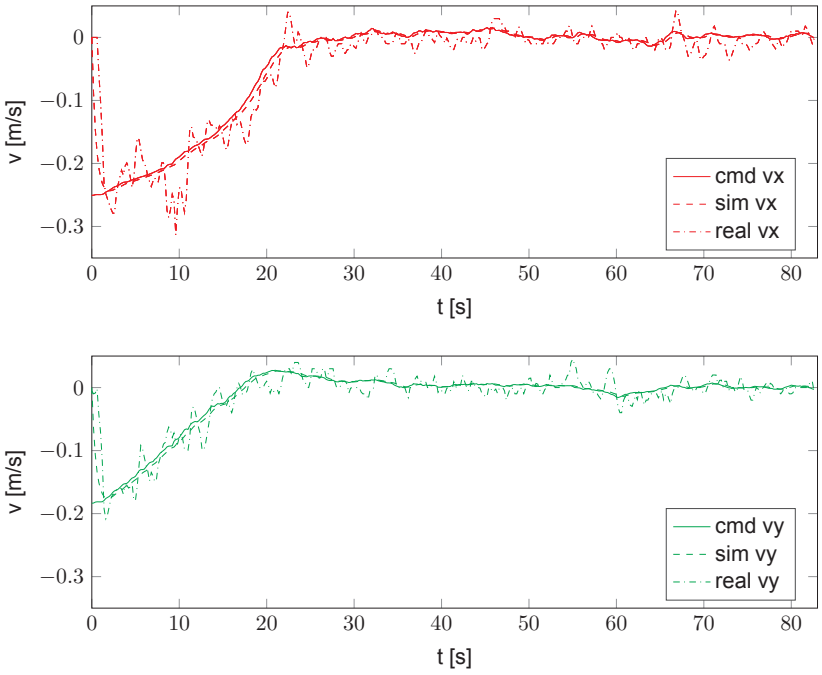


Figure 3.9: Tracking performance of the velocity controller on the simulated and real ARDrone 2 given the same velocity command highlighting the reality gap.

difference in simulated and real dynamics is one clear source of the reality gap. To investigate the robustness of the optimized behavior this controller, we implemented a second order system in our simulator with the following transfer function $\omega_0^2/(s^2 + 2\zeta\omega_0 + \omega_0^2)$. The natural frequency (ω_0) and damping ratio (ζ) was varied and tested with the previously optimized behavior. Table 3.2 summarizes the results of this investigation. The parameters identified to best fit the observed velocity response were ω_0 = 3.854 and ζ = 0.9. Interestingly, the behavior was robust to a wide range of velocity dynamics breaking down when the natural frequency dropped below 0.85. The damping ratio had little effect on the final formation performance but did have effect in the number of collisions as the behavior became more oscillatory.

3.8. ANALYSIS OF THE SENSORY-MOTOR COORDINATION

To analyze the effect of abstraction on the extent to which evolved robots exploit their environment and make use of SMC, we must first dive deeper into the optimized behavior. Firstly we will investigate an observation made in [89]. There, although the task was solved by the genetic optimization during analysis, the evolved behavior, researchers fixed two of robot satellites such that one leg of the formation was correct. When the third satellite was left free to move it did not settle into the cor-

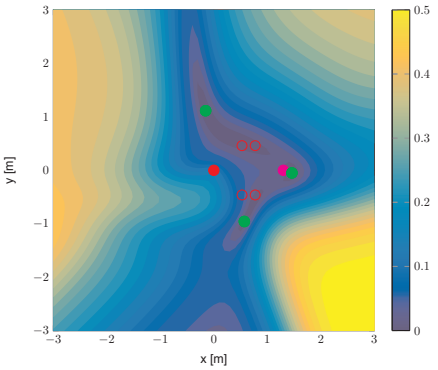
Table 3.2: Success rate of high-level controller when simulated with varying second order vehicle dynamics.

		ζ		
		0.9	0.6	0.3
ω_0	3.854	96.0%	95.6%	97.2%
	2.854	95.6%	94.0%	97.2%
	1.854	90.4%	94.4%	96.8%
	0.854	87.2%	85.2%	0%

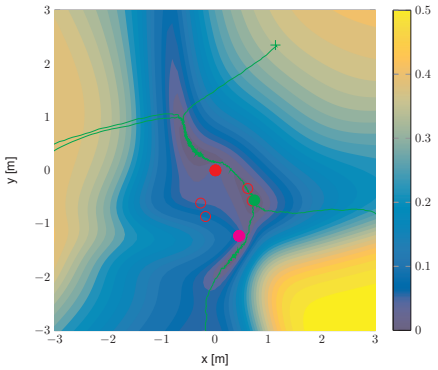
rect location to complete the formation. This led them to an interesting hypothesis: perhaps the asymmetric formation could only be reached if all three satellites were free to move. Here we will investigate if we can observe a similar phenomenon for our specific evolved solution, and evaluate whether SMC plays a role in successful formation flight.

In the flight tests performed in this chapter, it was observed that all successful formations resulted in a triangular formation with the same rotational orientation to the Cartesian axis system. The orientation can be seen in Fig. 3.7 and Fig. 3.8. Given the location of any two vehicles separated by any of the three length of the formation, there are four possible locations for the third vehicle that would satisfy the task given. The optimized solution seems to only utilize one of these. Additionally, there is a unique set of inputs to describe every possible rotational orientation of the triangle formation. The fact that the formation always converges to the same orientation suggests that the solution optimized in evolution is to define the output of the ANN such that the commanded velocities forces the vehicle into a specific relative location where the value for the inputs p_x, p_y and d produces a minimum velocity command which is stable to disturbances.

If we fix two of the vehicles as done in [89] and evaluate the velocity output of the ANN of the third vehicle when in various locations around the arena we can get an idea for the solution strategy as shown in Fig. 3.10. Fig. 3.10a shows the velocity map of the third vehicle when the other members of the swarm are aligned with the x-axis, an orientation not seen when the vehicles were all free. It can be seen that there are three stable minimums in this basin of attraction and none of them satisfy the given formation. Notably, although the highlighted spots are stable points when the two other vehicles are free, the commanded velocities of the other two vehicles is non-zero showing that this formation is not stable. In Fig. 3.10b, the two fixed vehicles are situated in a similar orientation to that seen when the vehicles were all free. This basin of attraction shows only one stable minimum which does satisfy the formation. We also performed real flight tests with two vehicles fixed along the correct orientation with Fig. 3.10b showing that the ground tracks of the real flights overlap almost exactly with this velocity field. This confirms the hypothesis of output shaping to converge to a single relative position and therefore set of inputs p_x, p_y and d and is a strong indication that the high-level controller is in fact using the SMC to complete the task.



(a) Velicity map when two fixed vehicles are fixed in the wrong rotational orientation to facilitate the formation.



(b) Velocity map when two fixed vehicles are fixed in the correct rotational orientation to facilitate the formation. Overlain in green are ground tracks of all real world flights. This shows that the real world performance mirrors what is expected from simulation quite well despite a clear reality gap.

Figure 3.10: Basin of attraction for one vehicle given the other two vehicles (shown in red and blue) are fixed in space. The hollow red circle highlights the solution to the formation problem and the green dots show the stable attractor points. Color bar represents the magnitude of the commanded velocity $|\mathbf{v_{cmd}}|$.

When considered from the point of view of the decentralized swarm, as mentioned when two vehicles are fixed, it is possible to enter a local minimum that is not observed when all are free to move. This would suggest that the solution strategy is based on the fact that the swarm is homogeneous and therefore there is an inherent modeling of the motion of the other members of the swarm. This implicit modeling can be also expressed as a form of environmental exploitation which is a hallmark of SMC. Given Nolfi's suggestion that the emergence of behavioral attractors is indicative of SMC [129], the high-level controller would appear to effectively exhibit SMC albeit on a more abstract level. Although the evolution has no access to the low-level control or sensory inputs, the resultant behavior was still able to exploit the implicit knowledge that the other members of the swarm would implicitly cooperate to solve the task.

3.9. DISCUSSION

It was not the goal of this chapter to suggest that the use of low-level primitives should not be used, or that it is not possible to evolve complex behavior using these primitives. Although we were unable to bridge the reality gap for our low-level controller, it does not mean that it is not possible. Likely, revising the modeled dy-

namics, use of a recurrent neural network or incremental evolution may provide the tools necessary to cross the gap but that is in itself one argument for using abstraction. We investigated what effect abstracting away from low-level primitives has on the effectiveness of the optimized behavior and the optimization process itself. We have shown that the abstraction of actions can reduce the effective dynamics of the robotic platform which in combination with the reduced simulation fidelity required, can reduce the computational load of simulations, speeding up optimization times. Shifting the action space to a higher level also helps to reduce the search space of the optimization reducing the bootstrapping problem. Importantly, we have demonstrated that abstraction does not necessarily prevent the emergence of SMC as some have suggested.

This chapter used abstraction on the actions but the authors expect similar success when applied to the sensory inputs where large gains can be seen in the use of preprocessed visual inputs. This however comes with a few caveats, by preselecting a subset of abstracted inputs for evolution we are effectively reducing the search space, which is beneficial for the optimization but also removing degrees of freedom which can be potentially detrimental. Open questions are: How do you select the level of abstraction; How do we determine the inputs necessary to complete the task at hand; How do we abstract away from raw inputs without injecting unnecessary amounts of designer bias? We hope to answer some of these questions in future work.

Reflecting on the evolution of the behavior itself, it is interesting to note that both behaviors did not find solutions which solved the formation while guaranteeing collision free flight. Collisions were in-fact the highest source of failed flights. This may be due to the fact that collision was used to preemptively end the simulation without an explicit negative reward. It may have been more effective to have some explicit fitness impact when a collision occurred.

The fact that the optimization selected a rotationally fixed solution to the formation problem may be a result of the selection of the fitness functions. The fitness functions used in this chapter tried to promote the formation of a triangle with three fixed length sides but gives no definition of the required orientation. Given that freedom, the optimization simply found the simplest solution to the problem, which in this case is a formation resulting in a unique combination of the relative position inputs.

3.10. Conclusion

In this chapter we investigated the application of abstraction on the inputs and outputs of a neural network controller within the Evolutionary Robotics paradigm. Two flight controllers using differing levels of abstraction were evolved to solve an asymmetric triangle formation task with a swarm of three MAVs. Both controllers were able to control the vehicles to complete the swarming task however only the controller using the higher level of abstraction successfully bridged the reality gap and

worked on the real robot. We have shown that abstraction can be a useful tool in making evolved behavior robust to the reality gap. Additionally, abstraction has been shown to reduce the level of fidelity required of the simulator as low-level interactions of the sensors and actuators are modeled by a higher level system thereby reducing optimization time. Most importantly, we have demonstrated that sensory-motor coordination, a typical emergent phenomenon of reactive agents, is not necessarily lost when abstracting away from the raw inputs and outputs, as some have suggested, but is rather is simply shifted to a higher level of abstraction.

The use of abstraction has the potential to accelerate the development of more complex robotic behaviors than have been previously attempted. There are however some open questions that must be answered before it can be widely used. The most significant of which is how this abstraction is selected. By only providing the evolutionary optimization with a subset of possible sensory inputs or actions, the user may inadvertently limit or unnecessarily guide the optimization process away from a potential optimum. Future work will investigate how the optimal input and output set can be automatically determined or optimized within the evolutionary process.

4

Efficient Event-Based Optic flow

Things are only impossible until they're not.

- Jean-Luc Picard

Star Trek: The Next Generation, Season 1 Episode 17

The contents of this chapter have been published as:

Title	Vertical landing for micro air vehicles using event-based optical flow
Journal	Journal of Field Robotics, 35(1), 69–01, 2018
Authors	B.J. Pijnacker Hordijk, K.Y.W. Scheper and G.C.H.E. de Croon
Contribution	The research that lead to the original published work was the result of the MSc graduate research of Bas Pijnacker Hordijk, whom I supervised together with Dr. Guido de Croon. Apart from the natural conceptual collaboration, I specifically prepared the onboard implementation of the software required to capture the data for the presented results. Additionally, I performed supplementary flight tests and data analysis required for the revised version of the paper which was eventually published.
Expansion	Since the early work presented in the Journal of Field Robotics, I have continued developing the software. Specifically, I implemented the optical flow algorithms on the Parrot Bebop 2 with the SEEM1 event-camera from Insightness. This re-implementation has resulted in better optical flow results and higher processing rate. The following will first present an edited version of the published paper followed by some updated results with the updated setup.

The tight coupling sensori-motor coordination used by embodied robots makes them susceptible to small changes in the environment. The previous chapter delved into improving robustness of the actuation side of this loop whilst this and the next chapter will delve into the perception side.

To investigate the how robotic behavior can be made robust to its input we will first introduce a novel robotic input, namely the dynamic vision sensor. The low

data throughput and low latency of the data produced by this event-based camera promises to facilitate high speed perception of the surrounding world. This novel camera architecture unfortunately requires novel computational infrastructure to convert the raw data into a usable form.

This chapter presents an efficient method to compute optical flow from data produced by a dynamic vision sensor and show the first use of this camera in the control loop of an autonomous robotic vehicle. This sensor system will be used in the following chapter to investigate how the reality gap is influenced by the type of robotic input used.

4.1. Introduction

Rapid advances in micro-electronics catalyzed the development of tiny flying robots [65], formally referred to as Micro Air Vehicles (MAVs). Due to their size and agility, MAVs have the potential to perform activities in confined and cluttered environments. However, achieving autonomous flight with very small MAVs (for example, the 20-gram DelFly Explorer [46]) is a significant challenge due to strict weight and power limitations for on-board equipment. The speed of cutting edge autonomous MAV navigation pales in comparison to their natural counterparts, the insect.

The main sensor system of most insects uses some form of visual light to perceive the environment around them. Visual navigation in flying insects is primarily based on optical flow, the apparent motion of brightness patterns perceived by an observer due to relative motion with respect to the environment [69]. In essence, optical flow provides information on the ratio of velocity to distance, such that the actual metric distance to the environment is not directly available. Instead, flying insects navigate based on certain visual observables extracted from the optical flow field that relate to ego-motion. Honeybees were seen to control their descent during landings based on the *divergence* of the optical flow field perceived from the ground [7]. When looking down, flow field divergence conveys the ratio of vertical velocity and height above the ground. By maintaining a constant divergence in downward motion, an observer approaches the ground while exponentially decreasing its downward speed. For flying robots capable of vertical landing, this is an interesting strategy. This application has been explored in several experiments with rotorcraft MAVs [41, 81, 82, 84].

While such optical flow based navigation strategies are bio-inspired, most visual sensors employed for measuring optical flow differ significantly from their natural counterparts. Commonly used miniature cameras operate in a *frame-based* manner: full frames are obtained by periodically measuring brightness at all pixels. This is a relatively inefficient process for motion perception, since the information output rate is independent of the actual dynamics in the scene. Static parts of a frame are recorded as well as changing parts, even though only the latter are relevant for motion perception. Therefore, follow-up processing of a full frame is necessary, which at present still requires significant processing. In addition, fast inter-frame displacements lead to motion blur, which limits the accuracy of resulting optical flow estimates or requires complex algorithms to account for this. These characteristics are undesirable for optical flow measurement on board miniature flying robots, which are subject to strict computational limits and exhibit fast dynamics.

In contrast, biological vision systems, such as the compound eyes of insects, employ an *event-driven* mechanism; they measure changes in the perceived scene at the moment of detection [147]. Several researchers have attempted to mimic the sensory system in insects in order to measure optical flow. For example, in [151] a tethered rotorcraft MAV was equipped with a 2-photodetector neuromorphic chip for measuring translational optical flow. In [63] a miniature curved compound eye

design, inspired by the fruit fly Drosophila, was presented.

In particular, *event-based cameras* are a promising class of sensors for optical flow perception. When a pixel of an event-based camera measures a relative increase or decrease in brightness, it registers an event, mapping its pixel location to the timestamp and sign of the change. This timestamp is obtained with microsecond resolution and latencies in the range of 3 to 15 μs. In addition, event-based pixel architectures enable intrascene dynamic ranges exceeding 120 dB [174]. These characteristics are highly desirable in robotic visual navigation. Experiments using event-based cameras demonstrated high performance of visual control systems through low-latency state updates, efficient data processing, and operation over a wide range of illumination conditions [37, 48].

This novel approach to visual sensing is in general incompatible with state-of-the-art computer vision algorithms for estimating optical flow, due to the lack of absolute brightness measurements. Therefore, several event-based methods for optical flow estimation [10, 11, 16, 17, 23] as well as benchmarking datasets [13, 150] have been developed. Of the existing techniques, the local plane fitting approach by [16] is the most promising based on its application to estimating time-to-contact (the reciprocal of flow field divergence) in simple scenes [31] and good results estimating optical flow visual observables in [150]. However, until now, no study has discussed an implementation of event-based optical flow into an optical flow based control loop of an MAV.

This chapter contains *three main contributions*. First, a novel method for estimating event-based optical flow inspired by [16] is introduced. Its applicability is extended to a wider range of velocities, while improving computational efficiency. Second, a method for incorporating event-based optical flow into visual estimation of divergence is proposed, which accounts for the aperture problem that occurs in most existing event-based optical flow methods. Third, the proposed algorithms for event-based optical flow divergence estimation are incorporated in a constant divergence landing controller on-board of a quadrotor. To the best of the authors' knowledge, this chapter presents the first functional event-based visual controller on-board of an MAV.

Figure 4.1: MavTec MAV flying outdoors with the DVS128 event based camera attached.

We begin by discussing related work concerning landing using optical flow, event-based vision, and optical flow estimation in Section 4.2. In Section 4.3 the estimation method for event-based optical flow is described and evaluated. Subsequently, Section 4.4 introduces the approach to estimating visual observables from event-based optical flow, followed by an assessment of its performance in combination with the optical flow method. Section 4.5 presents flight test results of the full estimation pipeline during constant divergence landing maneuvers. Lastly, the main conclusions and recommendations for future work are presented in Section 4.7.

4.2. Related Work

4.2.1. Event-Based Cameras

Inspired by the workings of biological retinas, event-based cameras rely on a sensing mechanism that fundamentally differs from their frame-based counterparts. In frame-based cameras the pixel values from all pixels are measured at fixed time intervals to produce a sequence of images. In event-based cameras, on the other hand, pixel activity is driven by light intensity changes. Whenever a pixel measures a local change, it produces an *event*. Specifically, this occurs when the pixel's logarithmic intensity measurement $I(x, y, t)$ (at pixel location (x, y) and timestamp t) increases or decreases beyond a threshold C:

$$|\Delta (\log I (x, y, t))| > C \tag{4.1}$$

Events are encoded according to an Address-Event Representation (AER) [107], which consists of event information encoded by an address and the timestamp of detection. Typically, an event encodes the pixel position (x, y), the timestamp t, and the polarity $P \in \{-1, 1\}$, which indicates the sign of the intensity change. A visualization of a basic stream of events, in comparison to an equivalent set of frames, is shown in Fig. 4.2.

The sensor used in this work is the Dynamic Vision Sensor (DVS) - specifically, the DVS128 - which is the first commercially available event-based camera [29]. It features a 128x128 pixel grid operating at an intrascene dynamic range of 120 dB, measuring events at 1 μs timing resolution with a latency of 15 μs [107]. A picture of the DVS is shown in Fig. 4.3. Since the availability of the DVS, other event-based cameras have been developed. Most notable are the Asynchronous Time-based Image Sensor (ATIS) [146], which measures absolute intensity as well as polarity for each event, and the Dynamic and Active pixel Vision Sensor (DAVIS) [22], whose pixels record events as well as full frames. Interesting in the context of this work is the 2.2 gram micro embedded DVS (meDVS) [36], which is highly suitable for on-board MAV applications.

Event-based cameras have several interesting applications for robotic navigation. Initial work has been performed on visual SLAM with event-based cameras [171,

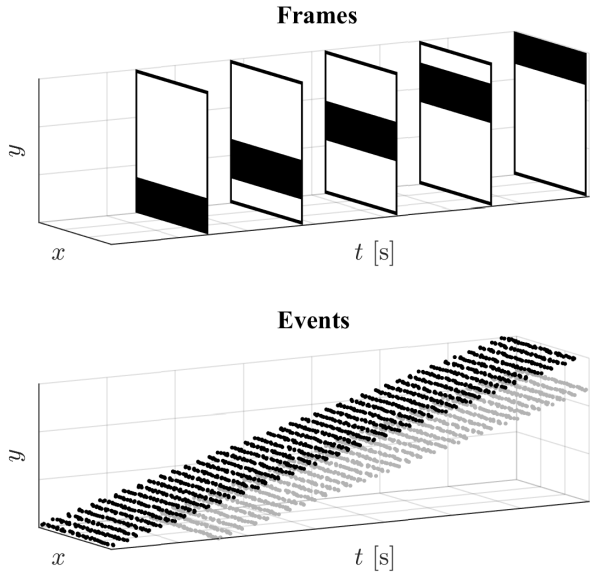


Figure 4.2: Frame-based and event-based visual output generated from a simple synthetic scene, in which a black horizontal bar moves upward. The events are visualized as points in space-time, hence showing the trajectory of the leading and trailing edges of the black bar. Events with positive polarity are highlighted as light colored points; those with negative polarity are marked as black.

172]. In [125] a pose estimation algorithm based on line tracking is applied to a quadrotor, enabling it to perform aggressive maneuvers. Some studies demonstrate the ability to simultaneously reconstruct intensity maps and relative pose [97] and, more recently, three-dimensional structure [98]. Others aim at combining the benefits of event-based and frame-based vision using the DAVIS. For example, the method presented in [104] uses frames to identify visual features and events to track their position in high-speed motion, in order to perform visual odometry. Early work on optical flow estimation using the DVS was shown in [38].

4.2.2. EVENT-BASED OPTICAL FLOW ESTIMATION

Since the introduction of the DVS and subsequently developed sensors, several different approaches to event-based optical flow estimation have been developed. Most of these techniques operate on each newly detected event and its spatiotemporal neighborhood, providing sparse optical flow estimates. However, in most cases, the algorithms do not distinguish between corner points and other visual features. Thus, they primarily estimate normal flow. In the following, a brief review

Figure 4.3: Picture of the event-based camera employed in this work, the DVS128.

of recent approaches is presented.

An adaptation of the frame-based Lucas-Kanade tracker is introduced in [17]. Similar to the original algorithm, it solves the optical flow constraint by including the local neighborhood of a pixel. Since absolute measurements of I are not available, the authors replaced the intensity I by the sum of event polarities at a pixel location, obtained over a fixed time window. The reconstructed 'relative intensity' is used to numerically estimate I_x, I_y, and I_t. However, the number of events is generally too low for this approach to provide accurate gradient estimates, in particular for the temporal gradient I_t.

In [16] an algorithm is presented that operates on the spatiotemporal representation of events as a point cloud (as shown in Fig. 4.2). When representing a sequence of events by three-dimensional points of (x, y, t), they form surface-like structures. The gradient of such a surface relates to the motion of the object that triggered the events. By computing a local tangent plane to an event and its neighbor events, normal flow for that event is estimated. A follow-up study employs this algorithm for detecting and tracking corners from neighboring normal flow vectors, hence obtaining fully observable optical flow [32]. However, real-time results are not yet demonstrated with a non-parallelized implementation.

In [11], a technique is introduced that estimates optical flow on object contours, based on both events and absolute intensity measurements. Input events are used to locate motion boundaries on contours. Along each boundary, motion is estimated using the width of the contour, which is computed from the local event distribution and the absolute intensity. The latter can be reconstructed from events, but having separate intensity measurements (e.g. from a DAVIS or ATIS sensor) simplifies the

process. [36] uses no spatio-temporal fitting but simply estimates the flow of events by looking at neighboring pixels and the time difference between these events.

A bio-inspired approach is proposed in [23]. In this approach, optical flow is estimated using direction- and speed-selective filters based on the first stages of visual processing in humans. A bank of spatiotemporal filters is employed, each of which is maximally selective for a certain direction and speed of optical flow. For each new event, the neighboring event cloud is convolved with the filters to obtain a confidence measure for each filter. Optical flow for that event is then obtained from the sum of the confidence measures weighted by direction.

More complex event-based algorithms have also been developed, which have not demonstrated real-time performance, but show promising results. In [12] a phase-based optical flow method is discussed, which is developed for high-frequency textures. The algorithm is compared to other event-based methods [11, 16, 17], indeed showing significant accuracy improvements. Also, an approach was presented for simultaneous estimation of dense optical flow and absolute intensity [10]. This is the only available approach aimed towards dense optical flow estimation. Visual results of this method are encouraging, yet a quantitative evaluation has not been performed.

Recently, several datasets for event-based visual navigation have been published. The set in [13] provides both frame and event measurements from a DAVIS sensor accompanied by odometry measurements. This facilitates comparison between frame-based and event-based techniques for optical flow estimation or visual odometry. However, to the best of the authors' knowledge, an actual comparison of existing techniques has not yet been published for this set. In this respect, the work in [150] is more relevant for this work, as it features both an event-based dataset and a comparison of various optical flow algorithms. These are variants of the techniques in [17] and [16], as well as a basic direction selective algorithm.

We select the local plane fitting algorithm in [16] as the basis of the approach in our work. It has shown the most promising results in [150] and has recently been incorporated into follow-up experiments [31, 32]. In addition, its implementations yielded real-time operation for high event measurement rates.

4.3. REALTIME EVENT-BASED OPTIC FLOW ESTIMATION

This section describes our optical flow estimation approach. Since it is based on the work in [16], this baseline approach is explained first in Section 4.3.1. Then, the proposed modifications for achieving higher efficiency (Section 4.3.2) and timestamp-based selection (Section 4.3.3) are discussed. In Section 4.3.4 the result of our improvements is evaluated in comparison to the baseline algorithm. This is done as it is unlikely that a single edge will consist of both positive and negative contrast points.

4.3.1. THE BASELINE PLANE FITTING ALGORITHM

The main working principle of the baseline algorithm is based on the space-time representation of events as a point cloud. In the following, an event is denoted as a space-time point according to $\mathbf{e}_n = (x, y, t)$, where x and y represent the undistorted pixel locations. Note that the polarity P is not considered; we group positive and negative polarity events and process them separately.

Let $\Sigma_e(x, y) = t$ be a mapping describing the surface along which events are positioned. The shape of Σ_e is a result of the feature geometry and, in particular, its motion. In the case of a locally linear feature (such as an edge) and constant motion, this surface reduces to a plane. This is clearly visible in the example scene in Fig. 4.2. With these assumptions, Σ_e can be approximated by a tangent plane within a limited range of x, y, and t.

