Faculty of Aerospace Engineering

Obstacle avoidance for quadrotors using reinforcement learning and obstacle-airflow interactions

Thesis

G.J. van Dam

16 April 2019

TU Delft Delft University of Technology

Challenge the future

Obstacle avoidance for quadrotors using reinforcement learning and obstacle-airflow interactions

Thesis

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

G.J. van Dam

16 April 2019

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Obstacle avoidance for quadrotors using reinforcement learning and obstacle-airflow interactions"** by **G.J. van Dam** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 16 April 2019

Readers:

dr.ir. E. van Kampen

dr. Q.P. Chu

dr. R.P. Dwight

Executive summary

This report presents the work carried out for the research into using Reinforcement Learning (RL) and obstacle-airflow interactions to detect and avoid obstacles on a quadrotor. The goal of the research is to **investigate and propose an alternative method for obstacle avoidance and detection, one that does not require the addition of any sensors but relies solely on measurements of the accelerometer and rotor controllers**.

In recent years, Unmanned Aerial Vehicles (UAVs), and quadrotors in specific, have risen in popularity, with applications ranging from industrial inspection tasks to disaster response and package delivery applications. Increasing autonomy for these vehicles could bring down cost significantly and allow for larger scale deployments of these solutions. A key remaining challenge is in-flight autonomous object detection and avoidance. Most current obstacle avoidance methods are either based on computer vision, requiring good lighting conditions, or based on the addition of a range sensor.

The obstacle avoidance method proposed in this research could be used as a stand-alone method for small quadrotors, thereby removing the need for additional sensors, thus saving cost and weight while increasing flight time. Alternatively, when being used as an addition to other obstacle avoidance solutions, like computer vision methods, it can increase safety and reliability by functioning as a backup method. An example of this would be a disaster response drone entering a room filled with smoke, or a warehouse inspection drone returning safely to its base when the lights turn off during a power outage.

The detection of obstacles is based on the principle that the airflow around a multirotor changes when the multirotor is flying near a surface. A well-known example of this is the ground effect, an increase in lift force close to a ground surface. Similarly, a change in dynamics occurs when a multirotor is flying close to a wall or ceiling. The proposed method uses a reinforcement learning controller to detect obstacles based on these measurements, and take action to lead the multirotor back to safety.

A proof-of-concept of this method is developed by training a reinforcement learning agent to avoid obstacles beneath a descending quadrotor. This is first done in a simulated environment, where the influence of hyperparameters, the amount of noise in the state signal, and the number of training episodes are investigated. The best performing agent from simulation is evaluated during a flight experiment with the Parrot Bebop 1 drone, where it is able to prevent the quadrotor from hitting the obstacle in 80% of the episodes. Furthermore, it is shown that the same level of performance can be achieved, by learning fully from scratch, in-flight, without prior knowledge or training, during 50 real flight training episodes

An approach for extending this method to the avoidance of walls, ceilings, and smaller obstacles is discussed and the expected performance when doing so on the Parrot Bebop 1 is assessed. Additionally, it is shown that this method can easily be extended to other quadrotors, by demonstrating this on the Parrot Bebop 2 quadrotor.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
C&S	Control & Simulation
CDF	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DQN	Deep Q-Network
FOE	Focus of Expansion
GPS	Global positioning system
HRL	Hierarchical Reinforcement Learning
IMU	Inertial Measurement Unit
INDI	Incremental Nonlinear Dynamic Inversion
IRL	Inverse Reinforcement Learning
LIDAR	Light Detection and Ranging
MAV	Micro Air Vehicle
MDP	Markov Decision Process
MEMS	MicroElectroMechanical Systems
PBVI	Point-based Value Iteration
POMD	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RMSE	Root-Mean-Square Error
RPM	Revolutions Per Minute
SHERPA	Safety Handling Exploration with Risk Perception Algorithm
SNR	Signal to Noise Ratio
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TD	Temporal-Difference
TRF	Trust Region Reflective
TTC	Time To Contact
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UKF	Unscented Kalman Filter
VTOL	Vertical Take-Off and Landing

List of Symbols

Greek Symbols

α	Learning rate
δ_t	Temporal Difference error
ϵ	Exploration rate
γ	Discount rate
λ	Decay-rate for eligibility traces
μ	Mean
ω_i	Rotational speed of rotor i [rad/s]
ϕ	Roll angle [rad]
ψ	Heading angle [rad]
σ	Standard deviation
$\tau_{ext,x}$	Estimated external torque around the x-axis [Nm]
$\tau_{ext,y}$	Estimated external torque around the y-axis [Nm]
$\tau_{ext,z}$	Estimated external torque around the z-axis [Nm]
θ	Pitch angle [rad]

Roman Symbols

A_{signal}	Strength of a signal
a_t	Action at time t
b	Distance between two diagonally opposite rotor axes [m]
c	Distance to a surface above [m]
d	Distance between two adjacent rotor axes [m]
$d_{i,x}$	Distance from rotor i to the x-axis [m]
$d_{i,y}$	Distance from rotor i to the y-axis [m]
$d_{\mathbf{wall}}$	Maximum distance from which the wall force is exerted [m]
e	Ground effect model bias [N/kg]
E_t	Eligibility traces at time t
$F_{\mathbf{conv}}$	Force that is applied by the conventional controller [N]
$F_{\mathbf{drag},x}$	Drag force in body x-axis [N]
$F_{\mathbf{drag},y}$	Drag force in body y-axis [N]
$F_{\mathbf{drag},z}$	Drag force in body z-axis [N]

F_{ext}	External force estimate [N]
$F_{ext,x}$	Estimated external force in body x-axis [N]
$F_{ext,y}$	Estimated external force in body y-axis [N]
$F_{ext,z}$	Estimated external force in body z-axis [N]
F_g	Force that is applied to the ball by the combined controllers [N]
F_i	Thrust produced by rotor i [N]
$F_{\mathbf{rl}}$	Force that is applied by the reinforcement learning controller [N]
$F_{\mathbf{wall}}$	Force that is exerted by the wall [N]
g	Standard gravitational acceleration [m/s^2]
G_t	Return
I_{xx}	Moment of inertia about the x-axis [kg·m^2]
I_{yy}	Moment of inertia about the y-axis [kg·m^2]
I_{zz}	Moment of inertia about the z-axis [kg·m^2]
K_b	Body lift coefficient
$k_{D,x}$	Drag coefficient in body x-axis [N·s^2/m]
$k_{D,y}$	Drag coefficient in body y-axis [N·s^2/m]
$k_{D,z}$	Drag coefficient in body z-axis [N·s^2/m]
k_i	Rotor gain [N·s^2/rad]
$K_{\mathbf{wall}}$	Maximum force exerted by a wall [N]
l_i	Rotor torque gain [kg·m^2/rad]
m	Mass [kg]
$N_{\mathbf{episodes}}$	Number of episodes
$N_{\mathbf{sin}}$	Number of sine waves in the reference signal
o	Distance to a surface on the same level [m]
p	Body rate around the x-axis [rad/s]
Q	Action-value function
q	Body rate around the y-axis [rad/s]
$R_{\mathbf{rotor}}$	Rotor radius [m]
r	Body rate around the z-axis [rad/s]
r_t	Reward at time t
s_t	State at time t
T	Torque [Nm]
t	Time [s]

T_c	Thrust in ceiling effect [N]
T_g	Thrust in ground effect [N]
T_∞	Thrust in free flight [N]
u	Speed in the body x-axis [m/s]
V	State-value function
v	Speed in the body y-axis [m/s]
w	Speed in the body z-axis [m/s]
x	Position [m]
$\dot{x}$	Velocity [m/s]
$\ddot{x}$	Acceleration [m/s^2]
$x_{\mathbf{ref}}$	Reference signal [m]
$x_{\mathbf{wall}}$	Position of the wall [m]
z	Distance to a surface below [m]

Chapter 1

Introduction

1-1 Background

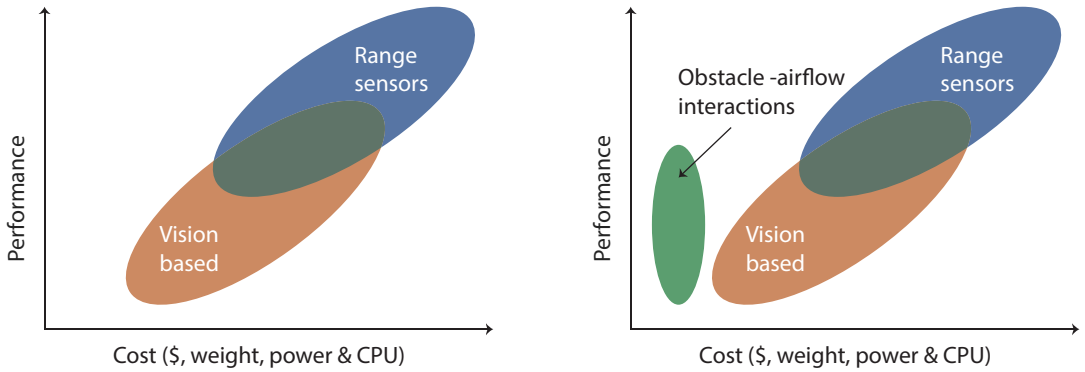
In recent years multirotors, and quadrotors in specific, have grown to be the most popular UAV platform. [12] This can be attributed to their simplicity, ability to hover and vertical take-off and landing (VTOL) capability, all driving commercial, academic and private demand for these vehicles.

With the acceptance of multirotors now high, the next step in development is largely expected to be the development of fully autonomous control for these UAVs. A key challenge here, similar to that for other UAV-platforms, is in-flight autonomous object detection and avoidance. Currently, partial solutions to this problem are usually either vision-based or range-sensor-based.

Vision-based methods rely on an onboard camera and the use of computer vision algorithms to detect and avoid obstacles. While this area of investigation certainly looks promising, it is also dependent on good lighting conditions, and the implementation on multirotors is usually limited by the amount of computational resources available on such a platform.

Other obstacle detection solutions usually rely on the addition of some range sensor for proximity measurement, like a laser rangefinder or ultrasonic sensors, to detect obstacles. Because of the added cost, weight and power usage of such a sensor, this can decrease flight time significantly and restrict their usability to large multirotors.

A qualitative assessment of these classes of methods, as well as of the method that will be proposed in this thesis, is shown in figure 1-1. Performance is considered to be a combination of accuracy and reliability.



(a) Current obstacle detection methods **(b)** Placement of the proposed new method

Figure 1-1: Qualitative assessment of current obstacle detection solutions.

1-2 Problem statement

The objective of this research is to investigate and propose an alternative method for obstacle avoidance and detection, one that does not require the addition of any sensors but relies solely on measurements of the accelerometer and rotor controllers, both present on almost all multirotors. This new obstacle avoidance method could potentially be used either as a stand-alone method, for multirotors that are not equipped with another form of obstacle avoidance, or as an addition to another obstacle avoidance solution.

The detection of obstacles is based on the principle that the flight dynamics of a multirotor change when the multirotor is flying near a surface. A well-known example of this is the so-called ground effect, an increase in lift force close to a ground surface, an effect also seen in helicopters and fixed-wing aircraft. Similarly, a change in dynamics occurs when a multirotor is flying close to a wall [37], or ceiling [52]. The proposed method then uses a reinforcement learning controller to detect obstacles based on these measurements and take action to lead the multirotor back to safety.

In the research, the focus is on quadrotors, a multirotor helicopter that is lifted and propelled by four rotors. However, it is expected that, following a similar approach, this obstacle avoidance technique can be extended to other types of multirotors as well.

In order to achieve the objective the following main research question and underlying sub-questions have been defined:

How can a reinforcement learning-based quadrotor control system avoid obstacles using the obstacle-airflow interactions caused by these obstacles?

SQ1 What is the state of the art in the related fields?

- SQ1.1 What is the state of the art in the field of obstacle avoidance?
- SQ1.2 What is the state of the art in the field of obstacle-airflow interactions between a quadcopter and obstacles?
- SQ1.3 What is the state of the art in the field of reinforcement learning?

SQ2 Given this state of the art, can a reinforcement learning-based quadrotor control system avoid obstacles, using these obstacle-airflow interaction effects?

- SQ2.1 How can the reinforcement learning-based control system best be setup?
- SQ2.2 How can the reinforcement learning agent be trained and tested in simulation?
- SQ2.3 How can this agent be trained and tested in a real-life experiment?
- SQ2.4 What is the performance of the trained and tested control system?

SQ3 How can the knowledge gained from developing this reinforcement learning-based quadrotor control system be applied to other quadrotors?

1-3 Considerations

1-3-1 Potential advantages of this new method

Since this new method does not require the addition of any sensors, there is a lot of potential for multirotors that need to be either really light or really cheap. Opening up the ability to also fly these type of multirotors autonomously in a safe manner.

Furthermore, by using this new method as an addition to another obstacle avoidance method three major things can be achieved. These would be applicable to all multirotors, regardless of cost or weight restrictions.

First of all, none of the current obstacle avoidance methods are 100% accurate, even in their designed operational environment. By implementing a secondary method, relying on a different type of detection technology, it is expected that this accuracy can be improved. This could be done by using both methods simultaneously, or by using this new method as an extra check when the primary obstacle detection method is not a 100% certain.

Secondly, a UAV flying autonomously can encounter situations that fall outside of their expected operational environment. An obvious example here would be a UAV operating indoors, using computer vision for obstacle avoidance, when suddenly the lights go out. In such a case a secondary method, not relying on visual light, could be a savior.

Thirdly all systems are subject to the risk of failure. As such, having a certain redundancy in your obstacle avoidance method would be an advantage. Especially when it is a method that does not require the addition of any weight to the UAV.

1-3-2 Potential disadvantages of this new method

There are of course also some potential downsides that need to be considered. First of all, the obstacle-airflow interaction effects used by this method to detect obstacles only occurs when the multirotor is close to the obstacle. Initial analysis from [37] and [53] shows that the effect is measurable at a distance between 3 and 5 times the rotor radius. The direct impact of this would be that the multirotor would probably need to fly at relatively low speeds.

Secondly, since the method relies on aerodynamic effects, its reliability would be influenced by other aerodynamic effects like wind or turbulence. Leading to either false positives or false negatives. As such, the application of this technique should probably first be applied indoor before it is tested in outdoor environments.

Third and finally, there are certain types of obstacles that this method would have a hard time detecting. Especially obstacles where the aerodynamic interaction between the multirotor and the obstacle is small. Either because the obstacle itself is small compared to the size of the multirotor, or because the obstacle does not have a solid surface, for example, a net.

1-4 Research approach

The research has been divided into three main parts. A literature survey, the preliminary research, and the final thesis research.

In the literature survey, relevant literature will be identified and studied. The focus will be on three topics: obstacle avoidance, reinforcement learning and the obstacle-airflow interactions between a quadcopter and obstacles. For all three topics, both the basic principles and the state of the art will be researched. By doing so, research question 1 (SQ1) will be answered.

In the preliminary research phase, an initial experiment will be performed in simulation, this will provide some first experience with solving a problem like this. Furthermore, the results from this preliminary research will help to partially answer subquestion 2.1 (SQ2.1) by providing insight in the influence of reinforcement learning algorithms and hyperparameters.

Finally, in the final thesis research, subquestions 2 and 3 will be answered by training a RL agent to intervene when it notices an obstacle beneath a descending quadrotor. This will first be done in a simulated environment, where the influence of hyperparameters, the amount of noise in the state signal, and the number of training episodes can be investigated. The best performing agent from simulation will then be evaluated during a flight experiment with the Parrot Bebop 1 drone in the CyberZoo at Delft University of Technology. Using the same quadrotor, an experiment will also be conducted where the RL agent will learn from scratch. Using these results the extension of the developed method to obstacles above, and on the same level as the quadrotor will be assessed.

1-5 Report structure

First, in chapter 2, the relevance of this research from both a social and academic perspective will be highlighted. With respect to the social relevance both the social relevance of autonomous aerial drones in general, and the social impact of this new obstacle avoidance method will be discussed. In the academic relevance section, the threefold addition of this research to the state of the art will be introduced.

The main thesis research and its most interesting results will be discussed in a stand-alone article. This article is included in chapter 3. In the article, subquestion 2 and 3 will be answered implicitly. An explicit answer to these subquestions will be provided directly after the article, in section 3-2. For readers not familiar with reinforcement learning it is suggested that section 4-3 of this thesis is read prior to the article. Similarly, readers not familiar with obstacle avoidance might consider reading section 4-1 prior to the article.

In addition to providing an introduction into some of the topics touched upon in the article, the literature survey in chapter 4 also discusses the state of the art in the fields related to this research. By doing so, subquestion 1 will be answered. In chapter 5, the preliminary investigation that preceded the main thesis research, and its results will be discussed. Thirdly, any additional results of the final thesis research not discussed in the article will be discussed in chapter 6.

The overall conclusions of the research will be drawn in chapter 7. After this, recommendations regarding future research will be made in chapter 8.

Chapter 2

Social and academic relevance

The relevance of this research can be seen from both a social and academic perspective. To highlight the importance of this research, and help place the conclusions in the right context, the research will be described from both perspectives.

2-1 Social relevance

The social relevance can be split up into two key parts: the social relevance of autonomous aerial drones in general, and the social impact of this new obstacle avoidance method.

The social relevance of autonomous aerial drones is mainly driven by the discovery and development of new applications, which may range from industrial inspection tasks to disaster response or package delivery applications. Covering all fields and applications would be impractical, but some examples of successful or promising applications are given below.

- *Package delivery* Of all drone applications, delivery drones have generated the most media coverage and public attention. Promises of half-hour shipping times, reduced greenhouse gas emissions and reduction of road traffic make this a promising application. [60] The actual use of delivery drones is however still in its early development phases.

Figure 2-1: One of the prototype package delivery quadcopters from Amazon [4].

- *Disaster response* In the future, autonomous aerial drones can be of great assistance in disaster prediction, assessment and response. For example by surveying an affected area, delivering medical supplies or establishing vital wireless communication links for survivors. [14]

- *Industrial inspection* Using aerial drones, like quadcopters, to inspect existing infrastructure can be cheaper, faster and safer than current inspection methods. [71] Several utility companies, like AT&T, are already using aerial drones to inspect (cell) towers.
- *Construction* The ability of autonomous aerial drones to perform real-time aerial inspections of large construction and infrastructure sites allows for quick surveys and mistake prevention. According to a study from PWC in 2017, dangerous accidents on construction sites monitored by drones decreased by up to 91 percent. [46]
- *Warehousing and inventory* In addition to using drones to deliver packages, companies like Wal-Mart are also testing with quadcopters for warehouse inventory management. According to the company, these UAVs can do a full inventory check in a day, a task that currently takes a month for people to do manually.[13]

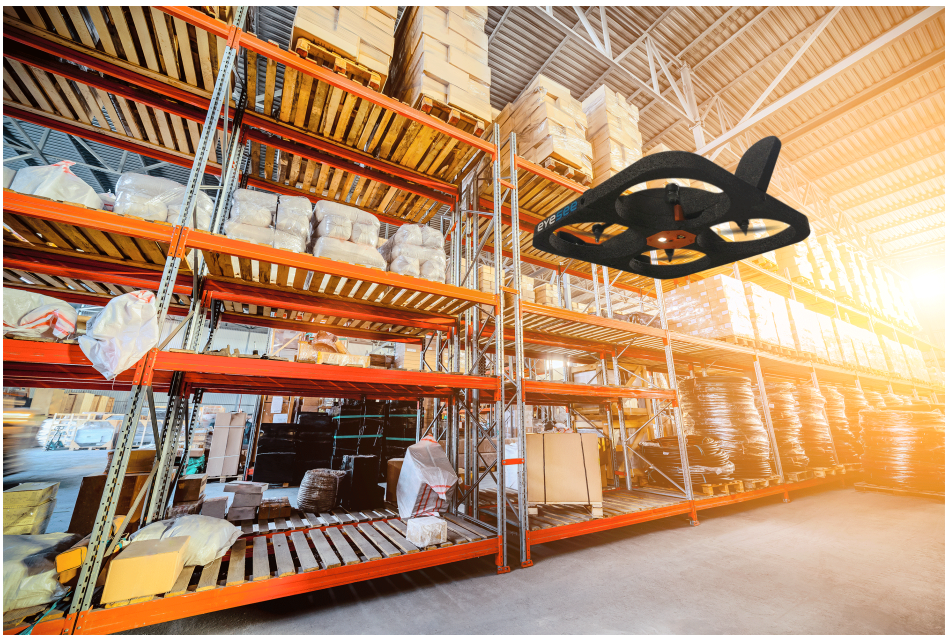


Figure 2-2: Eyesee, a fully autonomous quadrotor for inventory inspection [16].

For these, and other applications, one might argue that these aerial drones could also be controlled by human pilots instead of flying autonomously. While there is certainly some truth in this, increasing the level of autonomy can bring down cost significantly and allow for larger scale deployments of these systems. For example, by having a human supervise or monitor the deployment of multiple drones, efficiency can be increased, while still leaving higher level decisions to humans.

While the application areas described above show the social relevance of developing autonomous UAVs, which will require obstacle avoidance at early levels of autonomy, the social relevance of this specific obstacle avoidance method can best be explained by looking into the specific advantages of this new method.

As mentioned earlier, this new method could either be used as the primary obstacle avoidance method for small, lightweight, multirotors or as a secondary obstacle avoidance method for

larger multirotors. Both applications profit greatly from the very limited cost and resource requirements of this method.

When this method is applied as a primary obstacle avoidance method for small quadcopters, it removes the need for additional sensors, which you would need for range-sensor based methods, and large computational capacity, which you usually need for vision-based methods. By doing so, not only cost is saved, but both weight and power consumption can be decreased. This can greatly increase the flight time and range of such a quadcopter, thereby increasing the usability and applications.

When using this new method as a secondary obstacle avoidance method for larger quadcopters, it can act as a backup system or as an extra obstacle detector. Vision-based methods, for example, require good lighting conditions, so drones using that as their primary obstacle avoidance method could benefit from having the option to switch to this new method when lighting conditions worsen. An example of this would be a disaster response drone entering a room filled with smoke, or a warehouse inspection drone returning safely to its base when the lights turn off during a power outage. The reliability and safety of drone operations can thus be increased by using this new method as a secondary obstacle avoidance method.

2-2 Academic relevance

The addition of this research to the state of the art is threefold. This research offers a distinct addition to all the three fields it is part of: obstacle avoidance, reinforcement learning and the research of obstacle-airflow interaction between a quadcopter and an obstacle.

First of all, this research contributes to the field of obstacle avoidance by offering an innovative new method of obstacle avoidance. Furthermore, this new method is one that does not require the addition of any sensors, solely relying on the IMU and rotor speed measurements already available in most UAVs. As such, the method can not only be used as a stand-alone obstacle avoidance solution, but also as a low-cost addition, an extra layer of safety, for other obstacle avoidance solutions. The potential advantages and disadvantages of this method in this context are described in more detail in section 1-3.

Secondly, in the field of reinforcement learning, this research adds to the state of the art by its unconventional placement of the reinforcement learning agent within the control loop. In most current reinforcement learning applications the agent is in full control over the actions taken; the agent chooses an action, which results in a reward and a new state. In this application, however, the reinforcement learning agent is used in combination with a conventional controller. The conventional controller controls the UAV during most of the flight, but its actions can be overridden at any time by an action of the reinforcement learning agent.

Finally, regarding the research of obstacle-airflow interaction between a quadcopter and an obstacle, this research will provide additional measurements of the wall-effect, the aerodynamic interaction between a quadcopter and a large vertical surface on the same level as the quadcopter. Measurement data of this effect is currently scarce, so additional measurement data would be of value to any research looking into this effect.

Chapter 3

Article

3-1 Article

In the following stand-alone article the main research of this thesis will be presented. For readers unfamiliar with reinforcement learning it is suggested that section 4-3 of this thesis is read prior to the article, as it will introduce some of the reinforcement learning concepts being mentioned in the article. Furthermore, readers not familiar with obstacle detection and avoidance might consider reading section 4-1 prior to continuing with the article.

Obstacle avoidance for quadrotors using reinforcement learning and obstacle-airflow interactions

G.J. van Dam*
Delft University of Technology, Delft, 2629 HS, The Netherlands

This research investigates and proposes a new method for obstacle detection and avoidance on quadrotors. One that does not require the addition of any sensors, but relies solely on measurements from the accelerometer and rotor controllers. The detection of obstacles is based on the principle that the airflow around a quadrotor changes when the quadrotor is flying near a surface. A well-known example of this is the ground effect, an increase in lift force close to a ground surface. Similarly, a change in dynamics occurs when a quadrotor is flying close to a wall or ceiling. The proposed method uses a reinforcement learning controller to detect obstacles based on these measurements, and take action to lead the quadrotor back to safety. A proof-of-concept of this method is developed by training a reinforcement learning agent to avoid obstacles beneath a descending quadrotor. This is first done in a simulated environment, where the influence of hyperparameters, the amount of noise in the state signal, and the number of training episodes are investigated. The best performing agent from simulation is evaluated during a flight experiment with the Parrot Bebop 1 drone, where it is able to prevent the quadrotor from hitting the obstacle in 80% of the episodes. Furthermore, it is shown that the same level of performance can be achieved, by learning fully from scratch, in-flight, without prior knowledge or training, during 50 real flight training episodes. An approach for extending this method to the avoidance of walls, ceilings, and smaller obstacles is discussed and the expected performance when doing so on the Parrot Bebop 1 is assessed. Additionally, it is shown that this method can easily be extended to other quadrotors, by demonstrating this on the Parrot Bebop 2 quadrotor.

Nomenclature

α	Learning rate	$F_{ext,x}, F_{ext,y}, F_{ext,z}$	Estimated external forces [N]
γ	Discount rate	F_i	Thrust produced by rotor i [N]
δ_t	Temporal-Difference error	g	Standard gravitational acceleration [m/s^2]
ϵ	Exploration rate	I_{ss_t}, I_{aa_t}	Identity-indicator functions
θ	Pitch angle [rad]	I_{xx}, I_{yy}, I_{zz}	Moments of inertia [kg·m^2]
λ	Decay-rate for eligibility traces	K_b	Body lift coefficient
μ	Mean	$k_{D,x}, k_{D,y}, k_{D,z}$	Drag coefficients [N·s^2/m]
σ	Standard deviation	k_i	Rotor gain [N·s^2/rad]
$\tau_{ext,x}, \tau_{ext,y}, \tau_{ext,z}$	Estimated external torques [Nm]	l_i	Rotor torque gain [kg·m^2/rad]
ϕ	Roll angle [rad]	m	Quadrotor mass [kg]
ω_i	Rotational speed of rotor i [rad/s]	N_{episodes}	Number of episodes
a	Action	p, q, r	Body rates around x,y and z axes [rad/s]
A_{signal}	Strength of the signal	Q	Action-value function
b	Distance between opposite rotors [m]	r	Reward
c	Distance to a surface above [m]	R_{rotor}	Rotor radius [m]
d	Distance between adjacent rotors [m]	s	State
$d_{i,x}$	Distance from rotor i to the x-axis [m]	T_c	Thrust in ceiling effect [N]
$d_{i,y}$	Distance from rotor i to the y-axis [m]	T_g	Thrust in ground effect [N]
e	Ground effect model bias [N/kg]	T_∞	Thrust in free flight [N]
E_t	Eligibility traces	u, v, w	Speed in body x,y and z axes [m/s]
$F_{\text{drag},x}, F_{\text{drag},y}, F_{\text{drag},z}$	Drag forces [N]	z	Distance to a surface below [m]

*MSc Student, Control and Simulation Division, Faculty of Aerospace Engineering, g.j.vandam@student.tudelft.nl

I. Introduction

In recent years, Unmanned Aerial Vehicles (UAVs), and quadrotors in specific, have risen in popularity. Their low-cost, small volume and Vertical Take-Off and Landing (VTOL) capability have driven their application outside of aerial photography to many new applications, which may range from industrial inspection tasks to disaster response or package delivery applications [1].

The next step in development is expected to be increasing the level of autonomy for these quadrotors [2]. This could bring down the cost significantly and allow for larger scale deployments of these solutions. A key remaining challenge for this, similar to that for other UAV-platforms, is in-flight autonomous object detection and avoidance. Currently, solutions to this problem are usually either vision-based or range-sensor-based.

Vision-based methods rely on an onboard camera and the use of computer vision algorithms to detect and avoid obstacles. While this area of investigation and initial applications certainly look promising [3], it is also dependent on good lighting conditions. Furthermore, the implementation on quadrotors can be limited by the amount of computational resources available on such a platform.

Range-sensor solutions rely on the addition of some range sensor, like a laser rangefinder or ultrasonic sensor, for proximity measurement in order to detect obstacles. These solutions can achieve good accuracy and have been successfully implemented on quadrotors [4]. However, because of the added cost, weight and power usage of such a sensor, they can decrease flight time and limit their usability to only larger quadrotors.

This research investigates and proposes an alternative method for obstacle avoidance and detection, one that does not require the addition of any sensors but relies solely on measurements of the accelerometer and rotor controllers, both present on almost all quadrotors. This new obstacle avoidance method could be used as a stand-alone method for small quadrotors, thereby removing the need for additional sensors, thus saving cost and weight while increasing flight time. Alternatively, when being used as an addition to other obstacle avoidance solutions it can increase safety and reliability.

The detection of obstacles is based on the principle that the airflow around a quadrotor changes when the quadrotor is flying near a surface. A well-known example of this is the so-called ground effect, an increase in lift force close to a ground surface, an effect also seen in helicopters [5] and fixed-wing aircraft [6]. Similarly, a change in airflow occurs when a quadrotor is flying close to a wall [7], or ceiling [8]. The proposed method uses a Reinforcement Learning (RL) controller to detect obstacles based on these effects and take action to lead the quadrotor back to safety.

In this article, the initial development of a proof-of-concept of this low-cost method is described. This proof-of-concept is limited to the detection of large obstacles underneath a quadrotor, using the so-called ground effect. The results from this are used to assess and discuss the extension of this method to the avoidance of obstacles above and on the same level as the quadrotor.

The contribution of this research to the state of the art is twofold. This research offers a distinct addition to both fields it is part of: obstacle avoidance and reinforcement learning. First of all, this research contributes to the field of obstacle avoidance by offering an innovative method of obstacle avoidance. Furthermore, this new method is one that does not require the addition of any sensors, solely relying on the Inertial Measurement Unit (IMU) and RPM measurements already available in most UAVs.

Secondly, in the field of reinforcement learning, this research adds to the state of the art by its unconventional placement of the reinforcement learning agent within the control loop. Instead of being in full control, the RL agent is run in combination with an inner loop flight control following a predetermined flight plan. The UAV will follow the flight plan during most of the flight, but this can be overridden at any time by an intervention action of the reinforcement learning agent.