For each newly detected event $\mathbf{e}_n$, a plane $\mathbf{\Pi}$ is computed that fits best to all neighboring events $\mathbf{e}_i$ for which $x_i \in \left[x_n - \frac{1}{2}\Delta x,\, x_n + \frac{1}{2}\Delta x\right]$, $y_i \in \left[y_n - \frac{1}{2}\Delta y,\, y_n + \frac{1}{2}\Delta y\right]$, and $t_i \in [t_n - \Delta t,\, t_n]$, where $\Delta x, \Delta y, \Delta t$ indicate spatial and temporal windows. The spatial windows are generally small and are both set to 5 pixels. The temporal window setting has a large influence on the detectable speed and interference of multiple features, which is discussed further in Section 4.3.3.

The plane $\mathbf{\Pi} = [p_x,\, p_y,\, p_t,\, p_0]^T$ is computed through an iterative process of linear least squares regression and outlier rejection. It is represented in homogeneous coordinates, such that the following hold for any event $\mathbf{e}_i$ that intersects with $\mathbf{\Pi}$:

$$p_x x_i + p_y y_i + p_t t_i + p_0 = 0 \tag{4.2}$$

Extending Eq. (4.2) with at least four neighboring events, an overdetermined system of equations is obtained, which is solved through linear least-squares. After an initial fit, the Euclidean distance of each event to the plane is computed. All events for which the distance exceeds a threshold d_{max}, are rejected from this fit. Using the remaining events, a new least-squares plane fit is computed. In [16], this process is repeated until the change in $\mathbf{\Pi}$ is no longer significant. This is the case if the norm of the change in all components in $\mathbf{\Pi}$ is smaller than a second threshold k_d, i.e. $\|\mathbf{\Pi}(i) - \mathbf{\Pi}(i-1)\| < k_d$. In practice, the latter often occurs already after one or two iterations. In this work, the values for d_{max} and k_d specified in [150] are applied, which are 0.01 and 1e-5 respectively.

The final plane is preserved and used to compute the local gradients of Σ_e:

$$\nabla\Sigma_e\,(x, y) = \left[\frac{\partial\Sigma_e}{\partial x},\, \frac{\partial\Sigma_e}{\partial y}\right]^T = \left[-\frac{p_x}{p_t},\, -\frac{p_y}{p_t}\right]^T \tag{4.3}$$

In [16] the gradient components of Σ_e are assumed to be inversely related to the optical flow components (u, v):

$$\nabla\Sigma_e\,(x, y) = \left[\frac{1}{u\,(x, y)},\, \frac{1}{v\,(x, y)}\right]^T \tag{4.4}$$

However, as is also noted in [23, 150], Eq. (4.4) is subject to singularities when computing u and v. If either component of $\nabla\Sigma_e$ tends to zero, the corresponding optical flow component grows to infinity, which is incorrect. For example, consider a horizontally moving vertical line. Along the y-direction, temporal differences between the resulting events are in this case very small. Therefore, $\frac{\partial\Sigma_e}{\partial y}$ is also small, which leads to a high value of the vertical component v, even though the line is moving horizontally.

In recent work, two modifications to the previously discussed methodology have been proposed. First, in [23] an approach is presented that is robust to singularities in u and v, which led to significant accuracy improvements in the comparison in [150]. In this approach, an orthogonality constraint is imposed on the plane's normal vector $[p_x,\, p_y,\, p_t]^t$, the optical flow vector $[u,\, v,\, 1]$ and the orientation $[l_x,\, l_y,\, 0]$ of the edge in homogeneous coordinates. This constraint leads to a new expression of the optical flow components u and v in terms of the plane's normal vector:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\|\nabla\Sigma_e\|^2}\nabla\Sigma_e = -\frac{p_t}{{p_x}^2 + {p_y}^2}\begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{4.5}$$

Second, in the implementation in [150] not all events within the space-time window are considered. For each pixel location, only the most recent event is used for computing optical flow. However, high contrast edges tend to produce multiple events in quick succession at a single pixel. Hence, the most recent event at a pixel occurs slightly later than the first event caused by such an edge, which leads to over-estimation of its speed. It may also lead to optical flow estimates in the opposite direction of the edge motion. To prevent this, a refractory period Δt_R (typically 0.3 s) is applied. Events that occur within Δt_R are neither processed nor preserved to support future events.

The discussed algorithm with the previously proposed modifications forms our baseline algorithm. In the following, methods are proposed to increase its efficiency and range of application.

4.3.2. EFFICIENCY IMPROVEMENTS

In order to enable faster computation and scale the algorithm towards low-end processing hardware, we propose two modifications.

The first modification is to reduce the number of parameters of the local plane. We reduce Eq. (4.2) by introducing the new parameters p_x^*, p_y^*, p_0^*:

$$p_x^* = \frac{p_x}{p_t},\; p_y^* = \frac{p_y}{p_t},\; p_0^* = \frac{p_0}{p_t} \tag{4.6}$$

hence obtaining the nonhomogeneous, three-parameter form of Eq. (4.2):

$$p_x^* x_i + p_y^* y_i + p_0^* = -t_i \tag{4.7}$$

A second reduction is performed by assuming that the new event $\mathbf{e}_n$ intersects with the plane, which enables the definition of relative coordinates for neighbor events as follows. Given $\mathbf{e}_n$ and a previously identified neighbor event $\mathbf{e}_i$, the relative coordinates of the neighbor event are defined as $\delta x_i = x_i - x_n,\ \delta y_i = y_i - y_n,\ \delta t_i = t_i - t_n$. Substituting these coordinates into Eq. (4.7) and rearranging terms, the following relation is obtained:

$$p_x^* \delta x_i + p_y^* \delta y_i + \delta t_i = -p_x^* x_n - p_y^* y_n - p_0^* - t_n \tag{4.8}$$

By substituting Eq. (4.7) (for which we set $i = n$ to enforce that e_n intersects with the plane) into Eq. (4.8), the right-hand side of the latter equation reduces to zero. Thus, the final plane $\mathbf{\Pi}^* = \left[p_x^*,\ p_y^*\right]$ is described by two parameters, the slopes:

$$p_x^* \delta x_i + p_y^* \delta y_i = -\delta t_i \tag{4.9}$$

This reduced approach requires significantly less computational effort than the baseline. While solving a homogeneous least squares system is generally performed using a Singular Value Decomposition (SVD), more efficient solvers such as the commonly used QR-decomposition are applicable to nonhomogeneous problems. With a system of M events and N parameters, the computational complexity of the SVD scales with $O(MN^2+N^3)$. In comparison, the complexity of the QR decomposition scales with $O(MN^2 - N^3/3)$ [78]. Hence, a four-parameter SVD solution has a cost that is approximately proportional to $16M + 64$, which compares to $4M - 8/3$ for a two-parameter QR-decomposition. Note that this is only a rough indication of the true complexity, but it suffices for illustrating the efficiency gain of the parameter reduction. Only a slight reduction in accuracy is introduced with this simplification.

The second modification consists of capping the rate at which optical flow vectors are identified, denoted as the output rate ρ_F. Depending on the computational resources available, input events can be processed at a limited rate to maintain real-time performance. In addition, since the approach assumes that for each individual event, motion needs to be estimated, neighboring events produce highly similar optical flow vectors, making the information increasingly redundant with increasing ρ_F. Therefore, capping this value also prevents unnecessary computational load on follow-up processes. To achieve this, we keep track of the timestamp t_f of the event for which the latest optical flow vector was identified. If a new event $\mathbf{e}_n$ occurs, optical flow is only estimated if $t_n - t_f > 1/\rho_{F_{max}}$, where $\rho_{F_{max}}$ denotes the output rate limit. The event is, however, still stored to support future events, taking into account the refractory period. Hence, accuracy of the estimates that are still performed, is unaffected. The resulting effect of the value of $\rho_{F_{max}}$ on computational performance is explored in Section 4.3.4.

4.3.3. TIMESTAMP-BASED CLUSTERING OF RECENT EVENTS

The baseline algorithm incorporates a fixed setting for the time window Δt to collect recent events. There are two main drawbacks of using a fixed time window, which

are illustrated in Fig. 4.4 for two simple one-dimensional cases. First, Δt defines the lower limit for the magnitude of observable optical flow. For slower motion, the time difference between neighboring events increases. If this difference is too large, all neighboring events fall outside the time window (as illustrated in Fig. 4.4a), such that the motion cannot be observed. Second, a larger time window can result in the inclusion of unrelated events. For example, in Fig. 4.4b events are shown which clearly belong to separate features. Still, a part of the outdated features falls within the time window, which leads to an inaccurate fit. In some cases outlier rejection may prevent this, but with tightly packed features this may still cause a failed estimate. A fixed time window therefore imposes a fundamental trade-off between minimal observable speed and feature density. Since MAVs tend to move at a wide range of velocities, from hovering to fast maneuvers, the capability of observing both fast and slow motion is desirable. This issue has also been discussed in [124].

To accomplish this, we propose a very simple clustering method based on the time order of events, which is illustrated in Fig. 4.4c for a one-dimensional motion case. First, the minimum number of most recent events $\mathbf{e}_i$ for observing velocity is found. In the one-dimensional case in Fig. 4.4c, only one point is necessary for constructing a line; for two-dimensional image motion, two linearly independent points $(\delta x_i, \delta y_i, \delta t_i)$ are required in order to construct a plane. From the point with the largest δt_i, the relative timestamp defines a maximum time increment Δt_S between the timestamps of consecutive events. To provide a margin for noise, δt_i is scaled with a factor k_S (which has a value of 8 in our experiments), such that $\Delta t_S = -\delta t_i k_S$ (since δt_i should be negative). Second, we iterate through the remaining recent events, ordered by decreasing value of δt_i. If the time difference between two consecutive events $\mathbf{e}_i$ and $\mathbf{e}_{i-1}$ exceeds Δt_S, $\mathbf{e}_{i-1}$ and all events that occurred before it are assumed to belong to different features, and are rejected.

Note that this approach does not take spatial location into account, except for finding the first linearly independent events. Therefore, a variation on the baseline process of outlier rejection is still applied, which is independent of time-scale. Instead of rejection based on a distance threshold, the overall fit quality is assessed through the Normalized Root Mean Square Error (*NMRSE*), defined here as follows:

$$NRMSE = \frac{n}{\sum_{i=1}^{n} \delta t_i} \sqrt{\frac{\sum_{i=1}^{n} \left(\delta t_i - p_x^* \delta x_i - p_y^* \delta y_i\right)^2}{n}} \tag{4.10}$$

Then, while $NRMSE > NRMSE_{max}$, only the event having the maximum distance to $\mathbf{\Pi}^*$ is rejected. This is repeated until a maximum number of n_R events are rejected. If n_R is exceeded, the estimated plane is rejected and no optical flow is computed. Suitable values for attaining a high number of successful estimates, without sacrificing significant quality, are empirically set at $NRMSE_{max} = 0.3$ and $n_R = 4$.

Still, incorrect optical flow estimates may be detected, either due to noise in the event stream or due to undesired inclusion of outlier events. Depending on the

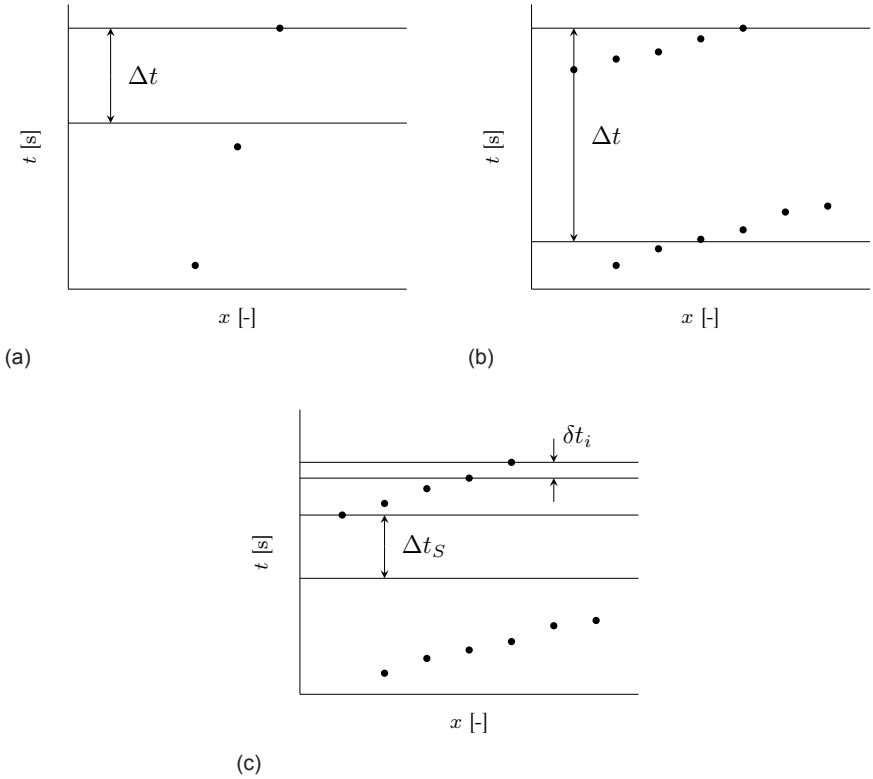


Figure 4.4: Examples with one-dimensional (x-t) event structures representing motion using time windows to estimate motion. In (a), the time window is too small to perceive the slow motion that triggers the events. (b) shows events from two sequential fast-moving features enter the same time window. (c) illustrates the proposed clustering approach, in which the time difference between the current event and the next most recent event δt_i defines the maximum time difference $\Delta t_S = k_S \delta t_i$, older events are ignored.

application, certain optical flow magnitudes can be deemed unrealistic in advance. Optical flow estimates are therefore rejected if their magnitude exceeds a threshold V_{max}, which is set to 1000 pixels/s. In addition, a minimum number of events n_{min} must be found through the clustering mechanism in order to have sufficient support for a reliable fit. This number is set to 10 events. Further, a maximal time window of $\Delta t = 1.5$ s is maintained such that unnecessary event checking is prevented. Note, however, that this time window can now be much larger than in the baseline approach. All the parameters used in this chapter are summarized in Table 4.3 and were determined through genetic optimization and represent a parameter set which performed best on a training dataset of real DVS landing data described in the following section. Note that all results shown below are based on a validation dataset not used in training.

4.3.4. EVALUATION

To evaluate optical flow estimation performance, several datasets were recorded in which the DVS was moved by hand, facing towards a ground surface covered with a textured mat. The recordings are performed indoors, using an Optitrack motion tracking system to measure ground truth position and orientation of the DVS. This is performed for each event for which optical flow is identified, hence providing the means for quantitative accuracy evaluation. Although currently datasets are already available [13, 150], the recorded set specifically represents motion above a flat surface in indoor lighting conditions, i.e. the environment in which flight tests are performed in Section 4.5. The DVS bias settings used to make the sets is summarized in Table 4.2.

Images of the recorded ground surfaces are shown in Fig. 4.5. The checkerboard in Fig. 4.5a provides high contrast and clear edges and is hence relatively simple for optical flow estimation. The roadmap texture in Fig. 4.5b has largely unstructured features and lower contrast. It is used to show that our approach extends to more general situations as well. Eight short sequences were selected to evaluate the performance of the proposed method. Each sequence is approximately 1.0 s long and consists of event and pose measurements in which one primary motion type is present. Five sets are selected for the checkerboard; one for vertical translational image motion ($\vartheta_y \approx 1.0$), one for rotational motion ($r \approx -1.3$) rad/s, and three sets with diverging motion of different speeds ($\vartheta_z \approx \{0.2, 0.5, 2.0\}$). For the roadmap texture, three sets with diverging motion were selected as well ($\vartheta_z \approx \{0.1, 0.5, 1.0\}$).

In the following analysis, the cap on ρ_F is not applied (i.e. $\rho_{F_{max}} = \infty$), except for the assessment of computational complexity in Section 4.3.4.

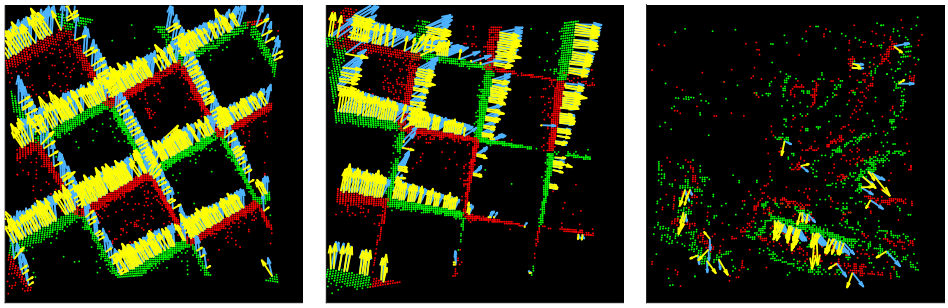
(a) Checkerboard (b) Roadmap

Figure 4.5: Ground surface textures used during the experiments.

QUALITATIVE EVALUATION

Fig. 4.6 shows optical flow vectors (yellow arrows) estimated using the improved algorithm during three of the selected sequences, along with ground truth flow vectors (blue arrows). Note that, for clarity, the time window for visualizing events is

larger than for visualizing optical flow, which is why for some event locations, it appears that no optical flow estimates are found. Accurate normal flow estimates are visible for the checkerboard datasets. In Fig. 4.6a optical flow is generally constant along the checkerboard edges, matching well to the normal component of the ground truth vectors. The rotating checkerboard sequence (Fig. 4.6b) also provides accurate optical flow estimates. Clear variation of the normal flow magnitude along the lines is seen.



(a) Translating checkerboard (b) Rotating checkerboard (c) Diverging roadmap ($\vartheta_z = 1.0$)

Figure 4.6: Optical flow estimated in several sequences, shown as yellow arrows. The accompanying blue arrows show the ground truth optical flow. Events are shown as green dots (positive polarity) or red dots (negative polarity). The time window for displaying optical flow in each sequence is 10 ms. To better visualize the event input, a larger window of 50 ms is applied for the events.

Last, in Fig. 4.6c it is clearly visible that the roadmap texture is more challenging. Event structures are less coherent and the visible features are more noisy. Optical flow vectors are sparsely present, yet the available estimates are sufficient for observing the global motion. At some location with noisy features the motion tends to be underestimated, but the majority of the estimates is very similar to the normal direction of the ground truth motion.

QUANTITATIVE EVALUATION

For quantitative evaluation, a comparison was made of the proposed optical flow algorithm and the baseline algorithm by [16] (as detailed in Section 4.3.1). In this comparison the fixed time window Δt for the original approach is set to 100 ms, and the rejection distance d_{max} is set to 0.001 to obtain a similar event density in both algorithms. For a consistent comparison, the baseline algorithm incorporate the same refractory period Δt_R, maximum speed limit V_{max}, and minimum number of events n_{min} as the proposed algorithm.

For benchmarking optical flow accuracy, several error metrics have been introduced such as the endpoint error and angular error [8]. These metrics have been incorporated into recent event-based optical flow benchmarks as well [150]. However,

they are defined for optical flow that is fully determinable and is not subject to the aperture problem. The algorithms in this section both estimate normal flow. In [11] a version of the endpoint error is applied that indicates the magnitude error of the normal flow with respect to the *projection* of ground truth optical flow along the normal flow vector. This metric is employed here as well. For each optical flow vector, we compute the Normalized Projection Endpoint Error (NPEE), which is defined as follows:

$$\text{NPEE} = \frac{\left| \|\mathbf{V}\| - \frac{\mathbf{V}}{\|\mathbf{V}\|} \cdot \mathbf{V}_{GT} \right|}{|\mathbf{V}_{GT}|} \tag{4.11}$$

where $\mathbf{V} = [u,\, v]^T$ is the normal flow estimate and $\mathbf{V}_{GT}$ denotes the ground truth optical flow vector.

A comparison of the resulting mean absolute NPEE values, and their standard deviations, is presented in Section 4.3.4. The optical flow density is also shown (abbreviated here as η) which indicates the percentage of events for which an optical flow estimate was found. A high value of η indicates that more motion information can be obtained with a given event input.

Table 4.1: Mean Normalized Projection Endpoint Error {25%, 50%, 75% percentile} and density results of the baseline plane fitting algorithm, and the new algorithm proposed in this work. Values highlighted are the lowest NPEE or highest density result of both algorithms.

	Baseline [16]		This work	
	NPEE [pix/s]	η [%]	NPEE [pix/s]	η [%]
Checker, $\vartheta_y = 1.0$	**0.331** {0.058, 0.128, 0.295}	37.2	0.465 {0.057, 0.118, 0.205}	**42.1**
Checker, $r = -1.3$	**0.622** {0.097, 0.219, 0.679}	32.3	1.503 {0.066, 0.138, 0.249}	**36.4**
Checker, $\vartheta_z = 0.2$	**0.568** {0.176, 0.350, 0.588}	19.8	0.572 {0.140, 0.290, 0.529}	**21.6**
Checker, $\vartheta_z = 0.5$	0.519 {0.102, 0.290, 0.752}	24.7	**0.480** {0.086, 0.202, 0.449}	**29.2**
Checker, $\vartheta_z = 2.0$	0.766 {0.305, 0.928, 1.011}	**42.8**	**0.504** {0.075, 0.159, 0.273}	35.6
Roadmap, $\vartheta_z = 0.1$	**0.671** {0.271, 0.585, 0.861}	7.5	0.737 {0.191, 0.472, 0.819}	**8.0**
Roadmap, $\vartheta_z = 0.5$	0.720 {0.342, 0.719, 0.987}	7.9	**0.618** {0.136, 0.293, 0.557}	**12.0**
Roadmap, $\vartheta_z = 1.0$	0.743 {0.444, 0.822, 1.002}	10.3	**0.605** {0.148, 0.310, 0.559}	**11.9**

Overall, the results are very similar. Both algorithms reach good scores on the checkerboard sets with translation, rotation, and medium divergence ($\vartheta_z = 0.5$). However, some differences are observable. The proposed algorithm tends to reach a higher optical flow density in the slow divergence ($\vartheta_z = 0.2$) checkerboard scene and in all roadmap scenes, since the baseline algorithm fails to perceive slowly moving features. Still, this does not degrade the estimate accuracy with respect to the baseline algorithm, or only to a limited extent. Note also that in both fast diverging sequences (Checkerboard, $\vartheta_z = 2.0$, and Roadmap, $\vartheta_z = 1.0$) a lower mean absolute NPEE is achieved with our approach.

Looking at the percentile distribution, we can see that the proposed algorithm consistently has a lower error for each percentile bracket. As the mean value is typically

worse, this would suggest that the proposed algorithm has lower absolute error but with some significant outliers. We performed some genetic optimization trials to investigate the effect of our parameter selection on the mean performance and the density of the generated flow vectors. We found that generally there is a strong relation between these two parameters that must be balanced, to increase the mean performance you will reduce the number of generated vectors. This suggests that some more advanced outlier detection may be required in the future to further improve the performance of our estimator.

COMPUTATIONAL PERFORMANCE EVALUATION

An assessment of computational complexity is made using two datasets, one for both texture types. Both sets have a duration of 12 s and contain approximately 40k events/s. This enables quantifying the potential of $\rho_{F_{max}}$ to regulate processing time, as well as the effect of texture. The algorithm is implemented in C and interfaced with MATLAB through MEX, running single-threaded on a 64bit desktop computer running Ubuntu 16.04 with an Intel Xeon E5-1620 octacore CPU. Each dataset and setting of $\rho_{F_{max}}$ is processed ten times for consistent results.

The resulting computation time per event for several settings of $\rho_{F_{max}}$ is shown in Fig. 4.7, in comparison with the maximal computation time with no control of $\rho_{F_{max}}$. For both textures it is clearly possible to regulate processing time by $\rho_{F_{max}}$. A lower limit appears to be present, which is due to the remaining overhead related to event timestamp copying. Interestingly, there is a clear influence of texture. This difference is due to higher contrast edges in the checkerboard texture, at which several successive events are generated per pixel. Therefore, the refractory period filter rejects these duplicate events before optical flow computation.

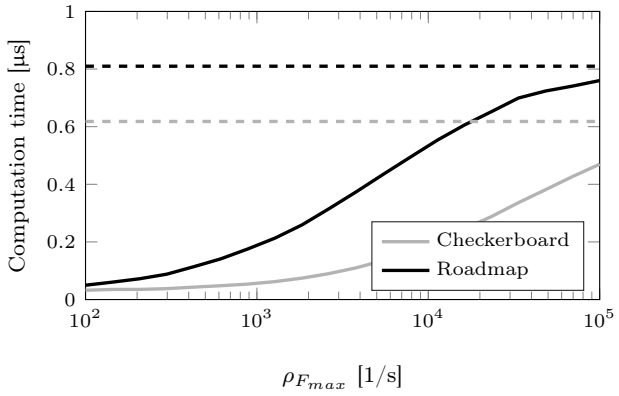


Figure 4.7: Processing time per event for checkerboard and roadmap datasets, for different settings of $\rho_{F_{max}}$. The dashed lines indicate the computation times when no limit on $\rho_{F_{max}}$.

Without the refractory period, computational effort is similar for both textures. In this case, the maximal computation time per event, i.e. without control of ρ_F, is

0.81 μs. This is equivalent to processing 1.23M events/s in real-time, which is easily sufficient for processing realistic scenes on the test machine. Event sequences recorded for post-processing contained event peaks below 150k events/s. If hardware capabilities are more restricted (e.g. in on-board applications for MAVs), control of ρ_F can be applied to scale the computational complexity of the algorithm down if necessary.

4.4. ESTIMATION OF VISUAL OBSERVABLES FROM EVENT-BASED OPTICAL FLOW

This section describes our approach for estimating visual observables from event-based optical flow. While optic flow estimation is performed asynchronously, most existing control systems still operate on a periodic basis. Similarly, the proposed algorithm aims to update the estimates of visual observables at a fixed rate. For each periodic iteration, all newly detected optical flow vectors between the current iteration and the previous one form a planar optical flow field, of which the parameters are estimated.

The algorithm is based on two components. First, newly detected optical flow vectors are grouped per direction and incorporated into a weighted least-squares estimator for the visual observables, as discussed in Section 4.4.1. To enable preservation of flow field information over subsequent periodic iterations, a recursive update technique is introduced in Section 4.4.2. In addition, a confidence value is computed and applied to filter the visual observable estimates, as is described in Section 4.4.3. The estimator is evaluated in combination with our event-based optical flow algorithm in Section 4.4.4.

4.4.1. DIRECTIONAL FLOW FIELD PARAMETER ESTIMATION

The presented approach is based on techniques introduced in [44] and used in [1, 84], in which fully defined optical flow estimates are available. Since our optical flow algorithm provides normal flow output, a regular optical flow field representation leads to inaccurate parameter estimates. However, in planar flow fields, normal flow may already provide sufficient information for computing the visual observables. Along the direction of the flow vector, normal flow does provide accurate information.

An example diverging flow field with both optical flow and normal flow is sketched in Fig. 4.8. Note that the normal flow in some cases deviates significantly from the optical flow equivalent, which leads to significant errors when computing the flow field parameters. However, when grouped by direction (which is done in Fig. 4.8 through the arrow colors and their accompanying numbering), the normal flow vectors indeed show the original pattern of divergence. This idea is central to the proposed directional flow fields approach.

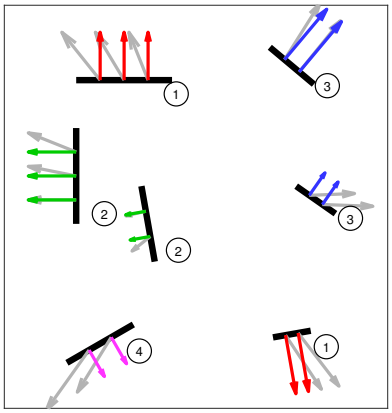


Figure 4.8: Example of a diverging flow field resulting from several randomly oriented moving edges. The gray vectors indicate the true flow field, while the colored vectors show the normal flow along the edge orientation. Vectors with a similar direction can be sorted into one of 6 directions where the inset numbering, as well as the colors, highlight the grouping for this example.

When considering the flow vectors grouped by similar direction, at least two normal flow vectors from separate non-parallel edges are required to observe flow field divergence in that direction. The further apart these edges are the more reliably the divergence can be estimated. For example, in Fig. 4.8 the purple group of normal flow vectors does not, by itself, provide sufficient information for perceiving divergence. Also, if the flow vectors are located in close proximity, errors in normal flow magnitude have a larger influence. In Fig. 4.8 the green group is more sensitive to these errors than the red group, since the edges are located closely together. Grouping per direction enables assessment of the reliability of the flow field in each direction, taking the previous issues into account.

A set of m directions $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ is defined, where $\alpha_1 = 0$ and $\alpha_i - \alpha_{i-1} = \pi/m$. In this work, $m = 6$ directions are used. For each newly available flow vector, we first determine the closest match of α_i to the flow direction α_f. Each direction α_i accommodates both flow in similar and opposite direction, i.e. when $-\pi < \alpha_f < 0$, a match is computed for $\alpha_f + \pi$.

Along the selected direction α_i, the projected normal flow position S and magnitude V are computed, hence obtaining a one-dimensional representation of the flow along α_i:

$$\begin{bmatrix} S \\ V \end{bmatrix} = \begin{bmatrix} \hat{x} & \hat{y} \\ \hat{u} & \hat{v} \end{bmatrix} \begin{bmatrix} \cos\alpha_i \\ \sin\alpha_i \end{bmatrix} \tag{4.12}$$

Subsequently, it is corrected for rotational motion by subtracting the normal com-

ponent of the rotational flow:

$$\begin{aligned} V_T = V &- \cos\alpha_i \left(p - \hat{y}r - q\hat{x}\hat{y} + p\hat{x}^2\right) \\ &+ \sin\alpha_i \left(q - \hat{x}r - p\hat{x}\hat{y} + q\hat{y}^2\right) \end{aligned} \tag{4.13}$$

For each direction, a one-dimensional flow field is maintained. From Eq. (4.12), the flow field in a single direction is expressed as:

$$V_T = -\vartheta_x \cos\alpha_i - \vartheta_y \sin\alpha_i + \vartheta_z S \tag{4.14}$$

To solve Eq. (4.14) for the visual observables, a weighted least-squares solution is computed using the flow vectors from all directions. Let $\mathrm{c}_\alpha = \cos\alpha$ and $\mathrm{s}_\alpha = \sin\alpha$. The overdetermined system to be solved is composed as follows:

$$\begin{bmatrix} -\mathrm{c}_{\alpha_1} & -\mathrm{s}_{\alpha_1} & S_{1,1} \\ \vdots & \vdots & \vdots \\ -\mathrm{c}_{\alpha_1} & -\mathrm{s}_{\alpha_1} & S_{1,n_1} \\ -\mathrm{c}_{\alpha_2} & -\mathrm{s}_{\alpha_2} & S_{2,1} \\ \vdots & \vdots & \vdots \\ -\mathrm{c}_{\alpha_2} & -\mathrm{s}_{\alpha_2} & S_{1,n_2} \\ \vdots & \vdots & \vdots \\ -\mathrm{c}_{\alpha_m} & -\mathrm{s}_{\alpha_m} & S_{m,n_m} \end{bmatrix} \begin{bmatrix} \vartheta_x \\ \vartheta_y \\ \vartheta_z \end{bmatrix} \approx \begin{bmatrix} V_{1,1} \\ \vdots \\ V_{1,n_1} \\ V_{2,1} \\ \vdots \\ V_{2,n_2} \\ \vdots \\ V_{m,n_m} \end{bmatrix} \tag{4.15}$$

which has the form $\mathbf{A\Theta} \approx \mathbf{y}$. The weighted least-squares solution is then obtained from the normal equations:

$$\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{\Theta} = \mathbf{A}^T\mathbf{W}\mathbf{y} \tag{4.16}$$

in which $\mathbf{W}$ a diagonal matrix composed of the weights per direction:

$$\mathbf{W} = \mathrm{diag}\Big(W_1, \cdots, W_1, W_2, \cdots, W_2, \cdots, W_m\Big) \tag{4.17}$$

The weight W_i is used to represent the reliability of normal flow along a direction i based on the spread of S_i along that direction. Its value is determined by the variance $\mathrm{Var}\{S_i\}$. We let W_i scale linearly with $\mathrm{Var}\{S_i\}$, up to a maximum of $\mathrm{Var}\{S\}_{min}$:

$$W_i = \begin{cases} \frac{\mathrm{Var}\{S_i\}}{\mathrm{Var}\{S\}_{min}} & \mathrm{Var}\{S_i\} < \mathrm{Var}\{S\}_{min} \\ 1 & \mathrm{Var}\{S_i\} \geq \mathrm{Var}\{S\}_{min} \end{cases} \tag{4.18}$$

The minimum variance $\mathrm{Var}\{S\}_{min}$ is set to 600 pixels^2.

Note also that, through the formulation of Eq. (4.15), directions with more normal flow estimates have a larger influence on $\mathbf{\Theta}$. Hence, directions for which more information is available, contribute more to the solution.

4.4.2. RECURSIVE UPDATING OF THE FLOW FIELD

The solution to Eq. (4.16) for $\boldsymbol{\Theta}$ provides the estimate for the visual observables. However, depending on the sampling rate of the estimator, it is possible that, during a single periodic iteration, too few normal flow estimates are available for an accurate fit. This leads to noise peaks in the measurement of $\boldsymbol{\Theta}$, especially during low speed motion. To limit this effect, the matrices $\mathbf{A}$ and $\mathbf{y}$ are not completely renewed at each iteration. Instead, rows from previous iterations are retained and assigned an exponentially decreasing weight, similar to an exponential moving average filter.

For an efficient implementation of the former, $\mathbf{A}$ and $\mathbf{y}$ are not explicitly composed as shown in Eq. (4.15). Instead, our approach operates on the normal equations in Eq. (4.16). For each direction independently, we recursively update parts of the matrices $\mathbf{B} = \mathbf{A}^T\mathbf{W}\mathbf{A}$ and $\mathbf{C} = \mathbf{A}^T\mathbf{W}\mathbf{y}$. These matrices are composed by the following elements:

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{21} & b_{22} & b_{32} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}, \; \mathbf{C} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \tag{4.19}$$

From Eq. (4.15), it can be shown that the elements of $\mathbf{B}$ are expressed as:

$$\begin{aligned} b_{11} &= \sum_{i=1}^{m} W_i n_i (\mathbf{c}_{\alpha_i})^2, \quad b_{21} = \sum_{i=1}^{m} W_i n_i \mathbf{c}_{\alpha_i} \mathbf{s}_{\alpha_i} \\ b_{22} &= \sum_{i=1}^{m} W_i n_i (\mathbf{s}_{\alpha_i})^2, \quad b_{31} = \sum_{i=1}^{m} W_i \mathbf{c}_{\alpha_i} \sum_{j=1}^{n_i} S_{i,j} \\ b_{33} &= \sum_{i=1}^{m} W_i \sum_{j=1}^{n_i} S_{i,j}^2, \quad b_{32} = \sum_{i=1}^{m} W_i \mathbf{s}_{\alpha_i} \sum_{j=1}^{n_i} S_{i,j} \end{aligned} \tag{4.20}$$

and those of $\mathbf{C}$ are expressed as:

$$\begin{aligned} c_1 &= \sum_{i=1}^{m} W_i \mathbf{c}_{\alpha_i} \sum_{j=1}^{n_i} V_{i,j} \\ c_2 &= \sum_{i=1}^{m} W_i \mathbf{s}_{\alpha_i} \sum_{j=1}^{n_i} V_{i,j} \\ c_3 &= \sum_{i=1}^{m} W_i \sum_{j=1}^{n_i} S_{i,j} V_{i,j} \end{aligned} \tag{4.21}$$

We introduce a shorthand notation $\Sigma_S^i = \sum_{j=1}^{n_i} S_{i,j}$ to represent the sums, cross-product sums, and sums of squares of S and V for direction i. The unweighted contribution of the associated flow vectors is then contained in n_i and the sums Σ_S^i, $\Sigma_{S^2}^i$, Σ_V^i, and Σ_{SV}^i. These values are further referred to as the *flow field statistics*. Hence, a newly detected flow vector is included in the flow field estimate by incrementing these quantities according to the values S and V of the new vector.

What makes this decomposition interesting, is that the flow field statistics form a compact summary of the flow field, independent of the actual number of flow vectors. Thus, flow field information from a previous iteration can be efficiently included in subsequent ones, without increasing the size of the system in Eq. (4.15). Now, at the start of each iteration, it is possible to include information from the flow field of the previous iteration, simply by preserving a fraction F of the previous flow field statistics. Hence, the estimator accuracy is less dependent on the sampling rate of the algorithm.

The preservation process is illustrated using the statistic Σ_V^i. At the start of iteration k, Σ_V^i is initialized as $\Sigma_V^i(k) = F\Sigma_V^i(k-1)$. During iteration k, Σ_V^i is then updated using newly available normal flow vectors that are allocated to direction i. Hence, the complete update for Σ_V^i is performed as follows:

$$\Sigma_V^i(k) = F\Sigma_V^i(k-1) + \sum_{j=1}^{n_i} V_{i,j} \tag{4.22}$$

The value of F is computed as:

$$F = 1 - \frac{t(k) - t(k-1)}{k_f} \tag{4.23}$$

where the time constant k_f is assigned a value of 0.02 s. This step is similar for all statistics. When all newly available vectors are categorized and processed, the flow field is recomputed using Eq. (4.16).

4.4.3. CONFIDENCE ESTIMATION AND FILTERING

In visual sensing, the reliability of motion estimates varies greatly depending on the environment. Factors such as visible texture and scene illumination have an effect on the estimate. With event-based sensing, motion in the scene is another key factor.

Therefore, a confidence value is computed based on several characteristics of the flow field, in order to quantify the reliability of the estimate. This confidence value is defined as a product of three individual confidence metrics based on the following statistical quantities:

- The flow estimation rate ρ_F.
- The maximal variance $\mathrm{Var}\{S\}$ of all flow directions.
- The coefficient of determination R^2 of the solution to Eq. (4.16), applied here as a nondimensional measure of the fit quality.

R^2 is generally computed through the following [173]:

$$R^2 = 1 - \frac{RSS}{TSS} \tag{4.24}$$

In this work, the Residual Sum of Squares (RSS) and Total Sum of Squares (TSS) are computed in weighted form as follows:

$$\begin{aligned} RSS = &\quad \mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{\Theta}^T\mathbf{A}^T\mathbf{W}\mathbf{y} \\ TSS = &\quad \mathbf{y}^T\mathbf{W}\mathbf{y} - \frac{\left(\sum_{i=1}^{m} W\Sigma_V^i\right)^2}{\sum_{i=1}^{m} Wn_i} \end{aligned} \tag{4.25}$$

For each indicator, a confidence value k is computed ranging from 0 to 1 (higher is better), similar to the variance weight in Eq. (4.18). The individual confidence values are thus dependent on settings for R^2_{min}, $\mathrm{Var}\{S\}_{min}$, and $\rho_{F_{min}}$ (not to be confused with $\rho_{F_{max}}$). The values of R^2_{min} and $\rho_{F_{min}}$ are set to 0.8 and 500 respectively. Note that, since Eq. (4.18) already provides individual confidence values per direction in the form of W, we simply let $k_{\mathrm{Var}\{S\}} = \max(W_i : i = 1, \ldots, m)$.

The total confidence value K is then the product of k_{ρ_F}, $k_{\mathrm{Var}\{S\}}$, and k_{R^2}. Hence, each individual confidence factor needs to be close to 1 in order to obtain a high K. For example, when ρ_F and R^2 are very large, but the flow is very localized (the maximal value for $\mathrm{Var}\{S\}$ is small), the estimate is still not reliable. In this case, it is likely that a single visual feature causes the normal flow, which is insufficient for computing the visual observables.

The confidence K is useful to monitor the estimate quality of the visual observables during flight. In addition, it is the main component of a *confidence filter* for $\mathbf{\Theta}$. This filter is based on a conventional infinite impulse response low-pass filter, in which K is multiplied with the filter's update constant. The final estimate for the visual observables $\hat{\mathbf{\Theta}}$ is determined through the following update equation at iteration k:

$$\hat{\mathbf{\Theta}}(k) = \hat{\mathbf{\Theta}}(k-1) + \Big(\mathbf{\Theta}(k) - \hat{\mathbf{\Theta}}(k-1)\Big)\, K \frac{t(k) - t(k-1)}{k_t} \tag{4.26}$$

where k_t is the time constant of the low-pass filter, which is set to 0.04 s. Lastly, a saturation limit is applied that caps the magnitude of the update of each individual value in $\mathbf{\Theta}$ to $\Delta\vartheta_{max}$ in order to reject significant outliers. The value for $\Delta\vartheta_{max}$ is set to 0.3. A summary of all the settings used in this chapter are summarized in Table 4.3.

4.4.4. RESULTS

To evaluate the accuracy of the presented visual observable estimator, we use the measurements generated for evaluating optical flow performance in Section 4.3.4, which are generated through handheld motion. Optitrack position measurements provide the ground truth estimates for ϑ_x, ϑ_y, and ϑ_z. For each set, normal flow estimates are computed using the C-based implementation discussed in Section 4.3.4.

The flow detection rate cap $\rho_{F_{max}}$ is set to 2500 flow vectors/s and the periodic estimator samples the visual observables at 100 Hz, similar to the on-board implementation in Section 4.5.

In our experiments the main variable of interest is ϑ_z, as it forms the basis for the constant divergence controller. Therefore, this variable is investigated over a wide range of velocities. However, the estimates of ϑ_x and ϑ_y are also interesting to assess, since a more elaborate optical flow based controller may also include the horizontal components for hover stabilization. The latter process does require the MAV to perform rolling and pitching motion, inducing rotational normal flow. Therefore, the effectiveness of derotation is evaluated as well.

For assessment of ϑ_z estimates, vertical oscillating motion was performed above both texture types. The vertical speed of these oscillations was gradually increased, hence covering a wide range of divergence values. This enables a first-order characterization of the estimator behavior.

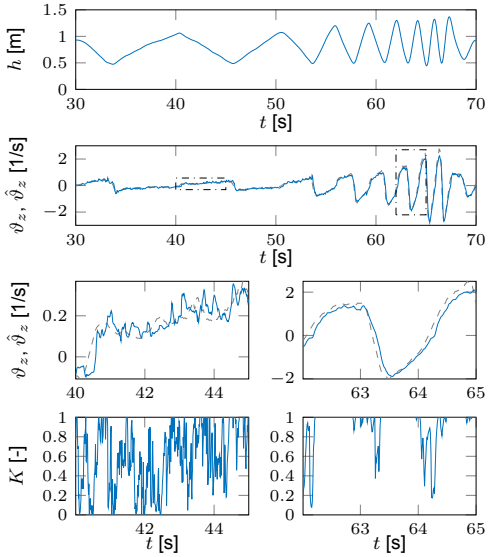
Fig. 4.9 shows the resulting estimates compared to ground truth measurements. Detail sections are shown for low and high divergence motion, for which also the confidence value is shown.

The detail plots show that the estimator is relatively sensitive to local outliers in normal flow at low speeds. In addition, the confidence K is generally low due to lower detection rates of optical flow and low value of R^2. At higher speeds, the errors are relatively smaller. Note that K is also generally higher there. Somewhat lower confidence values are seen for the roadmap texture.

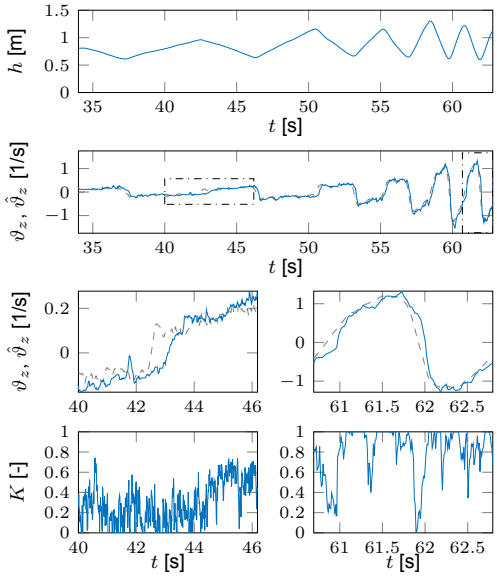
Around sign changes, brief moments are present where the confidence value K is low. The result of this is that, due to the confidence filter, the update of $\hat{\vartheta}_z$ at these points is limited, which leads to a local delay with respect to the ground truth. However, when higher confidence estimates are available, the estimate quickly converges back to the ground truth value.

Based on the estimator results in Fig. 4.9 we can assess how the error varies with the ground truth divergence. Fig. 4.10a shows the variation of the absolute error $\varepsilon_{\vartheta_z} = |\hat{\vartheta}_z - \vartheta_z|$ with the magnitude of ϑ_z. The errors of both the checkerboard set and the roadmap set are combined, since the estimator shows roughly the same error distribution for both cases with the quantile plot almost flat for the divergences tested. Interestingly, the largest absolute errors appear to be present at low divergence. This results from the local delay occurring around zero-crossings in Fig. 4.9 where the confidence value of the filter is low due to the low number of events produced by the slow motion. Note, however, that the error increase with the magnitude of $|\vartheta_z|$ is limited, which enables application of the presented pipeline to a wide range of velocities.

An extensive characterization of two frame-based visual estimators for ϑ_z was performed in [85], which includes an assessment of the error distribution up to $\vartheta_z \approx 1.3$ (Fig. 10 in the chapter). For a first-order comparison, Fig. 4.9 also shows the quadratic error fit obtained for the frame-based 'size divergence' estimator, which

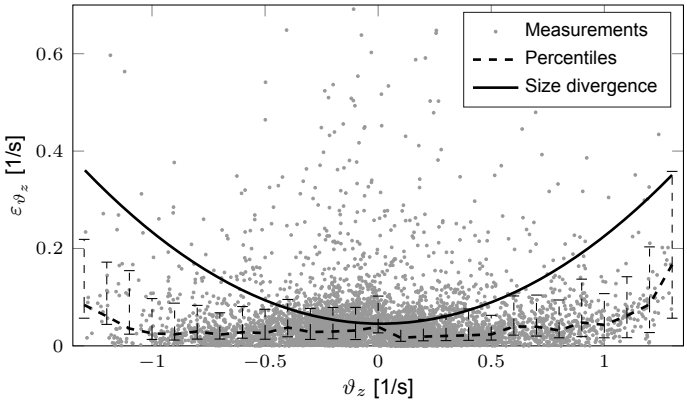


(a) Checkerboard texture

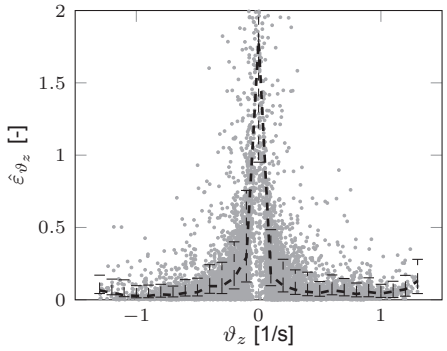


(b) Roadmap texture

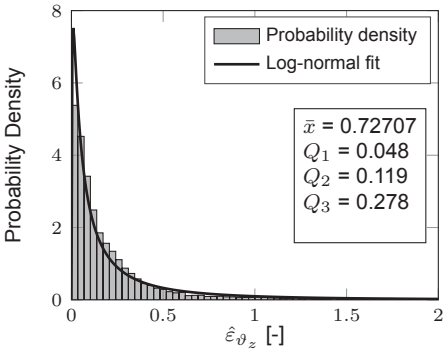
Figure 4.9: From top to bottom: Height measurements and estimates of ϑ_z (solid line) in comparison to ground truth measurements (dashed line), detail sections of ϑ_z estimates at low speed and high speed, as well as the accompanying estimate confidence value K. Measurements are shown for (a) checkerboard and (b) roadmap textures separately.



(a) Absolute error distribution.



(b) Normalized error distribution of ϑ_z.



(c) Absolute error probability density distribution.

Figure 4.10: Error distribution for a set of landing tests performed at different constant vertical speeds above the roadmap texture. (a) Absolute error distribution with error bars. For comparison, the model obtained for the frame-based size divergence estimator [85] is shown as well. (b) Normalized error distribution of ϑ_z. The dashed black line shows the 25, 50 and 75% percentiles of the data. (c) Absolute error probability density distribution with a $Lognormal(-2.18, 1.56)$ fit. The mean ($\bar{x}$) and quantiles are show in the insert.

performed best in [85]. Compared to the presented event-based estimator, the size divergence estimator achieves slightly lower errors in the region of $\vartheta_z < 0.3$. However, for faster motion, the error is lower for our event-based estimator.

Fig. 4.10b shows the distribution or error as a function fo the divergence. Here we can see that the minimum divergence that our estimator produces accurate results is 0.1. Values below this have a large uncertainty. Above 0.1 however, the estimator seemingly improves the faster you go. Additionally, unlike [85], the error seen with our estimator is not well captured with a Gaussian distribution but is fit better by a log-normal error distribution with $Lognormal(-2.18, 1.56)$, as shown in Fig. 4.10c.

4.5. CONSTANT DIVERGENCE LANDING EXPERIMENTS

This section presents experimental results of constant divergence landings with the presented algorithms in the control loop. In Section 4.5.1 the divergence control law is defined, after which the experimental setup is detailed in Section 4.5.2. Results from the experiments are presented and discussed in Section 4.5.3.

4.5.1. DIVERGENCE CONTROLLER

The control law regulates ϑ_z through the vertical thrust T. The controller applies a thrust difference ΔT with respect to a nominal hover thrust T_0, such that $T = T_0 + \Delta T$. A simple proportional control law is applied to ΔT based on ϑ_z, similar to [41]:

$$\Delta T = k_P \left(\vartheta_{z_r} - \vartheta_z \right) \tag{4.27}$$

The nominal hover thrust T_0 counteracts the weight of the test vehicle. Its value is adapted in-flight in the height control loop of the test vehicle's autopilot software. Before the start of each landing, the vehicle first performs automatic hover to obtain a stable estimate for T_0. During the subsequent landing maneuver its value is kept constant.

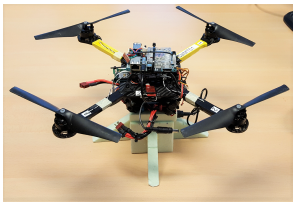
4.5.2. EXPERIMENTAL SETUP

The flying platform used in this work is a customized quadrotor referred to as the MavTec. Its main component is a Lisa/MX board, which features a 168MHz 32bit ARM microprocessor as well as a pressure sensor and 3-axis rate gyros, accelerometers, and magnetometers. The Lisa/MX runs the open-source autopilot software Paparazzi[1], which handles the control of the drone. The DVS is mounted at the bottom of the MavTec facing downwards. Experiments are performed indoors, using an Optitrack motion tracking system to measure ground truth position and attitude.

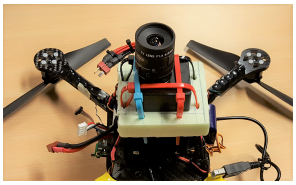
In addition, an Odroid XU4 board is mounted on the quadrotor, which processes the event output of the DVS. It features a Samsung Exynos 5422 octacore CPU (four cores at 2.1 GHz and four at 1.5 GHz). The Odroid receives the events from the DVS through a USB 2.0 connection and processes these through the C-based open-source software cAER [110].

An overview of the experimental setup is shown in Fig. 4.11, including an overview of the on-board processing workflow in Fig. 4.11c. The estimation pipeline is subdivided in two stages. First, raw events are transmitted from the DVS to the Odroid through a USB interface. In cAER, optical flow is computed from the events using an implementation of our optical flow algorithm. Any event for which flow is estimated, is transmitted to the Lisa/MX board through a serial UART interface. This process

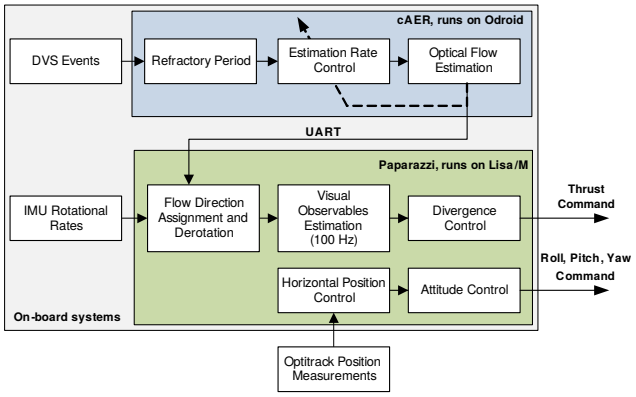
[1] Paparazzi UAV, `http://wiki.paparazziuav.org/`



(a) Top view



(b) Bottom view showing the DVS



(c) Overview of the implementation

Figure 4.11: Overview of the experimental setup, including pictures of the MavTec. In (a) a top view of the vehicle is shown. The DVS is located at the bottom, protected by a foam cover. In (b) the cover is removed to expose the DVS. In (c) an overview of the processing workflow is shown, indicating the distribution of processes over the Odroid and the Lisa/MX processors.

is completely event-based and is performed in a single thread. Separate threads handle event reception and transmission through the USB and UART interfaces.