This article is structured as follows. First, a brief background on the obstacle-airflow interactions between a quadrotor and obstacles, and RL is given in section II. Then, in section III, the method in which obstacle-airflow interactions are used to detect obstacles is discussed. The reinforcement learning setup is discussed in section IV. The setup and results of the experiments carried out in simulation are discussed in section V. Similarly, the setup and results of the flight experiments are discussed in section VI. In section VII the extension of this obstacle avoidance method to walls, ceilings, and other quadrotors is discussed. The conclusions of this article are presented in section VIII, after which recommendations for future research are made in section IX.

II. Background

Background information is presented on the obstacle-airflow interactions between a quadrotor and obstacles. Furthermore, the basics of reinforcement learning, as well as the application for flight control of quadrotors and the specific reinforcement learning algorithm used for this research, are discussed.

A. Obstacle-airflow interactions between a quadrotor and obstacles

The dynamics of a quadrotor are greatly dependent on its aerodynamics, most importantly the airflow around the thrust-producing rotors. This airflow can be influenced by surfaces in proximity to the quadrotors. In the proposed method, this change in dynamics will effectively function as the source of information used by the reinforcement learning agent to determine the presence of obstacles. Therefore, it is of key importance that their effect on the quadrotor is known and can be estimated in flight. Furthermore, training in simulation will require a model of these effects.

1. Obstacles underneath the quadrotor

The influence of a horizontal surface underneath a rotor has been well researched in literature [5, 9]. Most of this research has been focused on helicopters, but in general, it can be noted that for all rotorcraft operating closely above a ground surface, the produced thrust increases [10].

To estimate this thrust increase for quadrotors, often the classical model for ground effect in helicopters is used. This analytical model is derived by Cheeseman and Bennett [5] and shown in Eq. 1. It is based on potential flow theory, under both the assumption that the helicopter is hovering and the assumption that the rotor can be modeled as a point source. The method of images is then used to derive the ratio between the thrust produced by a helicopter in ground effect (T_g) and the thrust out of ground effect (T_∞), as a function of the radius of the rotor (R_{rotor}) and the distance to the surface underneath (z).

$$\frac{T_g}{T_\infty} = \frac{1}{1 - (\frac{R_{\text{rotor}}}{4z})^2} \tag{1}$$

Validation using experimental measurements has since shown that the ground effect for a quadrotor is larger than predicted by this equation [10, 11]. Furthermore, these experiments showed that the influence of the ground effect in quadcopters was apparent up to heights of 5 times the rotor radius.

A new model, specifically for quadrotors was recently proposed by Sanchez-Cuevas et al. [10]. It was shown to represent their experimental results more closely than Eq. 1. This model accounts for the presence of multiple rotors by representing them not as one but as four sources. It assumes a quadrotor hovering above a ground surface with four co-planar rotors.

Furthermore, this new model accounts for an effect called the *fountain effect*, an additional increase in lift previously seen in tandem helicopters [12] and quadcopter experiments [11]. It can best be explained by looking at the CFD simulation of a simplified quadrotor in ground effect, as shown in Fig. 1. As expected, the wakes from each rotor spread out to the sides as they near the ground, however in the center area where the two airflows interact with each other a vortex ring appears. Due to this aerodynamic effect, an upwards force is applied to the body of the quadrotor, leading to a greater ground effect. This effect is represented in this new model by an empirical body lift coefficient [10].

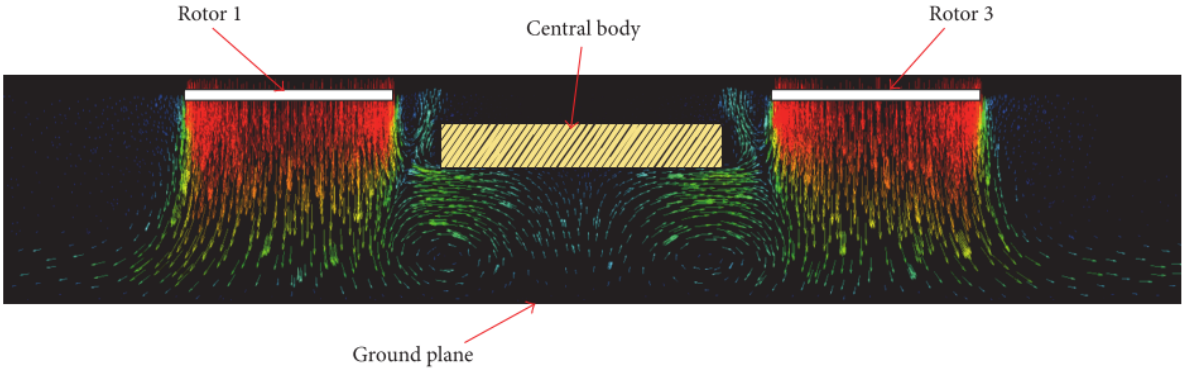


Fig. 1 CFD simulation of a simplified quadrotor model hovering close to a ground surface plane [10].

2. Obstacles above the quadrotor

Little research has been conducted on the influence of obstacles or surfaces above a quadrotor, the ceiling effect. The most relevant description and experiments are performed by Sanchez-Cuevas et al. [8]. In this research, the thrust produced by both a single rotor and a quadcopter, at varying distances to a ceiling surface, were measured and compared. These measurements were performed on a static test bench.

In Sanchez-Cuevas et al. [8], the increment in the thrust of a single rotor due to the ceiling effect is approximated by an analytical function similar to that of the ground effect shown in Eq. 1. The ratio between the thrust in (T_c) and out of the ceiling effect (T_∞) is given by equation 2, where c is the distance to the ceiling, and K_1 and K_2 are determined experimentally using ordinary least squares. A model for the increase in thrust for a complete quadcopter is not given, but it is shown that the relative increase is larger than that for a single rotor.

$$\frac{T_c}{T_\infty} = \frac{1}{1 - \frac{1}{K_1}\left(\frac{R_{\text{rotor}}}{c+K_2}\right)^2} \tag{2}$$

3. Obstacles on the same level as the quadrotor

The influence of large vertical surfaces like walls, on the same level as the quadrotor, has been described by Lee et al. [13], and Mckinnon [7]. Where the first article focuses only on reference tracking in the aerodynamic effects caused by such a wall, the second provides actual measurements of the effects caused by a wall. In that research, an Unscented Kalman Filter (UKF), based on a known model of the UAV, is used to estimate the external forces and torques near ground and wall surfaces whilst hovering.

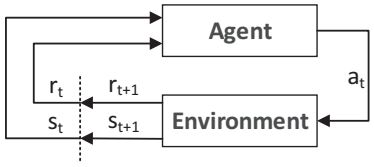
These measurements indicate a small external force away from the wall in the horizontal plane. Furthermore, a small external torque around the pitch and roll axes was noticed. Finally, a small downward external force was measured in the vertical plane. A clear increase in these three effects can be seen from a distance of 0.35 meter from the wall, about 3 times the rotor radius of the quadrotor that was used for that particular research.

B. Reinforcement learning

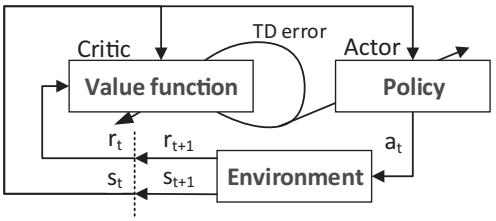
Reinforcement learning is a large and quickly developing field, containing a multitude of learning algorithms and architectures. In this research, reinforcement learning is used to find an optimal policy for the detection and avoidance of obstacles underneath a descending quadrotor. In the section below, the principles of RL and their application to flight control of multirotors is discussed. Furthermore, the method used for this research, Q-learning, is introduced.

1. Principles of reinforcement learning

Reinforcement learning is a computational approach to machine learning where an agent learns to maximize the cumulative rewards it receives when interacting with an environment. At each timestep t the agent chooses an action a_t, based on the current state s_t and its policy function, a mapping from state-space to action-space. The environment responds to this action by transitioning to state s_{t+1} and providing the agent with a numerical reward r_{t+1}, a process depicted in Fig. 2a.



(a) The agent-environment interaction in reinforcement learning.



(b) The actor-critic architecture.

Based on this reward and new state, a reinforcement learning algorithm specifies how the policy should be updated. The RL method does so with the goal of maximizing the sum of rewards received by the agent.

A special case of reinforcement learning methods are the actor-critic methods, in which the action selection and value estimation are split into two separate structures. The *actor* selects the actions and the *critic* estimates the (action-)value function and uses this to criticize the actor, as shown in Fig. 2b. This critique signal provided by the critic can be a scalar and is called the Temporal-Difference (TD) error (δ_t). It is used by the actor to adjust its policy.

2. Applying reinforcement learning to flight control of quadrotors

There are multiple examples of reinforcement learning techniques being applied to multirotors, and quadrotors in specific. Reinforcement learning agents have been used both for full control of a quadrotor [14] and for adjustment of a conventional controller [15]. Key challenges in applying reinforcement learning to flight control remain the challenge of safety, the challenge of robustness, the challenge of online efficiency and the challenge of sample efficiency.

3. Q-learning

One often used reinforcement learning method is Q-learning. Q-learning is a model-free, off-policy, Temporal-Difference method, which has three key implications. First of all, it does not require a model of the environment but instead learns solely from interacting with the environment. Secondly, the policy it uses to select actions, its behavior policy, is not necessarily equal to the estimation policy, the policy that is being improved to approach the optimal policy. Finally, methods of the Temporal-Difference learning class makes use of bootstrapping, using previously estimated values for the action-value function $Q_t(s, a)$ for its new estimate $Q_{t+1}(s, a)$, with the goal of approaching the optimal action-value function $Q^*(s, a)$.

Q-learning is often combined with eligibility traces, another key reinforcement learning mechanism, to obtain a more general method that may learn more efficiently. One implementation of this is Watkins's Q(λ) method [16]. The update equations for this method, using replacing traces, are given in equations 3 and 4.

The choice for Watkins's Q(λ) is based on a preliminary investigation where Monte Carlo, SARSA and Q-Learning methods were tested on a simplified version of the problem at hand. Results of this investigation suggest that Watkin's Q(λ) is best suited for problems with this particular setup.

$$E_t(s,a) = \begin{cases} \min\left(\gamma\lambda E_{t-1}(s,a) + I_{ss_t} \cdot I_{aa_t}, 1\right) & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a) \\ I_{ss_t} \cdot I_{aa_t} & \text{otherwise.} \end{cases} \tag{3}$$

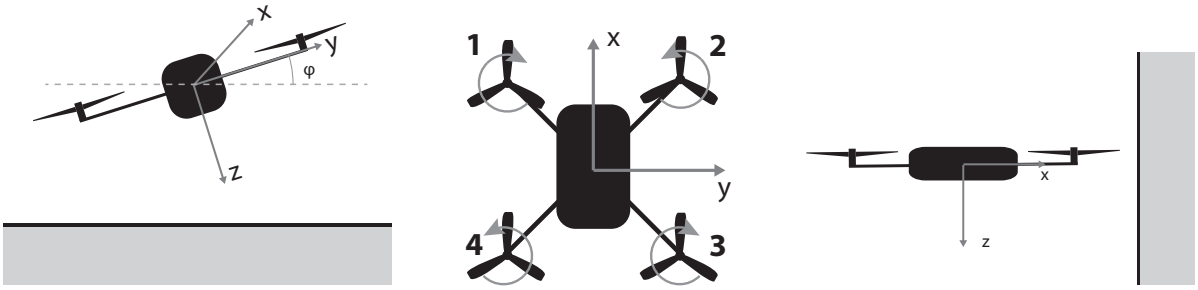
$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t E_t(s,a) \quad \text{for all s,a, with} \quad \delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{4}$$

III. Using obstacle-airflow interactions for obstacle detection

The result of the obstacle-airflow interactions on the quadrotor can be estimated by using a simple quadrotor model to estimate external forces and torques in and around all three axes. A procedure to do so is explained in detail for the external force in the vertical direction. This estimate is used to create a model for the ground effect, using measurement data gathered with the same drone that is used for the final flight experiments.

A. Frame of reference & equations of motion

For the purpose of this article, all derivations are performed within the quadrotor body frame and with the assumption of a rigid body. This right-handed coordinate frame is fixed to the quadrotor at the center of gravity, with the rotor axes pointing in the vertical z direction, the direction of thrust being negative, and the arms pointing in the x and y directions. A sketch of a quadrotor, showing this reference frame can be seen in Fig. 3.



(a) Quadrotor near a ground surface. **(b) Top view of a quadrotor.** **(c) Side view of a quadrotor near a wall.**

Fig. 3 Quadrotor body frame of reference.

The translational set of equations of motion for the quadrotor is given in Eq. 5. The gravitational acceleration is denoted as g, the mass of the quadrotor is m, the body attitude angles in pitch, roll, and yaw are given by θ, ϕ, ψ, the drag forces as F_{drag} and the thrust produced by each rotor i as F_i. Furthermore, the body speeds in forward, sideways and vertical direction are given by u, v, w, so the accelerations in the body frame are $\dot{u}, \dot{v}, \dot{w}$. Finally, p, q and r refer to the angular rates for pitch, roll, and yaw.

$$\begin{bmatrix} -mg\sin\theta + F_{\text{drag},x} \\ mg\sin\phi\cos\theta + F_{\text{drag},y} \\ mg\cos\phi\cos\theta - \sum_i F_i + F_{\text{drag},z} \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{5}$$

Assuming symmetry about the x and y axes of the body frame, as well as negligible rate damping, the rotational set of equations of motion is given by Eq. 6. Here $\dot{p}, \dot{q}$ and $\dot{r}$, refer to the angular acceleration rates, $d_{i,x}$ to the shortest distance from rotor i to the x-axis and $d_{i,y}$ to the shortest distance to the y-axis. The moments of inertia are given as I_{xx}, I_{yy} and I_{zz}. Finally, the torque produced by each rotor is assumed to be a function of rotor speed $T(\omega_i)$.

$$\begin{bmatrix} F_1 d_{1,x} - F_2 d_{2,x} - F_3 d_{3,x} + F_4 d_{4,x} \\ F_1 d_{1,y} + F_2 d_{2,y} - F_3 d_{3,y} - F_4 d_{4,y} \\ \sum_i T(\omega_i) \end{bmatrix} = \begin{bmatrix} I_{xx}\dot{p} \\ I_{yy}\dot{q} \\ I_{zz}\dot{r} \end{bmatrix} + \begin{bmatrix} -I_{yy}qr + I_{zz}qr \\ I_{xx}pr - I_{zz}pr \\ I_{zz}pq + I_{yy}pq \end{bmatrix} \tag{6}$$

B. Estimating external forces & torques

While the equations of motion given in Eq. 5 and 6 provide a model of the quadrotor in perfect free flight conditions, without any disturbances, wind or obstacle-airflow interactions, this is not always the situation in reality. To the contrary, these obstacle-airflow interactions are exactly the subject of interest. Therefore an external force, representing the difference between reality and the simplified quadrotor model, is added to each of the translational equations of motion; $F_{ext,x}, F_{ext,y}, F_{ext,z}$. Similarly an external torque is added to each of the rotational equations of motion: $\tau_{ext,x}, \tau_{ext,y}, \tau_{ext,z}$.

Since the external forces and torques represent all unmodelled dynamics, like the influence of obstacle-airflow interactions, these forces and torques can also be used to identify these dynamics. This can be accomplished by rewriting the equations of motion and solving for the external force or torque, using the known or approximated states. An example of this will be given below for the external force in the z-direction, as this force can be caused by obstacle-airflow interactions with obstacles underneath the quadrotor. The derivation starts with the translational equation of motion in the z-direction, as shown in Eq. 7.

$$mg\cos(\phi)\cos(\theta) - \sum_i F_i + F_{\text{drag},z} + F_{ext,z} = m(\dot{w} + pv - qu) \tag{7}$$

The IMU sensor of the quadrotor consists of a 3-axis accelerometer, providing accelerations, and 3-axis gyroscope, providing angular rates. The body speeds can either be derived from an external positioning system (like GPS, or OptiTrack), integration of body accelerations, or a combination of both using sensor fusion.

The onboard accelerometer measures in the body frame, however, it does not just measure $\dot{w}$. Instead it is influenced by the gravity vector, as it measures $\hat{\dot{w}}$, with $\hat{\dot{w}} = \dot{w} - g\cos(\phi)\cos(\theta)$. So if one substitutes this in Eq. 7 and solves for $F_{ext,z}$, the following estimate for the external force in the vertical direction is derived:

$$\frac{F_{ext,z}}{m} = \hat{\dot{w}} + pv - qu + \frac{1}{m}\sum_i F_i - \frac{1}{m}F_{\text{drag},z} \tag{8}$$

Now if the following model for the thrust produced by each rotor is assumed: $F_i = k_i\omega_i^2$ [17], this can be rewritten to:

$$\frac{F_{ext,z}}{m} = \underbrace{\hat{\dot{w}} + pv - qu}_{\text{from IMU}} + \sum_i \frac{k_i}{m} \underbrace{\omega_i^2}_{\text{from motors}} - \frac{1}{m}F_{drag,z} \tag{9}$$

To estimate $\frac{k_i}{m}$ an initialization procedure is conducted when the quadrotor is hovering in free flight, without closeby obstacles or disturbances. The external force $F_{ext,z}$ and drag $F_{drag,z}$ can then assumed to be zero. The equation above can then be solved for $\frac{k_i}{m}$ if either the assumption is made that $\frac{k_i}{m}$ is equal for all four rotors, or that the force produced by each of the four rotors during this initialization procedure is equal.

$$\frac{k_i}{m} = \frac{-\hat{\dot{w}} - pv + qu}{\sum_i \omega_i^2} \quad \text{assuming equal } k_i \qquad (10) \qquad \frac{k_i}{m} = \frac{-\hat{\dot{w}} - pv + qu}{4\omega_i^2} \quad \text{assuming equal thrust} \qquad (11)$$

The drag force can be estimated by performing another experiment in free flight, without closeby obstacles or disturbances. However, instead of hovering the quadrotor should now move up and down. Using the previously found values for $\frac{k_i}{m}$, Eq. 9 can be solved for $F_{\text{drag},z}/m$ for each measurement point of the experiment. It can be expected that the drag is of the form $|F_{\text{drag},z}/m| = k_{D,z}w^2$, where the direction of the force is opposite to the speed [18]. A function of this form can thus be fitted to the experiment data, and solved for $k_{D,z}$ using for example linear least squares, providing a model for $F_{\text{drag},z/m}$.

$$\frac{F_{ext,z}}{m} = \underbrace{\hat{\dot{w}} + pv - qu}_{\text{from IMU}} + \sum_i \frac{k_i}{m} \underbrace{\omega_i^2}_{\text{from motors}} - F_{drag,z}/m \quad \text{, with } F_{\text{drag},z}/m = \begin{cases} -k_{D,z}w^2 & \text{if } w > 0 \\ k_{D,z}w^2 & \text{otherwise} \end{cases} \qquad (12)$$

Similarly, the following equations can be derived to estimate the external forces in x and y direction.

$$\frac{F_{ext,x}}{m} = \underbrace{\hat{\dot{u}} + qw - rv}_{\text{from IMU}} - F_{\text{drag},x}/m \quad \text{, with } F_{\text{drag},x}/m = \begin{cases} -k_{D,x}u^2 & \text{if } u > 0 \\ k_{D,x}u^2 & \text{otherwise} \end{cases} \qquad (13)$$

$$\frac{F_{ext,y}}{m} = \underbrace{\hat{\dot{v}} + ru - pw}_{\text{from IMU}} - F_{\text{drag},y}/m \quad \text{, with } F_{\text{drag},y}/m = \begin{cases} -k_{D,y}v^2 & \text{if } v > 0 \\ k_{D,y}v^2 & \text{otherwise} \end{cases} \qquad (14)$$

Following a similar process, equations can be derived to estimate the external torque around the x, y, z axes. The resulting estimators are shown in Eq. 15, 16 and 17. To get to these estimators, it is assumed that the angular rates are relatively small and qr, pr and pq can be considered negligible. Furthermore, these estimators make use of $\frac{k_i}{m}$, this is the same parameter that is used for estimating the external force in the z-direction.

The moments of inertia around the x and y-axis can be estimated by performing an experiment in free flight, where the quadrotor performs sequentially a pitch and roll changing maneuver. If the external torques are assumed to be zero, Eq. 15 and 16 can be solved for I_{xx} and I_{yy}.

$$\frac{\tau_{ext,x}}{I_{xx}} = \dot{p} - \frac{1}{I_{xx}}\left[\frac{k_1}{m}\omega_1^2 d_{1,x} - \frac{k_2}{m}\omega_2^2 d_{2,x} - \frac{k_3}{m}\omega_3^2 d_{3,x} + \frac{k_4}{m}\omega_4^2 d_{4,x}\right] \qquad (15)$$

$$\frac{\tau_{ext,y}}{I_{yy}} = \dot{q} - \frac{1}{I_{yy}}\left[\frac{k_1}{m}\omega_1^2 d_{1,y} + \frac{k_2}{m}\omega_2^2 d_{2,y} - \frac{k_3}{m}\omega_3^2 d_{3,y} - \frac{k_4}{m}\omega_4^2 d_{4,y}\right] \qquad (16)$$

To estimate the external force around the z-axis, as shown in Eq. 17 it is assumed that the torque produced by one rotor can be approximated by $T(\omega_i) \approx l_i\omega_i^2$ [17]. The parameters $\frac{l_i}{I_{zz}}$ can be estimated by letting the quadrotor perform an initialization procedure in free flight, where $\tau_{ext,z}$ is assumed to be zero. If one then either assumes equal l_i, or equal torque, Eq. 17 can be rewritten to estimate the torque parameters.

$$\frac{\tau_{ext,z}}{I_{zz}} = \dot{r} + \frac{l_1}{I_{zz}}\omega_1^2 - \frac{l_2}{I_{zz}}\omega_2^2 + \frac{l_3}{I_{zz}}\omega_3^2 - \frac{l_4}{I_{zz}}\omega_4^2 \qquad (17)$$

Finally, with respect to the accuracy of the three external torque estimates, it is important to note that angular acceleration rates are often not provided directly by an IMU. Instead, they might need to be acquired by taking the derivative of the angular rates, which introduces noticeable noise into the estimation.

C. Challenges

While the method described above provides estimates of the external forces, they are not perfect estimates. There are three main challenges for achieving accurate estimates: inaccuracies in the estimated model, noisy measurements and delays in measurements.

Inaccuracies in the estimated model can be a result of incorrect assumptions, disturbances during initialization or changing system dynamics (e.g. a difference in mass due to a change in the payload). To cope with this, especially the latter, the estimation of the rotor gains is performed on board the quadrotor at the start of every flight. This is fully automated and takes only 2 seconds. In addition to being influenced by the gravity factor, the measured body accelerations $\hat{u}, \hat{v}, \hat{w}$ are also susceptible to sensor noise and vibrations in the body frame. One common source of such vibrations are the rotors. In this research, a 4th order Butterworth filter with a cutoff frequency of 3Hz is applied to all accelerometer and gyroscope measurements in an effort to remove sensor noise and rotor-induced vibrations. This choice is based on a frequency spectrum analysis and a trade-off between remaining noise and introduced delay.

Finally, delays in the measured states can lead to inaccuracies in the estimations. One of such delays is caused by the low-pass filter discussed above, which is expected to introduce a 140ms delay [19]. Therefore, the same filter is also applied to the speed, attitude, and rotor measurements in order to keep the signals synchronized. At a descend speed of 0.3m/s this thus corresponds to a distance of 4.2cm.

D. Model for the external force in the vertical direction when flying close to the ground

In order to train reinforcement learning agents to avoid obstacles underneath the quadrotor, a training environment is created. To make this environment as realistic as possible, a model of the net vertical force caused by the ground effect is created. This model is created based on measurements gathered during three separate measurement flights with the Parrot Bebop 1 drone, the same drone that is used for the flight experiments. In these measurement flights, the quadrotor descends from 1m height, with constant speed, to 1cm above a large surface of artificial grass. At each timestep, the external force in z-direction is estimated.

The model for the external force is based on the recently proposed formula for thrust increase in ground effect provided by Sanchez-Cuevas et al. [10]. In this formula, the increase in thrust near a surface underneath the quadrotor is not only dependent on the rotor radius R_{rotor} and distance to the ground z, but also on the distance between adjacent rotors d, the distance between opposite rotors b and an empirical body lift coefficient K_b. For this research, the model is extended with a bias e and rewritten to approximate $F_{ext,z}/m$, as can be seen in Eq. 18.

$$\frac{F_{ext,z}}{m} = -g\Big[\frac{1}{1-(\frac{R_{\text{rotor}}}{4z})^2 - R_{\text{rotor}}^2(\frac{z}{\sqrt{(d^2+4z^2)^3}}) - (\frac{R_{\text{rotor}}^2}{2})(\frac{z}{\sqrt{(2d^2+4z^2)^3}}) - 2R_{\text{rotor}}^2(\frac{z}{\sqrt{(b^2+4z^2)^3}})K_b} - 1\Big] + e \tag{18}$$

Non-linear least squares, using the Trust Region Reflective (TRF) algorithm and boundaries based on the physical properties of the quadrotor, is used to fit this function to the measurement data. This results in the following parameters: $R = 0.05$m, $d = 0.181$m, $b = 0.253$m, $K_b = 0.474$, $e = -0.150$. From Fig. 4 it can be seen that this function better replicates the measurement data than the classical formula from Cheeseman and Bennett [5].

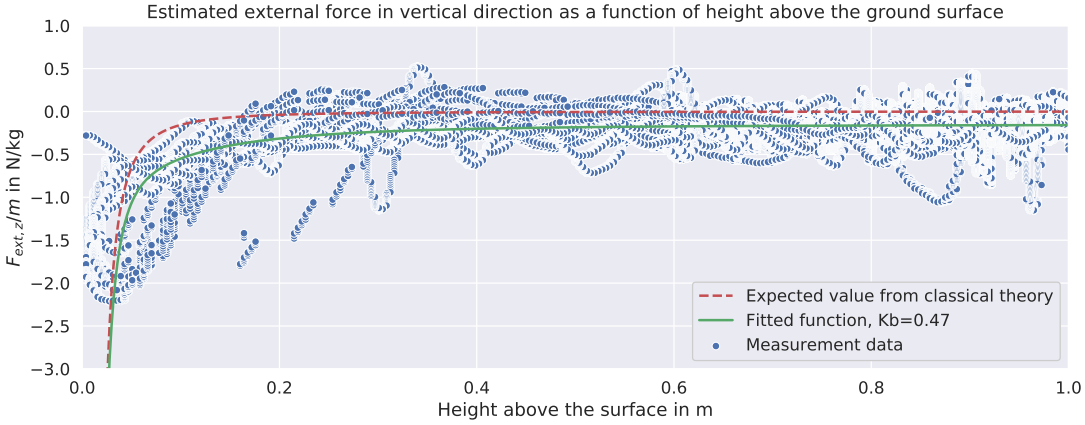


Fig. 4 Estimated external force in the vertical direction as a function of height above the ground surface.

IV. Reinforcement learning setup

In this section, the position of the reinforcement learning agent in the general control scheme are discussed. Furthermore, the states, actions, and rewards provided to the agent are introduced. Additionally, the initialization and termination conditions are highlighted. Finally, the exploration strategy, reinforcement learning algorithm, and hyperparameter are discussed.

A. General control scheme

The general control scheme of both the simulation and real flight experiment is depicted in Fig. 5. As can be seen from this figure, the RL agent is set up as an actor-critic method, where the actor is only updated after an episode has ended. The reason for this is practical, on the Parrot Bebop 1 the onboard computational capacity is limited. Therefore it is expected that the update rate of the RL agent will not be able to keep up with the frequency of the control loop, which runs at 512Hz. This high frequency is chosen to limit reaction time and give the agent the best chance of preventing collision with the obstacle underneath.

The state estimator that is indicated in the control scheme is the function that estimates $F_{ext,z}/m$. In the simulation experiments, this is estimated using the ground model discussed in section III.D. In the actual flight experiments, it is estimated based on the actual measured signals using Eq. 12.

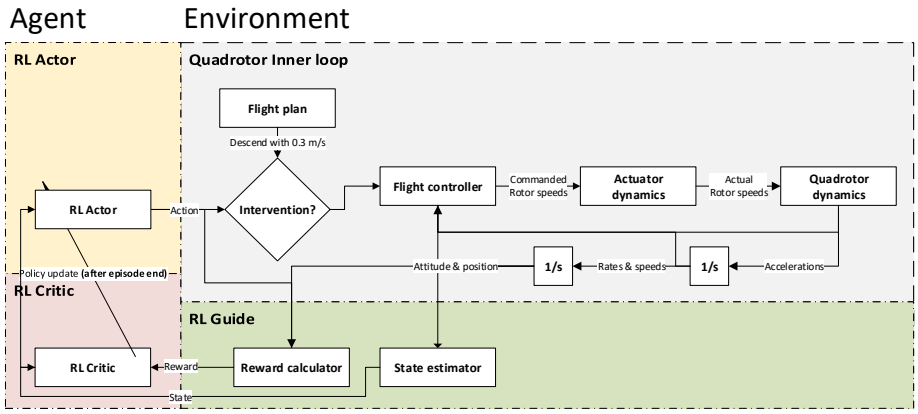


Fig. 5 High-level overview of the control scheme used in the experiments.

B. States & actions

Two states are available to the reinforcement learning agent; the current estimate of $F_{ext,z}/m$ and the action chosen in the previous timestep a_{t-1}. The estimate of external force $F_{ext,z}/m$ is discretized into 9 equally spaced bins, ranging from -0.2N/kg up to -1.00N/kg. The action in the previous timestep a_{t-1} is provided to the RL agent because the reward given by the environment depends as much on the previously chosen action as on the external force.

Based on these states, the agent then decides which action to perform. There are three actions available to the agent; No action ($a_{\text{no-action}}$), save (a_{save}) and hover (a_{hover}). When $a_{\text{no-action}}$ is chosen, the quadrotor continues on its original flight plan for one timestep, thus continuing the 0.3m/s descend. When choosing a_{save}, a short 1-second full-thrust command is sent to the quadrotor inner control loop. Finally, when choosing a_{hover}, the quadrotor inner control loop is given the command to hover for 0.5 seconds.

As mentioned above, two of the three potential actions take more than 1 timestep to execute. Hovering will take 256 timesteps and a save 512 timesteps. During this time the reinforcement learning agent is considered frozen, no new states are provided to the agent, no actions are picked by the agent and neither the action-value function, policy nor eligibility traces of the agent are updated. Any rewards that the agent might receive during this period are summed and provided to the agent for processing at the final timestep of the multistep action, together with the new state.

C. Rewards & termination

The goal of the reinforcement learning agent is to prevent the quadrotor from hitting any obstacles underneath. As such, the largest negative reward is given when the quadrotor comes to close to the obstacle below, as determined by the termination height $z_{\text{termination}}$, with height referring to the height above the obstacle. While this is referred to as a crash,

it must be noted that it is not an actual crash of the quadrotor, instead, the safety controller intervenes, preventing an actual collision, and leading the quadrotor back to safety. In all experiments, a termination height of 0.05 meter is used.

$$R(s,a) = \begin{cases} -2000 + R_a & \text{if } z \leq z_{\text{termination}} \\ R_a & \text{otherwise} \end{cases} \tag{19}$$

Furthermore, a negative action-based reward R_a is given when the agent intervenes, especially when the intervention is false. In this case, false is defined as outside of the area from which the ground effect can be measured, which was estimated to be 0.25 meter, corresponding to 4 times the rotor radius.

$$R_a(s|a = a_{\text{save}}) = \begin{cases} -500 & \text{if } z > 0.25 \\ -50 * \frac{0.25 - z}{0.25 - z_{\text{termination}}} & \text{if } z \leq 0.25 \end{cases}, \quad R_a(s|a = a_{\text{hover}}) = \begin{cases} -100 & \text{if } z > 0.25 \\ -25 & \text{if } z \leq 0.25 \end{cases} \tag{20}$$

There are three ways in which an episode can end. First of all, when the agent intervenes by performing a save, this is an episode-ending action. Secondly when the quadrotor comes to close to the surface underneath, as defined by Eq. 19. In either case, the final reward is processed by the agent and the episode ends. If neither of the two happens the episode automatically ends after 10 seconds.

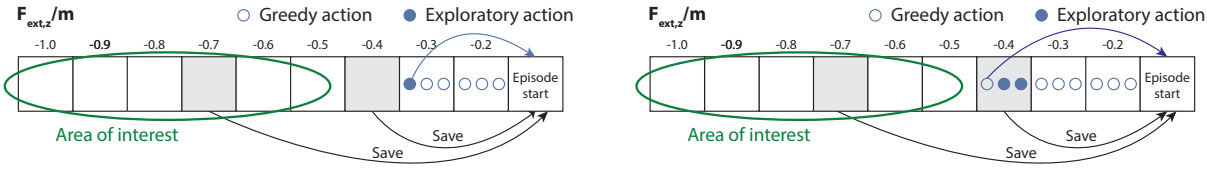
As can be concluded from these termination conditions, all episodes are finite, therefore it is not an absolute requirement to have a reward discount factor $\gamma < 1$. In the context of obstacle avoidance, it can even be argued that the negative impact of a collision in the future should not be discounted at all. Therefore, in this reinforcement learning problem, it is chosen to have $\gamma = 1.0$.

D. Exploration & initialization

Initial results with ϵ-greedy exploration strategies [20] showed that exploration was quite challenging for the RL agents, the number of state visits was highly skewed towards states with $F_{ext,z}/m$ close to zero. These states typically correspond to heights out of the ground effect. The states with more negative $F_{ext,z}/m$, that generally correspond to heights within the ground effect, were rarely visited by the agents. Further studies showed that the limited exploration of states close to the ground is inherent to the problem, for which there are two reasons.

First, the save action is episode-ending. As such, any exploration strategy which relies on random actions being picked runs the risk of ending the episode, and thus stopping further exploration, every time it does so. This is problematic when it is prone to happen early on in the episode, as it prevents the quadrotor from coming close to the ground. Therefore, the RL agent will rarely experience those states. An illustration of this can be seen in Fig. 6a.

Secondly, there are a lot of timesteps compared to the number of potential discrete states. This means that during the descent the agent will often be in the same state for many steps. This is a challenge for exploration because it requires many subsequent exploratory actions to be taken in order to reach another state. An example of this is depicted in Fig. 6b, where it must be noted that in reality it usually takes way more than three steps to progress from one state to the next. This reason relates back to the unconventional placement of the RL agent in the control scheme, as running the agent at a high frequency allows for quick interventions, but leads to this large number of timesteps per discrete state.



(a) Early ending of an episode due to exploration.

(b) Multiple consecutive exploratory actions required to go past a state where the greedy action is *save*.

Fig. 6 Two typical exploration challenges for this particular reinforcement learning problem. Note: only one dimension of the policy is shown.

To handle these issues, a combination of three different exploration strategies is implemented; exploring starts, episode-long exploratory actions, and epsilon-greedy exploration. An example of this strategy is shown in Fig. 7.

First of all, exploring starts are implemented [20]. From the perspective of the simulated or actual quadrotor, each episode always starts when hovering at 1-meter height, where it is given the command to descend with 0.3m/s. The RL

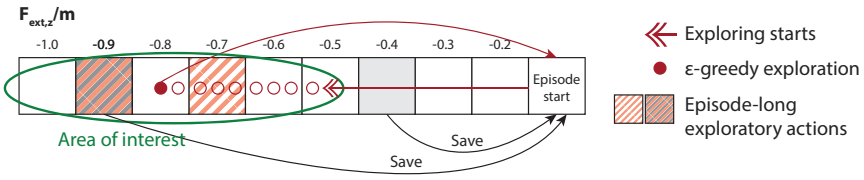


Fig. 7 Implemented threefold exploration strategy.

agent is however not initialized until height z_{expl} is reached. This exploratory starting height z_{expl} is randomly taken from the uniform distribution $[0.45, 1.0]$ before the start of each episode. This part of the exploration strategy encourages the exploration of the states that typically occur closer to the ground.

Secondly, episode-long exploratory actions are randomly generated before the start of each episode; for each state s in the state-space there is a chance $\epsilon_{\text{episode}}$ that random action a will be taken on every visit of that state, instead of the greedy action. By doing so, this approach can mitigate the challenge of requiring multiple consecutive exploratory actions to get to another state.

Finally, the methods above are combined with ϵ-greedy exploration at each timestep. The chance of picking a random action, instead of the greedy or episode-long exploratory action for that state, is then given by ϵ_{step}.

E. Hyperparameters

There are four key hyperparameters that determine the behavior of the RL agent; the learning rate (α), exploration rate at each step (ϵ_{step}), episode-long exploration ($\epsilon_{\text{episode}}$) and decay of eligibility traces (λ). The results from the previously mentioned preliminary investigation suggest that, when using Q-learning, a high, non-decreasing, exploration rate at each step ϵ_{step}, a λ between 0.1 and 0.5, and a learning rate α that decreases to a quarter of its initial value during the first half of the episodes, produce the best-performing agents.

Based on these preliminary results, 216 different combinations of these hyperparameters were selected for further investigation. Two grid searches were carried out in the simulation environment described below. The results from the first grid search are used to determine the best set of hyperparameters for training an agent from scratch. The results from the second grid search are used to determine the set of hyperparameters that are best when a previously trained agent is placed in a new, slightly different, environment. In each grid search, 2160 agents were trained during 500 training episodes, 10 for each of the 216 different hyperparameter sets.

The sets of hyperparameters that performed the best in these two grid searches are shown in table 1. For both sets, it is found that the best performance is achieved when $\epsilon_{\text{episode}}$ linearly decreases to zero during the first half of the episodes. The learning rate decreases as well, with the learning rate in episode i given by Eq. 21.

	λ	ϵ_{step}	α_0	$\epsilon_{\text{episode}}$
Initial training	0.1	0.01	0.5	$0.5 \rightarrow 0.0$
Continued training	0.1	0.01	0.1	$0.01 \rightarrow 0.0$

Table 1 Best hyperparameter sets found for initial and continued training.

$$\text{with } \alpha_i = \alpha_0 \frac{N_{\text{episodes}}}{N_{\text{episodes}} + i} \tag{21}$$

It is important to note here that the found sets of hyperparameters are the local optimum, the best available set from the 216 analyzed sets of hyperparameters. While these 216 options were selected carefully, based on both literature [21] and a preliminary investigation, there might exist a better set of hyperparameters globally.

V. Simulation

For the first phase of the experiments, a simulation environment is created that represents the in-flight environment as much as possible. To this end, the quadrotor inner vertical loop is replicated, the quadrotor dynamics are implemented and the ground effect is simulated. This simulated environment is then used to train and evaluate multiple agents, investigate the influence of noise and the number of required training episodes.

A. Experiment setup

First, the setup of the simulation environment and the experiments conducted within this environment are presented.

1. Creating the simulation environment

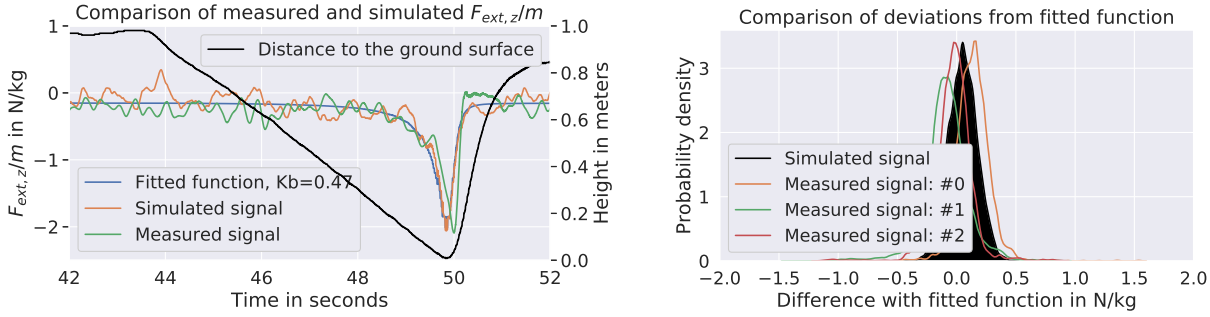
The quadrotor dynamics are implemented as described by the equations of motion given in Eq. 5 and 6. A key addition to this are the actuator dynamics. These provide the relationship between the commanded rotor speed $\omega_{cmd,i}$, from the inner control loop to the actual rotor speed ω_i. These are simulated using a 2nd order lower-pass Butterworth filter with a cut-off frequency of 15Hz. This actuator model is based on in-flight measurements from the system identification experiment described below.

The quadrotor inner vertical control loop is replicated in simulation to better reproduce the behavior of the quadrotor when it receives a command. This is especially relevant at the start of the episode when the quadrotor is hovering and receives the 0.3m/s descend command, and during interventions, when it receives a hover (a_{hover}) or save (a_{save}) command. The replication of the inner loop is based on the control scheme that is provided for the open-source flight control software *. The gains are determined by performing small system identification experiments with the Parrot Bebop 1 drone, where a step command is given on the vertical reference speed. The same step command is given in the simulation environment, and the gains are tuned based on the comparison.

The ground effect is simulated using the fitted function, as given in Eq. 18, and shown in Fig. 4. However, as can be seen from this figure, $F_{ext,z}/m$ is not a perfect estimator of height above the ground. As discussed in section III.C, the estimate is also influenced by inaccuracies in the estimated model, and noise or delays in measurements. As a result, the difference between the estimated external force at one timestep and the fitted model cannot be considered white noise. Instead, the $F_{ext,z}/m$ signal is quite smooth, as can be seen from Fig. 8.

In the simulation, an effort is made to replicate the stochastic deviations from the fitted function, but to keep the smoothness of the estimated signal. To do so, random normally-distributed noise is generated and filtered using a 2nd order Butterworth low-pass filter with a cutoff frequency of 1.8Hz. The mean ($\mu_{noise} = 0.055$) and standard deviation ($\sigma_{noise} = 1.392$) of this normal distribution are chosen such that the filtered noise distribution replicates the distribution seen in measurements. This can be confirmed by looking at Fig. 8, as the signals are similar, both in terms of smoothness and deviation from the fitted function.

A small difference can, however, be seen in the timing of the signal; the $F_{ext,z}/m$ signal measured in flight is delayed around 140 milliseconds. This difference can be explained by considering the 140ms delay introduced by the 4th order low-pass filter that is used to filter the in-flight measurements, as discussed in section III.C. While a low-pass filter is also used in the simulation, only the noise is filtered, not the underlying signal, as such no delay is introduced into the actual information-carrying signal, explaining the 140ms difference.



(a) $F_{ext,z}/m$ during one episode. **(b) Deviations from the fitted function.**

Fig. 8 Comparison of measured and simulated noise in the $F_{ext,z}/m$ state signal.

2. Training agents in simulation

Using the created simulation environment, and the set of hyperparameters for initial training discussed in section IV.E, 100 agents are trained in the simulation environment. Each agent is trained for 500 episodes, using the exploration strategy described in section IV.D. Their performance is evaluated during 100 fully greedy evaluation episodes. In these greedy episodes, there is no exploration and no learning. The comparison of agents is based on the average total reward

*`http://wiki.paparazziuav.org/wiki/Control_Loops#Vertical_loop`, as accessed on 23/01/2019

during these evaluation episodes, a measure of agent performance, and the number of episodes since the agent's policy last changed, a measure of the agent's stability. Based on this comparison the top performing agent is then selected as the *top agent*.

3. Investigating the influence of noise on agent performance

As discussed in sections III.C and V.A.1, the external force estimators are not perfect estimators of the distance to an obstacle underneath. Instead, they are stochastic signals, influenced by both inaccuracies in the estimated external force, like sensor noise, and external forces not resulting from obstacle-airflow interactions, like gusts. Furthermore, the amount of stochasticity in comparison to the underlying obstacle-airflow interaction is likely to depend on the quadrotor, obstacle characteristics and location of the obstacle with respect to the quadrotor. As such, getting an understanding of the effect of this stochasticity on the performance achievable by the RL agent is of key importance. Especially when considering the application of this object avoidance technique to other quadrotors or other types of obstacles.

Therefore, an experiment is carried out in which the noise on the $F_{ext,z}/m$ state is varied. The noise is varied from no noise, so a completely deterministic signal ($\sigma_{noise} = 0$), to 10 times as much noise as measured on the Parrot Bebop 1 drone for obstacles underneath ($\sigma_{noise} = 13.92$). The ratio between the strength of the signal (A_{signal}) and the noise (σ_{noise}) is captured by the Signal to Noise Ratio (SNR), as given in 22 [22]. Since this equation uses the strength of the signal, it also depends on the distance to the obstacle. In this case, the strength of the signal, $F_{ext,z}/m$ as caused by the ground effect, at 0.15m distance from the surface is used. This is approximately three times the rotor radius and exactly halfway between the estimated start of the ground effect area (0.25m), and the termination distance (0.05m).

$$SNR = \frac{A_{signal}^2}{\sigma_{noise}^2} \tag{22}$$

For each SNR level, 100 RL agents are trained for 500 episodes, using the hyperparameter set for initial training. Each agent is then evaluated during 100 fully greedy episodes. Both training and evaluation are carried out in the simulation environment, with the noise on the $F_{ext,z}/m$ state as defined by the respective SNR levels.

4. Investigating the number of required training episodes

In order to determine the influence of the number of training episodes on the performance of the agents, another experiment is carried out in simulation. Using the top initial hyperparameter set determined before, 100 agents are trained for each of the following number of episodes: $N_{episodes} = \{25, 50, 100, 500, 1000\}$. Evaluation is once again performed during 100 fully greedy episodes.

The results from this experiment are expected to help determine the feasibility of training an agent fully online during real flight, as the number of episodes that can be performed in real flight in a practical manner is limited.

B. Results

In the section below the results of the experiments carried out in the simulation environment are discussed.

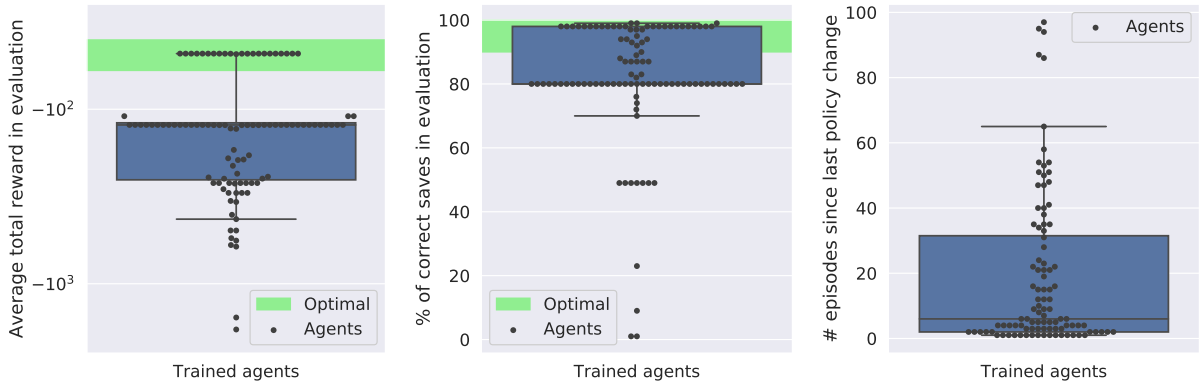
1. Training agents in simulation

The results of training 100 agents in simulation are shown in Fig. 9. They respectively show the performance of the agent, as measured by the average total reward (Fig. 9a) and the percentage of episodes resulting in a correct save (Fig. 9b). The number of episodes since the last policy change, as shown in Fig. 9c, is a metric indicating the stability of the agent.

From these figures, it can be seen that a number of agents exist with similar top performance. Further inspection shows that the similar performance of these agents is due to the fact that they have converged to a policy that is, for all practical purposes, equal. From this point forward, this policy shall be referred to as the *optimal policy*.

Further evaluation of this optimal policy is performed during 10,000 fully greedy evaluation episodes, each with uniquely random generated noise on the $F_{ext,z}/m$ signal. This results in an average total reward of -86.5 ($\sigma = 32.1$). Furthermore, 96.4% ($\sigma = 2.0$) of the episodes result in a correct save.

Of the 100 trained agents, 22% have found this optimal policy after training for 500 episodes. Given the level of performance achieved by this optimal policy, one could conclude that the RL setup works. Furthermore, 85% of all trained agents save the quadrotor successfully in ≥80% of the episodes.



(a) Boxplot showing the average total reward in the evaluation episodes. **(b) Boxplot of the percentage of evaluation episodes resulting in a correct save.** **(c) Boxplot showing the number of episodes since the last policy change.**

Fig. 9 Performance and stability metrics for the 100 agents trained in simulation.

In addition to providing insight into the performance distribution of agents trained in this environment, this experiment also set forth to select a *top agent*. This is the agent that will be used for the experiments in the flight phase. As further selection from 22 agents with the optimal policy based on performance is not evident, selection among these agents is based on the perceived stability of the agent.

The convergence of the selected *top agent* can be seen in Fig. 11. Furthermore, the final policy of this top agent can be seen in Fig. 10. For clarity, only the policy for states visited more than 10 times during the 500 training episodes are shown.

Indicated in the policy is the key intervention state (circled), the first intervention that the RL agent is likely to encounter when descending towards an obstacle underneath. Accounting for discretization, this part of the policy thus says: perform a save when: $-0.65\text{N/kg} < F_{ext,z}/m \leq -0.55\text{N/kg}$ and the previous action is $a_{no-action}$.

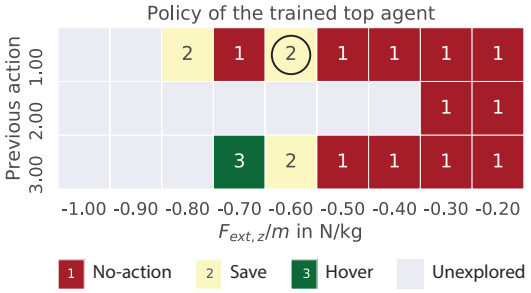


Fig. 10 Policy of the selected top agent, including the key interevention state.

From this experiment, the following conclusions can be drawn. First of all, in the simulation environment, there is one clear optimal policy, achieving high performance both in terms of average total reward (-86.5) and percentage of episodes resulting in a correct save (96.4%). Secondly, 85% of all the trained agents are able to save the quadrotor successfully in ≥80% of the episodes. Thirdly, only a small percentage of the agents (22%) converges to the optimal solution.

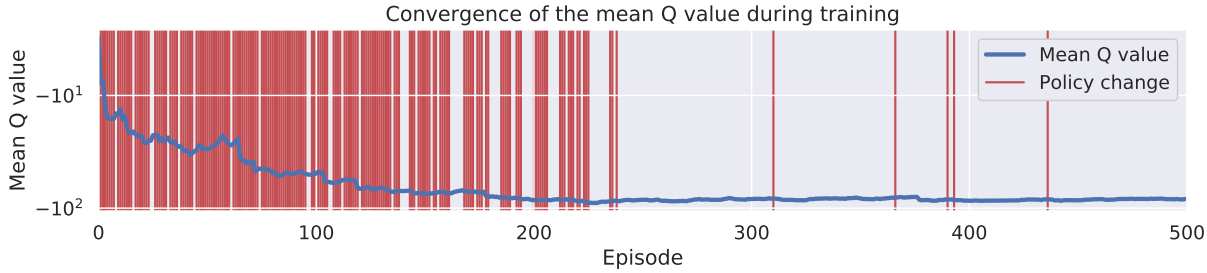


Fig. 11 Convergence of the selected top agent, as seen from the mean Q value and policy changes.

2. The influence of noise on agent performance

To investigate the effect of noise in the force and torque estimations, an experiment is conducted where 100 RL agents are trained with 26 different levels of noise added to the $F_{ext,z}/m$ state signal. The results of these experiments are shown in Fig. 12 and Fig. 13. These figures respectively show the influence of the noise on the average total reward

and the percentage of episodes resulting in a correct save. They do so as a function of the SNR level, which is taken at 0.15m distance to the obstacle underneath.

The expected rewards for three key agent behaviors are also shown in these figures. First of all, the line at -2000 indicates the expected reward for a crash. This would be the performance of an agent that always performs $a_{no-action}$. The only way in which an agent could achieve a performance worse than this would be if it performed hovering actions but still crashed every episode. Secondly, the line at -500 indicates the expected reward for a policy that always performs a save at the start of every episode, the *always-save* policy. Since the quadrotor would not yet be in the ground effect, this would always result in a false save. As such, this is the level of performance that can be achieved regardless of the noise. Finally, the line at -50 indicates the expected reward for an agent that is able to correctly save the quadrotor in every episode, but do so at the last minute, so when $z - z_{termination}$ is close to zero. Any agents with a performance better than this are thus able to save the quadrotor almost every episode and do it farther away from the obstacle.

The following observations can be made from these figures. Below -9dB, almost all agents have an average reward of -500 and 0% save rate, suggesting that they are unable to detect the presence of an obstacle underneath and thus converge to an *always-save* policy. Between -9dB and -1dB, the performance of all agents is actually worse compared to the *always-save* policy. This suggests that it is possible to detect obstacles, but that the reliability of doing so is lacking, resulting in some saves, but mostly crashes. Between -1dB and 2dB there are some top agents that are able to achieve a performance better than *always-save*. However, the stochasticity of the signal makes it difficult for the algorithm to find these better performing policies. Beyond 6dB almost all agents find a policy better than the *always-save* policy, their policies resulting in a correct save in ≥70% of the episodes.

Furthermore, the distribution of the agents' performances becomes smaller. Between a SNR of 12dB and 26dB, all trained agents are able to perform a correct save in almost every episode. Improvement is found in performing a save at larger distances to the obstacle underneath. Beyond 26dB, the performance is constant. Almost all agents are able to save the quadrotor far away from the obstacle in all episodes.

From these results, it can be concluded that using the current setup, a SNR ≥-1dB is required to outperform a trivial always-save policy and a SNR ≥2dB for most agents to do so. Beyond 6dB most agents are able to perform the obstacle avoidance task quite well, performing a correct save in ≥70% of the episodes.

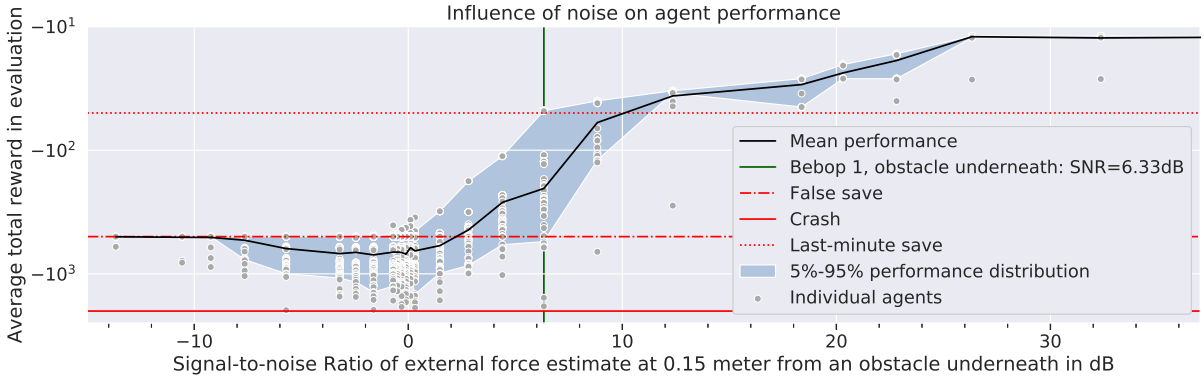


Fig. 12 Influence of noise on agent performance, as measured by the rewards.

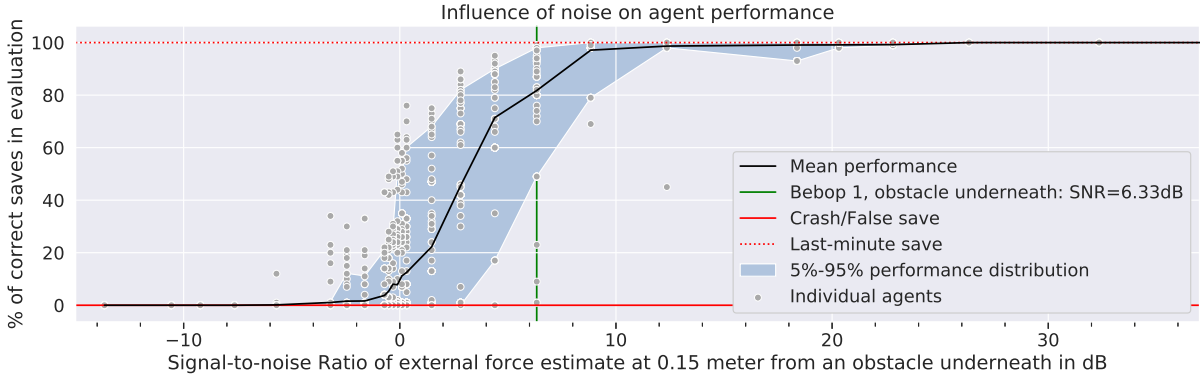


Fig. 13 Influence of noise on agent performance, as measured by the percentage of episodes resulting in a save.

3. The number of required training episodes

The results of the experiment to determine the number of required training episodes are shown in Fig. 14. Two key observations can be made. First of all, the performance of agents trained for 100 episodes is similar to those trained for 500 episodes or more. Secondly, while the mean performance of agents trained for 25 and 50 episodes is lower, in both cases there are still some agents which manage to learn the optimal policy. After training for only 50 episodes, 25% of the agents have learned the optimal policy. Considering the stochasticity involved in both the exploration and the state signal, this can be considered similar to the percentage of agents that have learned the optimal policy after training for 100 episodes (19%) and 500 episodes (22%).

It can thus be concluded that training for 100 episodes is sufficient when the purpose is to achieve the best performance for each of the trained agents. However, if one is only concerned with finding one agent with the optimal policy, it can be more efficient to train agents for only 25 or 50 episodes.

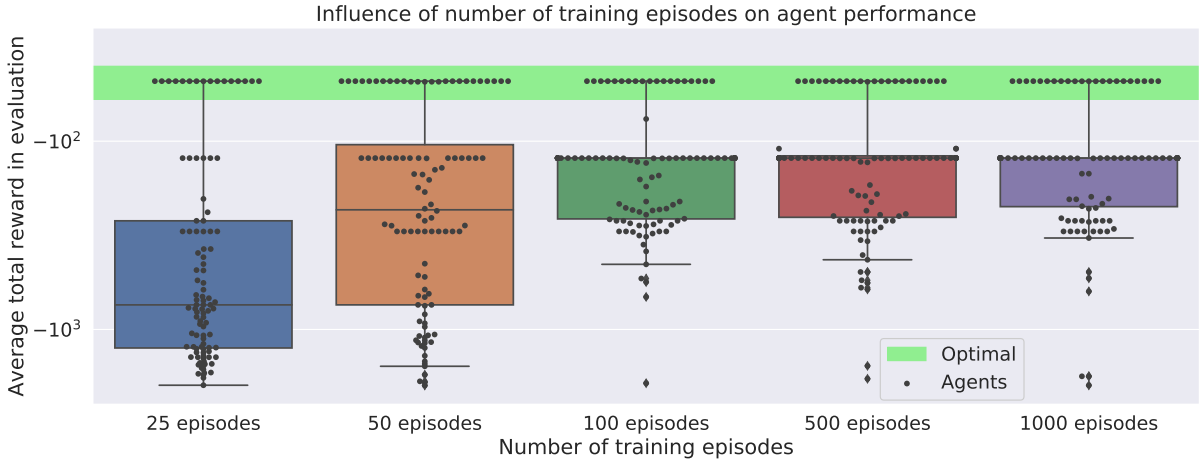


Fig. 14 Boxplot showing the influence of the number of training episodes on agent performance.

VI. Real flight

In the second phase, experiments are carried out in real flight, using a Parrot Bebop 1 quadrotor. First, the hardware and software setup that is developed for these experiments is discussed. Then the setups of three experiments that are carried out in real flight are presented; the evaluation of the top agent trained in simulation, continuing training of this top agent, and the training of an agent from scratch.

A. Experiment setup

First, the hardware and software setup, as well as the experiments carried out using this setup, are discussed.

1. Hardware and software setup

The quadrotor being used for the flight experiments is the Parrot Bebop 1 quadrotor shown in Fig. 15. This relatively inexpensive drone features 4 outrunner, brushless, motors, driving 4 rotors with a radius of 6.4cm. For the measurement of accelerations and angular rates, the Bebop relies on the MPU 6050 chip, which contains a 3-axis gyroscope and 3-axis accelerometer [24]. All flight experiments are carried out with the protection bumpers attached. With these bumpers, the drone weighs 420 grams.

Fig. 15 Parrot Bebop 1 quadrotor [23].

All experiments are carried out in the Cyberzoo of Delft University of Technology. This test area for ground robots and aerial vehicles spans 10m x 10m and is 7m high. It is equipped with the *Optitrack: Motive Tracker* optical tracking system, consisting of 24 cameras, enabling high precision positioning. This system provides the current position of the quadrotor, with 0.5cm accuracy at 120Hz, as GPS coordinates, via a wired connection to a laptop that functions as a ground station.