Second, in Paparazzi, a periodic follow-up processing thread runs at 100 Hz. At each iteration, all newly received optical flow events are collected and corrected for the quadrotor's attitude and rotational motion. When all new events are processed, new estimates of the scaled velocities are computed with accompanying confidence values. A separate thread running at 512 Hz performs divergence control using the new update for ϑ_z, as well as horizontal position control and stabilization.

4.5.3. RESULTS

CONSTANT DIVERGENCE, CONSTANT GAIN LANDING

Constant divergence landing maneuvers were performed for several values of the setpoint ϑ_{z_r}. During the tests, the target ground location was covered with the roadmap textured mat shown in Fig. 4.5b. In an ideal world, a fixed gain proportional controller will result in a smooth landing with the vehicle never actually touching down. In reality however, due to computational latency and actuator delays, the vehicle will be come unstable as it approaches the ground [41]. Notably, the gain which causes instability is directly proportional to the altitude of the vehicle above the ground. As in [41], this feature is used to estimate the altitude of the vehicle above the ground by analyzing the covariance of the thrust command. The gain of the controller is selected such that the vehicle will become unstable about 0.3

m above the ground, which in our case is a gain (k_P) of 0.25. Once a diverging oscillation is identified in the thrust command, a final landing procedure is activated which sets a constant thrust setting allowing the vehicle to perform a safe landing. In our experiments, the final throttle was set to 80% of the hover throttle setting.

Resulting flight profiles are shown in Fig. 4.12 for setpoints $\vartheta_{z_r} = \{0.3, 0.5, 0.7, 1.0\}$. Note that these values are much higher than the setpoints in comparable frame-based experiments [81, 84]. The estimates for ϑ_z are shown in comparison to the ground truth estimate and the corresponding setpoint.

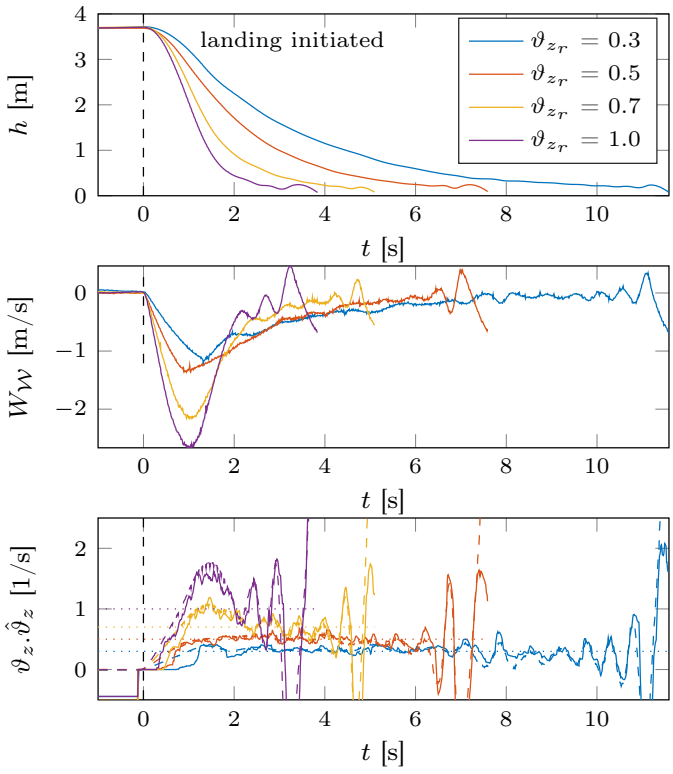


Figure 4.12: Height above ground, vertical speed, and divergence measurements with ground truth during a constant divergence landings performed at four different divergence setpoints. In the bottom graph, the dotted, dashed, and solid lines represent the setpoint, ground truth, and estimate for ϑ_z respectively.

First looking at the accuracy of the ϑ_z estimate we can see that at the start of the decent, there is a delay between the start of the landing and the any change in the ϑ_z estimate. This is particularly evident in the landing with ϑ_{z_r} 0.3. This is mainly due to the minimum resolution of the estimator of around $\vartheta_z = 0.1$, once ϑ_z is higher than 0.1 the estimate is quite accurate.

For landings with a low ϑ_{z_r} we can see that the tracking is quite good, although there

is a relatively long rise time of the controller, there is little overshoot. As expected, as the vehicle gets close to the ground the controller becomes increasingly unstable until the oscillation is identified by the controller and the final landing procedure is engaged. For the landings performed at $\vartheta_{z_r} > 0.5$ there is a noticeable overshoot of the set point. This is caused by the vehicle thrust being saturated at max throttle. Landing at speeds of around 2 m/s seems to be the limit of the MavTec in this configuration.

CONSTANT DIVERGENCE, ADAPTIVE GAIN LANDING

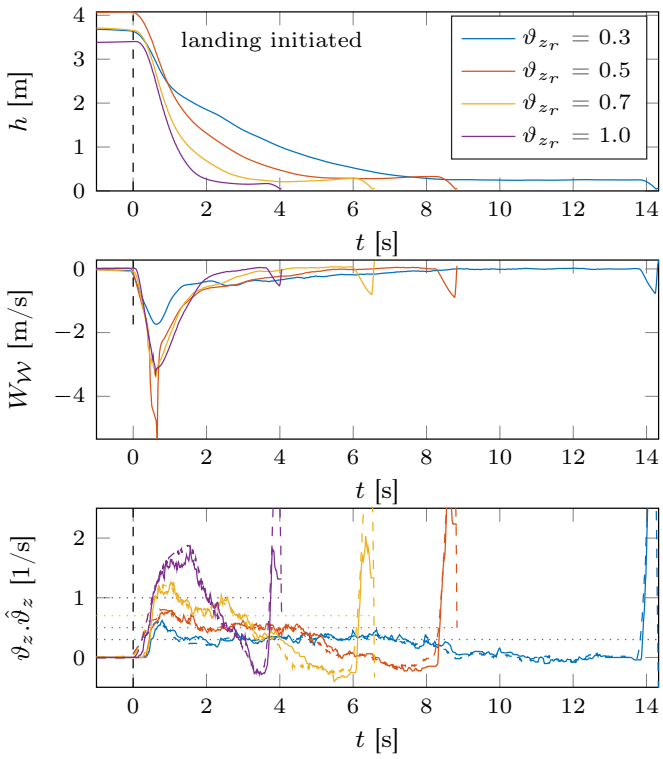


Figure 4.13: Height above ground, vertical speed, and divergence measurements with ground truth during a constant divergence landings with adaptive controller gains performed at four different divergence setpoints. In the bottom graph, the dotted, dashed, and solid lines represent the setpoint, ground truth, and estimate for ϑ_z respectively.

To improve the tracking performance and rise time of the landing controller as well as to remove the oscillation during landing we implemented a method to adapt the proportional gain of the controller as we descend. This has been shown to be an effective method for performing constant divergence landings [85]. The update rule

for the controller gain is defined as:

$$k_P = k_{P_0} e^{\vartheta_{z_r} t} \tag{4.28}$$

where k_{P_0} is the initial gain used at the start of the landing and t is the elapsed time of the landing.

Fig. 4.13 shows the results of landing with this adaptive approach where k_{P_0} was set to 0.8. This figure shows a faster rise time of the controller with reduced overshoot and generally better tracking performance. Some oscillation is still observed suggesting that a derivative component would be beneficial to the controller.

Landing with the higher gain helps to reduce the rise time of the controller while introducing a slight overshoot of the setpoint. The tracking of the setpoint is improved with no oscillations. At high values of ϑ_{z_r} there is still a clear overshoot of the setpoint due to the saturation of he throttle with landing speeds peaking at 4 m/s. Also interesting to note is at the end of the landings, the vehicle experiences an aerodynamic phenomenon called *ground effect*. When the vehicle is just above the ground a pocket of air form between the rotors changing the effectiveness of the rotors. This is a useful feature which helps to further guarantee a safe landing. If this is in-fact undesired, an integral gain may be added to take the change of rotor effectiveness into account.

4.6. EXTENSIONS

The above algorithm was re-implemented onboard the light weight with the Parrot Bebop 2 quadrotor MAV equipped with the significantly smaller Insightness SEEM1 event-based camera. This sensor has a more dense pixel density with 320×262 pixel resolution and a 16 g form factor. Directly connecting the SEEM1 to the Bebop removes the need for the Odroid co-processor significantly simplifying the system.

An additional benefit of the SEEM1 is the inclusion of an onboard IMU. This high speed data acquisition of rotational rates and acceleration using the same clock time as that used for the camera events facilitates more accurate derotation of the opticflow vectors. To achieve this, we pass the incoming rotation rates through a moving windowed median filter at 1 kHz. We sample this moving filter and store the rates at 250 Hz. As the opticflow method is based on a spatio-temporal plane fit, the derotation should consider this time window. We do this by averaging the rotational rates in the same time window used to generate the optical flow vector. This tight coupling of the derotation results in better ventral flow performance.

For the confidence estimation, we determined that discounting the confidence with low event rate was not appropriate and was therefore was removed. Only the variance and coefficient of determination are currently being used.

The high update rate of the pixels, although reducing the total data throughput of a comparable high speed camera, does produce a high event rate. The higher

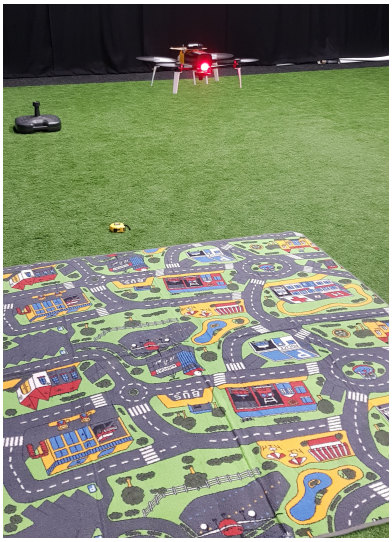


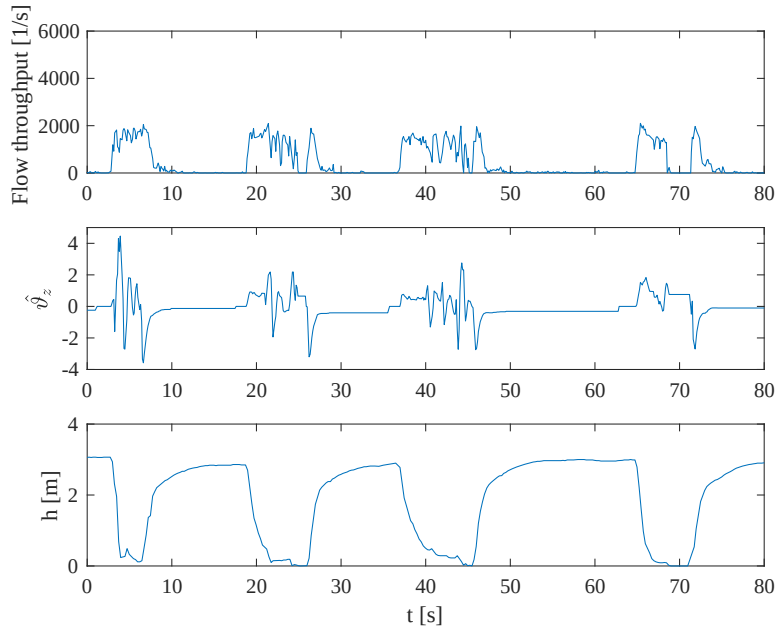
Figure 4.14: SEEM1 event-based camera mounted on the Parrot Bebop 2 quadrotor MAV.

resolution of the SEEM1 also results in a higher event throughput than the DVS128 with some recordings have spikes of over 4M events/s. Additionally, the processor on the Bebop is a dual core operating at 780 MHz, less powerful than the dual quadcore processor at 2.1 GHz and 1.5GHz on the Odroid. The flow computation takes about 30 μs onboard the Bebop, resulting in a maximum throughput of about 33k events/s. For this algorithm to run in real time, not all events can be processed.

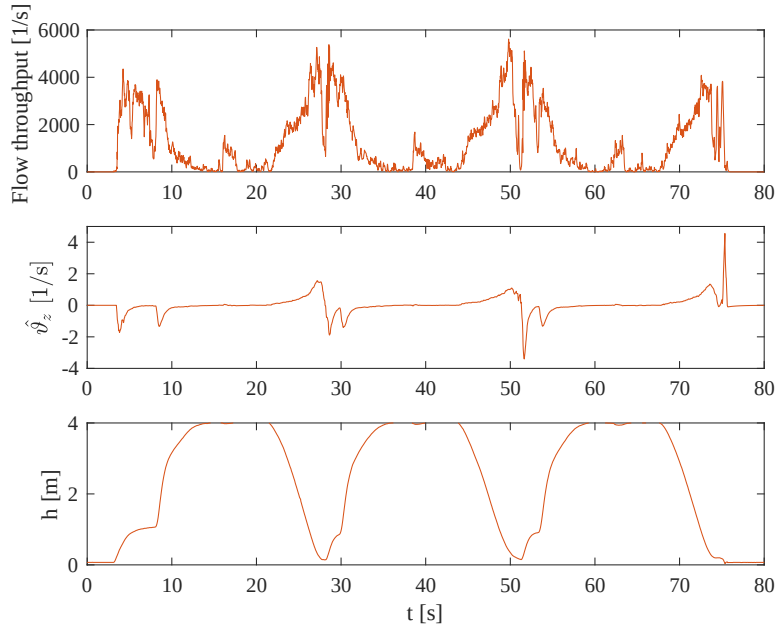
To achieve this reduction in throughput, we first artificially make the input square, reducing the resolution from 360$\times$262 to 262$\times$262. This results in about an 18% reduction in the maximum event throughput. Next, with the refractory period for a pixel is implemented as a combined check on both input channels (on and off events) together rather than separately as previously done. With this, we can reduce redundant spikes caused by rapid changes or flashing. This allowed us to reduce the refractory period 50 ms down from 300 ms, which should result in higher resolution results. Finally, instead of regulating the output flow vector rate, we limit the minimum time between computed flow vectors. This allows us to directly limit the maximum computational load. We set the limit to 10k events/s to ensure we can handle peak throughput with little added computational lag.

Despite the limitations with the new processor, we are able to consider more events than the original implementation by removing the communication limitation with the co-processor as can be seen in Fig. 4.15. The limiting factor is only the computational power of the flight platform and the performance can be scaled by a single parameter.

Fig. 4.16 shows a comparison for divergence output of the original and new implementations. We can see that the number and magnitude of outliers is significantly



(a) Original



(b) Updated

Figure 4.15: Comparison of flow vector throughput of original and updated implementations (for different flights).

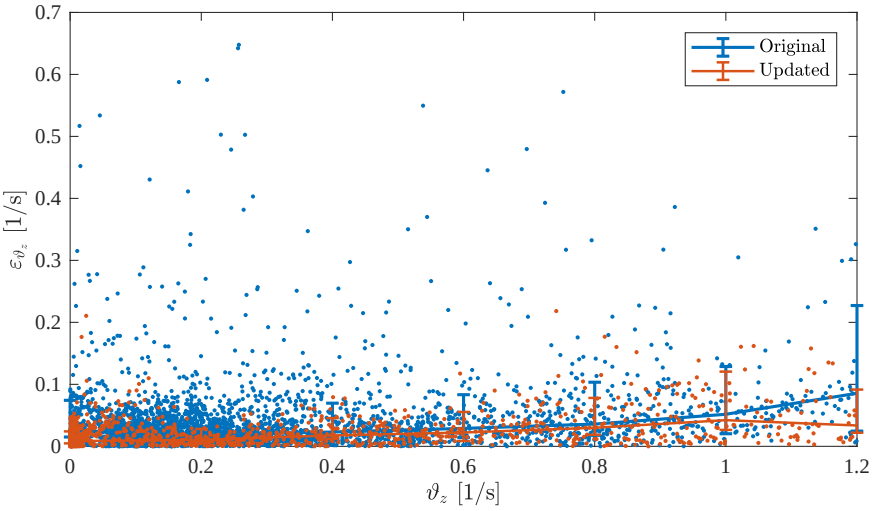


Figure 4.16: Comparison of divergence error of original and updated implementations.

reduced resulting in a slightly lower median and tighter percentile bounds when compared to the original implementation. This leads to a generally better divergence estimate overall.

4.7. CONCLUSION

In this chapter we present a successful implementation of event-based optical flow estimation into a constant divergence landing controller for flying robots. Three main contributions lead to this result.

First, a novel algorithm for computing event-based optical flow is derived from an existing local plane fitting technique. The algorithm is capable of estimating normal optical flow with a wide range of magnitudes through timestamp-based clustering of the event cloud. Its performance is evaluated in ground texture scenes recorded by a DVS. Accurate estimates are seen in real event scenes with sparse, high contrast edges, as well as in scenes with densely packed, lower contrast features. Compared to the existing technique, optical flow accuracy is slightly improved for fast motion, while a larger number of successful optical flow estimates is obtained during slow motion. In addition, it is shown that the optical flow detection rate can be capped to limit computational effort for the algorithm, which enables implementation on low-end platforms without sacrificing accuracy.

Second, we introduce an algorithm for estimating optical flow based visual observables from normal optical flow measurements. By grouping flow vectors by their direction, the aperture problem can be limited by estimating the parameters of a planar optical flow field. The estimator assesses the reliability of its output through

a confidence metric based on the flow estimation rate, the variance of optical flow positions, and the coefficient of determination of the flow field. When coupled to the optical flow algorithm, it is capable of estimating the visual observables accurately over a wide range of speeds. Also, the influence of fast rotational motion on the visual observables is adequately corrected through separate rotational rate measurements.

Third, using the developed pipeline, fast constant divergence landing maneuvers are demonstrated using a quadrotor equipped with a downward facing DVS. Decent tracking performance is achieved for the majority of the descent using a simple proportional controller. The final touchdown of the landing maneuver is not yet performed due to self-induced oscillations close to the ground. However, stability-based control methods have already been demonstrated that can resolve this issue. A future controller based on these methods can, for example, autonomously detect the oscillations and switch to a final touchdown phase based on constant thrust, or perform a complete landing maneuver using an adaptive gains. In addition, our controller does not yet incorporate the visual observables for horizontal stabilization, but relies on an external position tracking system. However, with the estimate accuracy and rotational motion correction presented in this work, this appears feasible.

In a first-order comparison to recent work on landing using frame-based cameras for estimating optical flow, the presented event-based pipeline demonstrates more accurate measurements at high speed and a higher sampling rate, which enable faster maneuvers than previously shown in literature. However, for a more solid conclusion regarding real-time computational benefits of event-based vision, a comparison should be performed where both frame-based and event-based cameras are incorporated in the same hardware configuration.

Supplementary material

- The dataset used to generate these plots and statistics is available at: `http://hdl.handle.net/10411/FBKJFH`
- DVS event processing software implemented in the cEAR framework is publicly available at: `https://github.com/tudelft/caer/tree/odroid-dvs`
- Flight control software implemented within the PaparazziUAV framework used in this chapter is publicly available at: `https://github.com/tudelft/paparazzi/tree/event_based_flow`
- Video of the experiments performed can be found at: `https://www.youtube.com/playlist?list=PL_KSX9GOn2P8RBdSyzngewi76G37PI3SF`

PARAMETER SETTINGS

Table 4.2: DVS bias settings applied in cAER during the experiments.

Bias	Value
cas	864
diff	30153
diffOff	132
diffOn	482443
foll	102
injGnd	1108364
pr	108
puX	8159221
puY	16777215
refr	242
req	159147
reqPd	16777215

Table 4.3: Overview of all algorithm parameter values used in the experiments.

Parameter	Value
Optical flow estimation	
Refractory period, Δt_R	0.3 s
Time window, Δt	1.5s
Spatial window, Δx, Δy	5 pix
Maximum number of rejected events n_R	4
$NRMSE_{max}$	0.3
Time difference factor for clustering, k_S	8
Minimum number of events in a fit, n_{min}	10
Maximum optical flow velocity, V_{max}	1000 pix/s
Optical flow output rate setpoint, $\rho_{F_{max}}$	2500 s^{-1}
Visual observable estimation	
Number of flow field directions, m	6
Flow field preservation time constant, k_f	0.02 s
Minimum variance, $\mathrm{Var}\{S\}_{min}$	500 pix^2
Confidence filter,	
Minimum optical flow estimation rate, $\rho_{F_{min}}$	500 s^{-1}
Minimum coefficient of determination, R^2_{min}	0.8
Low-pass filter time constant, k_t	0.04 s
Max filter update, $\Delta\vartheta_{max}$	0.3 s^{-1}

5

Masking the Reality Gap with Sensor Abstraction

Fear exists for one purpose: to be conquered.

- Kathryn Janeway
Star Trek: Voyager, Season 2 Episode 23

The contents of this chapter have been submitted for publication as:

Title	Evolution of Robust High Speed Divergence-Based Landing for Autonomous MAV
Journal	Under review at the Journal of Robotics and Autonomous Systems
Authors	K.Y.W. Scheper and G.C.H.E. de Croon

As robots are embodied agents, their perception of the world around them is directly coupled to the actions that they take. This sensor-action loop or sensorimotor coordination is integral to expressing the appropriate behavior on the robotic platform. The previous chapter presented an efficient method to process the data from a novel sensor, this chapter will use this to investigate the impact of abstracting the robotic sensory system on the reality gap.

This chapter delves into the effect of abstracting away from raw sensor inputs of a virtual robot on its transferability to the real world. We present a more detailed description of the robotic task and the method used to investigate the effect of abstraction on the reality gap. This is followed by the simulated and real world results with some conclusions.

The authorse would like to thank the team at Insightness AG for their assistance in porting the SEEM1 SDK to operate on the ARM processor of the Parrot Bebop 2. Without their assistance the results in the paper would not be possible.

5.1. Introduction

Insects are a significant source of inspiration for developing novel solutions to autonomous flight of Micro Air Vehicles (MAVs). Even with limited computational and energy resources, insects are able to effectively complete challenging tasks with relatively complex behaviors. One behavior that is of particular interest, is landing.

Insects have been shown to primarily rely on visual inputs when landing, whereby many insects regulate a constant rate of expansion (or *divergence*) of optical flow [68] to perform a smooth landing [7]. This approach has inspired some robotic implementations of constant divergence landing strategies [81]. Although simple in concept, this approach is quite difficult to implement in reality. A direct implementation causes the robot to become unstable as the vehicle nears the ground. This is a result of the non-linear interaction between the vehicle control and sensing, in the presence of delay, measurement noise and environmental disturbance [41]. Different augmentations to the standard control scheme have been made, most, simply perform slow landings [81], use long landing legs to delay the onset of this instability and touch the ground before they occur or by switching to alternative measures like time-to-contact [31, 96] or velocity in-plane of the camera [119, 149]. More recently, there have been attempts to identify the instability and adapt the landing strategy by adjusting control gains [85, 142].

An alternative to these manually designed approaches is to have an optimization technique automatically develop a suitable solution that is robust to these instabilities. This optimization may reveal new solutions to this problem, and perhaps even alternative hypotheses on what flying insects such as honeybees may be doing. Some attempts have been made to do this [87] but to the best knowledge of the authors, none of these have been implemented on real world robots.

This is due in part, to the effects of the differences between the simulated environment, commonly used for behavioral optimization, and the real world. This resultant difference in the robotic behavior is commonly referred to as the *reality gap* [19, 131]. Several approaches have been used to make controllers more robust to the reality gap with the most significant being adding appropriate and varied noise [92], co-evaluation of controllers in the real world to test transferability [101] and inspired by conventional control theory, there are approaches utilizing abstracted outputs from the neurocontroller with a closed loop control system to actively reduce the effect of reality gap [154, 162].

In this paper, we describe a method to optimize a quick but safe landing maneuver for a quadrotor MAV equipped with a downward facing camera. The neurocontrollers are given only the divergence of the optical flow field from the camera as input and its time derivative. Although this abstracted input may reduce the possible behaviors the controllers can express, building on the work in [155], we will demonstrate that this abstraction leads to a robust transfer from simulation to reality after virtual optimization.

The following consists of a summary definition of optical flow in Section 5.2. The flight platform and simulation environment is then described in Section 5.3. Next, the performance of conventional constant divergence landing approaches are presented in Section 5.4 to provide some baseline performance to compare the optimized policies against. The evolutionary setup and neural models used for the neurocontrollers is then described in Section 5.5. This is followed by a presentation and analysis of the optimized policies in Section 5.6. The reality gap and the results from the real world experiments are presented in Section 5.7 and Section 5.8 from which we draw some conclusions in Section 5.9.

5.2. OPTICAL FLOW DEFINITION

To perform optical flow based landing as in [44, 84, 85, 142], we must first define the optical flow parameters. The formulation here is a summarized version of that presented in [44]. This algorithm provides a good trade-off of accurate optical flow estimates while using relatively limited computational resources. This allows the perception and control loop to operate at high frequency and low latency on the embedded flight platform used in this paper. Alternative optical flow estimation methods could be used given that the estimation runs fast enough to facilitate the flight control. An investigation of the accuracy and reliability of the optical flow estimation is out of the scope of this paper.

If we assume that we have a downward-looking camera overlooking a static planar scene, as shown in Fig. 5.1, we can derive the perceived optical flow as the result of the camera ego-motion. The derivation of this optical flow model relies on the two reference frames, the inertial world frame is denoted by $\mathcal{W}$ and the camera frame centered at the focal point of the camera denoted by $\mathcal{C}$. In each of these frames, position is defined through the coordinates (X, Y, Z), with (U, V, W) as the corresponding velocity components. The orientation of $\mathcal{C}$ with respect to $\mathcal{W}$ is described by the Euler angles ϕ, θ, and ψ, denoting roll, pitch, and yaw, respectively. Similarly, p, q, and r denote the corresponding rotational rates.