Onboard the quadrotor runs version 5.13 of the open source autopilot software Paparazzi. [†] The Paparazzi software suite also contains software for the ground station. The ground station is an Elitebook 8570w laptop. Communication between the quadrotor and ground station, including the position as provided by Optitrack, is done over Wi-fi.

Paparazzi is a modular platform on which additional functionality can be easily be added by creating custom modules. For the purpose of this experiment, such a custom Paparazzi module has been developed. This module is written in C and performs the following functions in-flight:

- **State estimator**: estimating $F_{ext,z}/m$ based on Eq. 12, using the filtered accelerations, motor speeds, rotational rates, body speeds, and thrust model. To enable this, the state estimator is also responsible for conducting a 2-second initialization procedure at the start of each flight. During this procedure the gains for the thrust model (k_i/m) are estimated, using Eq. 10. For the drag coefficient a constant value of $k_{D,z} = 0.271\text{N} \cdot \text{s}^2/\text{m}$ is used.
- **Reinforcement learning actor**: at every step of the episode, this actor is presented with the reward and the state (consisting of the estimated $F_{ext,z}/m$ and the previous action), based on which the actor will select one of the three actions (no-action, hover or save). It does so, based on its exploration strategy and the policy it determines at the start of each episode. When the action is to intervene (a_{hover} or a_{save}), this action is passed to the Paparazzi inner vertical control loop as a command. When the chosen action is $a_{\text{no-action}}$, the inner vertical control loop will follow the flight plan, which in all of the conducted experiments commands it to descend with 0.3m/s.
- **Safety controller**: if the quadrotor comes to close to the obstacle underneath $z < z_{\text{termination}}$, the safety controller will end the episode and send the quadrotor back to its start point of 1 meter above the obstacle. This part of the module thus has access to the height above the obstacle, as provided by the Optitrack system. This information is however not shared with the RL actor or critic, except through the rewards they receive.
- **Data logger**: storing all relevant variables for later analysis, including measured accelerations, speeds, motor speeds, estimated $F_{ext,z}/m$ and chosen actions. All these variables are written to a .csv file every timestep.

Other than the custom module, the quadrotor uses only existing Paparazzi modules models for its flight control. For stabilization in the horizontal plane, usage is made of the Incremental Nonlinear Dynamic Inversion (INDI) module [25]. For speed and positional control, the Paparazzi default inner vertical and horizontal control loops are used. It must be noted however that the *HOVER_KD* gain for the vertical loop has been increased from 100 to 600, in order to ensure closer tracking of the desired descent speed. This is required to maintain constant descent speed when nearing the obstacle underneath, as the ground effect tends to reduce the descent speed. Furthermore, the importance of thrust control with respect to the roll and pitch axes was increased from 10 : 1000 to 100 : 1000. The complete control scheme for the flight experiments can be found in appendix A.

The reinforcement learning critic was custom developed for these particular experiments and runs on the ground station laptop. It was developed in Python and uses the codebase that was originally developed for the simulation experiments. At the end of every episode, the critic retrieves the data log file of that episode from the quadrotor over FTP. For each step in this episode, Watkin's Q(λ) algorithm is then used to calculate the change in action-value function (Q), based on the state, selected action, and resulting reward. To do so, the critic needs to distinguish between greedy actions and exploratory actions. The exploratory actions being either episode-long exploratory actions or an exploratory action taken at one specific timestep by the ϵ–greedy exploration. The critic uses this distinction to reset the eligibility traces when needed. After processing every step in the episode, the result is sent to the quadrotor over the Ivy bus, a text-based (ASCII) Publish-Subscribe protocol that communicates over the local Wi-Fi network.

2. Evaluation of the top agent trained in simulation

The first experiment carried out in real flight is the evaluation of the top agent trained in simulation. The purpose of this experiment is twofold. First of all, it serves to validate that the simulation environment is a reasonably accurate representation of the real flight environment. This can then be used to argue that results from experiments carried out in simulation, like the influence of hyperparameters, noise and number of training episodes, hold for real flight as well. Secondly, it provides a first performance assessment in terms of the obstacle avoidance capability of this novel method.

In the experiment, for each of the evaluation episodes, the Bebop 1 quadrotor is commanded to descend from 1m height with $0.3m/s$. Onboard the quadrotor runs the top RL agent trained in simulation. All episodes are carried out as fully greedy evaluation episodes, so no exploring starts, no exploration and no learning by the agent, just the evaluation of the found policy. Twenty evaluation episodes are carried out, using the artificial grass surface of the TU Delft CyberZoo as the obstacle underneath.

[†] `http://wiki.paparazziuav.org/wiki/Main_Page`, as accessed on 02/04/2019

3. Continuing training in real flight

Secondly, an experiment is carried out where training of the top agent from the simulation experiment is continued in real flight. This is done during 100 training episodes, using the same exploration strategy as used in the simulations, and the hyperparameters found to be best for helping an agent adjust to a slightly different environment. After training, the agent is once again evaluated during 20 fully greedy evaluation episodes. In all flights, the artificial grass surface functions as the obstacle underneath.

4. Training from scratch in real flight

Finally, using the knowledge gained during previous experiments, an experiment is conducted where multiple RL agents are trained fully from scratch, during a real flight. Training is conducted during 50 episodes, using the same exploration strategy as used in the simulations, and the hyperparameters found to be best for the initial training of agents. Evaluation is performed during 20 fully greedy evaluation episodes. In these flights the artificial grass surface functions as the obstacle underneath, however, if this yields promising results, an additional evaluation is performed using an alternative obstacle, a 75cm x 53cm x 17.5cm box placed underneath the quadrotor.

This experiment serves to determine whether a RL agent can be trained fully in flight to avoid obstacles underneath it, using the obstacle-airflow interactions caused by this obstacle. If this is indeed possible, it shows the potential for the extension of this method to other quadrotors, or other types of obstacles, without requiring simulation environments or models of the obstacle-airflow interaction.

B. Results

Using the hardware and software setup described in section VI.A.1, several experiments are carried out in real flight, using the Parrot Bebop 1 quadrotor. First, the top agent from the simulation phase is evaluated in flight. Secondly, an experiment is conducted in which training of the previously found top agent is continued in flight. Finally, an experiment is conducted where multiple agents are trained during real flight, without any prior knowledge of the environment.

1. Evaluation of the agent trained in simulation

Of the 20 in-flight evaluation episodes, the top RL agent trained in simulation is able to perform a correct save in 80%. As can be seen from Fig. 16, 5% of episodes result in a false save and 15% in a crash. This results in an average reward of -350. When comparing this to the performance of this agent in simulation; with an average reward of -86.5 (σ = 32.1), and 96.4% (σ = 2.0) of episodes ending in a correct save; it is clear that the performance in real flight is significantly worse. Based on these results, it is estimated to be around 20% worse.

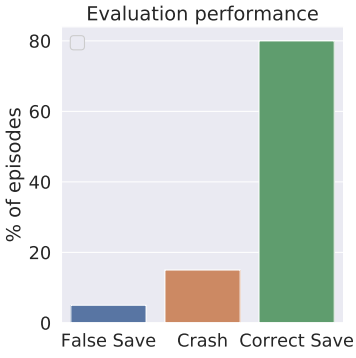


Fig. 16 Evaluation results of the agent trained in simulation.

An explanation for this difference in performance could be the 140ms delay in the $F_{ext,z}/m$ signal, introduced by the low-pass Butterworth filter, discussed in section III.C. As this delay is present in real flight, but not in the simulation, obstacles are expected to be detected slightly later in real flight, potentially leading to worse performance.

While the results above provide the first assessment of the obstacle avoidance capability of this new method, there is a second goal of the experiment; validating that the simulation environment is an accurate representation of the real flight environment. There are two key observations that are especially relevant to this. First of all, the observation that the agent is able to perform well in the real flight environment, while only having being trained in the simulation environment. This suggests that a good performance in simulation corresponds to a good performance in real flight. Secondly, the exact level of performance achieved by this policy in the simulation environment is different from the performance in real flight.

While the first observation speaks to validate the simulation model, the second raises the question of whether the optimal policy found in simulation is also the optimal policy in real flight. To answer this question, and further validate the simulation environment, this is investigated in the next two flight experiments. This is done by seeing if an even better policy can be found by continuing training of the top agent in real flight, or training a new agent from scratch in real flight.

2. Continuing training in real flight

To check if the performance of the top agent can be further improved, training is continued in flight for 100 episodes, using the previously determined set of hyperparameters deemed best for adjusting to a slightly different environment. Due to the stochasticity involved, both in the exploration and in the $F_{ext,z}/m$ signal, this experiment is conducted five times.

In all five training runs it can be seen that the agent experimented with different policies. After the 100 training episodes, 4 of the 5 agents have converged to the same policy they started with, the top policy from the simulations. The other agent has ended up with the policy shown in Fig. 17. Accounting for discretization this policy says: perform a save when: $-0.75\text{N/kg} < F_{ext,z}/m \leq -0.65\text{N/kg}$ and the previous action was $a_{\text{no-action}}$. Compared to the optimal policy found in simulation, the RL agent thus intervenes later, at a more negative $F_{ext,z}/m$.

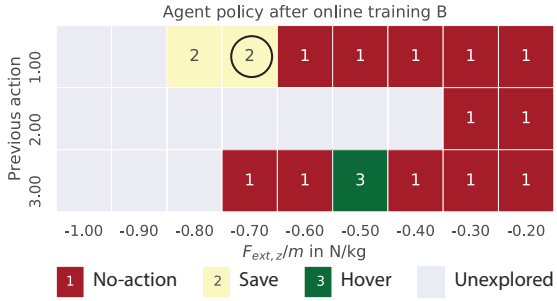


Fig. 17 Alternative policy found after continuing training online.

To test whether this policy is better than the initial policy, it is evaluated during 20 fully greedy episodes. The performance is clearly worse. Only 5% of the episodes results in a correct save, 95% result in a crash. This also becomes clear from the average total reward, which is -1902.

This result thus supports the hypothesis that the optimal policy found in simulation, is the optimal policy in real flight as well. Furthermore, this implies that the performance shown in Fig. 16, performing a correct save in 80% of the episodes, is the best achievable performance within the current setup.

3. Learning from scratch in real flight

Since the simulation experiments show that even after training for only 50 episodes, some RL agents have found the optimal policy, an attempt can be made to train a RL agent fully from scratch, during a real flight. Of the 5 agents trained in real flight, one has converged to a policy with the top performance, as can be seen from the evaluation results shown in Fig. 18. During the 20 evaluation episodes, this agent correctly performed a save 16 times (80%), crashed 2 times (10%) an performed a false save 2 times (10%).

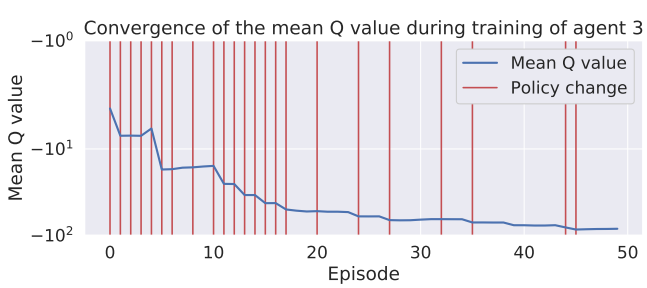
Fig. 18 Evaluation performance of the 5 agents trained from scratch in flight.

Further inspection of this particular agent shows that its policy, as shown in Fig. 19b, is very similar to the optimal policy found in simulation. Both have the same key intervention, a save when $-0.65\text{N/kg} < F_{ext,z}/m \leq -0.55\text{N/kg}$ and the previous action is $a_{\text{no-action}}$. It can thus be argued that for practical purposes this policy is equal to the optimal one found in simulation. As such, this speaks for validation of the simulation environment.

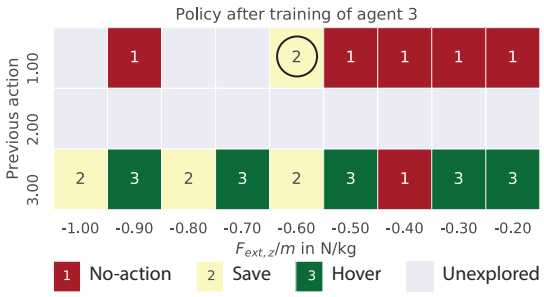
This agent is also evaluated using a 75cm x 53cm x 17.5cm box as the obstacle underneath, instead of the artificial grass surface. During the 20 fully greedy evaluation episodes, this results in the agent performing a correct save 15 times (75%) and a crash 5 times (5%).

From a practical perspective, it takes 15 minutes to train a single agent and evaluation takes another 7 minutes. This includes the flight time for the 50 episodes, the initialization procedures and the replacement of batteries (3x). If enough batteries are available, training 5 agents sequentially could thus be conducted in less than 2 hours. This is important when considering the extension of this method to other quadrotors, or other types of obstacles.

Overall, it can thus be concluded from this experiment that it is possible to successfully train an agent fully in flight, in only 50 episodes to detect and avoid obstacles underneath a descending quadrotor. Upon approaching an obstacle, a detect-and-avoid accuracy of at least 80% can be achieved by a reinforcement learning agent with the optimal policy.



(a) Convergence of the third agent trained from scratch.



(b) Policy of the third agent trained from scratch.

Fig. 19 Convergence and policy of the third agent trained from scratch.

There is however no guarantee that every agent that is trained will always converge to this optimal policy. Results from simulation and real flight suggest that only 20-25% of agents will converge to the optimal policy. Therefore, it is recommended to train multiple agents, evaluate them and select the best.

VII. Extension to other types of obstacles and other quadrotors

While the results discussed above look promising, they of course only demonstrate the obstacle avoidance capability for one specific drone, the Parrot Bebop 1, and only for large obstacles underneath the quadrotor. To assess the potential of this method as a general method of obstacle avoidance, the extension to other quadrotors and other types of obstacles must be considered.

A. Extension to other obstacles

When considering the extension of this method to other obstacles, one could consider extending it to detect large obstacles above, like a ceiling, obstacles on the same level, like a wall, or to smaller or otherwise different obstacles. The feasibility of using this method to detect walls and ceiling surfaces will depend on the strength of the effects caused by these surfaces and the amount of noise and other disturbances present in the estimated forces and torques. For the Parrot Bebop 1, an initial estimate of the SNR of the effects caused by the ceiling and wall is made as part of this research.

A large surface above the quadrotor, e.g. a ceiling, is known to cause an external force in the vertical direction [8]. The effect can be estimated using the measurement data from Sanchez-Cuevas et al. [8]. Adjusting for the different rotor radius, a first estimate of the external force caused by the ceiling would be $F_{ext,z}/m \approx -0.32$N/kg. If similar noise as seen in the presence of obstacles underneath is then assumed, this results in a SNR of 8.63dB. Since this is larger than the SNR seen for obstacles underneath, it implies that these obstacles are easier to detect. One must, however, note that the force, in this case, is pulling the quadrotor towards the surface above, in contrast to pushing the quadrotor away from a surface underneath. As such, the intervention action might be more difficult or require a sooner intervention, thereby perhaps reducing the performance.

To estimate the feasibility of detecting large vertical surfaces on the same level, like walls, an experiment is conducted using the Parrot Bebop 1 drone. In this experiment, a 1m wide, 2.05m high screen is placed inside the CyberZoo. The quadrotor then hovers at a height of 1.5 meter at varying distances from the wall, ranging from 1-meter to up to 1-centimeter distance. The measurement data is used to construct plots similar to Fig. 4, showing the estimated forces and torques versus the distance to the wall.

The clearest influence of the wall can be seen in the torque around the y-axis, the effect is however relatively small compared to the noise. An initial analysis estimates the SNR to be -4.15dB. For such a SNR it is not expected that agents will able to find a policy that results in correct saves, as can be seen from Fig. 20.

It must, however, be noted however that this is only an initial analysis. It is expected that by analyzing the effects in more detail, using better sensors, or using more than 1 external force or torque as a state, the performance might be further improved. Furthermore, the performance could also be improved by using some of the other recommendations discussed in section IX.

The extension to smaller or otherwise different types of obstacles is something that might be done incrementally. As mentioned in the results, the RL agent trained in this research is already able to detect a large box. By continuing training with smaller obstacles underneath, perhaps in combination with refining the used discretization, the detection limits can be found and improved.

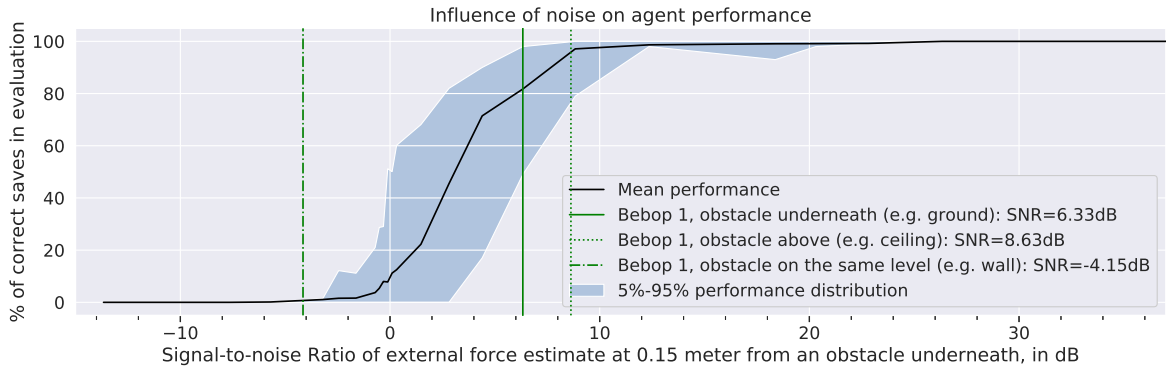


Fig. 20 Estimated Signal-to-Noise ratios and resulting performance, for the detection of walls, ceiling and ground surfaces.

Overall, the following conclusions can be drawn with respect to the extension of this method to other obstacles. The extension of this obstacle avoidance method to the detection of surfaces above the quadrotor, e.g. ceilings, is expected to achieve a similar or even better performance as for surfaces underneath. The extension to surfaces on the same level as the quadrotor, e.g. walls, is expected to require some significant improvements to the SNR of the estimated forces and torques, or to the usage thereof, before a similar performance can be reached. The extension to smaller or otherwise different types of obstacles requires further research, and possibly a more dense discretization of the states.

B. Extension to other quadrotors

Finally, the extension to other quadrotors. The whole method has been set up such that it requires little prior knowledge of the quadrotor. For example, no assumptions are made about the mass of the quadrotor and the moments of inertia are calculated in flight. Furthermore, the rotor gains k_i/m are automatically estimated during initialization procedures. The following steps are suggested to implement this obstacle avoidance method on another quadrotor:

1) Perform an experiment where the quadrotor first moves up and down without any nearby obstacles and then descends to 5cm above a ground surface three times. Use the gathered measurement data to estimate the drag coefficient as discussed in section III.B, and if need be, adjust the discretization bounds for the estimated external forces and/or torques.
2) Train multiple agents in flight, using a large horizontal surface as the obstacle underneath. Suggested is to train at least 5 agents for 50 episodes, using the hyperparameters discussed in section IV.E.
3) Evaluate all agents during a number of evaluation episodes, at least 20 is suggested, and pick the best agent for implementation.

While following these steps should provide a good basis, there is no guarantee on the performance of the optimal obstacle avoidance policy. This will especially depend on the ratio between noise and obstacle-airflow interaction effects within the state signal, the SNR, as discussed in section V.B.2.

To demonstrate the extension of this method to other quadrotors, an experiment is performed using the Parrot Bebop 2 drone. This successor to the Bebop 1 has a larger frame, larger rotors (R_{rotor} = 7.5cm) and new motors [26]. Furthermore, in the experiment the Bebop 2 is flown without the bumpers, thereby potentially altering the airflow, and thus the obstacle-airflow interactions. Other than this, the setup is as described in section VI.A.4. Two RL agents are then trained to avoid obstacles underneath this quadrotor. After 50 training episodes, one of the agents has already learned a policy with similar performance as was achieved on the Parrot Bebop 1 drone. In the 40 fully greedy evaluation episodes it is able to perform a correct save in 80% of the episodes, 2.5% resulted in a crash and 17.5% in a false save. It can thus be concluded that the obstacle avoidance method can be extended to other quadrotors in only a few steps.

VIII. Conclusion

In this research, a first step in the development of a novel obstacle avoidance method for quadrotors is taken. A reinforcement learning agent is successfully trained to detect and avoid obstacles underneath a descending quadrotor.

To accomplish this, a simple quadrotor model is introduced and used to estimate external forces and torques around all three axes. Measurement flights with a Parrot Bebop 1 quadrotor are performed in order to model one of these

estimators, the external force in the vertical direction ($F_{ext,z}/m$) as a function of distance to the ground. These models are used to create a simulation environment in which RL agents can be trained. This simulation environment represents the actual flight environment as much as possible, replicating not only the quadrotor equations of motion but also its inner loop flight control. Furthermore, the states ($F_{ext,z}$ and the previous action), actions ($a_{\text{no-action}}$, a_{hover} and a_{save}), and rewards of the environment are defined. In this environment, several experiments are then performed in simulation. These are performed using the Watkin's Q(λ) reinforcement learning algorithm, a custom threefold exploration strategy, and hyperparameters determined during an extensive grid search.

From these results, it can be concluded that the estimated external force in the vertical direction is a good indicator for the presence of large surfaces underneath. Furthermore, a reinforcement learning agent can be trained to use this estimated external force to detect and avoid obstacles underneath. However, due to the stochastic nature of the problem, not all agents will find the optimal policy. Multiple agents should thus be trained to ensure an optimal solution is found.

The best agent trained in simulation is evaluated in real flight, during an experiment with the Parrot Bebop 1, running the Paparazzi open-source flight software, inside the Delft University of Technology CyberZoo. The results show that the agent is able to save the quadrotor from hitting the obstacle underneath in 80% of the episodes. An attempt is made to further improve this performance by continuing training online, but this yields no further improvement, suggesting that the optimal policy found in simulation is also the optimal policy in real flight.

Finally, it is shown that it is possible to train an agent fully from scratch during a real flight. This is accomplished by training 5 agents during 50 episodes each, without prior knowledge or training in simulation. Of these 5 agents, one found the optimal policy, confirming the conclusion that multiple agents should be trained in order to ensure that the optimal policy is found. Since this did not require any simulation beforehand, it suggests that a RL agent can be trained to avoid other types of obstacles or obstacles on another quadrotor in real flight in a similar fashion.

For the extension of this method to other quadrotors, a procedure is presented. Furthermore, this extension is demonstrated using the Parrot Bebop 2 drone. Showing that similar performance can be achieved on another quadrotor in only a few short steps, without requiring any specific quadrotor model or simulation.

Furthermore, an approach for extending this method to the avoidance of walls, ceilings, and smaller obstacles is discussed and the expected performance when doing so on the Parrot Bebop 1 is assessed. For surfaces above the quadrotor, e.g. ceilings, the method is expected to achieve a similar or even better performance as for surfaces underneath. The extension to surfaces on the same level as the quadrotor, e.g. walls, is expected to require some improvements to the SNR of the estimated forces and torques, or to the usage thereof, before a similar performance can be achieved.

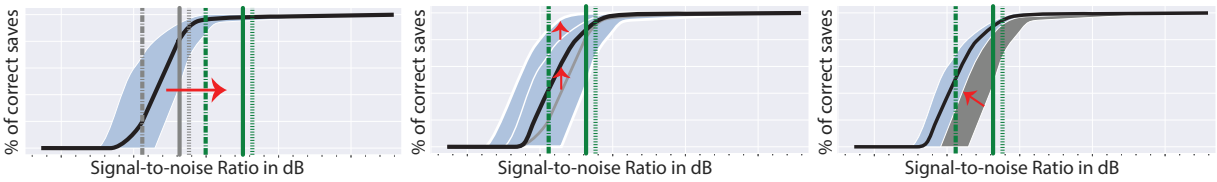
Overall, it can be concluded that it is possible to use reinforcement learning and obstacle-airflow interactions for the detection and avoidance of large obstacles underneath a Parrot Bebop 1 quadrotor. Furthermore, it is expected that this method can be extended to other quadrotors, as well as to large obstacles or surfaces above the quadrotor.

IX. Recommendations for future research

While initial success for this new obstacle avoidance method is shown, there are many points on which the method can be improved. First of all, this method can be extended to other quadrotors, or other types of obstacles, e.g. walls, ceilings, or smaller obstacles, as discussed in section VII.

Secondly, the estimations of the external forces and torques might be improved, thereby increasing the signal-to-noise ratios, the effect of which is shown in Fig. 21a. This could be done by improving the estimator, reducing noise or correcting for other disturbances. Improving the estimator could be accomplished by using more accurate models for the produced thrust, drag or quadrotor dynamics. Reducing the noise in the estimator could be achieved by using better sensors, improved filtering, or by reducing latencies in the underlying measurements. Finally, there can be other effects, like wind, turbulence or the airflow of other aerial vehicles, causing external forces and torques on the quadrotor. Methods might be developed by which they can be identified and corrected. In the case of wind and turbulence, a correction might reduce false positives or false negatives, improving the performance of the obstacle avoidance method. In the case of other aerial vehicles, identification might extend the applicability to dynamic obstacle avoidance.

Finally, the way the estimated external forces and torques are being used might be improved. This could lead to a better optimal policy, the expected effect of which is shown in Fig. 21b, a better distribution of performance among trained agents, as shown in Fig. 21c or a combination of the two. Two ways in which this might be accomplished are; the combination of multiple estimators and improvements to the reinforcement learning setup. For obstacle-airflow interactions expected to result in more than one external force or torque, e.g. those caused by a wall, providing multiple estimators as a state to the RL agent could improve performance. Another way the proposed method might be improved is by improving the reinforcement learning setup. Of the many ways in which this could potentially be accomplished,



(a) Improvements resulting in a better signal-to-noise ratio. **(b) Improvements resulting in a better optimal policy.** **(c) Improvements resulting in a better performance distribution.**

Fig. 21 Potential influence of proposed improvements on agent performance.

two interesting potential improvements can already be recommended for future research. First, the representation of the action-value might be improved; either by increasing the discretization density, or using another function approximator such as a Support Vector Machine (SVM) or a neural network. Secondly, the reward structure of the current setup might be improved, for example by providing a positive reward for a correct save, with a larger distance to the obstacle resulting in a larger positive reward. Initial experiments suggest that this can increase the percentage of the agents finding the optimal policy in 50 episodes, from 20-25% to 40%.

To conclude, future research could both improve upon the current obstacle avoidance capabilities of this low-cost method, and extend the method to other quadrotors and types of obstacles. Thereby increasing the potential of this method as a primary obstacle avoidance method for small quadrotors, or as a secondary obstacle detection method for larger quadrotors.

References

[1] DroneDeploy, "Commercial Drone Industry Trends," Tech. Rep. March, DroneDeploy, 2017.

[2] Zhang, T., Li, Q., Zhang, C.-S., Liang, H.-W., Li, P., Wang, T.-M., Li, S., Zhu, Y.-L., and Wu, C., "Current trends in the development of intelligent unmanned autonomous systems," *Front Inform Technol Electron Eng*, Vol. 18, No. 1, 2017, pp. 68–85. doi:10.1631/FITEE.1601650, URL `http://dx.doi.org/10.1631/FITEE.1601650`.

[3] Alberto, F. B.-f., and Gabriel, O., "Visual Navigation for Mobile Robots : A Survey," *Journal of intelligent and robotic systems*, Vol. 53, No. 3, 2008, pp. 263–296. doi:10.1007/s10846-008-9235-4.

[4] Stowers, J., Hayes, M., and Bainbridge-Smith, A., "Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor," *2011 IEEE International Conference on Mechatronics*, 2011, pp. 358–362. doi:10.1109/ICMECH.2011.5971311.

[5] Cheeseman, I. C., and Bennett, W. E., "The Effect of the Ground on a Helicopter Rotor in Forward Flight," *Aeronautical Research Council Reports and Memoranda*, Vol. 3021, 1955, p. 12. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.226.6371`.

[6] Rozhdestvensky, K. V., "Wing-in-ground effect vehicles," *Progress in Aerospace Sciences*, Vol. 42, No. 3, 2006, pp. 211–283. doi:10.1016/j.paerosci.2006.10.001.

[7] Mckinnon, C. D., "Data Driven , Force Based Interaction for Quadrotors by Data Driven , Force Based Interaction for Quadrotors," Ph.D. thesis, University of Toronto (Canada), 2015.

[8] Sanchez-Cuevas, P. J., Heredia, G., and Ollero, A., "Multirotor UAS for bridge inspection by contact using the ceiling effect," *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, 2017, pp. 767–774. doi:10.1109/ICUAS.2017.7991412.

[9] Rossow, V. J., "Effect of Ground and/or Ceiling Planes on Thrust of Rotors in Hover," *Nasa Technical Memorandum*, Vol. 86754, 1985.

[10] Sanchez-Cuevas, P., Heredia, G., and Ollero, A., "Characterization of the aerodynamic ground effect and its influence in multirotor control," *International Journal of Aerospace Engineering*, Vol. 2017, 2017. doi:10.1155/2017/1823056.

[11] Sharf, I., Nahon, M., Harmat, A., Khan, W., Michini, M., Speal, N., Trentini, M., Tsadok, T., and Wang, T., "Ground effect experiments and model validation with Draganflyer X8 rotorcraft," *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014*, 2014, pp. 1158–1166. doi:10.1109/ICUAS.2014.6842370.

[12] Griffiths, D. A., "A study of dual-rotor interference and ground effect using a free-vortex wake model," *American Helicopter Society 58th Annual Forum, Montreal, Canada, June 11-13*, 2002.

[13] Lee, D., Awan, A., Kim, S., and Kim, H. J., "Adaptive Control for a VTOL UAV Operating Near a Wall," *AIAA Guidance, Navigation, and Control Conference*, 2012, p. 4835. doi:10.2514/6.2012-4835, URL http://arc.aiaa.org/doi/10.2514/6.2012-4835.

[14] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M., "Control of a Quadrotor with Reinforcement Learning," *IEEE Robotics and Automation Letters*, Vol. 2, No. 4, 2017, pp. 1–8. doi:10.1109/LRA.2017.2720851, URL http://arxiv.org/abs/1707.05110.

[15] Junell, J., Mannucci, T., Zhou, Y., and Kampen, E., "Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning," *AIAA Guidance, Navigation, and Control Conference*, 2016, pp. 1–16. doi:10.2514/6.2016-1387, URL http://dx.doi.org/10.2514/6.2016-1387.

[16] Watkins, C., and Holloway, R., "Learning From Delayed Rewards," , 1989. URL https://www.researchgate.net/publication/33784417.

[17] García Carrillo, L. R., Dzul López, A. E., Lozano, R., and Pégard, C., "Modeling the Quad-Rotor Mini-Rotorcraft," *Quad Rotorcraft Control*, Springer, London, 2013, pp. 23–34. doi:10.1007/978-1-4471-4399-4, URL http://link.springer.com/10.1007/978-1-4471-4399-4_2.

[18] Salih, A. L., Moghavvemi, M., Mohamed, H. A. F., and Gaeid, K. S., "Flight PID controller design for a UAV quadrotor," *Scientific Research and Essays*, Vol. 5, No. 23, 2010, pp. 3660–3667. URL http://www.academicjournals.org/SRE.

[19] Manal, K., and Rose, W., "A general solution for the time delay introduced by a low-pass Butterworth digital filter: An application to musculoskeletal modeling," *Journal of Biomechanics*, Vol. 40, 2007, pp. 678–681. doi:10.1016/j.jbiomech.2006.02.001, URL www.elsevier.com/locate/jbiomech.

[20] Sutton, R. S., and Barto, A. G., *Reinforcement Learning : An Introduction*, 2015th ed., MIT press, 2015.

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

[22] Kourepenis, A., Borenstein, J., Connelly, J., Elliott, R., Ward, P., and Weinberg, M., "Performance of MEMS inertial sensors," *IEEE 1998 Position Location and Navigation Symposium (Cat. No.98CH36153)*, IEEE, 1998, pp. 1–8. doi:10.1109/PLANS.1998.669861, URL http://ieeexplore.ieee.org/document/669861/.

[23] RChelicoptershop.nl, "Parrot Bebop Drone red," , 2019. URL https://www.rchelicoptershop.nl/parrot-bebop-drone/parrot-bebob-drone-red.

[24] Paparazzi UAV, "Bebop," , 2019. URL https://wiki.paparazziuav.org/wiki/Bebop.

[25] Smeur, E. J. J., Chu, Q., and De Croon, G. C. H. E., "Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 3, 2015, pp. 450–461. doi:10.2514/1.G001490, URL http://arc.aiaa.org.

[26] Parrot, "Bebop 2 vs Bebop Drone comparison - Parrot Blog," , 2016. URL http://blog.parrot.com/2016/01/12/comparison-bebop-2-vs-bebop-drone/.

A. Appendix: Setup flight experiments

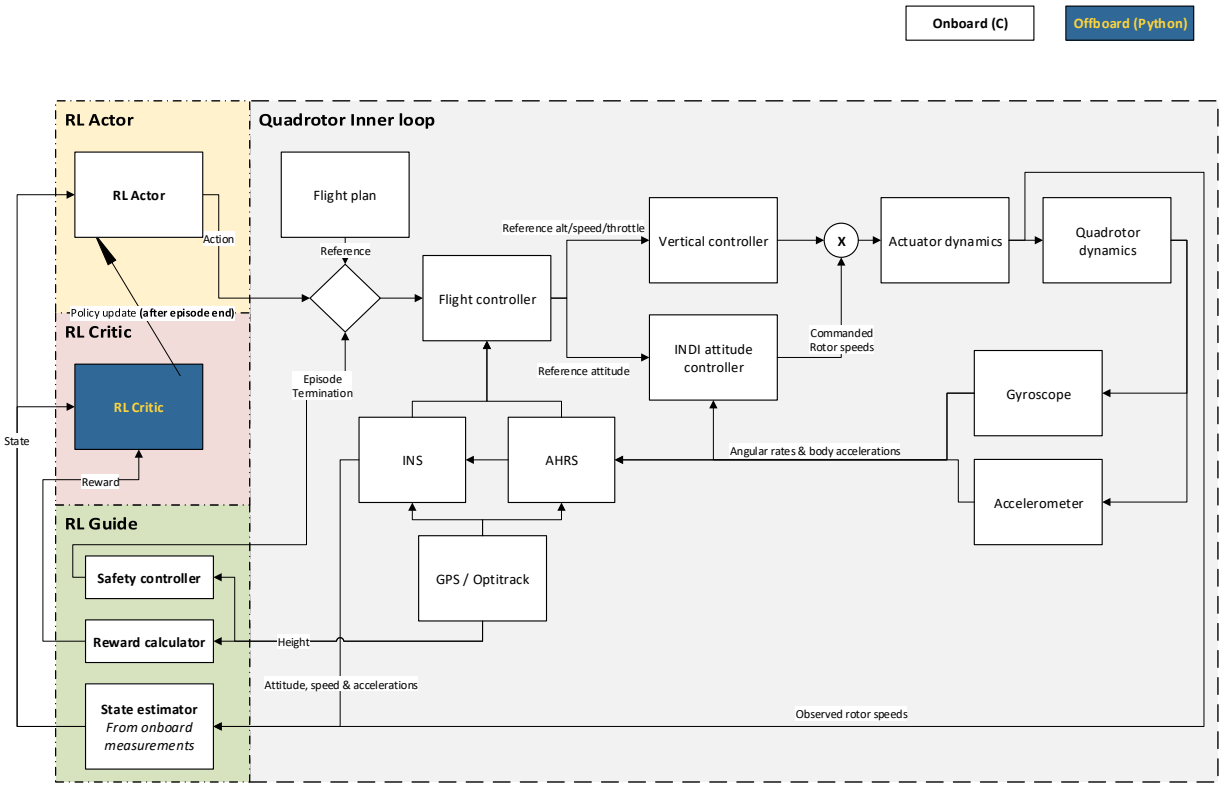


Fig. 22 Extended control scheme for the flight experiments

3-2 Conclusions

As the preceding article covers the core of the conducted thesis research, it also answers two of the three research questions (SQ2 & SQ3). The first research question being: *Given this state of the art, can a reinforcement learning-based quadrotor control system avoid obstacles, using these obstacle-airflow interaction effects?* The short answer to this question is: yes, at least for large obstacles underneath a quadrotor, as demonstrated in the article. A more comprehensive answer can, however, be provided by considering the underlying subquestions:

SQ2.1 *How can the reinforcement learning-based control system best be setup?*
From the development of a reinforcement learning-based control system, tasked with the detection and avoidance of obstacles underneath a descending quadrotor, several conclusions can be drawn with respect to the setup of such a system. First of all, the estimated external force in vertical direction can be a good indicator of the presence of large surface underneath the quadrotor. It is, however, susceptible to noise, both from other disturbances and sensor noise. While sensor noise can be reduced by filtering, this will introduce delays, thereby increasing the reaction time. Secondly, an actor-critic architecture can be used to limit onboard computational requirements for the quadrotor. Finally, it is shown that the unconventional placement of the RL agent within the control loop can work, but that it does introduce some challenges for exploration. It must be noted that these conclusions are in addition to the ones already drawn in the preliminary investigation, regarding the setup of such a system, as discussed in section 5-5 of the thesis. Together these conclusions describe several insights into the setup of such a control system.

SQ2.2 *How can the reinforcement learning agent be trained and tested in simulation?*
Several conclusions can be drawn with respect to training the reinforcement learning agent in simulation. First of all, it is important that in the simulated training environment not only the external force and torque estimations are simulated, but also the noise that is present within these estimations. For an accurate representation of real flight, the estimation should correspond in stochasticity, distribution, and smoothness of the resulting signal. Furthermore, it is expected to be beneficial to mimic any delays that are present in the real flight estimations. In addition to the creation of a simulation environment, training will also require a good set of hyperparameters. The sets of hyperparameters found to be best for this particular setup, when using Q-learning, are presented in section IV.E of the article. When using these hyperparameters, only a limited number of training episodes are required for some agents to find an optimal policy. However, not all agents will find the optimal policy, therefore instead of training a few agents for a long period, it can be more efficient to train a larger number of agents, evaluate and pick the best one.

SQ2.3 *How can this agent be trained and tested in a real-life experiment?*
In the article, it is demonstrated that a reinforcement learning agent can be trained and tested in real flight. This is accomplished by using the open-source flight controller Paparazzi, a Parrot Bebop 1 quadrotor, a commercial optical tracking system and custom software running both onboard the quadrotor and on a ground station. Either the continuation of training for an agent already trained in simulation or training from

scratch is possible this way. Furthermore, it is shown that only a small number of episodes is required to do so. This is an advantage, as the number of episodes that can be conducted in one flight is limited by the time required per episode and the battery capacity. As such, to increase the likelihood of finding a good policy, it is recommended to train a large number of agents for a limited number of episodes, instead of a few agents for many episodes. These agents can then be tested by running fully greedy episodes, with exploration and learning disabled, thereby providing a good measure of the obstacle avoidance performance.

SQ2.4 *What is the performance of the trained and tested control system?*
The prototype obstacle avoidance method described in the article is able to correctly detect and avoid a large obstacle underneath a descending quadrotor in 75-80% of the episodes. One could argue that this is a good performance, especially when considering the very limited cost of adding this method to a quadrotor. This of course only constitutes the performance with respect to large obstacles underneath. An initial assessment of the expected performance of this method for other obstacles yields the following results; for the avoidance of surfaces above the quadrotor, e.g. ceilings, a similar or even better performance is expected. The performance when considering the avoidance of surfaces on the same level as the quadrotor, e.g. walls, is expected to be worse. As such improvements, some of which are already discussed in the article, would be required to reach a satisfactory performance for detecting all obstacles.

While the second subquestion has been positively answered by the development and demonstration of the prototype method in the article, the social relevance is largely determined by subquestion three: *How can the knowledge gained from developing this reinforcement learning-based quadrotor control system be applied to other quadrotors?* As the ease and cost associated with the extension to other quadrotors will determine whether this method indeed is the low-cost obstacle avoidance method that is desired.

A procedure for the extension to other quadrotors is described in section VII.B of the article. It has already been demonstrated, as it was used to successfully train a reinforcement learning agent to avoid obstacles underneath, on the Parrot Bebop 2 drone. Furthermore, it is shown that this can be accomplished in only two 50 episode long training runs. The cost and time required to do so were negligible, showing that this system can easily be applied to other quadrotors. It must be noted however that this does not guarantee that the obstacle avoidance method will achieve satisfactory performance on every quadrotor. As shown in section V.B.2 of the article, the performance that can be achieved is highly dependent on the signal-to-noise ratio of the external force estimate. Since this ratio depends on, among others, the quadrotor rotor radius, sensor noise, and rotor-induced vibrations, the performance is expected to be quadrotor specific.

Chapter 4

Literature survey

In the literature survey, the state of the art in fields related to this research will be investigated. This will be done by studying both the basics from textbooks and the state of the art from recent publications

In section 4-1, existing obstacle avoidance methods will be analyzed in order to allow for comparison with the new method under investigation. This relates directly to research question 1.1 (SQ1.1).

In section 4-2, literature regarding the obstacle-airflow interactions between a quadcopter and obstacles will be investigated. This will be done by studying papers describing these effect and their (experimental) results. By doing so, research question 1.2 can be answered (SQ1.2).

Thirdly, in section 4-3, the basics of Reinforcement Learning will be described, as well as the state of the art. This will help answer research question 1.3 (SQ1.3)

Finally, based on this literature study, research question 1 (*What is the state of the art in the related fields?*) can be answered, this will be done in section 4-4.

4-1 Obstacle Avoidance

Obstacle avoidance is the task of preventing unwanted collisions. Within the domain of robotic control, it can be seen as one level above obstacle detection and one level below motion planning, as is illustrated in figure 4-1.

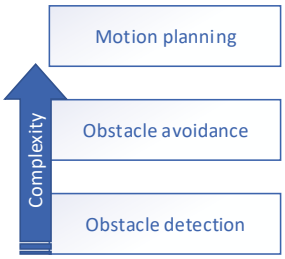


Figure 4-1: Obstacle avoidance as part of motion planning.

Obstacle detection is thus usually a prerequisite for obstacle avoidance. One must first be aware of the presence of an obstacle to take action and prevent a collision. Experience has shown that accurate obstacle detection at sufficient range is one of the main challenges for UAVs. Because of this, and because the obstacle avoidance method under investigation in this research relies on a new obstacle detection method, obstacle detection will be one of the main focus points of this literature survey.

Once an obstacle is successfully detected, a UAV or other robot can take action to prevent an unwanted collision. What is important here is that the mitigating action should be sufficient to avoid the collision, but also safe and not too restrictive in preventing the robot or UAV in pursuing its goal.

Motion planning can be considered the next step; based on the detected obstacles an obstacle-free path is calculated, which is then used as guidance. The planning involved is the key distinction between motion planning and obstacle avoidance. Motion planning is out of the

scope for this literature survey, but for a survey of different motion planning methods for UAVs the reader is referred to [38].

In section 4-1-1, different methods for obstacle detection on a UAV will be discussed. In section 4-1-2 different techniques within the category most relevant to this research will be highlighted. Then in section 4-1-3, different actions to prevent collision will be discussed, here the focus is purely on quadcopters.

4-1-1 Obstacle detection

Obstacle detection for UAVs is typically characterized by three main challenges: limited information about the environment, limitations on the placement and capabilities of onboard sensors, and uncertainty in vehicle state and sensor data.

On a general level, obstacle detection methods can be divided into three categories: Map-based, map-building and mapless. [3]

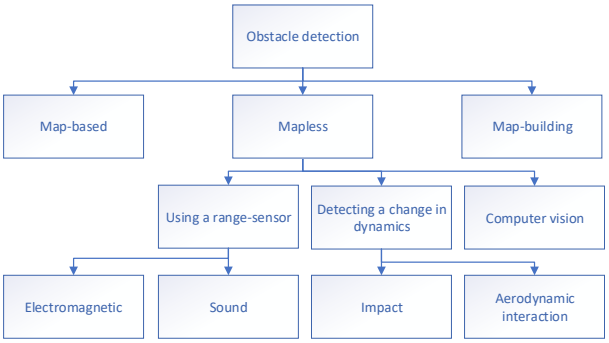


Figure 4-2: Methods of obstacle detection.

Map-based methods rely on a preexisting map containing either the location of obstacles or the location of obstacle-free paths. Because of this, they can rely on an internal (lookup) system to detect and avoid obstacles.

A map-based method can be a great obstacle detection method. There are however three main challenges for a map-based method. First of all, getting or creating a detailed map to make such a method work can be a great challenge. Secondly, it requires the environment to be time-invariant. Finally, the UAV needs to be able to localize itself on the map. This is especially challenging in indoor environments, where an accurate GPS signal is usually not available.

Map-building methods first explorer the environment to build a map and then use this map for navigation. By doing so, a map-building method can take away the first challenge in map-based methods, as discussed above. A prerequisite for this is however that the method is able to build a map that is accurate. This is is often a challenge for UAVs, because the accuracy of the map will depend on the accuracy of their self-localization during the map construction process. Another challenge for map-building methods on UAVs is that during the exploration, map-building, phase they already have to avoid obstacles in order to prevent collisions.

Mapless obstacle detection methods include all methods that do not need knowledge of the environment to run. Instead, they rely on some combination of sensors to detect obstacles. The main advantage of this is that it allows for deployments of the UAV in unknown or dynamic environments. A downside of these methods is however that they often rely on the placement of extra sensors on the UAV. This usually comes with added weight, cost and power usage, thereby increasing the cost of the UAV and/or limiting its flight time.

Because the new obstacle detection method investigated in this research falls into the category of mapless obstacle detection methods, the rest of this part of the literature study will focus on this category. It is, however, important to note that also map-based and map-building methods often rely on one or more of the mapless obstacle detection methods. They can, for example, use the results from these mapless methods to improve localization of the UAV, build or improve a map, or as an extra safety layer.

4-1-2 Mapless obstacle detection

Within the category of mapless obstacle detection, a wide variety of methods, sensors, and algorithms reside. A full subdivision is thus not obvious, however, for practical purposes, a subdivision has been made into methods that use a range-sensor, methods that rely on detecting a change in system dynamics and methods that rely on computer vision to detect obstacles.

Using a range/proximity-sensor

One common way of detecting obstacles is to use some kind of range sensor, sometimes also called proximity sensor, to measure the distance to the closest object in a certain direction. By doing this in multiple directions, an autopilot can get a decent overview of where obstacles are located with respect to the current position of the UAV.

The accuracy and reliability of such a range-sensor obstacle detection method then usually depend on two factors: the reliability of the distance measurement and the number of directions that can be measured per second. If both are high, a range-sensor based method can be one of the most reliable obstacle detection methods.

There is however also a downside to these methods, the range-sensors, especially those that can perform measurements in different directions quickly, are usually quite heavy and expensive. This limits their use mostly to larger UAVs.

In the list below some key examples of range-sensors and their implementation for obstacle detection will be given:

- *Radar*, an acronym for RAdio Detection And Ranging, is an object detection system that uses radio waves to detect objects. The basic technology was developed for military purposes between 1934 and 1943. Its main usage has historically been the detection of relatively large objects over large distances. Examples of these are the detection and tracking of aircraft and ships, tracking space vehicles and observing weather conditions.

  In the last few years, some research and experiments have been performed using RADAR for obstacle avoidance. Most notably by Moses et al. [41], who developed a lightweight (230 gram), X-band doppler RADAR system. This RADAR system was fitted on a quadrotor and used to detect and identify a conventional-type miniature helicopter. A picture of this setup can be seen in figure 4-3 .

  Initial results were promising but limited to the size of the radar lobe, around 12.5 degrees. The goal of this particular research was to develop a system to prevent mid-air collisions with other air traffic, but looking at the experiment setup it can be expected that a similar application might be developed for obstacle avoidance in general.

Figure 4-3: Quadrotor Equipped with RADAR Sensor [41].

- *Ultrasonic sensors* are, as the name suggests, sensors that measure distance by transmitting an ultrasonic wave. They measure the time between the emission and reception and combine this with the known speed of sound to come to an estimate of distance.

  Ultrasonic sensors are typically low-cost and low weight. They can achieve accuracy up to a few centimeters, but can only measure distances up to about 250cm reliably. Furthermore, soft surfaces, like foamed material and people wearing clothes, can be hard to detect for an ultrasonic sensor. [18] Also, these sensors, like most other range sensors, can only detect an obstacle within a certain cone from their installed direction. As a result, multiple of these sensors often need to be installed, as seen in figure 4-4.

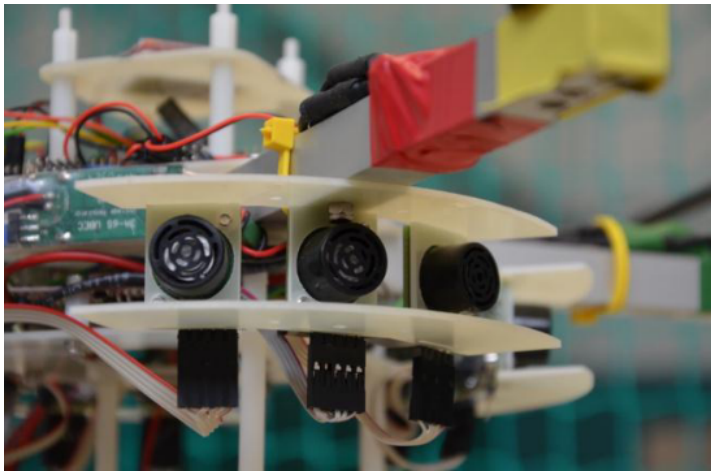


Figure 4-4: Installation of multiple ultrasonic sensors on a quadcopter [18].

- Most *Infrared range finders* work by transmitting an infrared beam. In contrast to the ultrasonic sensor, the transmitter and receiving-sensor are not placed at the same location. Instead of using the time difference between transmission and reception to calculate the distance, the infrared sensor uses the angle under which the infrared beam returns to calculate the distance. This is done using simple triangulation.

  Because infrared rangefinders use infrared light they are much more dependent on the

lighting conditions and reflective properties of the object that needs to be detected than similar ultrasonic sensors. However, infrared rangefinders can be effectively used on the softer surfaces which were a problem for ultrasonic sensors.

As a result, a combination of both ultrasonic and infrared sensor can be used as a successful low-cost obstacle detection and avoidance method on quadrotors. This is demonstrated in Gageik et al. [19]. This did, however, require a 360-degree setup with a total of 16 infrared sensors and 12 ultrasonic sensors for obstacles on the same level alone. The impact on the design of the drone is thus relatively large, as can be seen in figure 4-5.

Figure 4-5: The AQopterI8 equipped with 12 ultrasonic sensors and 16 (2 x 8) infrared sensors [19].

Additionally, more advanced infrared sensors exist that measure not just the distance to one single point but create a full depth map over a larger area. These devices are however also heavier and use more power. Historically, they were also quite expensive, however, there are now examples of cheaper implementations, like those using the Microsoft Kinect. [28] The first implementations of this Microsoft Kinect onto a quadrotor have already shown promising results. [61]

- *LIDAR.* One of the best-known methods is called Light Detection and Ranging (LIDAR), it is a surveying method that measures the distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor. LIDAR has previously been applied for cars, fixed-wing UAV's [51] [49], and thanks to recent advancements also on quadcopters [32] [78] [44].

- *Structured light* can be seen as a technique that is a mix between the range sensor methods previously discussed and the computer vision methods about to be discussed. Structured light is a technique whereby a known pattern, usually parallel stripes, are projected on a certain surface or scene. A camera is then used to capture this scene. From the deformation of the pattern, depth and surface information of the objects in the scene can then be extracted.

  Currently, structured light is being used for a variety of different applications, including 3D scanning of static objects. It has the potential for obstacle detection [17] and has previously been used for obstacle detection on mobile robotic devices. [73]

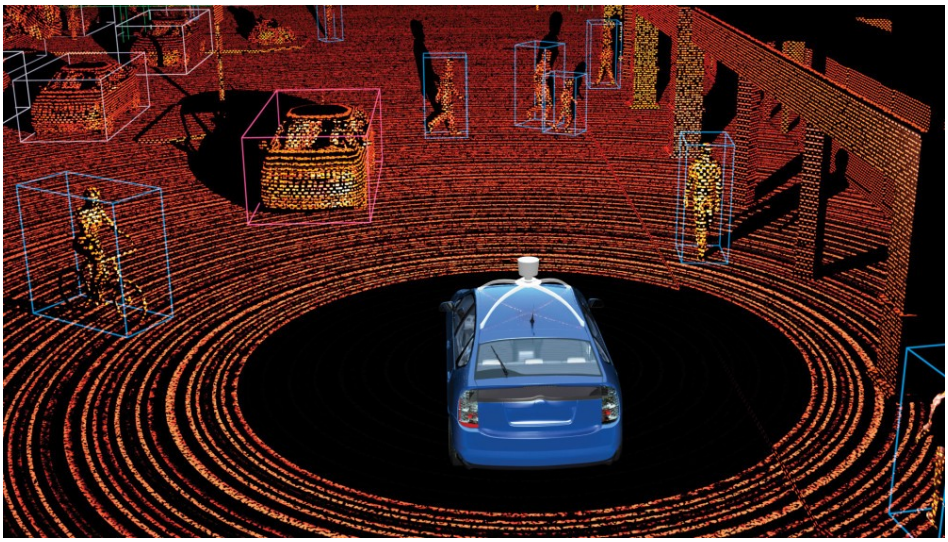


Figure 4-6: Obstacle detection for a car using LIDAR [70].

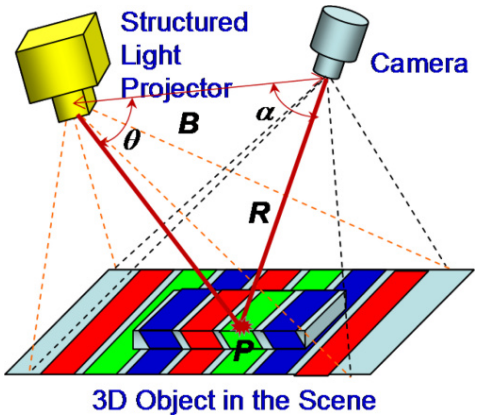


Figure 4-7: Working of structured light surface imaging [21].

In general, it can be observed that there are two categories of obstacle detection methods using range sensors. Both requiring the installation of specific sensors on the quadrotor.

The first category contains the simple sensors that measure the range to the closest obstacle in a single direction. The ultrasonic and simple infrared sensors fall under this category. They have the advantage of being low-cost, low-power and low-weight. However, they all have certain surface types on which their detection performance is significantly worsened. Furthermore, they only measure in one direction, so their obstacle detection capability is limited to one direction. Of course, it is in some cases possible to install multiple of these devices, but then also the cost, power and weight contribution starts to add up.

In the second category, there are sensors that are able to measure the range in multiple directions. This includes the radar systems, advanced infrared sensors like the Microsoft Kinect, structured light and of course LIDAR. They are able to achieve higher accuracy, but this also comes with a higher cost, power consumption, and weight.

Computer vision

The field of computer vision is a large and highly active field of research that has seen significant breakthroughs in previous years, mainly driven by the application of machine learning techniques. This progress can, for example, be seen by looking at the great advancement of image classification on the ImageNet dataset challenge. [50] Furthermore, face recognition using deep convolutional neural networks has become the standard for multiple large social media and photo sharing websites, bringing this new technology to the end user. [54]

When looking at using computer vision for obstacle avoidance, four main approaches can be seen, optical flow, appearance-based navigation, visual sonar, and stereo-vision. [3]

- *Optical flow* is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. Optical flow based solutions estimate this optical flow by using a sequence of images. The optical flow between frames is then usually represented by an optical flow vector for every pixel. The size of the vector indicates the motion speed, and the direction indicates the movement of the pixel. This information can then, for example, be used to estimate the layout of surfaces, the direction of the Focus of Expansion (FOE), the Time To Contact (TTC), or a depth map.[3] [23]

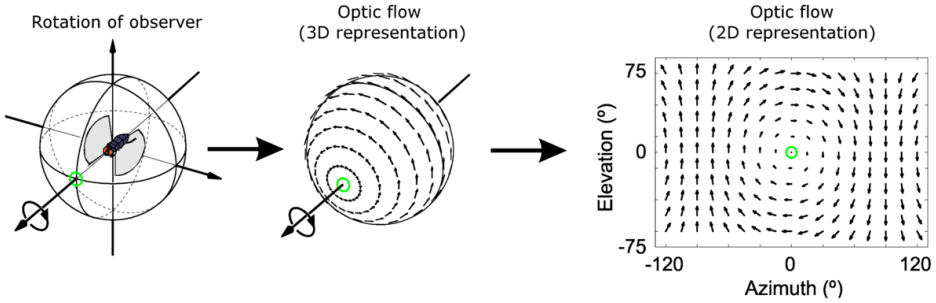


Figure 4-8: The optic flow experienced by a rotating observer [74].

  There exist a multitude of different platform independent algorithms using optical flow to detect obstacles. These methods are often insect-inspired or rely on the addition of edge and corner detection algorithms to improve performance. Furthermore, some articles have shown that the combination of optical flow and stereo vision can improve results. [3]

  There have also been multiple applications of optical flow methods to UAV's for navigation and obstacle avoidance. Examples of this are[15] and [9].

  A challenge for optical flow based methods is the detection of objects or surfaces with little texture. This, because here it is difficult to track individual pixels. Furthermore, changing light conditions and noise in the camera image can be problematic for obstacle detection methods relying on optical flow.

- *Appearance-based navigation* methods store (representations of) images encountered in previous stages, match this with the current camera image and use this for self-localization and/or direct steering commands.

There are multiple methods using this principle. The main challenge is usually the algorithm to create the stored imaged representation and to match this subsequently with an online image. This category also includes some of the machine learning methods discussed in the introduction, most specifically the image classification methods.

An example of using these methods for navigation is [40]. In this study, a SVM is used to store and match image representations, which are then used for navigation of a mobile robot. The result of which can be seen in figure 4-9.

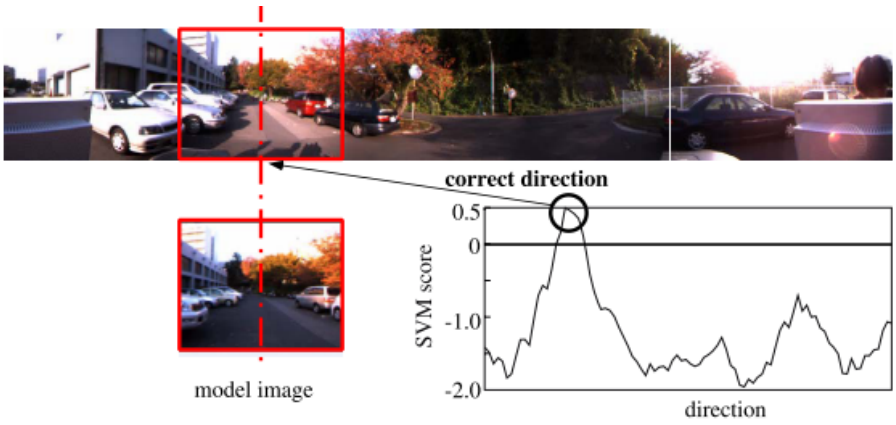


Figure 4-9: A computer vision method using a SVM for appearance-based navigation [40].

While this specific study is focused on localizing itself on a predetermined route, based on recognized appearances, a similar method could be used for obstacle detection. This would then be based on a stored set of known obstacle representations. However, the applicability would be challenging for environments in which not all potential obstacles are known.

- *Visual sonar* uses the idea that the distance in pixels from the bottom of an image to the object edge, is proportional to the real world distance from the robot to the detected object. In other words, this method looks at how much ground there is left between the robot and a particular object. This idea, first described by Horswill in 1994 [24], has seen multiple implementations on robotic platforms. Both on ground-based robots and UAV's. [36]

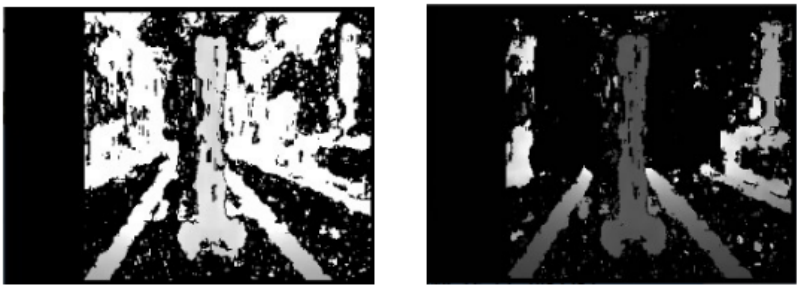
  This method is shown to work reasonably well, as long as the environment satisfies the specific requirements. Specifically, the floor needs to be easily recognizable for the algorithm. While this is certainly not the case everywhere, it can be argued that a lot of indoor environments actually do satisfy this requirement. [24]

- *Stereo-vision* methods can be used to generate a depth map and detect obstacles when not one, but two cameras are used. By comparing the images from the two cameras, and knowing the positions of the camera with respect to each other, 3D information can be extracted. This works similar to the binocular vision that allows humans to see depth.