The camera ego motion can be related to the optical flow, and visual observables based on the pinhole camera model [111] with camera pixel coordinates are denoted by (x, y), while (u,v) represent optical flow components, measured in pixels per second. These can be non-dimensionalized using the intrinsic calibration of the camera.

$$\begin{aligned} u &= -\frac{U_{\mathcal{C}}}{Z_{\mathcal{C}}} + \frac{W_{\mathcal{C}}}{Z_{\mathcal{C}}}x - q + ry + pxy - qx^2 \\ v &= -\frac{V_{\mathcal{C}}}{Z_{\mathcal{C}}} + \frac{W_{\mathcal{C}}}{Z_{\mathcal{C}}}y + p - rx - qxy + py^2 \end{aligned} \tag{5.1}$$

Eq. (5.1) shows that the optical flow of a point can be resolved into translational and rotational components. Since the latter is independent of the three-dimensional structure of the visual scene, these expressions can be derotated if information on the rotational rates of the camera is available. This derotation leads to pure

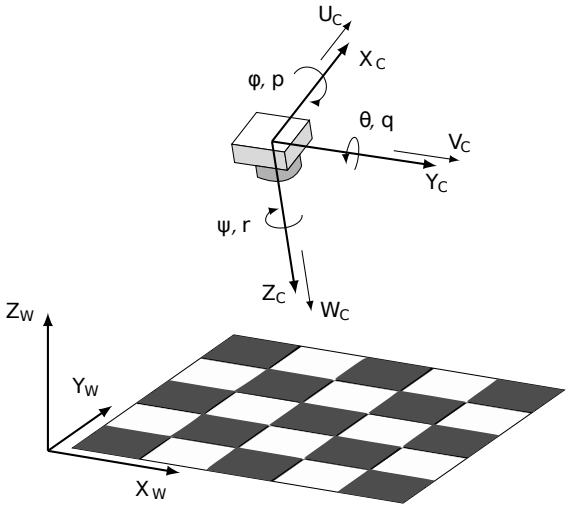


Figure 5.1: Definitions of the world ($\mathcal{W}$) and camera ($\mathcal{C}$) references frames. Also shown are the Euler angles (ϕ, θ, ψ), rotational rates (p, q r), and translational velocities (U, V, W) that describe the motion of $\mathcal{C}$ [135].

translational optical flow components, denoted by (u_T, v_T). Moreover, if the scene is a planar surface, the depth $Z_\mathcal{C}$ of all visible world points are interrelated through:

$$Z_\mathcal{C} = Z_0 + Z_X X_\mathcal{C} + Z_Y Y_\mathcal{C} \tag{5.2}$$

where Z_0 is defined as the distance to the surface along the optical axis of the camera, and Z_X and Z_Y represent the slopes of the planar scene with respect to the X- and Y-axis of $\mathcal{C}$.

In [111], the relation between the position of an arbitrary point in $\mathcal{C}$ and its projection onto the image plane is given by (x, y) = ($X_\mathcal{C}/Z_\mathcal{C}$, $Y_\mathcal{C}/Z_\mathcal{C}$). Consequently, Eq. (5.2) may also be written in the form:

$$\frac{Z_\mathcal{C} - Z_0}{Z_\mathcal{C}} = Z_X x + Z_Y y \tag{5.3}$$

Further, let the *scaled velocities* of the camera ϑ_x, ϑ_y, and ϑ_z be defined as follows:

$$\vartheta_x = \frac{U_\mathcal{C}}{Z_0}, \quad \vartheta_y = \frac{V_\mathcal{C}}{Z_0}, \quad \vartheta_z = \frac{W_\mathcal{C}}{Z_0} \tag{5.4}$$

Then, according to the derivations in [44], substituting Eq. (5.3) and Eq. (5.4) into Eq. (5.1) leads to the following expressions for translational optical flow:

$$\begin{aligned} u_T &= (-\vartheta_x + \vartheta_z x)(1 - Z_X x - Z_Y y) \\ v_T &= (-\vartheta_y + \vartheta_z y)(1 - Z_X x - Z_Y y) \end{aligned} \tag{5.5}$$

From Eq. (5.5), and under the aforementioned assumptions, the scaled velocities, which provide non-metric information on camera ego-motion, can be derived from the translational optical flow of multiple image points. ϑ_x and ϑ_y are the opposites of the so-called *ventral flows*, a quantification of the average flows in the X- and Y-axis of $\mathcal{C}$ respectively [142]. Hence, $\omega_x = -\vartheta_x$ and $\omega_y = -\vartheta_y$. On the other hand, ϑ_z is proportional to the *divergence* of the optical flow field, $D = 2\vartheta_z$ [142].

The flow divergence can alternatively be estimated simply by the relative change in the distance (l) between any two points at over time (t) [85]. This method is referred to as *size divergence* (D_{s_t}). A reliable estimate of the divergence ($\hat{D}$) can be generated by averaging the divergence estimate from a set of N points in the image. Throughout this paper, N is limited to 100 if there were more than 100 points or simply all points if fewer have been tracked.

$$\begin{aligned} D_{s_t} &= \frac{1}{\Delta t}\frac{l_{t-\Delta t} - l_t}{l_{t-\Delta t}} \\ \hat{D} &= \frac{1}{N}\sum_{i=1}^{N} D_{s_t} \end{aligned} \tag{5.6}$$

5.3. FLIGHT PLATFORM AND SIMULATION ENVIRONMENT

The flight platform used in this paper is the Parrot Bebop 2 quadrotor MAV[1], a picture of this vehicle flying in our indoor test environment can be found in Fig. 5.2. This vehicle is equipped with a 780 MHz dual-core Arm Cortex A9 processor, forward and downward facing CMOS cameras, sonar, and barometer enabling autonomous flight capabilities for up to 25 minutes. Full 3D flight control is enabled with the onboard use of the open source PaparazziUAV autopilot software [77]. In this work, we extract global optical flow from the downward facing camera using the Lucas-Kanade optical flow method executed onboard the vehicle [113][2].

Throughout the paper, an Optitrack[3] motion capture system was used to measure a ground truth of the vehicle position and motion. As the control task in this paper is only in the vertical axis, this ground truth measure was communicated to the vehicle to facilitate the control of the lateral axis of the vehicle. This was however not used in the vertical loop where instead the optical flow estimated onboard the vehicle was used.

To optimize our behavior in simulation, we first need a model of the vehicle. To this end we use a simple dynamical model of the vehicle, which is restricted to vertical motion only. The thrust generated by the rotors is modeled as a first order response

[1] https://www.parrot.com/nl/en/drones/parrot-bebop-2

[2] Code used in this paper can be found https://github.com/kirkscheper/paparazzi/tree/updated_event_based_flow

[3] https://optitrack.com/

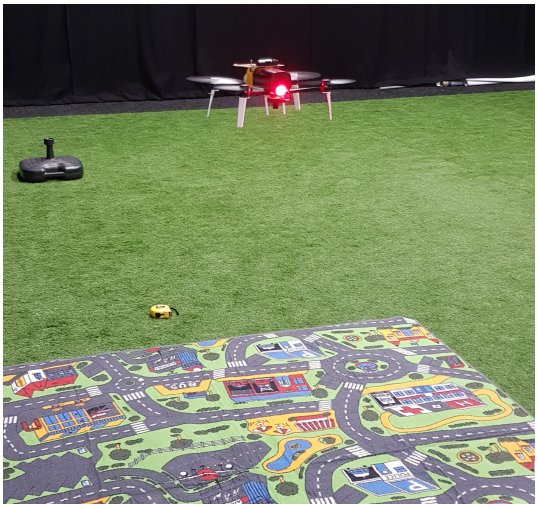


Figure 5.2: Parrot Bebop 2 quadrotor MAV equipped with SEEM1 Dynamic Vision Sensor.

with the dynamics defined in (5.7).

$$(\Delta t + \tau_T)\dot{T}_i = T_{sp} - T_{i-1} \tag{5.7}$$

where T_{sp} is the thrust set-point and spin-up spin-down time constant τ_T has a nominal value of 0.02 s. The thrust output is limited in the range [-0.8·g, 0.5·g] as a conservative model the maximum acceleration of the real vehicle.

The model used to describe the divergence estimation is based on the work presented in [84]. The observed divergence is the result of adding latency to the true divergence along with two types of noise, simple white noise drawn from $N(0,\sigma_w^2)$ and an additional noise proportional to the divergence magnitude drawn from $N(0,\sigma_p^2)$. [84] identified typical values for σ_w and σ_p as 0.1 s^{-1} and the latency L in the range of [50, 100] ms. We use similar nominal values for the standard deviations σ_w and σ_p are 0.1 s^{-1} and an exaggerated range for the latency L in the range of [1,4] samples or [20, 133] ms. Some additional computational jitter has been included, this simulates the situation of missed frames which happens when there are either insufficient or too many image features to be tracked. The chance of a missed frame is randomly determined with a given probability held constant for each simulation run randomly drawn from the range [0, 0.2].

5.4. BASELINE PERFORMANCE

Before we start to optimize a divergence based landing controller, let us first investigate the naive approach of a constant gain, constant divergence landing as studied in [41, 81, 170].

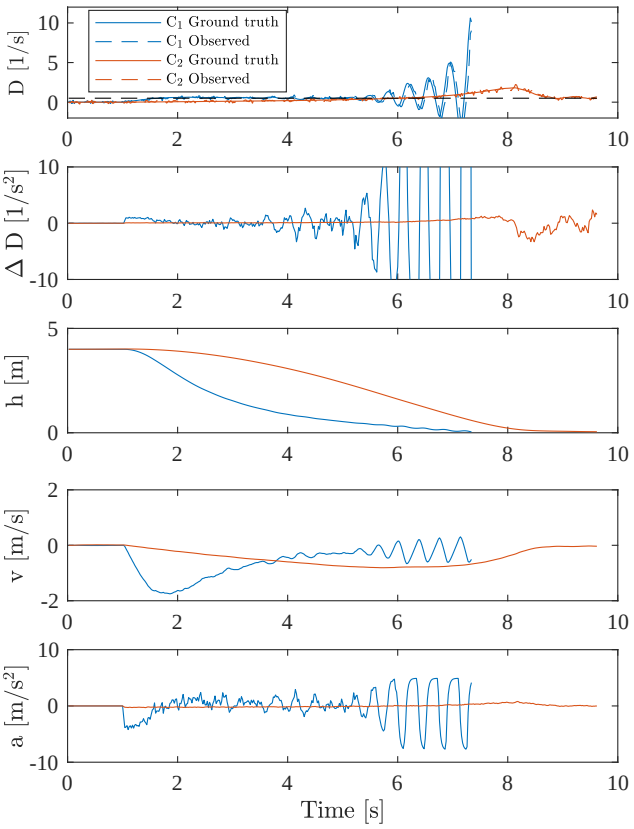


Figure 5.3: Time history of a constant divergence landing with two controllers of differing gain. C_1 is a controller with high gain and C_2 with low gain.

Fig. 5.3 shows the time history of a two constant divergence landing controllers performing a landing with a divergence set-point of 0.5. Controller C_1 has a relatively high gain and C_2 a low gain. The controllers are activated 1 s after the simulation starts. Fig. 5.16 shows the steady-state response of the controller with the time history control signal super-imposed.

These plots show that controller C_1 quickly reaches the desired divergence but as the vehicle descends the controller becomes increasingly unstable. This instability is due to delays in the sensing and control loop and is well described in [41]. In contrast, C_2 does not quickly achieve the desired divergence due to the low gain in the controller. Additionally, due to the non-linear relationship between the divergence and acceleration, the vehicle overshoots the desired set-point before finally slowing. Despite this poor tracking performance, the low gain does delay the onset of instability observed with C_1, resulting in a rather smooth landing profile.

5.5. EVOLUTIONARY OPTIMIZATION

Every evolutionary process is defined by: a population of candidate individuals, each with a given policy which must be evaluated; a way to evaluate these policies; a selection mechanism to filter out bad policies; and a method to alter the individuals to generate new policies. Here, we use a mutation-only evolutionary algorithm similar to (μ + λ) approach, where a population is maintained from which offspring are generated using a mutation operator. Offspring that are better than members of the population replace these members. The difference with the standard implementation is that we retest the current population on a random set of simulation parameters with every generation, rather than once as is commonly done. With the high level of non-determinism in our simulated environment, this ensures no individuals are preferred simply because they were tested on an *easy* set of conditions.

The evaluation of the individuals is done by simulating the policy on four independent simulation runs, initializing the vehicle at a standstill from four different altitudes, namely 2, 4, 6 and 8 m. The simulation ends when the vehicle exceeds 15 m above the ground, gets within 5 cm of the ground or exceeds 30 s of simulation time.

We use a multi-objective approach here, where the individuals must minimize three fitness functions measured at the end of a simulation: the total time to land (f_1); the final height (f_2); and the final velocity (f_3). NSGA-II is used to perform a non-dominated sorting of the population and determine which individuals are better than others in this multi-objective framework [47].

All the simulation parameters mentioned in Section 5.3 are randomly perturbed and set at the start of a generation. The ranges of the evolutionary parameters used in this work are summarized in Table 5.1. Altogether, this should encourage the optimization to develop a policy that can reliably land quickly and safely from different altitudes. This is all developed using the DEAP framework [66] with multi-threaded implementation utilizing scoop in Python[4][5].

The policy of each individual is encoded in a simple neural network. The neural potential (γ) of each neuron in the neurocontroller is updated with a simple discrete time Euler integration as described in Eq. (5.8).

$$\gamma(t) = \gamma(t-1) + \dot{\gamma}(t-1)\Delta t \tag{5.8}$$

where t represents the current time step and Δt is the time step of the integration.

The input to the neurocontroller is the simulated divergence and the derivative of the divergence $\Delta D = \frac{D_t - D_{t-\Delta T}}{\Delta t}$. The output was used to control the thrust of the vehicle leading to an acceleration. We utilized three types of neural networks to

[4]The software is openly available at: `https://github.com/DEAP/deap`

[5]Software used for the evolutionary process in this paper is openly available at: `https://github.com/kirkscheper/divergence_landing`

Table 5.1: Evolutionary parameters

Parameter	Value
Number of Generations	250
Number of Runs	4
Range of delays	[1, 4] samples
Range of computational jitter probability	[0, 0.2]
Range of divergence noise (σ_w^2)	[0.05, 0.15] 1/s
Range of divergence noise (σ_p^2)	[0, 0.25] 1/s
Range of thrust time constant (τT)	[0.005, 0.04]
Range of simulation frequency	[30, 50] Hz

investigate the effect of recurrent connections on the evolved solution. We implemented a feed-forward neural network (hereafter referred simply as NN), a recurrent neural network (RNN) and a continuous time recurrent neural network (CTRNN). All networks had three layers, the first with 2 neurons, the hidden layer with 8 neurons and the output with 1 neuron.

5.5.1. NN

The neural potential of an NN is defined by the instantaneous inputs to the network such that the potential of a neuron i in a given layer l connected to N^{l-1} neurons in the previous layer is determined simply by:

$$\gamma_i^l = \sigma^l(\sum_{j=1}^{N^{l-1}} w_{ij}^l \gamma_j^{l-1}) + \theta_i^l + I_i^l \tag{5.9}$$

where w_{ij} is the weight of the neural connection between the neuron i in layer $l = (2, 3, \cdots)$ and a given neuron j is the previous layer, θ is the neural bias, I is the external input to the neuron and σ is the activation function of the neuron. Here, the activation function is linear for the output layer and the Rectified Linear Unit (ReLU) function for the hidden layer. We can rewrite this equation using Eq. (5.8) to generate the neural dynamics.

$$\Delta t \dot{\gamma}_i = -\gamma_i + \sigma(\sum_{j=1}^{N} w_{ij}\gamma_j) + \theta_i + I_i \tag{5.10}$$

5.5.2. RNN

The NN has no effective way of explicitly considering previous states in determining it's current action. The behavior is simply a result of the emergent interaction of the vehicle actions and the environment. Adding explicit memory may enable the vehicle to exhibit more complex behaviors. RNNs are not only affected by an external

input and the weighted sum of connected neurons but also by a weighted internal connection to the previous potential as shown in Eq. (5.11).

$$\dot{\gamma}_i = \gamma_i r_i + \sigma(\sum_{j=1}^{N} w_{ij}\gamma_j) + \theta_i + I_i \tag{5.11}$$

where r is the weight applied to the recurrent connection. Like the NN, the activation function is linear for the output layer and the ReLU function for the hidden layer.

Again, using Eq. (5.8), we can derive the neural dynamics:

$$\Delta t\dot{\gamma}_i = \gamma_i(r_i - 1) + \sigma(\sum_{j=1}^{N} w_{ij}\gamma_j) + \theta_i + I_i \tag{5.12}$$

5.5.3. CTRNN

One pitfall of the RNN is that the recurrent connection does not consider the time between updates. This can be an important consideration for systems with variable time steps between updates. As such, we have also implemented the classic CTRNN as shown in Eq. (5.13) [14].

$$(\Delta t + \tau_i)\dot{\gamma}_i = -\gamma_i + \sum_{j=1}^{N} w_{ij}\sigma(\gamma_j + \theta_j) + I_i \tag{5.13}$$

where τ is its time constant ($\tau > 0$). The hyperbolic tangent activation function is used here ($\sigma(x)$ = tanh = $(e^x - e^{-x})/(e^x + e^{-x})$).

5.6. EVOLUTION RESULTS

The use of the non-dominated sorting and selection, results in the genetic optimization spreading the population of policies over the pareto front of the fitness landscape. As such, to evaluate the performance of the optimization, it is sensible to look at the ratio of the encapsulated volume and the area of the pareto front ($\nu = volume/area$) as shown in Fig. 5.4. This figure shows that the performance generally improves before leveling off after about 150 generations. This performance is also mostly stable as the performance remains flat after 150 generations. This also shows that this trend is consistent over the multiple initializations of the optimization and over the different types of neural architecture.

Fig. 5.5 shows the accumulated Pareto front of all individuals from the different runs of the optimization. This figure only shows the performance on the touchdown velocity and the time fitness as the fitness based on the height was consistently minimized for all individuals. As such, all optimized policies converged to perform the desired landing task. This figure shows how the policies are spread over the

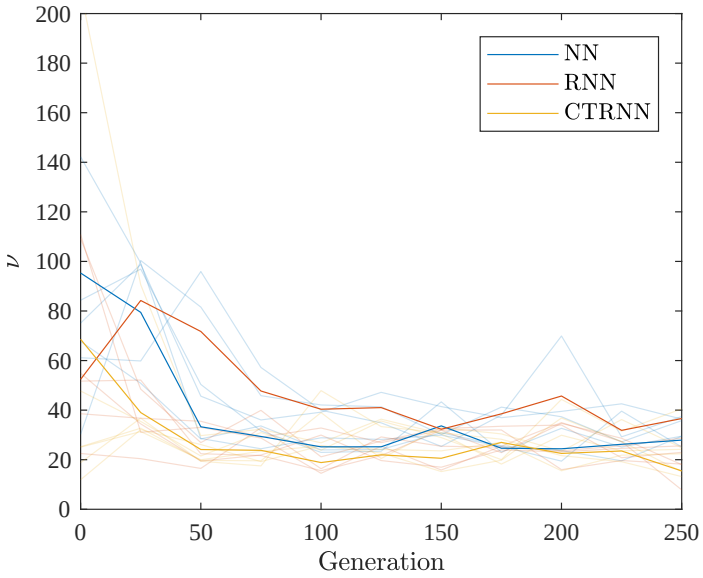


Figure 5.4: Performance of genetic optimization measured with ν which is the ratio of the encapsulated volume and area of the pareto front. A smaller ν would suggest a general improvement in the minimization of all individuals while they spread out over the available optimum policies. The pareto front was generated by evaluating the entire population at each generation to the same set of simulated environmental conditions.

inherent trade-off of reliable quick vs soft landing. The performance seems not to be strongly correlated with the neural architecture as all three types are represented in the pareto front. Additionally, the three architectures seem well distributed.

To investigate the sensitivity of the performance to different environmental conditions, we subjected the individuals of the pareto front to a validation test with 250 simulations while varying the environmental settings of the run. All individuals where subjected to the same set of conditions to make for a fair comparison. Fig. 5.6 shows that the individuals that optimized to have slow and soft landings have a small spread in the fitness performance whilst individuals that were optimized to faster landings have a larger variation in landing speed. This suggests that the policies that perform faster landings may show oscillatory behavior causing their touchdown speed to vary depending on the sensor noise or environmental perturbations. We will investigate this more below.

As it is not feasible to analyse them all, three individuals from the each architecture are be selected for further analysis of the landing behavior. The remainder of this section will dive deeper into the types of behaviors optimized by the different neurocontrollers.

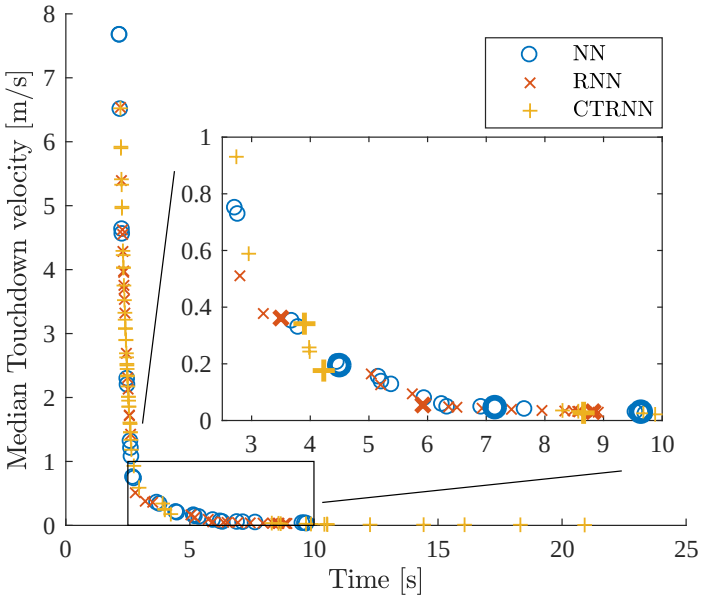


Figure 5.5: Performance of pareto front tested on 250 evaluations. Individuals selected for further analysis are bold.

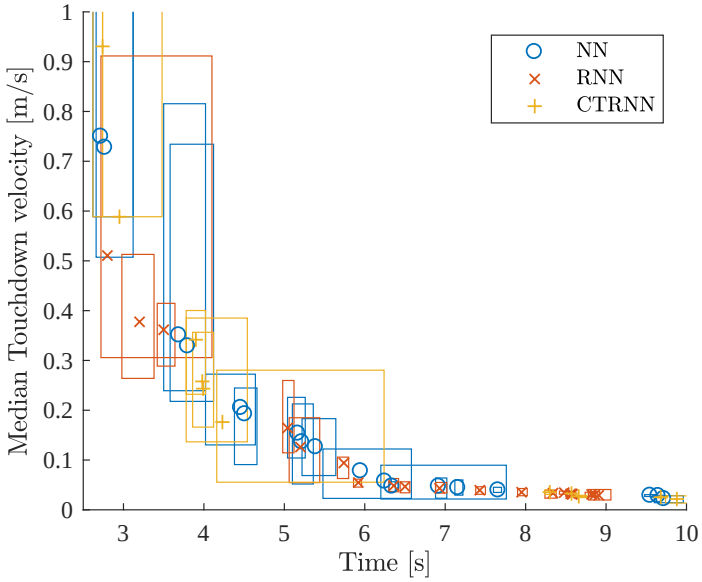


Figure 5.6: Sensitivity analysis of a portion of pareto front showing 25^{th}, 50^{th} and 75^{th} percentile of performance over 250 evaluations.

5.6.1. NN

Three neurocontrollers were selected from the Pareto front for some further analysis, their performance is shown in Fig. 5.7 and their steady-state response in Fig. 5.17. Controller NN_1 is a slow lander, NN_3 is a fast lander and NN_2 is intermediate.

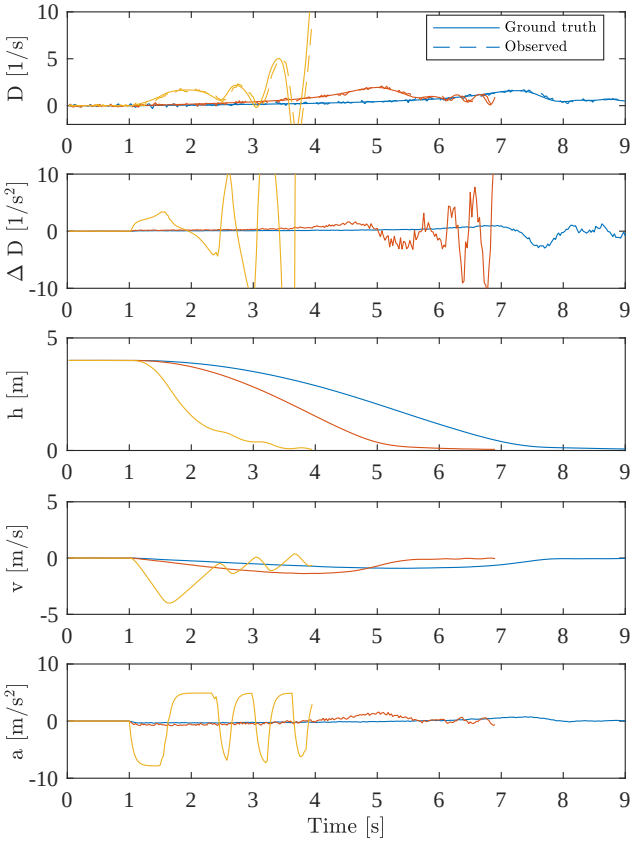


Figure 5.7: Vehicle states and observations from simulation with NN_1 (blue), NN_2 (red) and NN_3 (yellow).

Looking first at the world states in Fig. 5.7, we can see that NN_1 and NN_2 perform smooth landings with little oscillation and touchdown at very low velocities. Examining the steady-state response, NN_1 is similar to the low gain baseline C_2 except that instead of being a constant gain for all inputs, NN_1 is piece-wise linear with a high gain for positive values D and a low gain for negative values. This asymmetric control scheme is a significant result as this will delay the onset of the oscillation seen in higher gain controllers.

NN_2 is similar to NN_1 but has a higher gain and a noticeable gradient in the ΔD

with reduced thrust at negative ΔD. This is also an interesting result as a negative ΔD occurs when the vehicle is slowing while descending or accelerating while ascending. In both cases, it would indeed be desirable to reduce the control input.

NN_3 has clear oscillations and is similar to the high gain controller C_1 but is even higher gain, this causes the controller to act as a *bang-bang* controller. This type of control will cause the rotors of the vehicle to try to spin up and down very often, however, due to their inertia, they do not achieve the desired values. As such this control scheme relies on the simulated spin-up and spin-down reaction time of the rotors (τ_T) for the desired behavior to work well and will likely not transfer well to the real world. This may also explain why the controllers on the left of the pareto front in Fig. 5.6 have vertically elongated bounding boxes.