  Obstacle detection methods using stereo-vision have been successfully demonstrated on flying UAV's, for example by Yu et al. [77], [8] and [75].

Figure 4-10: Visual sonar, as implemented by Ulrich and Nourbakhsh [68].

As can be seen from 4-11, the direct results from these stereo-vision methods can have a low signal-to-noise ratio. By making certain assumptions about the scene, and applying for example contour detection, this noise can partly be removed.



(a) The original depth image

(b) The depth image after pretreatment

Figure 4-11: Depth image from stereo vision algorithm, before and after pretreatment [75].

In general, it can be concluded that a variety of computer vision methods exist, each with its own reliability in specific environments, however all depending on good lighting conditions. Furthermore, it must be noted that most UAV's use a front-facing camera, and thus these algorithms are only able to detect obstacles in front of the UAV.

Detecting a change in dynamics

Another way to detect obstacles would be to detect the change in flight dynamics that takes place when a quadrotor is either in contact with an obstacle or flying close to it.

In comparison to the other methods described above, very little research has been performed in this area. There are two key reasons for this. First of all, actually making contact with an obstacle can be damaging and even catastrophic when it is done at higher speeds, in other words, a collision. Secondly, the effect of changing flight dynamics when flying near a surface is relatively small and can only be experienced when flying close to the obstacle.

As such, these methods have historically been researched to a lesser extent. Within this research, their potential will be investigated further, as the proposed new obstacle avoidance method falls in this category.

When looking at the research that has been done, within this area of using a change in dynamics to detect obstacles, a couple of articles stand out.

- In Mckinnon [37] a SVM was trained to detect the presence of a wall using the change in flight dynamics. This research is the one that is most similar to the research at hand, therefore it is discussed in more detail in sections 4-2-3 and 4-2-4.
- In Tomic and Haddadin [66] a method was developed to detect and react to collisions, using an external wrench estimation. By doing so, they were able to map an environment with obstacles, simply by exploring and saving the location of collisions. This was shown to work in experiments using an AscTec Hummingbird quadrotor.

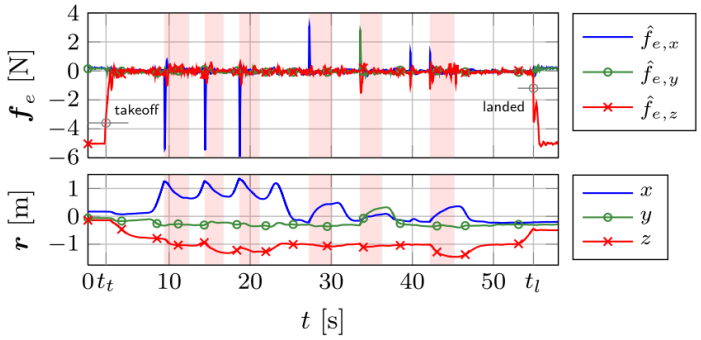


Figure 4-12: Experimental flight results from Tomic and Haddadin [66], showing the identification of multiple collisions, indicated by the peaks in the upper graph, and the recovery reaction, indicated by the shaded areas.

This work is then extended in [67], by filtering out the effects that wind might have on the external wrench estimation. Thereby enabling this method of obstacle detection in windy environments.

Overall it can be thus concluded that the research into obstacle avoidance methods based on detecting a change in system dynamics is limited. Further investigation of this would thus fill a specific literature gap.

4-1-3 Actions to prevent collision

Since the focus of this literature study is not just obstacle detection, but obstacle avoidance, the actual avoidance of detected obstacles needs to be considered. There are three key elements to consider when it comes to going from detection to avoidance.

- The location of the obstacle(s) w.r.t. the current position of the quadrotor.
- The current speed and acceleration of the quadrotor in the direction of the obstacle(s).
- The evasive maneuvers available to the quadrotor.

The first logical step for an autonomous quadrotor would be to determine the chance of collision with any detected obstacles. This can be done by combining the estimated location of the obstacle, the current speed and acceleration of the quadrotor, and any knowledge about the accuracy of both the detection system and the speed & acceleration estimates.

It must be noted here that not all obstacle avoidance methods calculate this as a chance in percentages, perhaps they use a certain threshold or conflict identification algorithm, but in general, there will be some way to determine which obstacles will potentially cause a collision. These obstacles must thus be acted upon.

A final check, before determining that an action needs to be taken to prevent collision, might be estimating the time till impact. This would be especially relevant to any obstacle detection method which is capable of detecting obstacles at larger distances. Not all obstacles might pose a direct threat to the quadrotor and therefore it might be beneficial to continue along a path in the short term, even if this path could lead to a collision in the long term. As long as there is enough time to prevent a collision at a later stage, the extra exploration could be beneficial. An example of this is shown in the figure below.

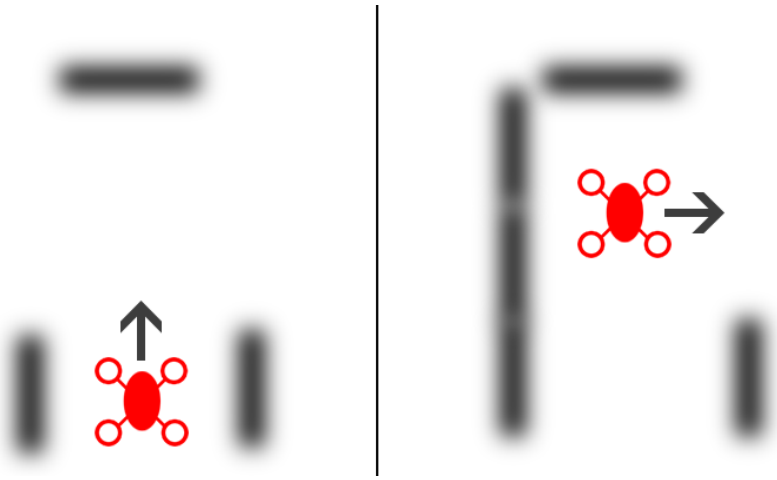


Figure 4-13: Example of a situation in which continuing along a certain path might be beneficial, even though an obstacle is detected on that path. In this case because it leads to the discovery of an alternative path.

In the cases where a collision with a detected obstacle is imminent, an autonomous quadrotor should take the right action to prevent a collision. Usually, this can be done in one of two ways.

- Decelerate such that the speed relative to the detected obstacle becomes zero.
- Accelerate in another direction, thereby adjusting the current flight path, such that the detected obstacle is no longer in its flight path.

The exact maneuver that is best for this depends greatly on the situation and the exact type of quadrotor that is being used. Depending for example on their power to weight ratio, certain quick deceleration's might, or might not be achievable.

In general, it can be said that deceleration is usually the less risky option but adjusting the flight path the more time-efficient in terms of achieving the overall flight mission. The choice for an exact maneuver is thus specific to the quadrotor, the situation and the mission objective.

A final interesting case that might be considered is the case where a collision cannot be avoided. An example of this would be a situation in which the quadrotor approaches an obstacle at such a speed that deceleration in time is no longer possible, and there is no alternative flight path available to avoid the obstacle. In that case, it might be desired to adjust the orientation of the quadrotor such that the impact of the collision is minimized. However, this is out of the scope for this particular research.

4-2 Obstacle-airflow interaction

The dynamics of a quadrotor are greatly dependent on its aerodynamics, most importantly the airflow around the thrust-producing propellers. These aerodynamics can be influenced by surfaces in proximity to the quadrotors. The most well-known example of this is the one produced by proximity to the ground surface, usually called the ground effect. This phenomenon has been shown to influence the dynamics not only in quadrotors [45], but also helicopters [7] and fixed-wing aircraft [25] [48]. Recently it has been shown that for quadrotors similar phenomena can also be seen for surfaces above [52], and on the same level as the quadrotor [37].

In the proposed new method, this change in dynamics will effectively function as the source of information used by the reinforcement learning agent to determine the presence of obstacles. Therefore, it is of key importance that these effects can be accurately measured in flight. Furthermore, it would be very useful if these effects could be modeled in simulation. Accuracy here is key because only then a policy learned in simulation can be successfully transferred to a real (physical) application without needing too much additional online training. In order to accurately simulate these effects, this research will use both known analytical models and measurement data provided in other publications.

The literature survey of obstacle-airflow interactions has been divided into three separate parts: obstacles underneath the quadrotor, obstacles above the quadrotor and obstacles on the same level as the quadrotor. This has been done for a couple of reasons. First of all, this is the division usually used in literature. Secondly, the aerodynamic effects causing the interaction are expected to be different. Finally, the resulting interactions, as experienced by a quadcopter, are expected to be distinctly different between the three cases.

4-2-1 Obstacles underneath the quadrotor

Also known as the ground effect, the influence of a horizontal surface underneath a rotor has been well researched in literature [7] [47]. Most of this research has been focused on helicopters, but in general, it can be noted that for all rotorcraft operating closely above a ground surface, the produced thrust increases. [53]

Classical theory

The classical model for ground effect in helicopters is provided by Cheeseman and Bennett [7]. In this article from 1955, I.C. Cheeseman and W.E. Bennett derive an analytical model based on potential flow theory and under both the assumption that the helicopter is hovering and the assumption that the rotor can be modeled as a point source. They then use the method of images to derive the following ratio between the thrust produced by a helicopter in ground effect (T_g) and the thrust out of ground effect (T_∞), as a function of the radius of the rotor (R_{rotor}) and the distance to the ground (z).

$$\frac{T_g}{T_\infty} = \frac{1}{1 - (\frac{R_{\text{rotor}}}{4z})^2} \tag{4-1}$$

Thanks to its simplicity, this model has also been used to estimate the ground effect in quadrotors [45], even though quadrotors have multiple rotors and the airflows from the different rotors could thus potentially interfere with each other.

Validation using experimental measurements have since shown that the ground effect for a quadrotor is larger than predicted by this formula. [57] [53] Furthermore, these experiments showed that the influence of the ground effect in quadcopters was apparent up to heights of 5 times the rotor radius. This is significantly larger than previously expected.

Recent developments

A new formula, specifically for quadrotors was recently (2017) proposed by Sanchez-Cuevas et al. [53], accounting for the presence of multiple rotors by representing them not as one but as four sources. It assumes a quadrotor hovering above a ground surface with four coplanar rotors, each rotor axis separated a distance d from its adjacent rotor axes.

Furthermore, this new formula accounts for an effect called the *fountain effect*. This effect has previously been seen in tandem helicopters [22] and quadcopter experiments [57]. It can best be explained by looking at the CDF simulation of a simplified quadrotor in ground effect shown in figure 4-14. As expected, the wakes from each rotor spread out to the sides as they near the ground, however in the center area where the two airflows interact with each other a vortex ring appears. Due to this aerodynamic effect, an upwards force is applied to the body of the quadrotor, leading to a greater ground effect. [53] This effect is represented in equation 4-2 by the empirical body lift coefficient K_b, where b is the distance between two diagonally opposite rotor axes.

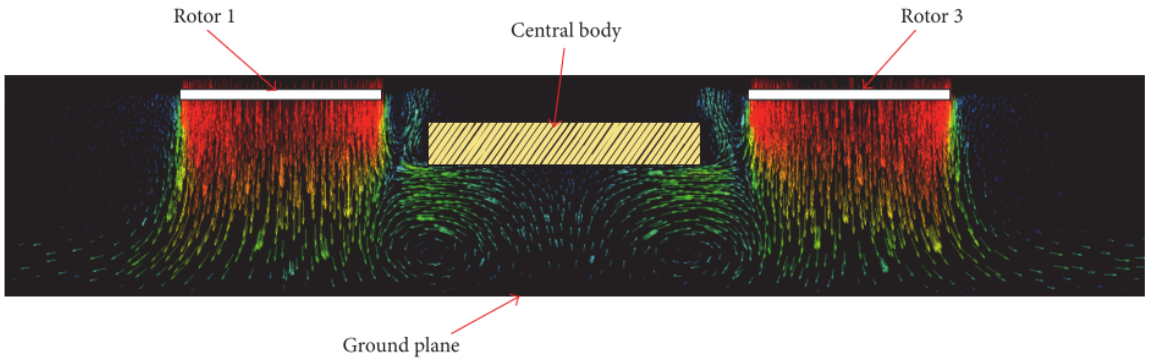


Figure 4-14: CDF simulation of a simplified quadrotor model hovering close to a ground surface plane [53].

In Sanchez-Cuevas et al. [53], equation 4-2 is shown to represent their experimental results more closely than equation 4-1. Unfortunately however, at the current moment, this has not yet been confirmed by third-party research.

$$\frac{T_g}{T_\infty} = \frac{1}{1 - (\frac{R_{\text{rotor}}}{4z})^2 - R_{\text{rotor}}^2(\frac{z}{\sqrt{(d^2+4z^2)^3}}) - (\frac{R_{\text{rotor}}^2}{2})(\frac{z}{\sqrt{(2d^2+4z^2)^3}}) - 2R_{\text{rotor}}^2(\frac{z}{\sqrt{(b^2+4z^2)^3}})K_b} \tag{4-2}$$

Another interesting part of ground effect analysis is the analysis of a situation in which only some of the rotors in a quadrotor are subject to the ground effect. This situation is

analyzed by Sanchez-Cuevas et al. [53], and referred to as the *quadrotor partial ground effect*. Experimental tests of this situation have also been performed by Seo et al. [55].

According to [53], if only a single rotor is under the influence of the ground effect, the increase in thrust of that specific rotor can be modeled using equation 4-1. The total trust of the quadrotor can then be calculated by summing the thrust of the 3 rotors outside of ground effect and the one rotor in ground effect. According to the same article, if three or more rotors are under the influence of the ground effect, each at the same distance z from the ground, equation 4-2 should be used.

This leaves open two groups of situations. First of all, situations wherein two rotors are experiencing the ground effect. Secondly, situations where two or more rotors are experiencing the ground effect, but at different distances (z_i). This can happen for example when the quadrotor is not parallel to the ground or the ground is uneven. To solve this problem, [53] introduces the following formula for the thrust generated by each of the motors.

$$F_i = k_i \omega_i^2 f_{GE}(z_{rmu}) \tag{4-3}$$

Where f_{GE} is the ground effect factor that accounts for the increment in thrust due to the (partial) ground effect. It is a function of the relative distance of each motor to the ground (z_r). Furthermore, this function varies linearly with the area of the rotor that is under the ground effect, an observation taken from Seo et al. [55].

The full equation for the ground effect factor (f_{GE}) is unfortunately not provided in the article. Therefore it is unclear whether equation 4-1, or an altered version of equation 4-2 is used in the calculation of f_{GE}.

4-2-2 Obstacles above the quadrotor

Little research has been done regarding the influence of obstacles or surfaces above a quadrotor. The most relevant description and experiments are performed by Sanchez-Cuevas et al. [52]. In this research, the thrust produced by both a single rotor and a quadcopter, at varying distances to a ceiling surface, were measured and compared. This was done using a static test bench, specifically built for these type of measurements.

In Sanchez-Cuevas et al. [52] the increment in thrust due to the ceiling effect is approximated by an analytical function similar to that of the ground effect shown in equation 4-1. The ratio between the thrust in (T_c) and out of the ceiling effect (T_∞) is in this case given by the following formula, where c is the distance to the ceiling, and K_1 and K_2 are determined experimentally using ordinary least squares.

$$\frac{T_c}{T_\infty} = \frac{1}{1 - \frac{1}{K_1}\left(\frac{R_{\text{rotor}}}{c+K_2}\right)^2} \tag{4-4}$$

Similar to the ground effect, it can be expected that there is also such a thing as *partial ceiling effect*. This would occur when some, but not all rotors are experiencing the ceiling effect. Unfortunately, at the current moment, no experimental data or formula is known that describes this effect. It can, however, be expected that a formula similar to equation 4-3 can be formulated.

4-2-3 Obstacles on the same level as the quadrotor

At the time of writing, only two previous research articles have been found in the literature that discuss the obstacle-airflow interactions with obstacles on the same level. The first one, [33], focused specifically on reference tracking in the presence of a wall. It shows that there is an interference, but does not discuss how to model it.

More detailed research is carried out in Mckinnon [37], where multiple measurements have been carried out, both in ground and wall effect. In this research, an Unscented Kalman Filter, based on a known model of the UAV, was used to estimate the resulting forces and torque due to the ground and the wall effect. Important to note here is that all experiments and tests were carried out in hover.

The results of this experiment can be seen in figure 4-15. On the x and y-axis, the distance to the wall and the ground at the moment of measurement are displayed. Indicated in color are the estimated external forces and torques: $||f_{x,y}||$ indicates the 2-norm of the force in the $x-y$ plane, f_z indicates the vertical force along the z-axis (perpendicular to the $x-y$-plane) and $||\tau_{x.y}||$ is the 2-norm of the torque about the $x-$ and $y-$ axis.

The force in the $x-y$ plane, shown in figure (a), can be seen to quickly increase from about 0.05N up to 0.1N between 0.5m and 0.25m. What is important to note here is that this effect seems stronger when the quadcopter is not only hovering close to the wall but is also close to the ground (<1m height).

When looking at the vertical force, shown figure (b), the ground effect can clearly be seen; f_z, the external force in vertical direction, becomes more positive as the height decreases. This effect, however, seems to be counteracted if the quadcopter is close to a wall, f_z even becomes negative when close to the wall (<0.35m), but far away from the ground (>1m).

Finally, from the graph showing the torque about the $x-$ and $y-$ axis, figure (c), a clear influence of the wall can be seen. Between 0.5m and 0.25m from the wall the torque doubles from 0.01Nm to 0.02Nm.

From these results, it can be seen that a small, but measurable increase in forces and torque are present when flying close to the wall, ground or both. In Mckinnon [37] an attempt was made to detect the presence of a wall from a distance of 0.35m or less. For the drone that was used, this distance corresponds to about 3 times the propeller radius. It is, however, unknown whether the minimum detection distance for walls depends on this radius.

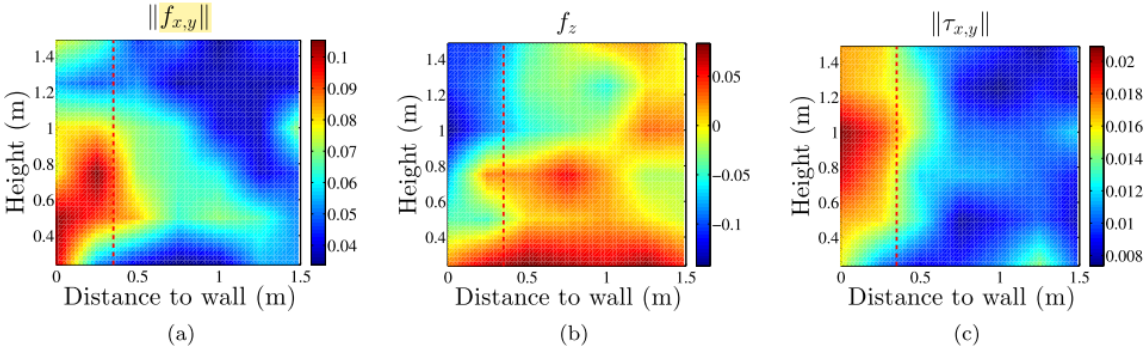
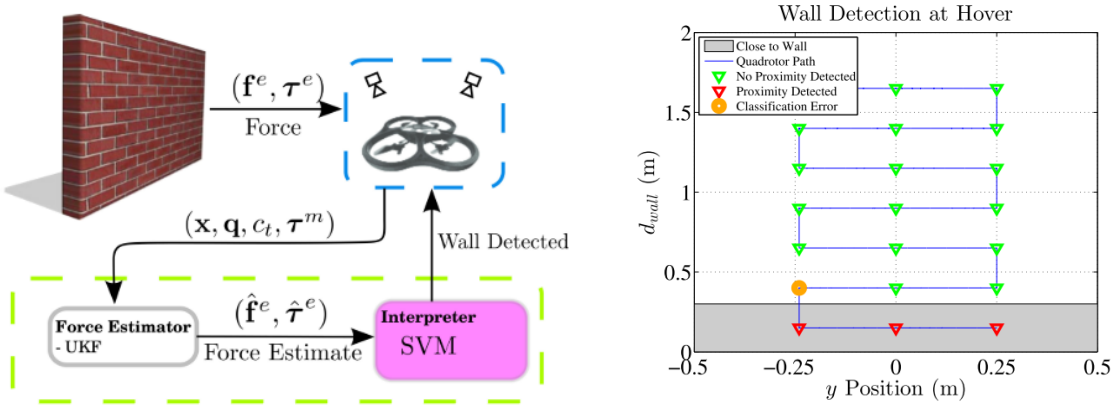


Figure 4-15: The estimated external forces and torque while hovering at different distances from a wall and the ground [37].

4-2-4 Using these effects for obstacle detection

In addition to describing the effect between quadcopters and obstacles, some effort was also made by Mckinnon [37] to detect obstacles based on these measurements. More specifically, in Mckinnon [37] a support vector machine was trained to detect the presence of a wall. A SVM is a machine learning method that learns a decision boundary between two classes as a function of feature vectors. Like other supervised machine learning methods, it learns this during a training phase, where it is exposed to multiple examples from both categories.

The setup of the experiment can be seen in figures 4-16a and 4-16b, what is important to note here is that the vehicle was hovering for 5 seconds at each measuring point.



(a) Setup of the wall detection using an SVM carried out by Mckinnon [37].

(b) Classification results for a quadrotor navigating a series of waypoints. [37]

Figure 4-16: Setup and results of the wall detection experiment carried out by Mckinnon.

Using random search on hyperparameters, to optimize classification accuracy on a separate dataset, the created SVM was able to accurately classify being close to the wall with an accuracy above 90%. Close to the wall was defined as being at a distance $\leq$0.35m. It was able to classify being not close to the wall with an accuracy of around 80%. It thus shows that the algorithm was capable of detecting proximity with consistently higher accuracy than free flight. The explanation for the difference in false negatives versus false positives, given in the publication is that the forces caused by the wall effect are so small that they can be felt during random disturbances in regular flight as well.

Other than that, no further research has been found that uses these effects for obstacle detection.

4-2-5 Final considerations

In the sections above, the influences of ground, ceiling, and wall, were all considered separately. It can however not be expected that their contribution to the increase in thrust can be considered independent. On the contrary, the experiment results of Mckinnon [37], as shown in figure 4-15, indicate a much more complex relationship.

Furthermore, the formulas given all approximate the increase in thrust when a quadcopter is hovering, near a flat surface. In applications for obstacle avoidance, one would want the obstacle detection especially to work while flying. Additionally, not all obstacles that are encountered are flat surfaces.

Based on these considerations, it can thus be concluded that the formulas found in literature can serve as a starting point for simulation, but not as a highly accurate representation of reality.

This highlights the reason why reinforcement learning will be used for this research. Because reinforcement learning can learn by interacting with these aerodynamic effects in real flight, it does not require a very accurate pre-existing model. It must be noted however that if some part of the training of the reinforcement learning agent were to be done in simulation, still some (simple) model of the obstacle-airflow interactions and their effect on the quadrotor would be required.

4-3 Reinforcement Learning

Reinforcement learning is a computational approach to machine learning whereby an agent tries to maximize the cumulative rewards it receives when interacting with a complex environment. In this section, both the basics and the state of the art will be discussed. If not otherwise indicated, all theory has been derived from Sutton and Barto [63].

4-3-1 The basics

Reinforcement learning is a type of machine learning, and thereby also a branch of artificial intelligence. The defining idea behind reinforcement learning is that an agent learns by interacting with an environment. By doing so, the agent tries to determine the ideal behavior in order to maximize its performance.

In reinforcement learning the agent is the learner and decision maker, and everything outside of the *agent* is considered the *environment*. The agent interacts with the environment by picking an action, the environment then responds to this action and presents the agent with a numerical reward and a new (updated) situation.

This constant loop of the agent picking an action and the environment responding with a reward plus a new situation is almost always considered in discrete time. Some work has been done to extend it to continuous time, but in practice, almost all applications of reinforcement learning are using discrete time timesteps. [63] The agent and environment thus interact at a sequence of discrete time steps.

At each discrete time step t, the agent not only receives a numerical reward (r_t), but also a representation of the environment's *state* s_t, the current situation. The agent then uses its policy function, a mapping from state-space to action-space, to determine the next action to take. This policy can be deterministic or stochastic and is usually denoted as $a_t = \pi(x_t)$. One time step later, this action a_t, chosen at state s_t, results in both a numerical reward r_{t+1} and a new state s_{t+1}. This process can be seen in figure 4-17.

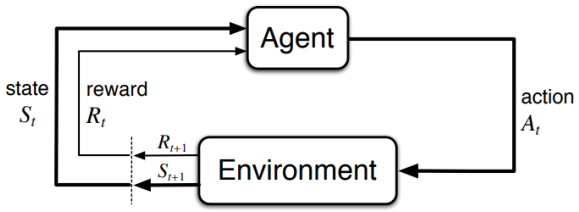


Figure 4-17: The agent environment interaction in reinforcement learning [63].

A reinforcement learning method then specifies how the policy should be updated, based on this reward and new state. The reinforcement learning method does so with the end goal of maximizing the total amount of rewards received by the agent.

4-3-2 Rewards & the return

This total amount of rewards received after some timestep t is often called the *return* and the simplest way to define it is as the sum of the rewards.

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{4-5}$$

In this equation T is the final time step, intermediately identifying a potential problem with this definition of the return; It only works for tasks in which the agent-environment interaction has a certain (natural) end. These kinds of tasks are usually called episodic tasks, examples include most games, and for example pathfinding. All tasks which can be naturally defined as a certain episode with a definitive start and end point.

There are however also a large class of tasks for which such a natural end of an episode does not exist. These tasks are called continuing tasks, and if there return is calculated using equation 4-5 a problem arises. The final time step would approach infinity and thus the return could easily be infinite, severely limiting a practical implementation.

Therefore, the *discounted return* is introduced. This discounted return is based around γ, the *discount rate*, a parameter determining the relative value of rewards in the future versus rewards now. This discount rate is bounded by $0 \leq \gamma \leq 1$, and if $\gamma < 1$ the discounted return will always have a finite value, as long as the rewards are also bounded.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4-6}$$

4-3-3 The state

Next to the immediate reward, another key source of information for the agent is the state. While the reward is used by a reinforcement learning algorithm to update the policy function, the state is the direct input to this policy function. As such, the state should contain all available and relevant information for the agent to make a decision.

The state signal usually includes immediate measurements, but can also include previous measurements and/or processed measurements. However, it is usually impractical and unnecessary to store all previous measurements in the state signal. Therefore a choice has to be made.

A goal is often to have the state signal compact, but also satisfying the *Markov property*. A state signal is said to have this Markov property if the signal succeeds in retaining all relevant information. Formally this can be defined as follows.

Considering a discrete process at time step t, with a finite number of states $(s_t, s_{t-1}, ...s_0)$, actions $(a_t, a_{t-1}, ..., a_0)$ and reward values $(r_t, r_{t-1}, ..., r_1)$. The state signal s is said to have the Markov property, if and only if, the probability distribution of transitioning from the current state s_t to a new state s' and receiving reward r, depends only on the current state s_t and the chosen action a_t. If this is the case then the transition probability is fully given by equation 4-7 for all s', r, s_t and a_t.

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{4-7}$$

If the Markov property does not hold for the state signal, the transition probability also depends on some previous states and actions. As shown in the equation below.

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \tag{4-8}$$

In other words, a state signal has the Markov property if and only if 4-7 is equal to 4-8 for all potential s' and r and all histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0$. Furthermore, one could predict future states and expected rewards from the knowledge of only the current Markov state and action as well as one could given the entire history of previous states, actions, and rewards. [63]

Because in reinforcement learning the value function, which will be discussed hereafter, and policy function are assumed to be functions of only the current state, the Markov property is of key importance to the application of reinforcement learning.

A reinforcement learning task that satisfies the Markov property is called a Markov Decision Process (MDP) and it is the most common type of task considered in the research into reinforcement learning. Markov state representations are thus a common assumption in reinforcement learning theory.

In real-life applications a perfect Markov representation is often not possible, however, experience has shown that if a state representation is informative enough, even though not strictly Markov, reinforcement learning can still successfully be applied. [63]

4-3-4 The value function

Almost all reinforcement learning algorithms are based on estimating some sort of value function, where the value function is an indication of the desirability of a certain state or state-action pair. The desirability is then usually an indication of the expected return from that state or state-action pair.

Most reinforcement learning algorithms use either an state-value function (V) or an action-value function (Q). Since the value function is an indication of the expected returns, it is also dependent on the actions that are taken after that state or state-action combination. Therefore the value function is usually evaluated under a certain policy π.

For an MDP, the state-value function at state s and for policy π is thus formally defined as in equation 4-9, where $E_\pi\{\}$ is the expected value given that the agent follows this policy π.

$$V^\pi(s) = E_\pi \{R_t \mid s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\} \tag{4-9}$$

Alternatively, some algorithms use the action-value function. This function not only takes into account the current state (s), but also the action that will be taken in the next time step (a). It can thus be seen as the expected return starting from state s, then taking action a and thereafter following policy π. The formal definition for the action-value function in an MDP can be seen in equation 4-10. [63] Important to note here is that for the value function $Q^\pi(s, a)$ the action a does not necessarily need to be in line with the policy π.

$$Q^\pi(s, a) = E_\phi \{R_t \mid s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right\} \tag{4-10}$$

4-3-5 On-policy and off-policy methods

In addition to the division into the three fundamental classes that will be discussed below, reinforcement learning methods can also be split between on-policy and off-policy methods. In on-policy methods, the reinforcement learning agent learns the value function for the policy it is currently using to select actions. In off-policy methods this is different, here the reinforcement learning agent learns the value function for the policy it currently thinks is best, however it is not necessarily following this policy to select actions. An off-policy method thus consists of a *behavior policy*, used to select actions, and an *estimation policy* that is being evaluated and improved in order to approach the optimal policy.

An advantage of off-policy methods is that sufficient exploration can be ensured by having a behavior policy with a nonzero probability of selecting each of the actions, while the estimation policy can be fully greedy in selecting only the action it thinks is best. As such, off-policy methods can be a great approach to the *exploration-exploitation dilemma*. This dilemma refers to the fundamental choice between exploration, where an agent gathers more information that might lead to better decisions in the future, and exploitation, where an agent makes the best decision given the current information.

A downside of off-policy methods is, however, that they are often hard to combine with bootstrapping, practice has shown that off-policy bootstrapping methods may even diverge to infinite error. [63]

4-3-6 Three main reinforcement learning methods

Reinforcement learning methods can be split up into three fundamental classes: dynamic programming, Monte Carlo methods, and temporal-difference learning. Each of these classes has its own specific strengths and weaknesses.

Dynamic programming

Dynamic programming refers to a collection of methods that can be used to find optimal policies for a given Markov decision process, provided however that a full and perfect model of the environment is known. This requirement, combined with the often great computational expense are the two main limitations of applying dynamic programming.

Monte Carlo methods

Monte Carlo methods, on the other hand, do not require a perfect and complete model of the environment, instead, they learn value functions and optimal policies directly from interaction with the environment. They learn from experience, by using sample episodes and estimating the value function by averaging the returns observed after visits to that state. What is important to note here is that Monte Carlo methods only update the estimated value function based on the returns observed and not on the basis of other value estimates. In other words, they are not *bootstrapping*. This can be an advantage, because they may be less harmed by violations of the Markov property [63]. This however also means that they are less suited for incremental step-by-step computation, something that is often desired for computational implementation.

Temporal-Difference learning

Like Monte Carlo methods, Temporal-Difference (TD) methods can learn directly from raw experiences and do not require a model of the environment. The key difference is that in TD methods, the value function is incrementally updated; Each timestep the values of the previous state or states are updated based on the observed reward and the estimated value of the new state. As such, TD is a bootstrapping method

The main advantage of TD methods, comparing to dynamic programming methods, is that they do not require a model of the environment. Secondly, due to the incremental nature of the algorithms, these TD methods can be naturally implemented in an online, fully incremental fashion. Taking immediate use of the knowledge gained during each episode. Finally, TD methods have usually been found to converge faster than Monte Carlo methods (with constant step-size) on stochastic tasks. [63]

The simplest TD method, known as $TD(0)$ is:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{4-11}$$

After taking action a from state s_t and receiving reward r_{t+1}, the estimated value of this previous state ($V(s_t)$) is thus updated based on the observed reward (r_{t+1}) and the estimated value of the next state ($V(s_{t+1})$). For any fixed policy π, the $TD(0)$ algorithm shown above has been proved to converge to V^π in the mean when α is constant and sufficiently small, and with probability 1 if α satisfies the following conditions.

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2(a) \leq \infty \tag{4-12}$$

With $\alpha_k(a)$ the step-size parameter used to precess the reward received after the kth selection of action a. A common choice is $\alpha_k(a) = 1/k$, which satisfies both conditions shown above.

Instead of estimating the value function (V^π) of a policy π, most reinforcement learning methods try to estimate the action-value function $Q^\pi(s, a)$ for the policy π and all states s and actions a. This is also the case for the most well-known implementation of a TD(0) method for on-policy control: SARSA.

As shown in equation 4-13, the SARSA algorithm uses every element in the quintuple $(s_t.a_t, r_{t+1}, s_{t+1}, a_{t+1})$ to update the estimated state-action value. This is also where the name SARSA comes from.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{4-13}$$

Because SARSA is an on-policy method, the policy π is constantly changed toward greediness, using the state-action value function Q^π. Since Q^π is an estimation of the value of behavior policy π this value function is then updated, to more closely approximate the value function for the current policy. This loop is proven to converge to an optimal policy and action-function as long as all state-action pairs are visited an infinite number of times and the policy function converges in the limit to a fully greedy policy.

Where SARSA is the well-known on-policy TD algorithm, the best known off-policy TD algorithm is Q-learning. The most basic, one-step, Q-learning method is defined as: [72]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{4-14}$$

The key difference with SARSA is thus that Q-learning uses the maximum value of the next state to update the previous state, instead of the value under policy π. As a result, Q-learning approximates Q^*, the optimal action-value function, instead of Q^π, the value of behavior policy π. This algorithm has been shown to converge with probability 1 to Q^* to as long as all pairs of (s,a) continue to be updated. [63] As with all reinforcement learning algorithms, sufficient exploration is thus required for both SARSA and Q-Learning implementations.

4-3-7 Eligibility traces

Eligibility traces are one of the basic mechanisms of reinforcement learning, they provide a theoretical bridge from Temporal-Difference methods to Monte Carlo methods. Furthermore, in a more mechanical view, they provide a way to temporarily store the occurrence of events (states and/or actions) and use this information for learning.

Regardless of which view is preferred for the explanation, eligibility traces are centered around the principle of using multiple, but not all rewards for backing up a value or action-value function. Instead of using just the direct reward in a state, like the previously discussed TD(0) methods, or using all subsequent rewards, like Monte Carlo methods, an intermediate number of rewards is used to update the value estimate.

From the perspective of the more theoretical view, sometimes referred to as the forward view, each state that is visited is updated by looking forward in time to all the future rewards. These are then combined and used to update the state. A graphical representation of this can be seen in figure 4-18.

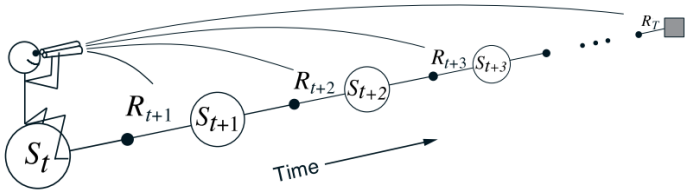


Figure 4-18: Illustration of the update mechanism according to the forward view [63].

The most used methods for the actual combination of these future rewards is called the λ-return. When using this method, the update to the current state is a weighted sum of future rewards, where the weight fades with λ on each step further in the future. This is then normalized with a factor $(1 - \lambda)$ to ensure that the sum of the weights is 1.

For an episodic task, with an terminal state of G_t, the λ-return is described by equation 4-15.

$$L_t = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{t+n}(V_t(S_{t+n})) + \lambda^{T-t-1} G_t \tag{4-15}$$

Where G_t^{t+n}, the general n-step return, is defined as following.

$$G_t^{t+n}(c) = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n c \tag{4-16}$$

Looking at these equations, the bridge from Temporal-Difference methods to Monte Carlo methods can be shown. When $\lambda = 1$, equation 4-15 reduces to the conventional return G_t, the same as the Monte Carlo algorithm. When $\lambda = 0$, the equation reduces to $G_t^{t+1}(V_t(S_{t+1}))$, the one-step temporal difference return TD(0). As such, both Monte Carlo and one-step temporal difference methods can be seen as special cases of the Temporal Difference method TD(λ).

While the beauty of this forward view is in the bridge it creates between Monte Carlo and the one-step temporal difference methods, a direct implementation is problematic, because it requires knowledge of rewards in future time steps. Instead, most implementations of eligibility traces follow the backward view, which provides a causal, incremental mechanism for approximating the forward view computationally. [63]

To do so, the backward view relies on memory variables called eligibility traces, associated with each of the states. These traces keep track of how recently a state, or state-action combination, has been visited. This is done for all states, except the current state S_t, using the following update equation.

$$E_t(s) = \gamma \lambda E_{t-1}(s), \quad \text{for } \forall s \in S, s \neq s_t \tag{4-17}$$

The eligibility thus decreases by the discount factor γ and the decay parameter λ every time step. The eligibility is only increased for the current state. There are three main methods to do so, as shown in equation 4-18 and visualized in figure 4-19.

$$E_t(s) = \gamma \lambda E_{t-1}(s) + 1, \quad \text{for } s = s_t \quad \text{Accumulating traces} \tag{4-18}$$
$$E_t(s) = (1 - \alpha)\gamma \lambda E_{t-1}(s) + 1, \quad \text{for } s = s_t \quad \text{Dutch traces} \tag{4-19}$$
$$E_t(s) = 1, \quad \text{for } s = s_t \quad \text{Replacing traces} \tag{4-20}$$

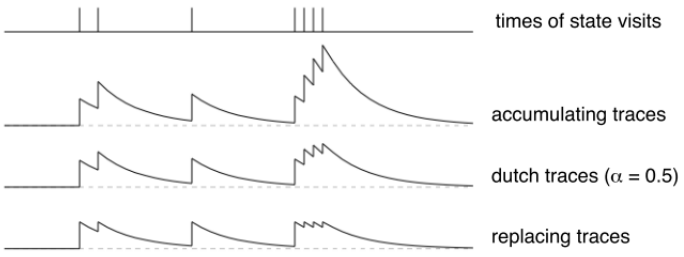


Figure 4-19: The three main tracing methods. [63]

When eligibility traces are not only used to track and estimate values for states but for full state-action pairs, they can then be used for policy evaluation, improvement and thus control.

This can be seen as an extension of the temporal difference methods discussed in section 4-3-6 with eligibility traces, and it can be done for both on-policy and off-policy algorithms.

When the on-policy TD method SARSA is extended with eligibility traces, it is referred to as SARSA(λ). The estimated action-value function Q is then updated, using the eligibility traces matrix E, as following

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t E_t(s,a) \quad \text{for all s,a} \tag{4-21}$$

With

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{4-22}$$

In contrast to the one-step SARSA update equation given in equation 4-13, multiple state-action pairs are thus updated in one single time step. This can improve the learning speed, as can be imagined when looking at an example of the impact seen in figure 4-20.

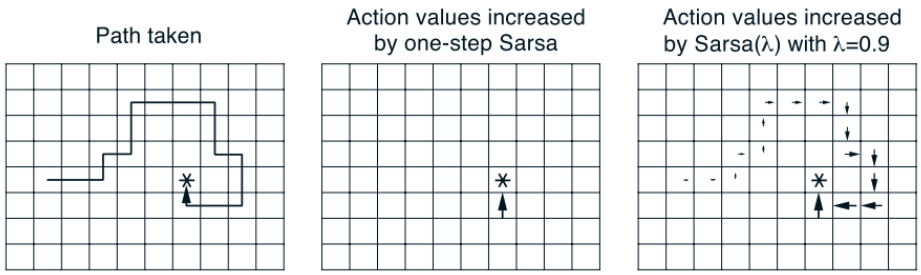


Figure 4-20: Gridworld example of the speedup of policy learning due to the use of eligibility traces [63].

Off-policy Temporal Difference methods can also be extended with eligibility traces. However, this requires special care, because in an off-policy the policy that is being learned is not necessarily the same as the one that is being used to pick actions. Instead, off-policy methods often rely on the occasional selection of sub-optimal exploratory actions.

A well-known implementation of Q-learning that deals with this problem and extends Q-learning with eligibility traces is Watkins's Q(λ). In this method, eligibility traces are used similar to SARSA(λ), however when an action is picked that is not in line with the policy being evaluated, for example, a non-greedy action, eligibility traces are set to zero. The update of the eligibility matrix, previously done using equation 4-17, thus changes to the following form [63].

$$E_t(s,a) = \begin{cases} \gamma\lambda E_{t-1}(s,a) + I_{ss_t} \cdot I_{aa_t} & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a) \\ I_{ss_t} \cdot I_{aa_t} & \text{otherwise.} \end{cases} \tag{4-23}$$

In this case accumulating traces are used, but a similar equation is easily derived for dutch or replacing traces. The estimated action-value function Q is then updated, as following

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t E_t(s,a) \quad \text{for all s,a} \tag{4-24}$$

With

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{4-25}$$

4-3-8 Partially observable Markov decision processes

In most of the reinforcement learning theory, the assumption is made that the state representation has the Markov property. In practical applications, this is however not always the case. The extension of reinforcement learning methods to non-Markov processes is an active and interesting research area.

One common approach is to try and construct a Markov or nearly Markov state representation, based on the non-Markov state provided by the environment. A well-known example of this is the theory of Partially Observable Markov Decision Processes (POMDs). POMDs are finite MDPs in which, even though the state is not fully observable, another signal is available that is stochastically related to the state.

A POMD can thus be seen as a generalization of a MDP, where it is assumed that the environment dynamics are a MDP, but the agent cannot directly observe the underlying state. Instead, the agent maintains a probability distribution over the set of possible states. More formally, a POMD is usually defined as the tuple $\langle S, A, T, R, \Omega \rangle$, where: [56]

- S is the complete set of possible environment states, also referred to as the *state-space*.
- A is the complete set of actions that the agent can execute, also referred to as the *action-space*.
- T is the (stochastic) transition function, the probability of reaching a state s' at time $t+1$ when executing action a from state s at time t.
  $T(s, a, s') = Pr(s_{t+a} = s' \mid s_t = s, a_t = a)$
- R is the reward function, $R(s, a)$ is the reward received by the environment at time $t+1$ when taking action a in state s at time t. In some special cases, when the reward is dependent on s_{t+1} and the transition is stochastic, it might be required to write the reward function in the form $R(s, a, s')$.
- Ω is the complete set of all possible observations that might be available to the agent, also referred to as the *observation-space*.
- O is an observation function, the probability of observing o at time $t+1$ given that the agent has executed action a at time t and reached state s'.
  $O(a, s', o) = Pr(o_{t+1} \mid a_t = a, s_{t+1} = s')$

Depending on the specifics of the problem at hand, sometimes this tuple can be extended with one of the following variables:

- b_0, a initial belief state, with $b \in B$ a vector of state probabilities, where for every $s \in S, b(s) \in [0, 1]$ and $\sum_{s \in S} b(s) = 1$. A POMD can be defined without specifying this initial belief, but specifying some initial belief distribution over initial states can help establish the boundaries of the reachable belief space. [56]
- $\gamma \in [0, 1]$, a discount factor for the rewards.

Historically, POMDs have been extensively studied for the case in which a full known model of the environment is available. In that case, the observation function O is known and a Bayesian method can be used to transform the observation signal into a probability distribution of being in a certain state. This signal can then be used as a new state signal for the original problem. [63]

In recent years, significant progress has been made in other POMD solving algorithms, improving the computational efficiency, supporting larger domains and loosening the requirement for a full model of the environment. [56] Key in this transition has been the development of Point-based Value Iteration (PBVI) methods like SARSOP, which explorer the belief space, focusing on reachable states, while continuously maintaining a value function approximation and applying a point-based backup operator.[31]

Finally, instead of trying to reconstruct some kind of Markov belief state representation, another approach is to leave the observation-representation unchanged and use methods that are not too adversely affected by the observations being non-Markov. Research into this area has been performed in Singh et al. [59].

The main proof provided by their research is that in a POMD, assuming ergodicity in the underlying MDP, both the $TD(0)$ and the Q-learning algorithm will converge to a value function that is not necessarily the correct value function. Secondly, they propose the use of stochastic policies instead of deterministic policies and prove that the return for a memory-less stochastic policy can be significantly better than the return for any memory-less deterministic policy. Finally, they present arguments why researchers should use the average payoff for POMDs instead of the discounted payoff. [59]

Based on this research, the authors, in a subsequent paper, present a new Monte Carlo algorithm for solving average payoff POMDs, without requiring the estimation of some belief state. This new algorithm includes stochastic policies in the search for an optimal policy in the policy space. [27]

4-3-9 State of the art in reinforcement learning methods

The field of reinforcement learning is rapidly developing, both in the application of reinforcement learning techniques to new applications and domains, as well as in the improvement and development of new techniques and methods. Due to the extent of the research being done, the summary of recent development below is not exhaustive but focuses instead on the developments most relevant to this thesis research.

Actor-critic methods

Actor-critic methods are special temporal difference methods in which the action selection and value estimation have been split up into two separate structures. The *actor* selects the actions and the *critic* estimates the (action-)value function and uses this to criticize the actor, as can be seen in figure 4-21.

This critique signal provided by the critic can be a scalar and is often called the TD error (δ_t). This value is then used by the actor to adjust its policy.

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{4-26}$$

Actor-critic methods have two main advantages. First of all, because the policy is explicitly stored, the effort of selecting an action is usually limited. This is especially relevant when the action space is continuous and thus an action must be picked from an infinite set.

Secondly, due to their structure, they allow for the implementation of a stochastic policy. As discussed in section 4-3-8, this can be advantageous when considering non-Markov problems.

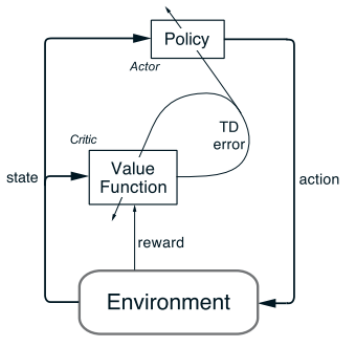


Figure 4-21: The actor-critic architecture [63].

Function approximation

So far, both the value function $V(s)$ and action-value function $Q(s, a)$ have been assumed to be of the tabular form, especially when discussing specific Temporal-Difference algorithms and eligibility traces. There are however two main challenges for such a representation of the (action-)value function when state-space or state-action space becomes large. In other words, when there are a large number of potential states and/or actions.

First, the computational requirements. The computational memory required to store large tables can become problematic and impractical. Similarly, the computational expense of manipulating such tables can increase significantly as well.

Secondly, the time and exploration needed to fill these large tables accurately can easily become problematic. If the required training becomes too time or resource intensive, using such an approach can quickly become impractical.

Combined, these challenges that arise when the combined state- and action-spaces become too large are often referred to as the *curse of dimensionality*.

To solve this issue, one might want to use a limited subset of the existing state space data and generalize this to produce a good approximation over a much larger state space. This is often referred to as *generalization*. [63]

Luckily, the idea of taking examples from a desired function and using these to construct a generalized approximation of the entire function is a well-studied area in mathematics. It is often referred to as *function approximation*. Within machine learning, this falls under *supervised learning*.

Gradient-descent methods are the most widely used category of methods used for function approximation in reinforcement learning. [63] They rely on a fixed number of real-valued parameters $(w_1, w_2, ..., w_n)^T$, that together are used to create a smooth differentiable function $\hat{v}$, which is then used to estimate the (action-)value function.

Gradient-descent methods approximate such a function by trying to minimize a certain cost function, for example, the Root-Mean-Square Error (RMSE). They do so by adjusting the parameters, after each sample data point, by a small amount in the direction that would most reduce the cost function for that data point.

When the approximate function $\hat{v}$ is a linear function with respect to the parameter vector $(w_1, w_2, ..., w_n)^T$, this method is called linear gradient descent. This is one of the most popular implementations of function approximation for reinforcement learning because the mathematical implementation is relatively simple. Furthermore, any method guaranteed to converge to a local optimum is guaranteed to converge to the global optimum. [63] Examples of linear gradient descent methods include radial basis functions, tile coding, and Kanerva coding.

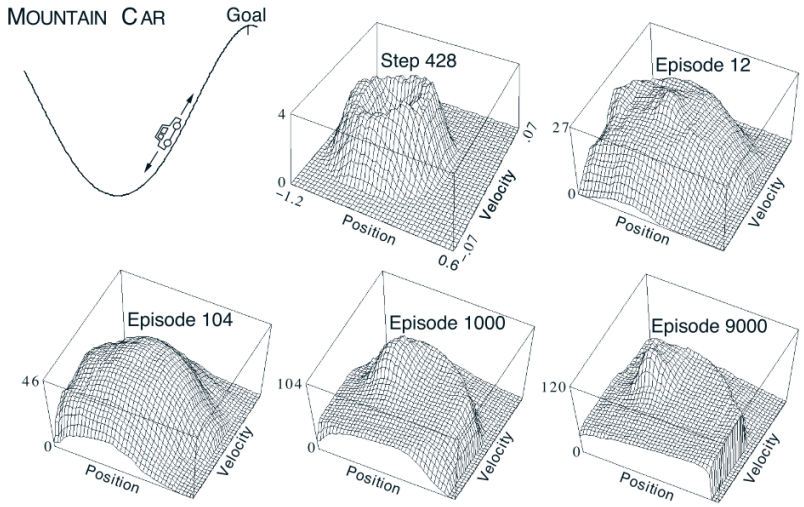


Figure 4-22: Function approximation of the cost-to-go function, the negative of the value function, learned during one run of the mountain-car task [63].

In addition to linear methods, also nonlinear gradient descent methods exist, the most famous one being multilayer neural networks. One of the first successful applications of this to reinforcement learning was the development of *TD-Gammon*, a reinforcement learning agent that learned how to play backgammon by combining the TD(λ) algorithm with a standard multilayer neural network as an approximation of the value function. The network consisted

of a layer of input units, a layer of hidden units and a final output unit. The input to the network was a representation of a Backgammon position and the output was an estimate of the value of that position. [63]

While some further progress was made in the years thereafter, the great breakthrough came with the successful usage of deep neural networks, networks with multiple hidden layers. This was termed *Deep Reinforcement Learning*. [5]

Deep Reinforcement Learning

The combination of deep neural networks with reinforcement learning has been an area of great recent interest and success. While there had been some research done in this area before, the kickoff to the real success of deep reinforcement learning was the work from Google Deepmind, published in 2015. [5] It described the development of an algorithm that could learn to play a range of Atari 2600 video games, directly from image pixels, and do so at a level comparable to professional gamers. They did so by developing a novel artificial agent, called a Deep Q-Network (DQN), based on recent advances in training of deep neural networks. This agent used a deep Convolutional Neural Network (CNN) to approximate the optimal action-value function. It did so using no prior knowledge and taking as input only the 4 most recent image frames and the current score, without explicitly designing the state space or action space.[39]

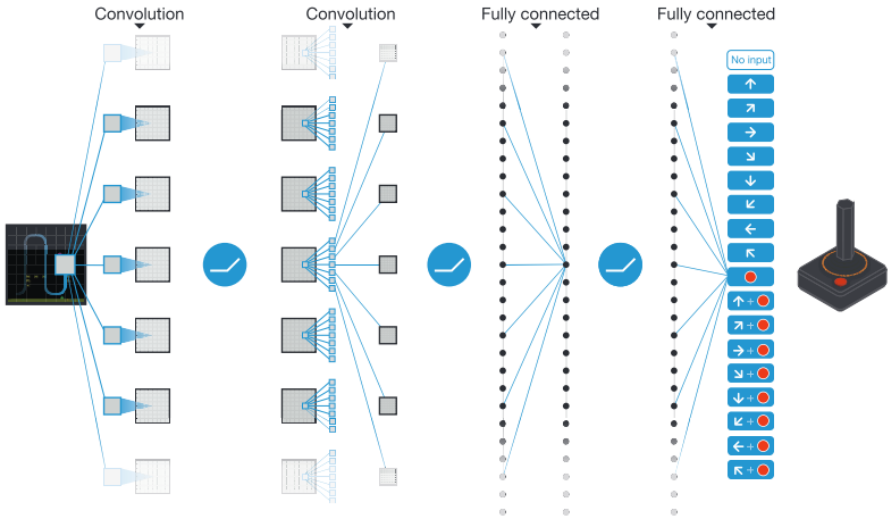


Figure 4-23: Schematic illustration of the convolutional neural network used by Google Deepminds DQN agent [39].

Another recent success of deep reinforcement learning is AlphaGo, the reinforcement learning agent that defeated the reigning human world champion in the game of Go. The game Go, is considered one of the hardest games in the world for Artificial Intelligence (AI), because of the incredible number of potential game scenarios and moves; there are around 250^{150} different sequences of moves. To master this game, AlphaGo used a combination of a multiple CNNs and heuristic search. [58]

While the given examples are merely two of the large success stories of reinforcement learning, the research currently being done in this area is extensive and the expectations are high. [58] They are however especially useful in problems with high dimensional raw input data (e.g., audio, text, and images), something that might be applicable, but is not necessarily expected in this research.

Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) is the idea of splitting up a large reinforcement learning problem into smaller pieces, under the assumption that solving them individually will solve the original task. [69] By doing so HRL can help deal with the *curse of dimensionality*, the well-known challenge for reinforcement learning problems when the combined state- and action-space become too large. It does so by splitting up to the state- and action-spaces in a hierarchy of sub-spaces, each with its own reinforcement learning problem.

Three main approaches to HRL covered in literature are the options formalism from Sutton et al. [62], the hierarchies of abstract machines (HAM) approach of Parr [43] and the MAXQ framework by Dietterich [11]. [6]

Transfer Learning

The core idea of transfer learning is that experience gained in learning to perform one task can help improve learning performance in a related, but different, task. [64] As such, it is not just relevant in reinforcement learning, but in all areas of machine learning. Furthermore, it is something that can be seen in humans as well. [65]

When the tasks are two truly independent tasks, it is referred to as inter-task transfer learning. When the initial task is part of the second task, or both are part of one larger task, it is often referred to as intra-task and it starts to partly overlap with hierarchical reinforcement learning.

Research has shown that transfer learning can speed up learning in reinforcement learning tasks, this research has however mostly been empirical [65].

Inverse RL

While normal reinforcement learning learns a policy from interaction with the environment, by selecting actions and receiving rewards, Inverse Reinforcement Learning (IRL) uses observations to learn a reward function. In other words; given the inputs to an existing agent, as well as the actions taken by the agent and a model of the environment, what is the reward function that the agent is optimizing?

Notable work in this area includes the development of apprenticeship learning by Pieter Abbeel and Andrew Y. Ng [1] in 2004. In apprenticeship learning, an expert is observed demonstrating the task that the RL-agent is trying to learn. It then uses IRL to learn the reward function, which is then used to develop a policy for the task itself.

One of the successes most relevant to flight control is the development of an autonomous helicopter capable of flying a wide range of highly challenging aerobatics, at the same level as a human expert pilot. [2]

4-3-10 State of the art in applying reinforcement learning methods on multirotors

While the previous section looked at the current state of the art in general reinforcement learning methods, it is also useful to investigate to what extent these methods have already been applied to the application domain under investigation, control of multirotors.

In general, it can be seen that most applications of reinforcement learning techniques for multirotors have focused on quadrotors, this could be attributed to the advantages of quadrotors as discussed in section 1. Furthermore, it is interesting to see how the reinforcement learning agent is placed within the control loop of a multirotor. This can range from full control, controlling the input to each motor, to only changing some parameters of a conventional controller.

A great example of applying reinforcement learning for full control is the paper *Control of a Quadrotor with Reinforcement Learning* [26]. In this research, a method is presented to control a quadrotor using a neural network trained with reinforcement learning methods. In this case, the created agent directly controls all actuators, the motors, of a quadcopter. The learning method used is a new algorithm, based on a value function trained using Monte-Carlo samples from on-policy trajectories and a policy optimization using natural gradient descent. The performance of the created policy is demonstrated by both reference tracking and stabilization tasks.

An implementation which focuses on the adjustment of parameters of a conventional controller using reinforcement learning techniques is the paper *Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning* [29]. In this research, a simple model of a quadrotor system is used in combination with gradient policy iteration to tune the gains on a quadrotor. Successful tests were performed in simulation as well as in a real flight test using the AR Drone 2 quadrotor.

Key challenges

When it comes to applying reinforcement learning to multirotors, some key remaining challenges can be identified that are especially important to this research. The most challenging being the challenge of safety, challenge of robustness, challenge of online efficiency and the challenge of sample efficiency. [35]

- The **challenge of safety** refers to limiting the risk of damaging the UAV itself, or its surroundings, due to unsafe actions, while still enabling and ensuring sufficient exploration. This risk is especially relevant for multirotors, because they are relatively fragile, expensive, can reach considerable velocities and have rotating parts.

  Reinforcement learning, on the other hand, requires states and action combinations to be visited, in order to learn from it. Especially in the beginning, when the RL agent has little experience and often relies on random actions to explore the learning space. [35]

  A proposed approach to handling this challenge of safety is the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) as proposed by Manucci in Mannucci

[35]. This algorithm functions as a sort of 'safety filter' between the agent and the environment. It monitors the states and actions and allows only actions that are not expected to cause a fatal transition. Furthermore, while the agent is exploring its environment, SHERPA accounts for backup-actions, a sequence of actions that cause the environment to transition (back) to a safe state. SHERPA has been successfully demonstrated on a simplified quadrotor control task and is as such very relevant to further research involving reinforcement learning and multirotors, including this research.

- Because of the challenge of safety, it could be desirable to train a RL agent for a multirotor in a safe or simulated environment, instead of in the actual environment. However, this brings rise to the **challenge of robustness**, the learned policy might not be safe or have satisfactory performance in the actual environment, due to differences between the training environment and the actual environment.

  This is often referred to as the reality gap and can be attributed to uncertainties in, or lack of, accurate environment and multirotor models. While this challenge can, in theory, be approached by using higher fidelity models, these models are sometimes nonexistent.

  Furthermore, if usage is made of some supervision during the training phase, the RL agent could start depending on this as part of its learned policy. This is something that special attention must be paid to, especially in the research proposed in this preliminary thesis. [35]

- The **challenge of online efficiency** concerns the combination of limited onboard computational capacity, and a highly dynamic platform requiring quick decisions. Reinforcement learning can become computationally expensive, especially when dealing with large state-action spaces and/or algorithms requiring many operations, like those using eligibility traces or function approximators.

  Furthermore, as suggested by Mannucci [35], the need to ensure safe exploration can add a further computational load, as well as the need for quicker reactions due to the insurgence of risks.

- The **challenge of sample efficiency** concerns the number of interactions with the environment (samples) that are required for an agent to find a good policy. Often reinforcement learning techniques have a low sample efficiency and thus require a large number of interactions with the environment. This requirement is mainly driven by the need for exploration of the environment. [76] This is especially the case when dealing with large state spaces, stochastic environments, or non-Markov Decision Processes since these will further increase the need for exploration of the environment, requiring more samples.

  A lot of the recent successes of reinforcement learning have thus been in the digital world, where samples can be generated fast, relatively cheap and sometimes even in parallel. When applying reinforcement learning to multirotors in the real world, this is usually not the case. The generation of samples is often limited by the time it takes to perform one episode, battery life, and availability of required assets.

  As a result, this challenge of sample efficiency might be another reason to perform (initial) training in a simulated environment, instead of the actual environment.

4-4 Conclusions

The goal of the literature study was to answer the first research question: *What is the state of the art in the related fields?* To do so, relevant literature has been identified and studied. Based on this, the subquestions of this first research question can be answered

SQ1.1 *What is the state of the art in the field of obstacle avoidance?*
Obstacle avoidance consists of two key parts: obstacle detection and taking mitigating action to prevent collision. Most of the current obstacle detection methods use either a range sensor or computer vision. Methods using a range sensor have achieved the highest accuracy and reliability. To reach this accuracy they require however heavy and costly sensors, which limits the flight time of a quadrotor. Computer vision methods have gained popularity and can achieve decent reliability, they are however dependent on good lighting conditions. It can thus be concluded that there would be added value in a low-cost obstacle avoidance method that does not require good lighting conditions or the addition of any sensors, especially for smaller quadcopters.