5.6.2. RNN

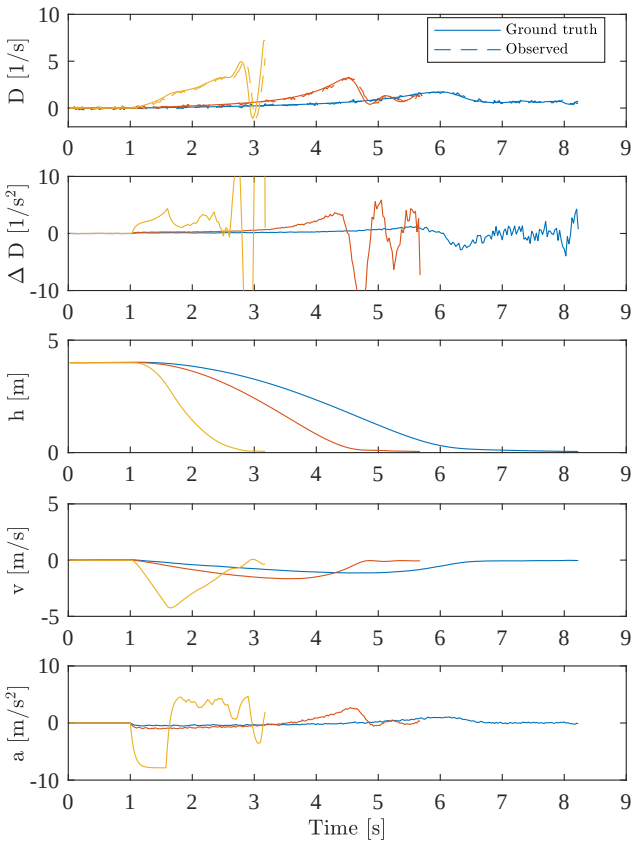


Figure 5.8: Vehicle states and observations from simulation with RNN_1 (blue), RNN_2 (red) and RNN_3 (yellow).

The selected RNN neurocontrollers are shown in Fig. 5.8 and Fig. 5.18. RNN_1 and RNN_2 show similar behavior to the policies with NN_1 and NN_2. These relatively high gain landings with a gradient on the ΔD seems to be a reliable way to perform this type of high speed yet smooth landing. RNN_3 is a little different than NN_3 in that high speed landings with a negative divergence rate have a lower throttle response. This would occur when the vehicle is descending quickly but slowing down. This control scheme would ensure that the vehicle doesn't over react when the vehicle is descending quickly but not accelerating towards the ground. This results in a reduced oscillatory behavior with the RNN_3 controller.

5.6.3. CTRNN

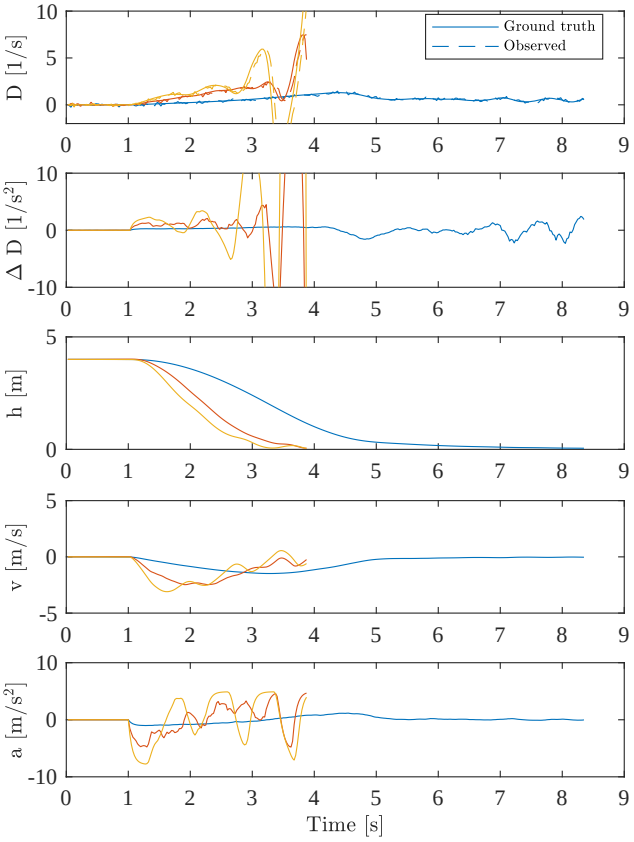


Figure 5.9: Vehicle states and observations from simulation with $CTRNN_1$ (blue), $CTRNN_2$ (red) and $CTRNN_3$ (yellow).

The three CTRNN neurocontrollers selected from the Pareto front in Fig. 5.9 and Fig. 5.19 all show variations on a similar control scheme. Effectively, these controllers split the control scheme into 4 segments, depending on the sign combina-

tion of the inputs D and ΔD. When descending with an increasing divergence, the vehicle will try to decelerate. When descending with an decreasing divergence, the vehicle will decelerate less aggressively. When ascending with increasing rate, the vehicle will reduce thrust. Finally, when the vehicle is descending with a negative divergence rate, the vehicle will reduce thrust less aggressively than the increasing rate case. This results in high speed yet smooth landings with little oscillation. This approach is similar to that shown by RNN_3. Portions of these controllers seem discretized, but as they only show the steady state throttle response, the temporal response may be more smooth.

5.7. Reality Gap

Everything discussed so far has been in a simulated environment, one that is a significant simplification of reality to facilitate high speed evaluation of the neurocontrollers. As we move to the real world, we can therefore expect various differences leading to a reality gap. This section aims to identify some of these differences.

Let us first look at the control of the vehicle, in simulation, the output of the neurocontroller was acceleration, which after being fed through a low pass filter to simulate the spin-up of the rotors was implemented by the simulated rotors. In reality, this desired acceleration must first be converted to a thrust command for the rotors. If we use a naive approach here, we can simply determine a linear transform from desired acceleration to thrust, the results of which are shown in the top plot of Fig. 5.10. This figure shows a set of real world neurocontroller landings using a constant scaling factor for desired acceleration to thrust. As the vehicle starts to move the command tracking is good but as it starts to descend the tracking degrades. This is almost certainly due, in part, to the unmodeled drag and non-linear aerodynamic effects of descending through the downwash of the propellers.

This poor tracking leads to a noticeable reality gap in the landing performance. This can be reduced with the use of a closed loop controller as proposed in [154], instead of a linear transform. The results using a Proportional-Integral (PI) controller, minimizing the error between the commanded and measured acceleration on the vehicle, are shown in the bottom plot of Fig. 5.10. The controller effectively abstracts away from the raw motor commands to a desired acceleration, which significantly improves the tracking performance allowing us to cross this reality gap.

The goal of this paper is to highlight the use of abstracted inputs to improve the robustness of optimized policies to differences in the input. To do this we will investigate the performance of the vehicle with the use of two different types of cameras with significantly different divergence signal output performance characteristics.

The first camera uses the bottom looking CMOS camera built into the Parrot Bebop 2 using the size divergence estimation method described in [85]. The second camera is the Insightness SEEM1 Dynamic Vision Sensor (DVS) using the efficient plane fitting optical flow estimation technique described in [142]. This event based

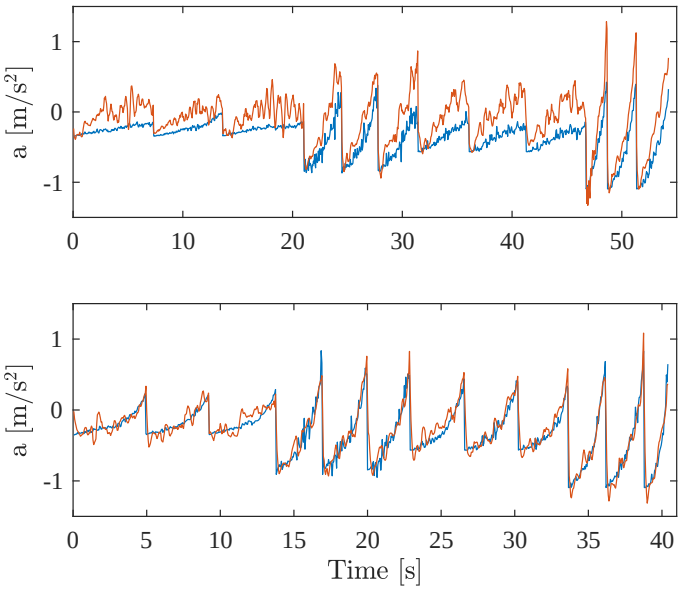


Figure 5.10: Thrust command (blue) and subsequent vehicle acceleration (red) for some real world landings. (Top) Unmodeled dynamics such as drag and other non-linear aerodynamic effects lead to a poor thrust command tracking using a naive approach. (Bottom) The tracking error can be substantially reduced with the use of a simple closed loop PI controller.

camera does not generate frames as a conventional image sensor but rather measures logarithmic light changes at each pixel independently and asynchronously. This makes the sensor conditioned to operate with high speed motion with low latency and relatively low data throughput. This type of camera has been previously used to facilitate high speed landings as shown in [142]. A schematic showing the optical flow processing pipeline for the CMOS and event-camera can be found in Fig. 5.11.

Fig. 5.12 shows a comparison of the divergence estimation error from these two cameras highlighting how different the output of these camera is and how different they both are to the camera statistics used in simulation. Some additional differences in the camera properties are summarized in Table 5.2. We will investigate in the next chapter if these differences result in a significant reality gap.

Table 5.2: Camera properties

Property	Simulated	CMOS	DVS
Sensor	-	mt9v117	SEES1
Resolution used	-	240 x 240	262 x 262
Field of View	-	58°	79°
Divergence Rate	[30, 50] Hz	45 Hz	100 Hz

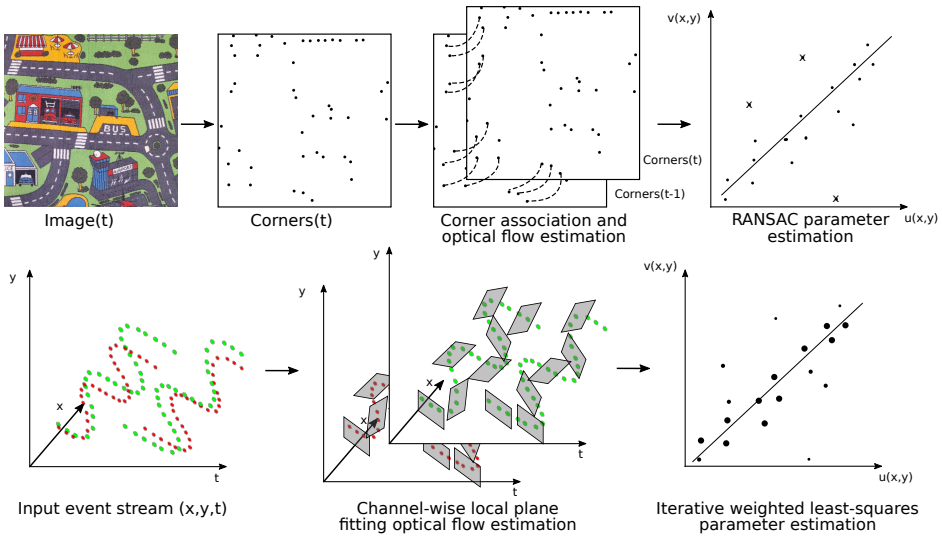


Figure 5.11: Process schematic of the optical flow computation for the CMOS camera (top) and the event-camera (below).

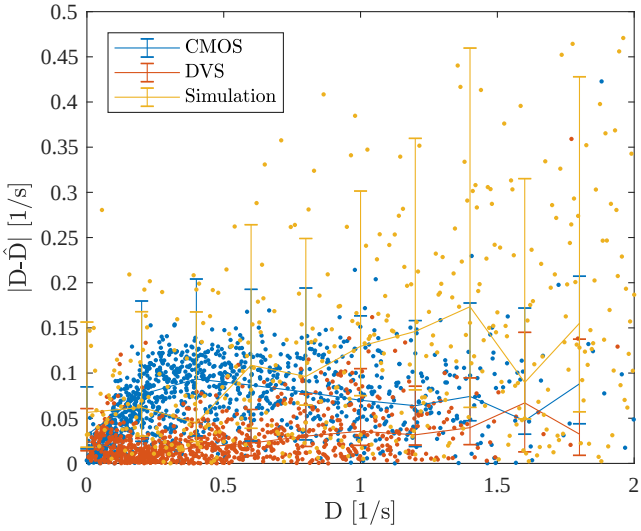


Figure 5.12: Divergence estimation error for the CMOS (blue) and DVS (red) cameras. The trend and distribution of these two cameras are quite different.

5.8. FLIGHT TEST RESULTS

Using the closed loop PI controller to control the thrust as described in the previous section, we performed a set of landings with some of the neurocontrollers identified

in Section 5.5. All flights were initiated from a steady hover at an altitude of 4 m. Fig. 5.13 shows the results from the controllers NN_1 and NN_2, Fig. 5.14 shows the results from RNN_1 and RNN_2 and Fig. 5.15 shows the results from $CTRNN_1$ and $CTRNN_2$. These results are plotted for both the CMOS and DVS cameras to see how well these two sensors affect the landing profile. The results from simulation have also been plotted to see how well the real world performance fits with the simulated, a measure for the eventual reality gap. NN_3, RNN_3 and $CTRNN_3$ were not tested in reality as their relatively high touchdown velocity in simulation may cause damage to our real world vehicle. This is in-fact a benefit of the multi-objective optimization scheme used here, the user can simply choose a policy that performs the trade-off of the fitness functions as desired.

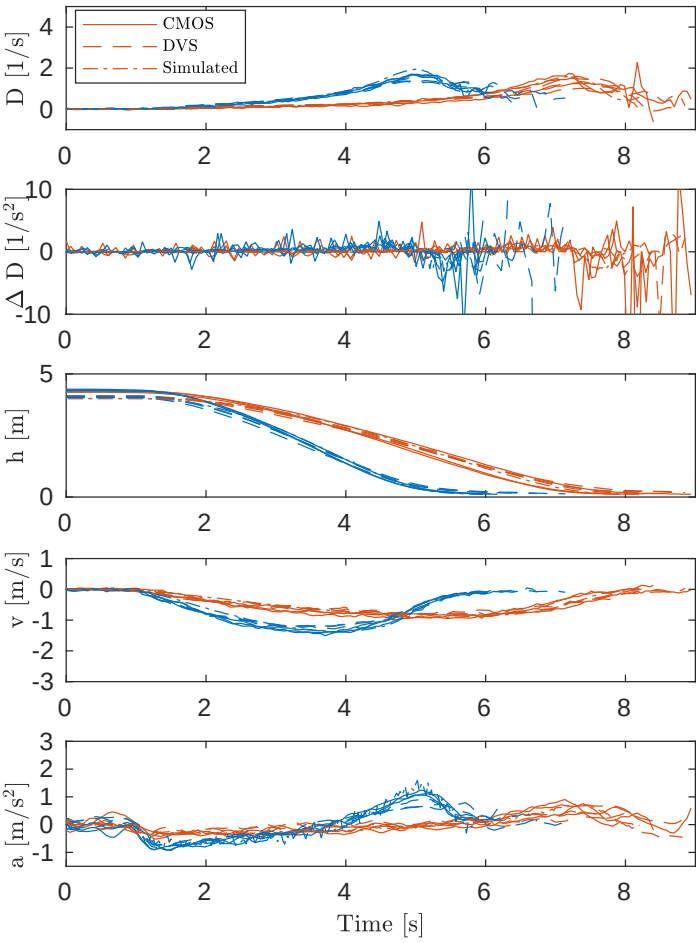


Figure 5.13: Vehicle states and observations from real world flights with the NN_1 (blue) and NN_2 (red) controllers. The results using the CMOS camera is shown in solid and the DVS in dashed. The simulated performance is also plotted in dot-dash for comparison.

In spite of the differences in the generation of the divergence, the landing performance is very similar. These two systems are also so similar to the simulated landing that the plot is hardly visible. This would suggest that the eventual reality gap is small despite significant differences in the way the input was generated.

Also notable is the repeatability of the landing maneuvers. The landings were performed three times each and each landing resulted in very similar trajectories. This shows that the evolutionary optimization converged to a robust solution as suggested by the analysis in Section 5.5.

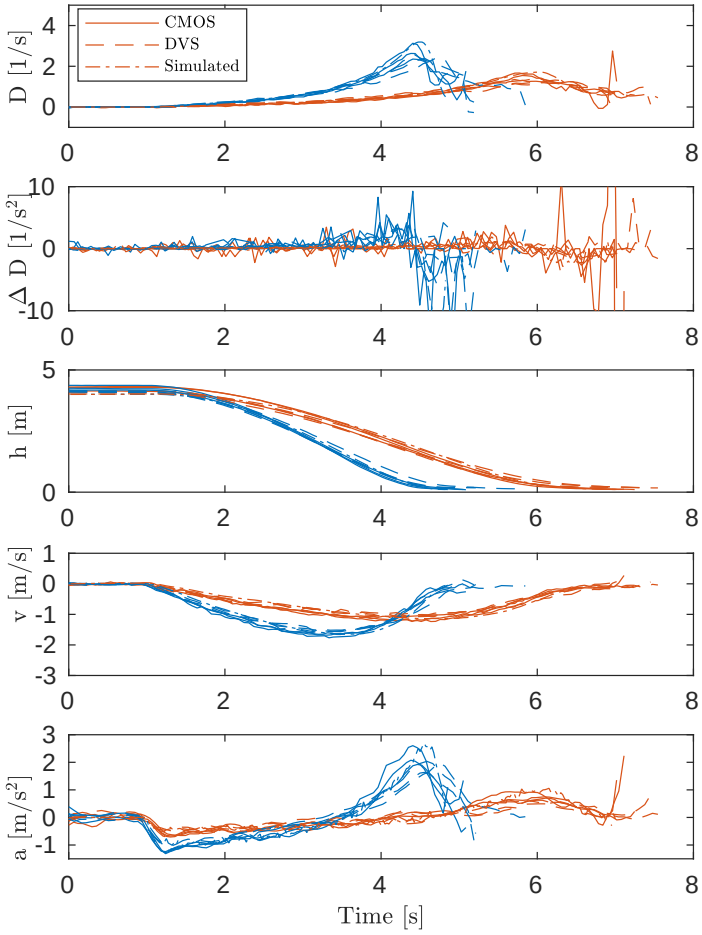


Figure 5.14: Vehicle states and observations from real world flights with the RNN_1 (blue) and RNN_2 (red). The results using the CMOS camera is shown in solid and the DVS in dashed. The simulated performance is also plotted in dot-dash for comparison.

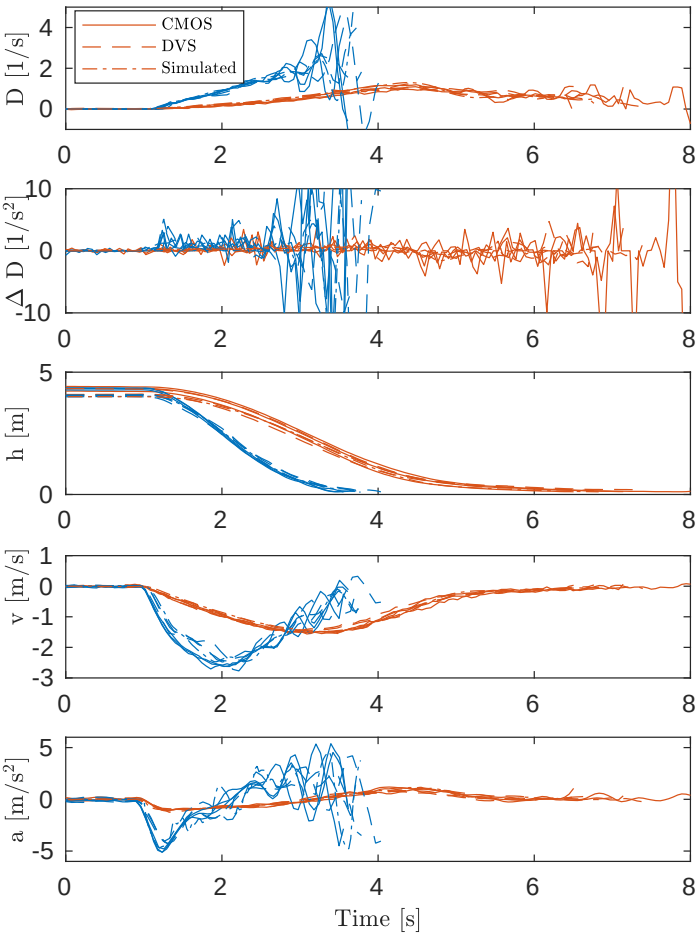


Figure 5.15: Vehicle states and observations from real world flights with the $CTRNN_1$ (blue) and $CTRNN_2$ (red). The results using the CMOS camera is shown in solid and the DVS in dashed. The simulated performance is also plotted in dot-dash for comparison.

5.9. CONCLUSION

This paper investigated the influence of abstraction of the sensory input to the reality gap for automatically optimized UAV agents tasked with performing quick yet safe landing. We have shown over multiple evolutionary runs and neurocontroller architectures, that abstraction does not unduly hamper the optimization power of the optimization as the agents developed a robust and effective method to land.

The optimized agents showed some landing strategies that were before not imagined by the human designers. One notable strategy is that instead of a simple proportional controller for the entire state space, an asymmetric response may be

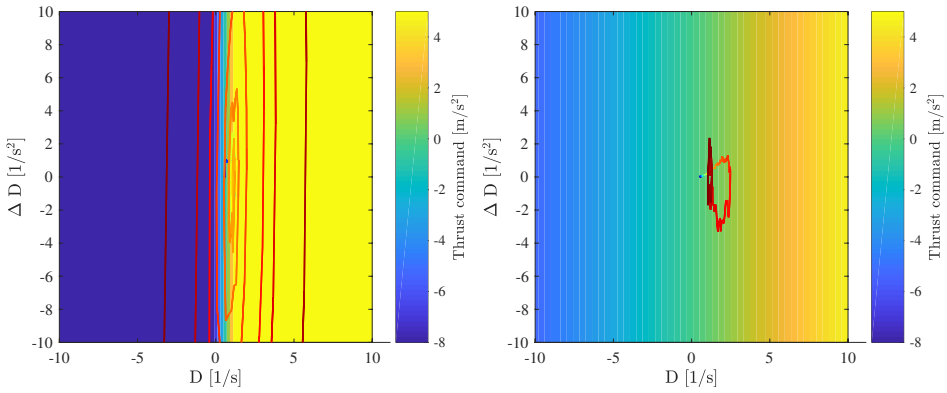
more appropriate to delay the onset of oscillations when performing the landing procedure. A strong response when the divergence error is positive and a weaker response when negative seems a good approach.

Tests in the real world showed the presence of significant differences between simulation and reality. The most significant was that the acceleration command tracking performance was poor, likely due to the drag and other non-linear aerodynamic effects which were not considered in simulation due to their modeling complexity. The resultant reality gap was crossed with the use of a closed loop controller representing another layer of abstraction, from the low-level raw motor control values to a desired vertical acceleration, which ensures robustness to the real world uncertainty.

Finally, we showed that abstraction on the sensory input of the neurocontroller was robust to the reality gap when using two different input estimation techniques. Although two cameras with different imaging techniques were used, the resultant landing profile was very similar. Abstraction can therefore be a powerful tool when crossing the reality gap.

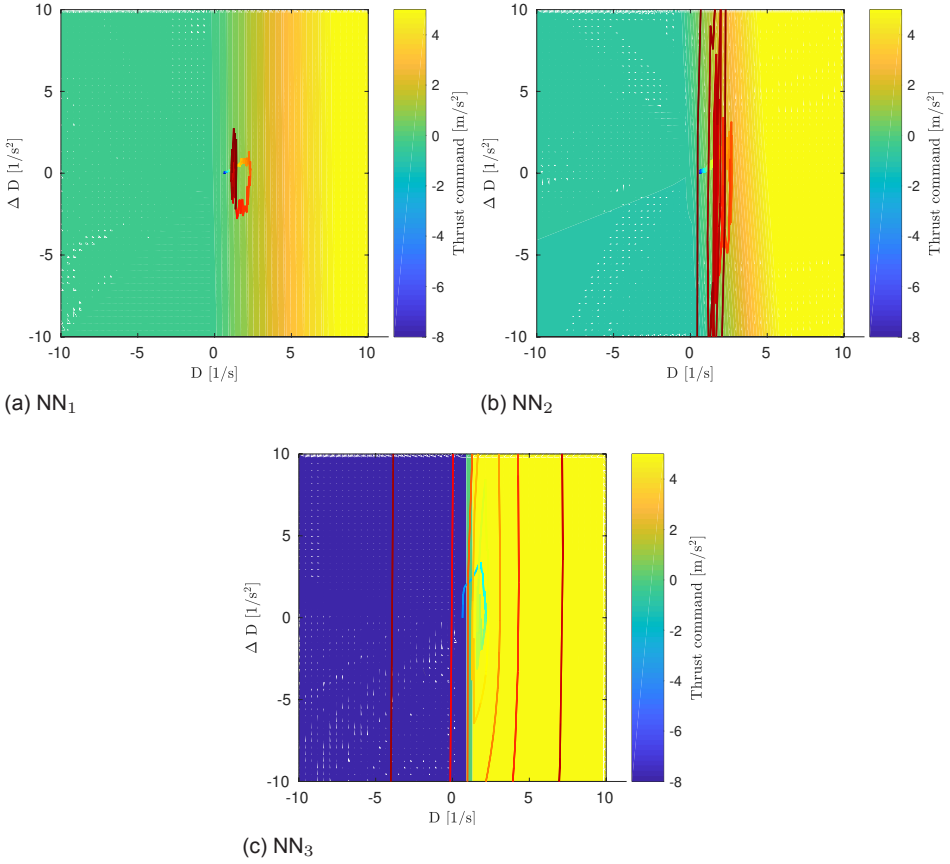
5

Additional Material: Controller Steady State Input-Output Mappings



(a) Controller C_1 (b) Controller C_2

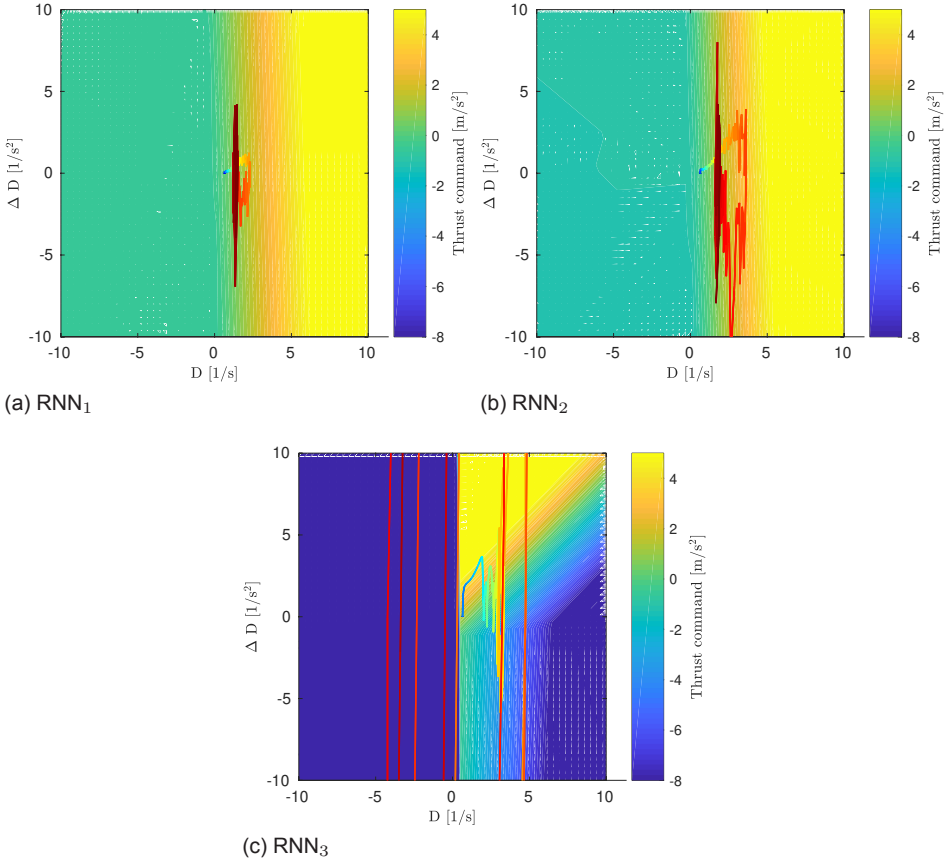
Figure 5.16: Steady state input-output mapping for hand designed controllers.



(a) NN_1 (b) NN_2

(c) NN_3

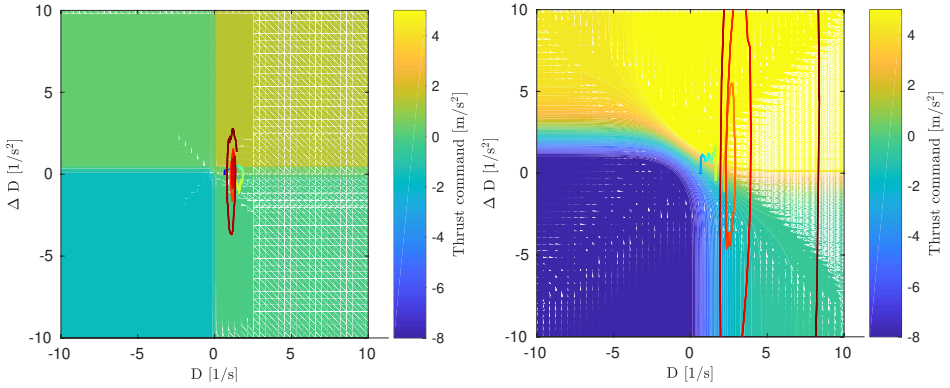
Figure 5.17: Steady state input-output mapping for NN controllers.



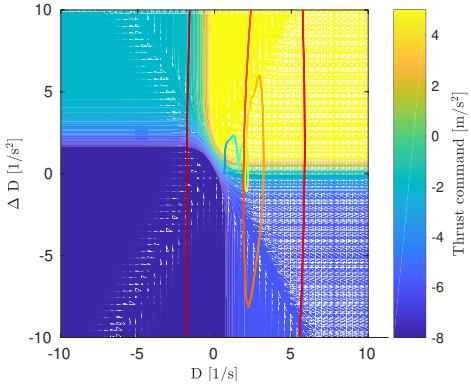
(a) RNN_1 (b) RNN_2

(c) RNN_3

Figure 5.18: Steady state input-output mapping for NN controllers.



(a) CTRNN$_1$ (b) CTRNN$_2$



(c) CTRNN$_3$

Figure 5.19: Steady state input-output mapping for NN controllers.

6

Conclusion

In this final chapter, we will first answer the research questions and problem statement laid out in the introduction. Next we reflect on how the work presented here can potentially affect the wider scientific research field. Finally, we summarize some ideas for future work.

6.1. Answers to Research Questions

RQ1: *How can abstraction in the behavioral representation help in crossing the reality gap?*

Chapter 2 investigated the use of a more intelligible behavioral representation for the robotic behavior than the commonly used neural network structure. This was tested by optimizing behavior for the 20 g DelFly Explorer flapping wing MAV using a Behavior Tree representational structure. The DelFly was tasked to fly around a small room, avoiding the walls while searching for and eventually flying through an open window. After being trained in simulation with success rates above 85%, there was a clear reality gap when testing on the real vehicle with a 0% success rate. The intelligible nature of the Behavior Tree allowed the human operator to quickly understand the desired robotic behavior and identify the source of the reality gap, in this case differences in the sensing and actuation of the real world vehicle. The structure of the Behavior Tree also easily lends itself to simple augmentation, allowing the user to update the behavior actively reducing the reality gap improving performance to 54%, higher than the baseline user-designed behavior at 46%. The resultant difference in performance was likely due to unmodeled aerodynamics effects and drafts.

The Behavior Tree achieves this increased intelligibility at the cost of representational power as compared to the neural network. With its subsumption-like architec-

ture, the behavior abstracts away from a global input-output mapping, segmenting the actions to a finite set of options. For the behavior to remain intelligible, the trees must remain small, somewhat limiting the ability of the evolutionary optimization to find the ideal solution to the problem. That said, without the intelligibility of this representational structure the optimized behavior would have been useless once transferred. This trade-off therefore seems worthwhile. Abstracting the behavioral representation can help to speed-up the re-optimization process by segmenting the behavior to simpler sub-behaviors.

RQ2: *How can abstraction in the control output be used to improve the robustness of a robot to differences between simulations and reality?*

Chapter 3 presented the optimization of two different neurocontrollers tasked to control a swarm of three quadcopters to form a predetermined pattern. Each quadcopter had the same neurocontroller making the swarm homogeneous, a non-trivial optimization task. The two different neurocontrollers optimized, differed in how they controlled the quadcopter. One controller directly controlled the rotor speed of the vehicle whilst the other controlled the velocity of the vehicle with the use of traditional nested closed-loop controllers to eventually actuate the rotors.

After optimization in simulation, both controllers were able to robustly achieve the desired swarming formation, however, when transferred to the real world, only the abstracted controller was successful. So successful in-fact, that with no augmentation, the real world flight behavior was near identical to that observed in simulation. The use of closed-loop controllers helps to increase the robustness of the behavior, actively rejecting environmental disturbances and adjusting for any unique vehicle bias due to manufacturing and operational differences.

Additionally, the abstraction allowed the simulation to be significantly sped-up. The abstracted neurocontroller only required a simple velocity model representing the response of the closed-loop controller whilst the non-abstracted neurocontroller required a significantly higher fidelity aerodynamic model.

Perhaps most significantly, this work shows that when done well, abstracting away from the low level details of the real world do not unduly limit the optimization scheme from finding a valid solution to the problem. The simplification of the task even reduces the computation required to find the optimal solution. This is a powerful tool to improve the robustness of robotic behavior developed in simulation to the reality gap.

RQ3: *How can abstraction on the sensory input be used to reduce the sensitivity of the robotic behavior to the reality gap?*

Chapter 4 and Chapter 5 presented the results of experiments testing the impact of abstraction of the inputs to the robot on the inevitable reality gap experienced. Here, a quadcopter was required to perform a high speed landing using an abstracted

input from a visual camera, namely the divergence of the optical flow field. This abstracted input gives the user the freedom to use any sensor that can generate this signal, reducing the dependence of the robotic behavior to the input used.

This was demonstrated by performing real world experiments using a conventional CMOS camera and the novel event-based Dynamic Vision Sensor. These two systems produced significantly different input signal statistics than that used to optimize the robotic behavior in simulation. Nevertheless, the resultant landing behavior of the real world vehicle was almost identical to that seen in simulation. This again demonstrates how abstraction provides the system developer with tools to scale the robotic application to various vehicles with different sensor solutions. Additionally, it improves the robustness of the optimized behavior to eventual environmental uncertainties. Notably, Chapter 5 also extended the investigation of the transfer problem to not just the reality gap by testing the solution with two different real world sensors. This highlights that this approach is more broadly applicable than just the reality gap.

6.2. Answer to Problem Statement

Problem Statement: *How can abstraction be used to bridge the reality gap in evolutionary robotics?*

All the results in this work have shown that abstraction can indeed be used as a tool to balance the implicit trade-off between optimization power and robust transfer from simulation to the real world. With this tool, the user is empowered to effectively control the reality gap. With limited reduction in the optimization power, robotic behavior can be developed in simulation and effectively transferred to the real world. This principle builds upon the envelope of noise presented by Jakobi. Similarly, we mask the details of the real world from the optimization, encouraging the development of robust control solutions, but we additionally ensure that the real world performance closely resembles that in simulation to ensure a small reality gap. We are essentially changing both simulation and reality to be more like each other, reducing the behavioral reality gap. In combination with a shift to a higher level of abstraction, this results in very robust behavioral development with a small reality gap. We effectively change the goal of the transfer problem to focus on the behavior rather than on performance.

6.3. Discussion

Advancements in the evolutionary robotics field have been relatively slow, due in large part to the reality gap. This was a constant impediment to implementing the approach of simulation-based robotic optimization with transfer to the real world. Even with compelling innovations in bridging this gap [91, 102], evolution in embodied agents directly in the real world [54], evolution of mind and body [138], and

smart adaptation on the real world vehicles [40], it was still difficult moving to more complex tasks. The abstraction techniques presented in this thesis should help both to simplify the optimization problem and make transfer to the real world more robust. This will eventually lead to the ability to develop ever more complex behaviors in complex environments.

Recently, the field of deep learning has seen a large surge in interest and development due to a break-through in the neural structure, allowing more complex information to be learned. This has for the most part been restricted to virtual agents. The few applications with embodied agents have been plagued with the reality gap [21, 30, 164]. The work presented in this thesis is not limited to ER but can also be applied to reinforcement learning. The expansion to a wider field of research will undoubtedly accelerate development, suggesting that major improvements are just around the corner.

6.4. FUTURE WORK

This thesis investigated abstraction on different aspects of the robotic platform with quite significant results but was limited to only three demonstrations of its efficacy. More work is needed to understand what aspects of abstraction are the most effective to improve robustness and what aspects are the most detrimental to the optimization process. This type of research would better inform the roboticist as to the effects of the trade-offs being made by selecting the level of abstraction.

Little adaptation was done to the behavior after transfer in this thesis. Although some behavioral augmentation was manually done by the human user in Chapter 2, this process can be automated. In fact, the abstracted behavioral architecture should aid in segmenting the re-optimization problem from a large global optimization to several smaller, simpler ones, potentially significantly speeding up the re-optimization process. Some early work has been done to achieve this automated re-optimization in the real world with the abstracted behavior [105] but more investigation is required.

As abstraction helps to segment and compartmentalize the behavior and reality gap, it should simplify the optimization while improving robustness. This should allow more complex behavior to be developed in simulation and successfully transferred to the real world. This hypothesis should be tested by actually applying the abstraction techniques presented to learn even more complex behavior.

Additionally, this segmentation of the optimization task can lead to significant reduction in the time to optimize a given task. This may aid the field of embodied evolutionary optimization. Abstraction may help speed up the development time making this approach more feasible.

References

[1] M. T. Alkowatly, V. M. Becerra, and W. Holderbaum. Bioinspired Autonomous Visual Vertical Control of a Quadrotor Unmanned Aerial Vehicle. *Journal of Guidance, Control, and Dynamics*, 38(2):249–262, 2015.

[2] M. H. J. Amelink, M. Mulder, and M. M. van Passen. Designing for Human-Automation Interaction: Abstraction-Sophistication Analysis for UAV Control. In *International MultiConference of Engineers and Computer Scientists*, volume I, 318–323, Hong Kong, Mar. 2008. IAE.

[3] P. J. Angeline. Subtree Crossover: Building Block Engine or Macromutation. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 9–17, Stanford University, CA, July 1997. Morgan Kaufmann.

[4] S. Ansari, R. Żbikowski, and K. Knowles. Aerodynamic modelling of insect-like flapping flight for micro air vehicles. *Progress in Aerospace Sciences*, 42(2):129–172, 2006.

[5] S. F. Armanini, C. C. D. Visser, G. C. H. E. de Croon, and M. Mulder. Time-varying model identification of a flapping-wing micro aerial vehicle using flight data. *Journal of Guidance, Control, and Dynamics*, 1–33, 2015.

[6] I. L. Ashkenas. Collected flight simulation and simulator induced sickness. *AGARD Flight Simulation*, 1986.

[7] E. Baird, N. Boeddekerb, M. R. Ibbotsonc, and M. V. Srinivasan. A universal strategy for visually guided landing. *Proceedings of the National Academy of Sciences*, 110(46):18686–18691, 2013.

[8] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[9] M. Bangura and R. Mahony. Nonlinear Dynamic Modeling for High Performance Control of a Quadrotor. *Australasian Conference on Robotics and Automation (ACRA 2012)*, 1–10, 2012.

[10] P. Bardow, A. J. Davison, and S. Leutenegger. Simultaneous Optical Flow and Intensity Estimation from an Event Camera. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 884–892. IEEE, 2016.

[11] F. Barranco, C. Fermuller, and Y. Aloimonos. Contour Motion Estimation for Asynchronous Event-Driven Cameras. *Proceedings of the IEEE*, 102(10):1537–1556, 2014.

[12] F. Barranco, C. Fermuller, and Y. Aloimonos. Bio-inspired Motion Estimation with Event-Driven Sensors. In *Advances in Computational Intelligence*, 309–321. Springer International Publishing, 2015.

[13] F. Barranco, C. Fermuller, Y. Aloimonos, and T. Delbrück. A Dataset for Visual Navigation with Neuromorphic Methods. *Frontiers in Neuroscience*, 10(February):1–9, 2016.

[14] R. D. Beer. On the Dynamics of Small Continuous-Time Recurrent Neural Networks. *Adaptive Behavior*, 3(4), 1995.

[15] R. D. Beer. Toward the evolution of dynamical neural networks for minimally cognitive behavior. In *From Animals to Animats 4: Proceedings of the 4th International Conference of Simulation of Adaptive Behavior (SAB1996)*, 421–429. Bradford Books, 1996.

[16] R. B. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE transactions on neural networks and learning systems*, 25(2):407–417, 2014.

[17] R. B. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. V. Srinivasan. Asynchronous frameless event-based optical flow. *Neural Networks*, 27:32–37, 2012.

[18] A. Boeing and T. Bräunl. Leveraging multiple simulators for crossing the reality gap. *2012 12th International Conference on Control, Automation, Robotics and Vision, ICARCV 2012*, 2012(December):1113–1119, 2012.

[19] J. C. Bongard. Evolutionary Robotics. *Communications of the ACM*, 56(8):74–83, 2013.

[20] J. C. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–21, 2006.

[21] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 4243–4250, 2017.

[22] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbrück. A 240x180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[23] T. Brosch, S. Tschechne, and H. Neumann. On event-based optical flow detection. *Frontiers in Neuroscience*, 9(137):1–15, Apr. 2015.

[24] J. V. Caetano, J. Verboom, C. C. de Visser, G. C. H. E. de Croon, B. D. W. Remes, C. De Wagter, and M. Mulder. Linear Aerodynamic Model Identification of a Flapping Wing MAV Based on Flight Test Data. *International Journal of Micro Air Vehicles*, 5(4):273–286, 2013.

[25] K. Čapek. R.U.R translated by Paul Selver and Nigel Playfair, 2001.

[26] G. Capi and K. Doya. Evolution of recurrent neural controllers using an extended parallel genetic algorithm. *Robotics and Autonomous Systems*, 52(2-3):148–159, 2005.

[27] A. J. Champandard. Behavior Trees for Next-Gen Game AI. In *Game Developers Conference '07*, 1–96, San Francisco, CA, 2007. GDC.

[28] H. J. Chiel and R. D. Beer. The brain has a body: Adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neurosciences*, 20(12):553–557, 1997.

[29] D.-i. Cho and T.-j. Lee. A Review of Bioinspired Vision Sensors and Their Applications. *Sensors and Materials*, 27(6):447–463, 2015.

[30] K. Choromanski, A. Iscen, V. Sindhwani, J. Tan, and E. Coumans. Optimizing Simulations with Noise-Tolerant Structured Exploration. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2970–2977, 2018.

[31] X. Clady, C. Clercq, S.-H. Ieng, F. Houseini, M. Randazzo, L. Natale, C. Bartolozzi, and R. B. Benosman. Asynchronous visual event-based time-to-contact. *Frontiers in neuroscience*, 8(9), 2014.

[32] X. Clady, S.-H. Ieng, and R. B. Benosman. Asynchronous event-based corner detection and matching. *Neural Networks*, 66:91–106, 2015.

[33] A. Clark. *Being There: Putting Brain, Body, and World Together Again*. MIT Press, 1998.

[34] A. Clarke and C. Thornton. Trading Spaces: computation, representation, and the limits of uninformed learning. *The Behavoral and Brain Sciences*, 20(1):57–66, 1997.

[35] M. Colledanchise and P. Ögren. How Behavior Trees modularize robustness and safety in hybrid systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1482–1488, Chicago, IL, 2014. IEEE.

[36] J. Conradt. On-Board Real-Time Optic-Flow for Miniature Event-Based Vision Sensors. In *IEEE International Conference on Robotics and Biomimetics*, Zhuhai, China, 2015.

[37] J. Conradt, R. Berner, M. Cook, and T. Delbrück. An embedded AER dynamic vision sensor for low-latency pole balancing. In *IEEE 12th International Conference on Computer Vision Workshops*, 780–785, 2009.

[38] M. Cook, L. Gugelmann, F. Jug, C. Krautz, and A. Steger. Interacting maps for fast visual interpretation. *Proceedings of the International Joint Conference on Neural Networks*, 770–776, 2011.

[39] F. C. Crow. Summed-Area Tables for Texture Mapping. *SIGGRAPH Computer Graphics*, 18(3):207–212, July 1984.

[40] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, May 2015.

[41] G. C. H. E. de Croon. Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. *Bioinspiration & Biomimetics*, 11(1):1–30, 2016.

[42] G. C. H. E. de Croon, L. M. O. Connor, C. Nicol, and D. Izzo. Evolutionary Robotics Approach to Odor Source Localization. *Neurocomputing*, 121(1):481–497, Dec. 2013.

[43] G. C. H. E. de Croon, K. M. E. de Clercq, R. Ruijsink, B. D. W. Remes, and C. De Wagter. Design, aerodynamics, and vision-based control of the DelFly. *International Journal of Micro Air Vehicles*, 1(2):71–97, 2009.

[44] G. C. H. E. de Croon, H. W. Ho, C. De Wagter, E.-J. van Kampen, B. D. W. Remes, and Q. P. Chu. Optic-flow based slope estimation for autonomous landing. *International Journal of Micro Air Vehicles*, 5(4):287–297, 2013.

[45] C. De Wagter, A. A. Proctor, and E. N. Johnson. Vision-Only Aircraft Flight Control. In *Digital Avionics Systems Conference*, Indianapolis, IN, oct 2003. IEEE.

[46] C. De Wagter, S. Tijmons, B. D. W. Remes, and G. C. H. E. de Croon. Autonomous Flight of a 20-gram Flapping Wing MAV with a 4-gram Onboard Stereo Vision System. In *International Conference on Robotics and Automation*, 4982–4987, Hong Kong, June 2014. IEEE.

[47] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[48] T. Delbrück and M. Lang. Robotic goalie with 3 ms reaction time at 4{%} CPU load using event-based dynamic vision sensor. *Frontiers in Neuroscience*, 7(223):1–7, 2013.

[49] J. Dewey. The reflex arc concept in psychology. *The Psychological Review*, 3(4):357–370, 1896.

[50] T. G. Dietterich, E. B. Kong, and D. Hall. Machine Learning Bias , Statistical Bias , and Statistical Variance of Decision Tree Algorithms. Technical report, Department of Computer Science, Oregon State University, Corvallis, OR, 1995.

[51] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. Eiben. Evolutionary Robotics: What, Why, and Where to. *Frontiers in Robotics and AI*, 2(March):1–18, 2015.

[52] R. G. Dromey. From Requirements to Design: Formalizing the Key Steps. In *First International Conference on Software Engineering and Formal Methods*, 2–11, Brisbane, Sept. 2003. IEEE.

[53] M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen. Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLoS ONE*, 11(3):1–25, 2016.

[54] A. E. Eiben. In Vivo Veritas: towards the Evolution of Things. *Proceedings of PPSN XIII*, 24–39, 2014.

[55] A. E. Eiben. EvoSphere: The world of robot evolution. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9477:3–19, 2015.

[56] A. E. Eiben, S. Kernbach, and E. Haasdijk. Embodied artificial evolution: Artificial evolutionary systems in the 21st Century. *Evolutionary Intelligence*, 5(4):261–272, 2012.

[57] A. E. Eiben, P.-E. Raue, and Z. Ruttkay. Genetic Algorithms with Multi-Parent Recombination. In *Parallel Problem Solving from Nature*, 78–87, Jerusalem, Israel, oct 1994. Springer Berlin Heidelberg.

[58] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg, 2nd Edition, 2015.

[59] K. Erol, J. Hendler, and D. S. Nau. HTN Planning: Complexity and Expressivity. In *Twelfth National Conference on Artificial Intelligence*, 1123–1128, Menlo Park, CA, 1994. AAIA Press.

[60] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from Architectures to Learning. *Evolutionary Intelligence*, 1(1):47–62, Jan. 2008.

[61] D. Floreano and F. Mondada. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven robot. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, 421–430, Cambridge, MA, 1994. MIT Press.

[62] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(3):396–407, 1996.

[63] D. Floreano, R. Pericet-Camara, S. Viollet, F. Ruffier, A. Brückner, R. Leitel, W. Buss, M. Menouni, F. Expert, R. Juston, M. K. Dobrzynski, G. L'Eplattenier, F. Recktenwald, H. A. Mallot, and N. Franceschini. Miniature curved artificial compound eyes. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 110, 9267–9272, 2013.

[64] D. Floreano and J. Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13:431–443, 2000.

[65] D. Floreano and R. J. Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460–466, 2015.

[66] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[67] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014.

[68] J. J. Gibson. *The perception of the visual world*. Boston, Houghton Mifflin, 1950.

[69] J. J. Gibson. *The ecological approach to visual perception*. Lawrence Erlbaum Associates, London, 1979.

[70] O. Gigliotta and S. Nolfi. Formation of spatial representations in evolving autonomous robots. *Proceedings of the 2007 IEEE Symposium on Artificial Life, CI-ALife 2007*, 171–178, 2007.

[71] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing, Boston, MA, 1989.

[72] J. Gomes, P. Urbano, and A. L. Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2-3):115–144, 2013.

[73] C. Hartland and N. Bredèche. Evolutionary Robotics, Anticipation and the Reality gap. In *Robotics and Biomimetics*, volume 2006, 1640–1645, Kunming, China, Dec. 2006. IEEE.

[74] I. Harvey. Evolutionary robotics and SAGA: The Case for Hill Crawling and Turnament Selection. *Artificial Life III*, XVI:229–326, 1993.

[75] I. Harvey, E. Di Paolo, R. Wood, M. Quinn, and E. Tuci. Evolutionary robotics: A new scientific tool for studying cognition. *Artificial life*, 11(1-2):79–98, 2005.

[76] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior from Animals to Animats 3*, 392–401, Cambridge, MA, Aug. 1994. MIT Press Bradford Books.

[77] G. Hattenberger, M. Bronz, and M. Gorraz. Using the Paparazzi UAV System for Scientific Research. In *International Micro Air Vehicle Conference and Competition 2014*, 247–252, Delft, Netherlands, 2014. IMAV.

[78] M. T. Heath. *Scientific Computation: an introductory survey*. McGraw-Hill, New York, 2002.

[79] F. W. P. Heckel, G. M. Youngblood, and N. S. Ketkar. Representational Complexity of Reactive Agents. In *Computational Intelligence and Games*, 257–264, Dublin, Aug. 2010. IEEE.

[80] R. K. Heffley and T. M. Schulman. Derivation of Human Pilot Control Laws Based on Literal Interpretation of Pilot Training Literature. Albuquerque, New Mexico, Aug. 1981.

[81] B. Hérissé, T. Hamel, R. Mahony, and F.-X. Russotto. Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow. *IEEE Transactions on Robotics*, 28(1):77–89, 2012.

[82] B. Hérissé, F. X. Russotto, T. Hamel, and R. Mahony. Hovering flight and vertical landing control of a VTOL Unmanned Aerial Vehicle using optical flow. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 801–806, 2008.

[83] F. S. Hiller and G. J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, New York, 9th Edition, 2010.

[84] H. W. Ho and G. C. H. E. de Croon. Characterization of Flow Field Divergence for MAVs Vertical Control Landing. In *AIAA Guidance, Navigation, and Control Conference*, 1–13, 2016.

[85] H. W. Ho, G. C. H. E. de Croon, E.-J. van Kampen, Q. P. Chu, and M. Mulder. Adaptive Gain Control Strategy for Constant Optical Flow Divergence Landing. *IEEE Transactions on Robotics*, 34(2):508–516, 2018.

[86] D. Howard, A. E. Eiben, D. F. Kennedy, J.-B. Mouret, P. Valencia, and D. Winkler. Evolving embodied intelligence from materials to machines. *Nature Machine Intelligence*, 1(1):12–19, 2019.

[87] D. Howard and F. Kendoul. Towards Evolved Time to Contact Neurocontrollers for Quadcopters. In *Australasian Conference on Artificial Life and Computational Intelligence (ACALCI)*, 336–347, 2016.

[88] D. Izzo and L. Pettazzi. Autonomous and Distributed Motion Planning for Satellite Swarm. *Journal of Guidance, Control, and Dynamics*, 30(2):449–459, 2007.

[89] D. Izzo, L. F. Simões, and G. C. H. E. de Croon. An evolutionary robotics approach for the distributed control of satellite formations. *Evolutionary Intelligence*, 7(2):107–118, 2014.

[90] N. Jakobi. Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis. *Adaptive Behaviour*, 6(2):325–368, 1997.

[91] N. Jakobi. *Minimal Simulations For Evolutionary Robotics*. PhD thesis, University of Sussex, 1998.

[92] N. Jakobi, P. Husbands, and I. Harvey. Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In *Advances in Artificial Life*, 704–720, Granada, June 1995. Springer Berlin Heidelberg.

[93] M. J. Jones and P. Viola. Robust Real-Time Object Detection. *International Journal of Computer Vision*, 57(2):137–154, 2001.

[94] R. C. Julian, C. J. Rose, H. Hu, and R. S. Fearing. Cooperative Control and Modeling for Narrow Passage Traversal with an Ornithopter MAV and Lightweight Ground Station. In *International conference on Autonomous agents and multi-agent systems*, 103–110, St. Paul, MN, May 2013. IFAA-MAS.

[95] M. Karásek, F. T. Muijres, C. De Wagter, B. D. W. Remes, and G. C. H. E. de Croon. A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns. *Science*, 361(6407):1089–1094, 2018.

[96] F. Kendoul. Four-dimensional guidance and control of movement using time-to-contact: Application to automated docking and landing of unmanned rotorcraft systems. *International Journal of Robotics Research*, 33(2):237–267, 2014.

[97] H. Kim. Simultaneous Mosaicing and Tracking with an Event Camera. In *Proceedings of the British Machine Vision Conference*, 2014.

[98] H. Kim, S. Leutenegger, and A. J. Davison. Real-time 3D reconstruction and 6-DoF tracking with an event camera. In *European Conference on Computer Vision*, 2016.

[99] A. Klöckner. Interfacing Behavior Trees with the World Using Description Logic. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, 57–68, Boston , MA, Sept. 2013. American Institute of Aeronautics and Astronautics.

[100] L. König, S. Mostaghim, H. Schmeck, L. Konig, S. Mostaghim, and H. Schmeck. Decentralized Evolution of Robotic Behavior using Finite State Machines. *International Journal of Intellegent Computing and Cybernetics*, 2(4):695–723, 2009.

[101] S. Koos, J.-B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, 119, 2010.

[102] S. Koos, J.-B. Mouret, and S. Doncieux. The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics. *Transactions on Evolutionary Computation*, 17(1):122–145, 2013.

[103] J. R. Koza. Genetic Programming as a Means for Programming Computers by Natural Selection. *Statistics and Computing*, 4(2):87–112, June 1994.

[104] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *IEEE International Conference on Intelligent Robots and Systems*, 16–23. IEEE, 2016.

[105] S. Leest. *Directed Increment Policy Search for Behavior Tree Task Performance Optimization*. Msc, Delft University of Technology, 2017.

[106] J. Lehman and K. O. Stanley. Novelty Search and the Problem with objectives. *Evolutionary Computation*, 19(2):189–223, 2011.

[107] P. Lichtsteiner, C. Posch, and T. Delbrück. A 128x128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[108] A. Ligot and M. Birattari. On mimicking the effects of the reality gap with simulation-only experiments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11172 LNCS(March):109–122, 2018.

[109] C.-U. Lim, R. Baumgarten, and S. Colton. Evolving Behaviour Trees for the Commercial Game DEFCON. In *Applications of Evolutionary Computation*, volume 6024, 100–110, Essex, 2010. Springer Berlin Heidelberg.

[110] L. Longinotti. *cAER: A framework for event-based processing on embedded systems*. Bsc thesis, University of Zürich, 2014.

[111] H. C. Longuet-Higgins and K. Prazdny. The interpretation of a moving retinal image. *Proceedings of the Royal Society of London Biological Sciences*, 208(1173):385–397, 1980.

[112] J. Love. *Process Automation Handbook*. Number 800 in Production & Process Engineering. Springer, London, 1st Edition, 2007.

[113] B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of Imaging Understanding Workshop*, 121–130, 1981.

[114] M. Lungarella and R. Pfeifer. Robots As Cognitive Tools: an Information theoretic analysis of sensory-motor data. In *Proceedings of the 2001 IEEE RAS International Conference on Humanoid Robots*, 245–252, 2001.

[115] O. Mayr. The Origins of Feedback Control, 1970.

[116] L. Meeden. Bridging the Gap Between Robot Simulations and Reality with Improved Models of Sensor Noise. In *Genetic Programming*, 824–831, San Francisco, CA, July 1998. Morgan Kaufmann.

[117] M. Melanie and M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998.

[118] O. Miglino, H. H. Lund, and S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments. *Artificial life*, 2(4):417–434, Jan. 1995.

[119] A. Miller, B. Miller, A. Popov, and K. Stepanyan. UAV Landing Based on the Optical Flow Videonavigation. *Sensors*, 19(6):1351, 2019.

[120] B. L. Miller and D. E. Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9(3):193–212, 1995.

[121] I. Millington and J. Funge. *Artificial Intelligence for Games*. Morgan Kaufmann, San Francisco, CA, 2nd Edition, 2009.

[122] A. Montebelli, C. Herrera, and T. Ziemke. On Cognition as Dynamical Coupling: An Analysis of Behavioral Attractor Dynamics. *Adaptive Behavior*, 16(2-3):182–195, 2008.

[123] J.-B. Mouret and S. Doncieux. Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. *Evolutionary Computation*, 20(1):91–133, 2012.

[124] E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza. Lifetime estimation of events from Dynamic Vision Sensors. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):4874–4881, 2015.

[125] E. Mueggler, B. Huber, and D. Scaramuzza. Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2761–2768, 2014.

[126] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness Functions in Evolutionary Robotics: A Survey and Analysis. *Robotics and Autonomous Systems*, 57(4):345–370, Apr. 2009.

[127] S. Nolfi. Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems*, 22(3-4):187–198, 1997.

[128] S. Nolfi. Evolutionary Robotics: Exploiting the Full Power of Self-organization. *Connection Science*, 10(3-4):167–184, 1998.

[129] S. Nolfi. Power and the Limits of Reactive Agents. *Neurocomputing*, 42(1-4):119–145, 2002.

[130] S. Nolfi and D. Floreano. Learning and Evolution. *Autonomous Robots*, 7:89–113, 1999.

[131] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence and Technology*. MIT Press, Cambridge, MA, 2000.

[132] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to Evolve autonomous robots: Different Approaches in evolutionary robotics. In *Artificial Life IV*, 190–197, Cambridge, MA, July 1994. MIT Press/Bradford Books.

[133] S. Nolfi and D. Parisi. Learning to Adapt to Changing Environments in Evolving Neural Networks. *Adaptive Behavior*, 5(1):75–98, 1996.

[134] C. Ollion, T. Pinville, and S. Doncieux. With a little help from selection pressures: evolution of memory in robot controllers. *Artificial Life 13*, 407–414, 2012.

[135] F. Paredes Vallés. *Neuromorphic Computing of Event-Based Data for High-Speed Vision-Based Navigation*. Msc, Delft University of Technology, 2018.

[136] D. Perez, M. Nicolau, M. O. Neill, A. Brabazon, and M. O'Neill. Evolving Behaviour Trees for the Mario AI Competition using Grammatical Evolution. In *Applications of evolutionary computation*, 123–132, Torino, Apr. 2011. Springer Berlin Heidelberg.

[137] P. Petrovi. Evolving Behavior Coordination for Mobile Robots using Distributed Finite-State Automata. In *Frontiers in Evolutionary Robotics*, chapter 23, 413–438. I-Tech Education and Publishing, Vienna, 2008.

[138] R. Pfeifer and J. Bongard. *How the body shapes the way we think: A new view of intelligence*. MIT Press, 2007.

[139] R. Pfeifer and C. Scheier. Sensory-motor coordination: The metaphor and beyond. *Robotics and Autonomous Systems*, 20:157–178, 1997.

[140] R. Pfeifer and C. Scheier. *Understanding Intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[141] J. Piaget. When Thinking Begins. In *The Origins of Intelligence in Children*, 25–36. International University Press, New York, 1952.

[142] B. J. Pijnacker Hordijk, K. Y. W. Scheper, and G. C. H. E. de Croon. Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, 35(1):69–90, 2018.

[143] Á. Pintér-Bartha, A. Sobe, and W. Elmenreich. Towards the Light: Comparing Evolved Neural Network Controllers and Finite State Machine Controllers. In *10th International Workshop on Intellegent Solutions in Embedded Systems*, 83–87, Klagenfurt, July 2012. IEEE.

[144] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust Adversarial Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[145] D. Pool. *Objective Evaluation of Flight Simulator Motion Cueing Fidelity Through a Cybernetic Approach*. PhD thesis, Delft University of Technology, 2012.

[146] C. Posch, D. Matolin, and R. Wohlgenannt. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[147] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbrück. Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output. *Proceedings of the IEEE*, 102(10):1470–1484, 2014.

[148] B. Remes, D. Hensen, F. van Tienen, C. de Wagter, E. van der Horst, and G. de Croon. Paparazzi: How to make a swarm of Parrot AR Drones fly autonomously based on GPS. In *Proceedings of the International Micro Air Vehicle Conference and Flight Competition*, 17–20, Toulouse, France, 2013. IMAV.

[149] L. Rodolfo Garcia Carrillo, I. Fantoni, E. Rondón, and A. Dzul. Three-dimensional Position and Velocity Regulation of a Quad-Rotorcraft Using Optical Flow. *Transactions on Aerospace and Electronic Systems*, 51(1):358–371, 2015.

[150] B. Rueckauer and T. Delbrück. Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Frontiers in Neuroscience*, 10(APR):1–17, 2016.

[151] F. Ruffier and N. Franceschini. Optic Flow Regulation in Unsteady Environments: A Tethered MAV Achieves Terrain Following and Targeted Landing Over a Moving Platform. *Journal of Intelligent & Robotic Systems*, 79:275–293, 2014.

[152] F. Ruini and A. Cangelosi. Extending the Evolutionary Robotics approach to flying machines: An application to MAV teams. *Neural Networks*, 22(5-6):812–821, 2009.

[153] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[154] K. Y. W. Scheper and G. C. H. E. de Croon. Abstraction as a Mechanism to Cross the Reality Gap in Evolutionary Robotics. In *From Animals to Animats 14: Proceedings of the 14th International Conference on Simulation of Adaptive Behavior (SAB2016)*, volume 9825 LNCS, 280–292. Springer International Publishing, Aberystwyth University, Wales, UK, 2016.

[155] K. Y. W. Scheper and G. C. H. E. de Croon. Abstraction, Sensory-Motor Coordination, and the Reality Gap in Evolutionary Robotics. *Artificial Life*, 23(2), 2017.

[156] K. Y. W. Scheper, B. D. W. Remes, C. De Wagter, M. Karásek, and G. C. H. E. de Croon. First Autonomous Multi-Room Exploration with an Insect-Inspired Flapping Wing Vehicle. In *International Conference on Robotics and Automation (ICRA2018)*, 5546–5552, Brisbane, 2018. IEEE.

[157] E. J. J. Smeur, Q. P. Chu, and G. C. H. E. de Croon. Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3):450–461, Dec. 2016.

[158] C. A. Smith and A. Corripio. *Principles and Practice of Automatic Process Control*. Wiley, 1985.

[159] T. Soule, B. D. Robison, and R. B. Heckendorn. Co-evolution of Sensor Morphology and Behavior. In *Genetic and Evolutionary Computation Conference*, 135–136, Denver, CO, USA, 2016.

[160] L. Steels. Emergent Functionality in Robotic Agents through On-line Evolution. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 8–14, Cambridge, MA, USA, July 1994. MIT Press.

[161] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, Sept. 1998.

[162] P. Szczawinski, M. Duarte, S. M. Oliveira, and A. L. Christensen. Toward Evolved Vision-based Control for a Quadrocopter. *Proceedings of the 9th Conference on Telecommunications (CONFTELE)*, 153–156, 2013.

[163] L. Takayama, W. Ju, and C. Nass. Beyond Dirty, Dangerous and Dull: What Everyday People Think Robots Should Do. *HRI 08 Proceedings of the 3rd ACMIEEE international conference on Human robot interaction*, 25–32, 2008.

[164] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. *ArXiv e-prints*, 2018, 10.15607/RSS.2018.XIV.010.

[165] S. Tijmons, G. C. H. E. de Croon, B. D. W. Remes, C. De Wagter, R. Ruijsink, E.-J. van Kampen, and Q. P. Chu. Stereo Vision Based Obstacle Avoidance on Flapping Wing MAVs. In *Advances in Aerospace Guidance, Navigation and Control*, 463–482, Delft, 2013. Springer Berlin Heidelberg.

[166] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *IEEE International Conference on Intelligent Robots and Systems*, 23–30, 2017.

[167] E. Tuci, V. Trianni, and M. Dorigo. 'Feeling' the flow of time through sensorimotor co-ordination. *Connection Science*, 16(4):301–324, 2004.

[168] J. Urzelai and D. Floreano. Evolutionary Robots with Fast Adaptive Behavior in New Environments. *Evolutionary Computation*, 9(4):495–524, 2007.

[169] A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, 429–528. Springer Berlin Heidelberg, 1998.

[170] F. van Breugel, K. Morgansen, and M. H. Dickinson. Monocular distance estimation from optic flow during active landing maneuvers. *Bioinspiration & Biomimetics*, 9, 2014.

[171] D. Weikersdorfer, D. B. Adrian, and D. Cremers. Event-based 3D SLAM with a depth-augmented dynamic vision sensor. *IEEE International Conference on Robotics and Automation*, 359–364, 2014.

[172] D. Weikersdorfer, R. Hoffmann, and J. Conradt. Simultaneous localization and mapping for event-based vision systems. In *International Conference on Computer Vision Systems*, 133–142. Springer-Verlag Berlin Heidelberg, 2013.

[173] S. Weisberg. *Applied linear regression*. John Wiley & Sons, Inc., St. Petersburg, 2005.

[174] M. Yang, S.-C. Liu, and T. Delbrück. A Dynamic Vision Sensor With 1{%} Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding. *IEEE Journal of Solid-State Circuits*, 50(9):2149–2160, 2015.

[175] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[176] Y. Yuan, H. Xu, and B. Wang. An Improved NSGA-III Procedure for Evolutionary Many-objective Optimization. *Genetic and Evolutionary Computation Conference (GECCO 2014)*, 661–668, 2014.

[177] J. C. Zagal and J. Ruiz-del Solar. Combining Simulation and Reality in Evolutionary Robotics. *Journal of Intelligent and Robotic Systems*, 50(1):19–39, Mar. 2007.

[178] J.-C. Zufferey and D. Floreano. Fly-inspired visual steering of an ultralight indoor aircraft. *Transactions on Robotics*, 22(1):137–146, Feb. 2006.

Acknowledgments

The work presented in this dissertation is not just the result of four years of research but the end of a long series of events, without which I would not have arrived at this here. Although I would like to, thanking every person I have ever interacted with would make this book very long. Let me then start with the most recent and arguably most significant persons of influence.

To Max, your years of experience and wide knowledge base helped shine a light on some of the weak areas of early versions of this thesis. This helped me to reflect, revise and restructure my ideas resulting in a body of work of which I am now quite proud. Thank you for being a constant rock of support throughout my PhD. You often thought of my well-being when I was too deep in my work to notice my own psychological state. Our iteration actually goes further back than my PhD as I also did my MSc in C&S. Thank you for helping me nurture the spark that was my interest in research by introducing me to Eric Johnson and helping me organize my visit to Georgia Tech. You put a lot of trust in me to be the first visiting researcher following Christophe, one of the smartest people I know.

To Guido, you have consistently helped me transform the unstructured ideas in my head into coherent stories and temper my sometimes hypercritical opinions. You were there starting from my MSc thesis through all my research papers, algorithm developments, student projects and this dissertation, I would not be where I am today without you. Thank you for your trust, guidance and advice, both professionally and personally, over the years.

To Christophe and Bart, thank you for founding the MAVLab and for making a safe space to make crazy ideas become reality. To my co-authors Matěj, Kimberly, Mario, Sjoerd, Fede and Coen, thank you for your ideas, discussions and feedback. To the other members of the MAVLab, Erik, Hann-Woei, Ewoud, Lodewijk, Andries, Roland, Freek, Kevin, Shou, Bart S., Diana, Shushuai, Tom and Yingfu thank you for making my time in the MAVLab not only productive but also thoroughly enjoyable.

To my office-mates: Ivan, Jaime, Jelmer, Sihao, Ye Z.; fellow PhDs present: Annemarie, Bo, Christian, Daniel, Dirk, Emmanuel, Ezgi, Isabel, Jerom, Julia, Junzi, Maarten, Malik, Noor, Rolf, Sarah, Sherry, Tim, Wei, Yingzhi, Ying; and past: Dyah, Henry, Jan, João, Kasper, Peng, Sophie, Tao, Tommaso; as well as all C&S staff: Alexander, Alwin, Andries, Bob, Clark, Coen, Daan, Erik-Jan, Ferdinand, Hans, Harold, Jacco, Joost, Marilena, Menno, Olaf, Dr. Chu, and Rene; thank you for the coffee corners, heidagen, flight practicals, BBQs, social drinks and vrijmibos. And to Bertine, I can't imagine our department operating without you, thank you for all of your help.

I was lucky enough to start my PhD around the same time that Matěj and Kimberly started their stints at C&S and we all got a seat in the office with Jaime. I am certain that my work life would have been quite a bit more boring if this had not happened. Matěj, thank you for being the voice of reason in my life, if ever I needed advice about my work or about my life, starting with 'hey Matěj, do you have a second?', you were always available and wise. Kim, thank you for being a true friend, always there to hear my crazy problems, letting me know when I was being too much and helping me to chill out. Jaime, you are one of the most authentic people I have every met. I cherish every conversation we have and I respect and actively request your opinion on different aspects of my life (even sometimes at your dismay). Although we are such different people, somehow we fit together and I think I am a better person as a result of knowing you. Thank you for putting up with me.

I can't mention my time during my PhD without mentioning Stabilo. To Niek and Vera, thank you for convincing me to join Stabilo and become the first PhD member, I think Stabilo would not be where it is today without your effort. To Jennifer and Rowenna, the way you grab problems by the horns and come 110% prepared to everything you set your minds to is really impressive. Thank you for your guidance and challenge, I learned many valuable skills by following your example. Ismael, Gideon, Bianca, Bart, Bram, Ankit, Taemin, Benyamin, Jacomijn, Thijs and Quincy, thank you for all the meetings, drinks, BBQs, karaokes, lunch lectures, career fairs and dinners. I can wholeheartedly say that we made Stabilo great again.

To my family, thank you for always supporting me even when my ambition pulled me away from you. Granny, after raising five quite troublesome boys, life threw you another one and you accepted me with open arms, thank you for your unconditional love over the years. Steven, thank you for being the best role model a young boy could ask for, I would be a lesser man without having had you in my life. Mummy and Daddy, you sacrificed so much for me over the years, more than I will be able to fully understand. I will spend the rest of my life putting that investment to work to make you proud. To my dear little sister, when we were young I was always annoyed with you cause you followed me around and tried your best to be like me, ironic then that now that I am the one that looks up to you. Your discipline and determination are unlike anything I have seen before and it inspires me to push for my dreams, keep doing what you are doing.

Moving to a foreign country where people speak a foreign language with foreign practices would seem to many a daunting experience, and for some it might be, but I quickly found a group of people that made my transition quite enjoyable. To Christophe and Eli, we met on the first day of Uni and have shared 10 years of friendship since, thank you for the parties, discussions, drinking, dinners, movies and so much more.

To my original housemates at the E du Porny: Q, Tom B., Tom S., Anne, Tim, Patrick and Louise, without you accepting an English speaker into the house two days before I was going to be kicked out of my previous house, I would have been in quite a bit of trouble. I would also like to thank the other 20 people at the instemming for

being so lame! Over the years we had several people come and go including Sacha, Kim, Steffen, Steven, Inge, Thijs, Menno, Jac and Donna, I can't imagine my life without all you in it. I enjoyed my time in the house more than I can possibly express. The house parties, house weekends, Christmas dinners in the summer, Queen's days (and Queen's nights), Carnivals, IO festivals, BK Beats, Aangeschoten Wilds, Sinterklasse, hungover Disney movies, cleaning sessions, dinners, conversations, teaching me Dutch, so many memories I will cherish forever. For all that and much more, thank you. To Inge and Merel, thank you for your dedication while helping me complete the Dutch Summary of this thesis during our Caribbean vacation.

Sacha, Inge, Christophe, Jaime you have been particularly influential during different phases of my life in the Netherlands. I don't know enough words to express my appreciation for you, so I will keep it simple, you are amazing people, thank you for being a part of my life.

Delft, September 2019

Curriculum Vitæ

Kirk Yannick Willehm Scheper

30-11-1989 Born in Castries, St. Lucia

Education

2014–2018 PhD. Aerospace Engineering
Delft University of Technology, Delft, The Netherlands
Thesis: Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics
Promotors: Prof. dr. ir. M. Mulder, Dr. G.C.H.E. de Croon

2011–2014 M.Sc. Aerospace Engineering
Delft University of Technology, Delft, The Netherlands
Thesis: Behaviour Trees for Evolutionary Robotics: Reducing the Reality Gap

2010 Minor Degree Program, Mechatronics
Nanyang Technological University, Singapore

2008–2011 B.Sc. Aerospace Engineering
Delft University of Technology, Delft, The Netherlands

2001–2008 Secondary School
Harrison College, Bridgetown, Barbados

Experience

2019– Engineer
Aerial Robotics Group, Sony
Schlieren, Switzerland

2017–2019 Board Member
Stabilo C&S Student Association
Delft, The Netherlands

2015–2019	Managing Director Nub Systems B.V. Delft, The Netherlands
2012	Visiting Research Intern Unmanned Aerial Vehicle Research Facility, Georgia Institute of Technology, USA

List of Publications

Journal Papers

6. **K. Y. W. Scheper**, G. C. H. E. de Croon, *Evolution of Robust High Speed Divergence-Based Landing for Autonomous MAVs*, [Under Review]

5. F. Paredes-Vallés, **K. Y. W. Scheper**, G. C. H. E. de Croon, *Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2019 [In Press]

4. M. Coppola, K. N. Mcguire, **K. Y. W. Scheper**, G. C. H. E. de Croon, *On-board Bluetooth-based Relative Localization for Collision Avoidance in Micro Air Vehicle Teams*, Autonomous Robots 42(8):1787–1805, 2018

3. B. J. Pijnacker Hordijk, **K. Y. W. Scheper**, G. C. H. E. de Croon, *Vertical Landing for Micro Air Vehicles using Event-Based Optical Flow*, Journal of Field Robotics 35(1):69–90, 2018

2. **K. Y. W. Scheper** and G. C. H. E. de Croon, *Abstraction, Sensory-Motor Coordination, and the Reality Gap in Evolutionary Robotics*, Artificial Life, 23(2):124–141, 2017

1. **K. Y. W. Scheper**, S. Tijmons, C. C. de Visser, and G. C. H. E. de Croon, *Behavior trees for evolutionary robotics*, Artificial Life, 22(1):23–48, 2016

Conference Papers

5. **K. Y. W. Scheper**, M. Karásek, C. De Wagter, B. D. W. Remes and G. C. H. E. de Croon, *First autonomous multi-room exploration with an insect-inspired flapping wing vehicle*. IEEE International Conference on Robotics and Automation (ICRA2018). May 2018, Brisbane, Australia

4. **K. Y. W. Scheper** and G. C. H. E. de Croon, *Abstraction as a Mechanism to Cross the Reality Gap in Evolutionary Robotics*, in From Animals to Animats 14: Proceedings of the 14th International Conference on Simulation of Adaptive Behavior (SAB2016), E. Tuci, A. Giagkos, M. Wilson, and J. Hallam, Eds. Aberystwyth University, Wales, UK: Springer International Publishing, 2016, pp. 280–292

3. **K. Y. W. Scheper**, G. Chowdhary, and E. N. Johnson, *Aerodynamic system identification of fixed-wing UAV*, in AIAA Atmospheric Flight Mechanics (AFM) Conference, 2013

2. **K. Y. W. Scheper**, D. Magree, T. Yucelen, G. De La Torre, and E. N. Johnson, *Application of Frequency-Limited Adaptive Quadrocopter Control*, in AIAA Guidance, Nav-

igation, and Control (GNC) Conference, 2013

1. R. van der Goot, J. Hendriks, **K. Y. W. Scheper**, W. van der Wal, D. G. Simons, *A low cost, high resolution acoustic camera with a flexible microphone configuration*, Proceedings of the 4th Berlin Beamforming Conference. February 2012, Berlin, Germany

One man cannot summon the future.

- Spock

But one man can change the present!

- James T. Kirk

Star Trek: The Original Series, Season 2 Episode 4

ISBN 978-94-6366-197-3

Propositions

accompanying the dissertation

Abstraction as a Tool to Bridge the Reality Gap in Evolutionary Robotics

by

Kirk Yannick Willehm Scheper

1. Abstraction can be used as a tool to control the trade-off between optimization power and robustness of robotic behavior to the transfer problem. [This thesis]
2. Given a sufficient model of the world, behavior optimized in simulation should result in similar performance in reality. As such, the goal of the transfer should be to reduce the behavioral reality gap rather than directly optimizing real world performance. [This thesis]
3. Abstraction segments the optimization problem into a set of smaller sub-problems, each of which can be independently optimized improving robustness and reducing overall optimization time. [This thesis]
4. Abstraction reduces the objective simulation fidelity required to optimize robotic behavior, leading to a faster optimization process. [This thesis]
5. To the perceptual system of a robot, reality is nothing more than a very complex simulation.
6. Science fiction as a looking glass to view our current and future world is a powerful method to perform thought experiments about what future technologies will benefit humanity and how they will impact on our society.
7. Lasting advances in science, as in life, often require a change in perspective.
8. Focusing solely on graduating on schedule comes at the cost of missing potentially life-changing opportunities and experiences.
9. Developing strong personal relationships with the locals in a foreign land is the most effective way for both parties to learn a new language.
10. Unlike methods commonly used in the military, the best method to generate group cohesion and trust, is to go on a self-organized vacation.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors Prof. dr. ir. M. Mulder and Dr. G.C.H.E. de Croon.