SQ1.2 *What is the state of the art in the field of obstacle-airflow interactions between a quadcopter and obstacles?*
Based on the limited literature and experimental data available, the ground and ceiling effect for a quadcopter with given dimensions can be approximated. According to the literature, these effects can be measured up to a distance of 5 times the rotor radius. For the wall effect, no approximation is available, however, an experiment by [37] indicated that the effect can be measured up to 0.35m for a drone of similar size as the Parrot Bebop 1 drone. It can thus be expected that the ground, ceiling and wall effects are large enough to be measured. Furthermore, the ground and ceiling effect can be simulated with some accuracy. For the wall effect, surfaces that are not flat and combinations of wall, ground and ceiling effect, it is however expected that simulation without further experimentation will be challenging.

SQ1.3 *What is the state of the art in the field of reinforcement learning?*
Reinforcement learning is a large and quickly developing field, containing a multitude of learning algorithms and architectures, each with its own strengths and weaknesses. Recent success in reinforcement learning has included the introduction of deep reinforcement learning, apprenticeship learning, and hierarchical reinforcement learning. A significant achievement has also been made in applying reinforcement learning to flight control and quadrotor control in particular. Reinforcement learning agents have been used both for full control of a quadrotor [26] and for adjustment of a conventional controller [29]. Key challenges in applying reinforcement learning to flight control remain the challenge of safety, the challenge of robustness, the challenge of online efficiency and the challenge of sample efficiency. Recently, a new algorithm called SHERPA was proposed by [35] that is potentially promising for this particular research. It deals with the challenge of safety by functioning as a safety net between the agent and the environment.

Chapter 5

Preliminary investigation

Based on the literature survey and the research questions, a preliminary investigation was carried out. The overall goal of this investigation will be described in section 5-1. Since the preliminary investigation concerns a reinforcement learning problem, the definition and setup of this problem will first be discussed in section 5-2, after which the approach to solving this problem using reinforcement learning is described in section 5-3. The results of this preliminary investigation will be described in section 5-4. Finally, the conclusions that can be drawn from these results, especially relating back to research subquestion 2.1 (SQ2.1), will be discussed in section 5-5.

5-1 Goal

The goal of the preliminary investigation is threefold: learning how to do such an experiment, investigating a novel RL-control scheme, and gathering an understanding of hyperparameters and their influence on the training & results.

First of all, by gathering experience with defining, analyzing and solving a reinforcement learning problem, valuable experience is gained that will help in the final research. This is especially important because of the large computational part that is involved with using reinforcement learning methods. While the theory of reinforcement learning follows from the literature survey, the actual computational implementation of these methods can be challenging. As such, both the experience of implementing these RL algorithms and the resulting code that might be reused, are expected to be of great use in the final research.

Secondly, as mentioned in section 2-2, this research adds to the state of the art by its unconventional placement of the reinforcement learning agent within the control loop. Because this is a novel method, the preliminary investigation serves to assess the viability of this method and identify challenges that would need to be addressed when using this control scheme for the final research.

Third and finally, there are not only multiple RL methods that can be used to solve a RL problem, but also a number of hyperparameters that can be set for each of these methods. Because, it is expected that in the full research it will be impractical to do a full grid search over all combinations of methods and different hyperparameter values, the preliminary research is used to gather an understanding of these methods, hyperparameters, and their influence on the training & result. By doing so it is expected that a substantiated choice can be made for the method & set of hyperparameters to be used in the final research.

By pursuing these goals, it is also expected that subquestion 2.1 from the problem statement can be answered. *How can the reinforcement learning-based control system best be setup?*

5-2 Setup

Based on the problem statement and research questions at hand, a discrete problem was created that contained a lot of the characteristics of the full problem but reduced in complexity. The setup of this problem is shown in figure 5-1.

In this setup, a ball with a certain mass is moving in a 1D world. Its position x lies on the vertical x-axis, and the ball can have a certain speed $\dot{x}$, and acceleration $\ddot{x}$, along this same axis.

The ball is controlled by a conventional PID controller which is performing a reference tracking task by applying an upward (positive) or downward (negative) force on the ball. The reference signal is a randomly generated sinusoidal signal.

Placed within the setup are two walls, one placed at a certain distance above the starting position of the ball, and one placed below. When the ball comes in the proximity of one of these walls, these walls will exert a certain pulling force on the ball, similar to the ceiling effect.

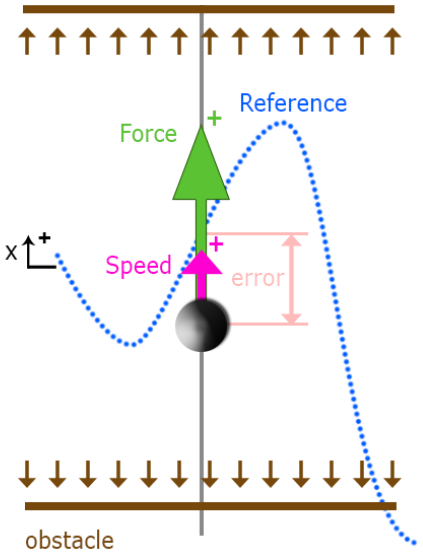


Figure 5-1: Sketch of the dynamics of the preliminary investigation .

A reinforcement learning agent is then tasked with preventing the ball from hitting the wall. It can do so by overriding the conventional controller and instead applying its own force to the ball. As in any reinforcement learning problem the agent receives a state and reward from the environment, based on the previous state, the action chosen by the agent and the environment dynamics.

5-2-1 Environment dynamics

The environment dynamics are defined by the following equations.

$$\ddot{x} = \frac{1}{m}(F_g + F_{wall}) \quad \dot{x} = \int_0^t \ddot{x}dt \quad x = \int_0^t \dot{x}dt \tag{5-1}$$

With m the mass of the ball, F_g the force exerted by either the conventional controller or the reinforcement learning agent, and F_{wall} the force exerted by the walls. Furthermore, x denotes the current position of the ball, $\dot{x}$ the current speed and $\ddot{x}$ the current acceleration.

Also part of the environment is the reference signal $x_{\text{ref}}(t)$. This is the signal that the conventional controller is trying to follow, and it is defined as

$$x_{ref}(t) = \sum_{i=1}^{N_{sin}} a_i * \sin(b_i 2\pi t + c_i) \tag{5-2}$$

Where N_{sin} is the number of individual sine waves that together comprise the reference signal, and a_i, b_i, c_i randomly generated amplitudes, periods and phase shifts for each of the sine waves.

From the perspective of the reinforcement learning agent, also the conventional controller is part of the environment. In this environment, the conventional controller is a PID controller, a widely used control loop feedback mechanism that uses the difference between the desired reference signal and the measured process variable. A PID controller then calculates the error between these two signals and applies a correction based on the weighted proportional, integral, and derivative terms of this error.

In this case, the desired reference signal is $x_{ref}(t)$, as given by equation 5-2, the measured process variable is $x(t)$ and the correction the PID controller can apply is F_{conv}. The PID controller is then described by equation 5-3, where K_P is the proportional gain, K_I the integral gain and K_D the derivative gain.

$$F_{conv}(t) = K_P e(t) + K_I \int_0^t e(t')dt' + K_D \frac{de(t)}{dt}, \quad e(t) = x_{ref}(t) - x(t) \tag{5-3}$$

It must be noted that within the environment, certain bounds are placed on the force that can be applied, either by the conventional controller or by the reinforcement learning agent.

Finally, the force exerted by the walls, F_{wall}, is defined as the sum of the forces exerted by each individual wall, where the force from each wall depends on some constant K_{wall}, the distance between the position of the wall x_{wall} and the ball x, and the minimum distance from which the wall exerts this effect d_{wall}.

The equation below is constructed such that the force exerted by each wall increases linearly from $|F_{wall,j}| = 0$ at distance d_{wall} from the wall, to $|F_{wall,j}| = K_{wall}$ when $x = x_{wall,j}$, the ball is effectively colliding with the wall. Furthermore, the direction of the force is always such that it is pulling the ball closer to the wall, given that $K_{wall} > 0$. This to prevent an inherent safety or stability that might otherwise occur in the environment.

An illustration of what the resulting wall forces would look like can be seen in figure 5-2.

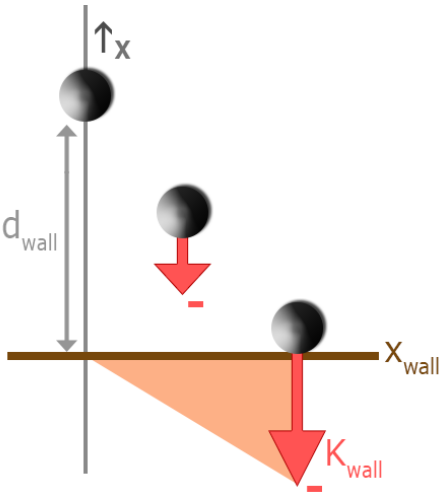


Figure 5-2: Linear pulling force exerted by a wall.

$$F_{wall} = \sum F_{wall,j}, \quad F_{wall,j} = \begin{cases} K_{wall}\left[\frac{x - x_{wall,j}}{d_{wall}} + \frac{x_{wall,j}}{|x_{wall,j}|}\right] & \text{if } x - x_{wall,j} < d_{wall} \\ 0 & \text{otherwise} \end{cases} \tag{5-4}$$

The specific parameters used during the preliminary investigation for the ball, conventional controller, reference signal and wall dynamics can be found in table 5-1.

Ball dynamics								
Ball mass (m)	1	kg	Force bounds	-20, 20	N			
Conventional controller & Reference signal								
K_P	5.0		a_i	$\in [-4, 4]$	-	N_{sin}	5	-
K_I	0.0		b_i	$\in [0.01, 0.05]$	-			
K_D	2.0		c_i	$\in [0.0, 0.0]$	-			
Walls								
d_{wall}	1	m	K_{wall}	1	-	x_{wall}	4 & -4	m

Table 5-1: Parameters used during the preliminary investigation.

5-2-2 Control scheme

As mentioned in the previous section, the ball has two controllers: a conventional controller, trying to follow a reference signal, and a reinforcement learning controller trying to prevent the ball from hitting the wall. The control scheme of this setup can be seen in figure 5-3.

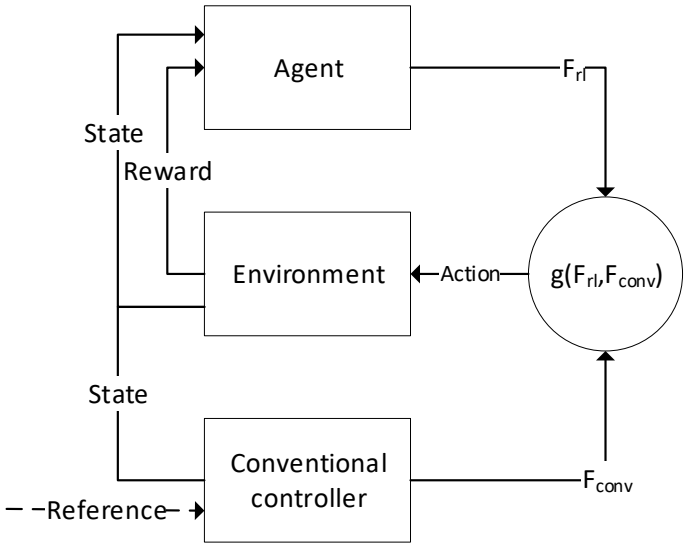


Figure 5-3: Control scheme of the preliminary investigation setup.

Because in the end there is only one input force to the ball F_g, the output from these controllers needs to be combined, this is done using the function $g(F_{rl}, F_{conv})$. When considering potential options for this function two immediate solutions come to mind.

A first solution could be to sum the inputs from the controllers, so $F_g = F_{rl} + F_{conv}$. That way the input from both controllers is truly combined, and both controllers have the same level of influence.

However, when looking at the problem as a representation of a quadcopter trying to follow a path, while simultaneously avoiding obstacles, it becomes clear that the obstacle avoidance part is often actually more important than the path following part. Because, if a collision occurs, the impact to the quadcopter is usually so high, e.g. a crash, that the path following

can't be continued anyhow. Therefore it was decided to give the controller trying to prevent a collision, the reinforcement learning agent, a higher level of control.

In this second solution, the reinforcement learning agent has the option to override any input from the conventional controller. The implementation of this is shown in equation: 5-5.

$$F_g = g(F_{rl}, F_{conv}) = \begin{cases} F_{rl}, & \text{if } F_{rl} \neq None \\ F_{conv}, & \text{otherwise} \end{cases} \tag{5-5}$$

One might argue that it would also be possible to give this higher level of influence to the reinforcement learning agent, while still defining F_g as the sum of F_{rl} and F_{conv}, by making for example use of boundaries on the input force. As previously shown in table 5-1 the absolute input force on the ball is limited to 20 Newton. By enforcing this limit on F_{conv} and the resulting F_g, but allowing F_{rl} to vary between -40 Newton and 40 Newton, the reinforcement learning agent could then effectively also override any conventional control.

While this might seem attractive from a theoretical perspective, this would also mean that the action space for the reinforcement learning agent would need to double to keep the same resolution. Furthermore, the state space would have to include F_{conv} as well, something that might not be required otherwise. As a result, it can be expected that such an implementation would require a longer training phase. Therefore the implementation given in equation 5-5 was preferred.

5-2-3 Making it a reinforcement learning problem

To make this problem a reinforcement learning problem, the states, actions, rewards, and termination need to be defined. Furthermore, it was chosen not only to simulate in discrete time but also to use discretization for the states and actions, therefore also these discretizations will be discussed.

States

Assuming the reference signal to be truly random, this RL problem can be seen as a partially observable Markov decision process. The speed ($\dot{x}$) and acceleration ($\ddot{x}$) are available to the agent, but not the position (x) or the reference signal (x_{ref}). What is available however is the estimate of external force F_{ext}, which can serve as an indication of the distance to the wall.

Overall, the following states are available to the reinforcement learning agent

- 'dx', the current speed of the ball, also referred to as $\dot{x}$.
- 'F_ext', an estimate of the external force on the ball, this is calculated, using the following equation.
$$F_{ext}(t) = \ddot{x}(t-1) * m - F_g(t-1) \tag{5-6}$$
- 'rl_intervention', a 1-bit memory state. It starts of as 0, but turns to 1 after the reinforcement learning agent has intervened, in other words, when it has performed an action which is not 'None'.

Actions

Based on these states, the agent can then determine his action. There is only 1 action dimension, which is the force F_{rl}. This action can be an actual force, which should be within the force bounds set by the environment, or 'None', an indication that the conventional controller can perform an action without being overruled by the reinforcement learning agent.

$$F_{rl} = \begin{cases} \text{a force } F, \text{ where: } F_{min} \leq F \leq F_{max} \\ None \end{cases} \tag{5-7}$$

Rewards

Because a reinforcement learning agent is trying to maximize its rewards, the choice of rewards can be key. For this problem, it was chosen to only work with negative rewards. This makes the theoretical maximum sum of rewards 0.

Most importantly, if the ball hits the wall. A negative reward of -1000 is given.

Furthermore, if the ball has not yet been in close proximity to the wall, the following rewards are provided to the RL agent:

- 0 when $F_{rl} = None$
- -10 in all other cases

However, if the distance of the ball to the closest wall has at one timestep in the episode been smaller than the wall effect distance, the following reward is provided instead.

- -0.5, regardless of the action taken

These rewards are chosen such that the reinforcement learning agent is stimulated to prevent collisions with the wall, but limit overriding of the conventional controller when not close to the wall.

The distinction between the rewards when the ball has been close to a wall, and the rewards when it has not, is made to prevent the reinforcement learning agent from relying on the conventional controller when performing an obstacle avoidance maneuver. This was implemented after initial results showed RL agents using the *None*-action while 'saving' the ball from hitting the wall, thereby preventing a non-zero reward, but introducing undesirable oscillations between the conventional controller and the reinforcement learning agent. An example of this can be seen in figure 5-4 This also leads to the first sub conclusion of the preliminary research:

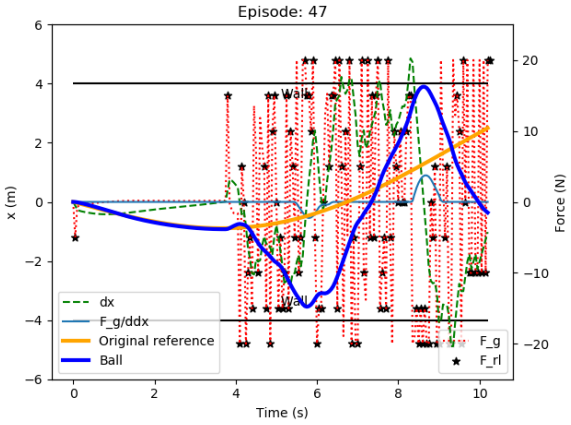


Figure 5-4: Typical episode before a distinction was made in the rewards before and after the ball had been close to a wall.

SC1 The reward structure must be set up such that it does not stimulate unwanted oscillatory behavior between the conventional controller and the reinforcement learning agent.

Initialization

While the environment dynamics describe the transition between states, based on the actions chosen by the agent, there must also be an initial state, a starting state. In his reinforcement learning problem this starting state was kept constant at $x = 0, \dot{x} = 0, \ddot{x} = 0$. Because $x = 0$ is not in proximity to any wall, also $F_{ext} = 0$. Furthermore, because the reinforcement learning agent has not yet performed an action 'rl_intervention' = 0.

Finally, because the reference signal is a sum of sine waves, and the deviation c_i for all these sine waves is set to zero, the reference signal will also be 0 at $t = 0$. The ball thus starts exactly on the reference signal, without a tracking error.

Termination

There are three possible endings to an episode.

- The ball hits the wall, resulting in a reward of -1000
- The ball was about to hit the wall but is successfully saved by the reinforcement learning agent. No additional rewards are given. The conditions for the end of a successful safe are defined to be the following:
  - The ball has been close to the wall; the distance of the ball to the closest wall has at one timestep in the episode been smaller than the wall effect distance.
  - The current speed of the ball is small; $-0.5 < \dot{x} < 0.5$.
  - The current distance to the wall is larger than the wall effect distance:

$$\min_j |x - x_{wall,j}| > d_{wall} \tag{5-8}$$

  - The rl agent has at one point in the episode performed an action, which was not 'None'; the state 'rl_intervention' is thus 1.
- The maximum number of timesteps is reached. In the training, this was set to 1,000 timesteps. No additional rewards are given.

Discount factor

As can be concluded from the termination requirements, all episodes are finite, therefore it is not an absolute requirement to have the discount factor $\gamma < 1$. In the context of obstacle avoidance, it might even be argued that the negative impact of a collision in the future shouldn't be discounted at all. Therefore, in this reinforcement learning $\gamma = 1.0$.

Discretization

Like most reinforcement learning setups, the training and evaluation of the agent were performed in discrete time. For this, a timestep of 0.05 seconds was used. In other words $dt = 0.05$.

Furthermore, the states and actions were also discretized, making it a fully discrete Reinforcement Learning problem. This discretization has been determined by trying different options and comparing results manually, using plots similar to figure 5-5 and figure 5-6. In the end, the following discretizations were found to adequately capture the environment dynamics.

- 'dx' was discretized linearly into 20 options, from $-5m/s$, up to, and including, $+5m/s$.
- 'F_ext' was discretized linearly into 11 options, starting from $-1.0N$ and up to, and including, $+1.0N$.
- 'rl_intervention' was already discrete (0 or 1)
- 'F_rl' has 10 options: 9 linear options starting from $-20N$ and up to, and including, $+20N$. Furthermore one option '99', which represented the 'None' action.

The size of the state-space is thus 20*11*2, for a total of 440 unique states. Combined with the 10 action possibilities this results in a state-action space of size 4400.

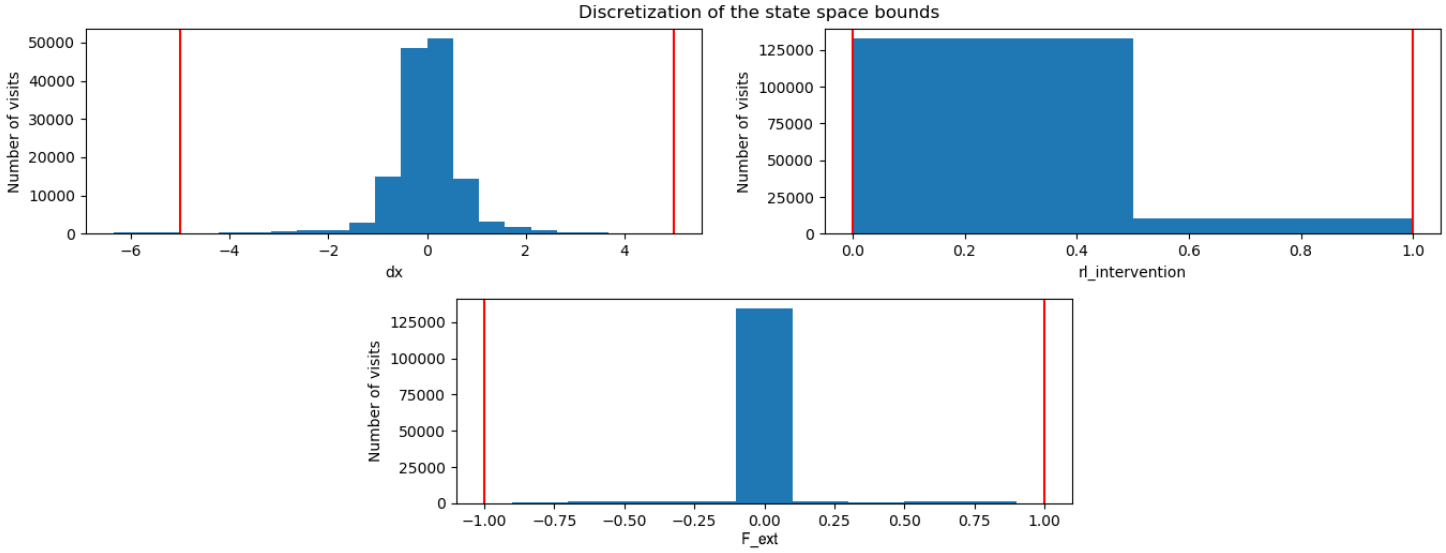


Figure 5-5: Number of visits for each of the discretized states, during 500 episodes.

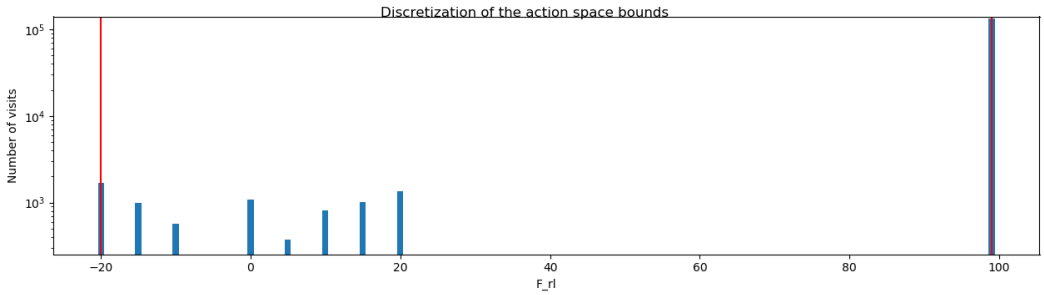


Figure 5-6: Number of times each of the discrete actions is taken by an RL-agent during 500 training episodes.

5-3 Approach

Now that the reinforcement learning problem has been defined, reinforcement learning agents can be trained. The chosen approach was to perform a grid search; training multiple agents, each with its own combination of reinforcement learning method and set of hyperparameters, evaluating these agents, and comparing the results.

5-3-1 Methods and hyperparameters

As mentioned in section 4-3-6, there are three main reinforcement learning solution methods: Dynamic programming, Monte Carlo methods, and Temporal-Difference learning. Since dynamic programming requires a complete model of the environment, which will not be available in the final object avoidance application, only Mont Carlo and Temporal-Difference methods were considered for the approach.

This still leaves open a wide variety of specific methods, variations, and implementations. Especially for the Temporal Difference methods. Therefore it was chosen to focus on the following 3 distinct methods that have been well-researched and form the basis of most other methods.

- Monte Carlo
- SARSA(λ), *on-policy TD*
- Q-learning(λ), *off-policy TD*

For both SARSA(λ) and Q-learning(λ), accumulating traces were used when $\lambda > 0$. To ease the computational load, traces were cutoff, set to zero, when $E_t(s) < 0.0001$.

In addition to the reinforcement learning methods, there are also some hyperparameters that have a large influence on the learning process of these agents. The three hyperparameters that are most relevant to these methods are the following:

- ϵ, the exploration rate. All algorithms were implemented with ϵ-greedy exploration, where this parameter determines the percentage of cases in which a non-greedy action is (randomly) chosen. It is applicable to all 3 methods.
- α, the learning rate. This parameter determines to what extent the value function should be updated based on the new estimate. How this is usually implemented can be seen in equation 4-11. It is applicable to both the SARSA and the Q-learning method.
- λ, the decaying factor of eligibility traces. This parameter determines whether eligibility traces are used, and if so, to what extent previous states are updated. A further description can be found in section 4-3-7. This parameter is applicable to both the SARSA and the Q-learning method.

Of course, defining which hyperparameters to variate in the grid search is not enough. The values that will be considered for these parameters will also need to be defined. Important here is that the options span the available parameter space adequately.

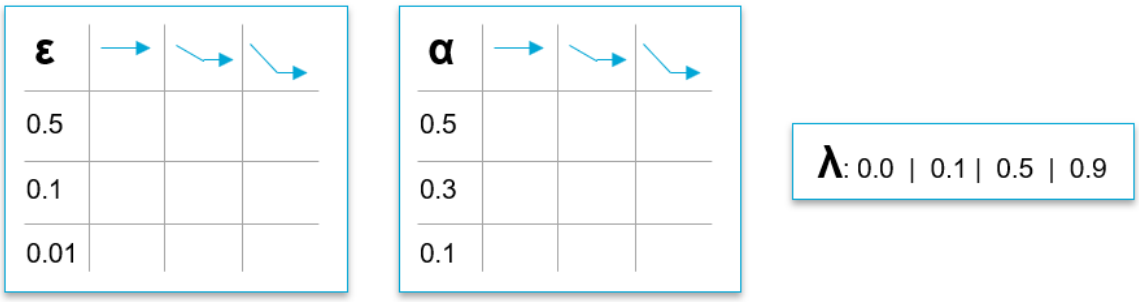


Figure 5-7: Grid search on the hyperparameters.

The exploration rate

For the exploration rate ϵ, three different starting values were considered: $\epsilon = 0.5, \epsilon = 0.1$ and $\epsilon = 0.01$. These values were chosen based on literature, specifically [63].

Furthermore, research has shown that in some situations reinforcement learning agents can benefit from a decreasing exploration rate. Therefore it was determined that next to the 3 potential starting values, there were also 3 potential transgression methods.

- ϵ is constant.
- ϵ linearly decreases to a tenth of its starting value in the first half of the episodes, and is constant thereafter.
- ϵ linearly decreases to zero in the first half of the episodes, and stays zero thereafter.

This gives a total of 9 options that are considered for the exploration rate.

The learning rate

For the learning rate α also three different starting values were considered: $\alpha = 0.5, \alpha = 0.3$ and $\alpha = 0.1$. These values were once again chosen based on [63].

Similar to the decay of the exploration rate, decreasing the learning rate throughout the episode has also been considered. For α, decay is given by the following formula. With i the current episode, α_0 the initial value and k a parameter that indicates the speed of decay.

$$\alpha_i = \alpha_0 \frac{k}{k+i} \tag{5-9}$$

Two potential values for k where considered, which resulted in the following 3 options for transgression.

- α is constant
- α decreases to half its initial value after the first half of the episodes: $k = N_{\text{episodes}}$
- α decreases to a quarter of its initial value after the first half of the episodes: $k = \frac{N_{\text{episodes}}}{2}$

In total there are thus 9 different options for α.

Eligibility traces

For λ, 4 potential values were considered: $\lambda = 0.0, \lambda = 0.1, \lambda = 0.5$ and $\lambda = 0.9$. The case where $\lambda = 0.0$ means that no eligibility traces are used.

Combinations

Since these hyperparameters cannot be considered independent, it was not only necessary to vary them individually, but also their combinations. This resulted in the following agents being trained:

- 9 Monte Carlo agents with varying ϵ
- 324 SARSA(λ) agents with varying ϵ, α and λ
- 324 Q-learning(λ) agents with varying ϵ, α and λ

For a grand total of 657 unique agents.

5-3-2 Initialization and training

Next to the choice of reinforcement learning method and hyperparameters, a choice also has to be made for the initialization of the value function Q for SARSA and Q-learning, since they make extensive use of bootstrapping. In this case, it was chosen to initialize the Q function at zero for both methods. Since all rewards received by the agent are negative this will ensure exploration.

Each of these 657 agents was then trained for 5,000 episodes with a maximum length of a 1,000 timesteps per episode. Each of these episodes was simulated using a randomly generated reference signal in order to ensure that the agent did not become dependent on one specific reference signal. The maximum number of timesteps was chosen such that most reference signals crossed the position of the wall at some point during an episode.

5-3-3 Evaluation

All 657 trained agents were then evaluated during an additional 50 episodes. Each of these episodes took once again place with a randomly generated reference signal. Furthermore, these evaluations were performed greedy, the exploration rate was set to zero for all agents during this evaluation phase. Also, learning was stopped, neither the value functions nor the policies were updated during these evaluation episodes.

After the evaluation episodes, the performances of all agents were compared. This was done based on two key performance indicators.

- The average total reward, a value that contains both information about the number of crashes (since these result in a high negative reward), and the number of actions required to prevent crashes.
- The % of evaluation episodes that resulted in a crash.

5-4 Results

In this section, the results of the preliminary investigation will be presented and discussed. First, a baseline, to compare reinforcement learning agents with, will be established. Then the training performance of different RL methods and hyperparameters will be presented, after which also their evaluation results will be discussed. Finally, the best-performing agents will be investigated.

5-4-1 The baseline

In order to understand and interpret results, it is important that a baseline is set. Therefore, the same training and evaluation process that will be used for the reinforcement learning agents has also been applied to two trivial agents; a no-action agent and a random-action agent.

The no-action agent performs no true action, instead, it always picks the *None* option, such that the conventional controller is fully in control. The ball will thus simply try and follow the reference signal. Its ability to do so accurately depends on the reference signal and the gains chosen for the PID controller.

This tracking ability can be confirmed by running the environment, with the no-action controller, but without walls placed in the environment. When doing so for 10,000 episodes, and looking at the difference between the position of the ball x and the reference signal x_{ref}, the mean absolute tracking error is found to be only 0.04m. Furthermore, this is fairly consistent among episodes, as seen in figure 5-8. Considering the size of the environment, the walls in this simulation are placed at -4 and +4 meters, so 8 meters apart, it can be concluded that the tracking error is negligible when there is no wall present.

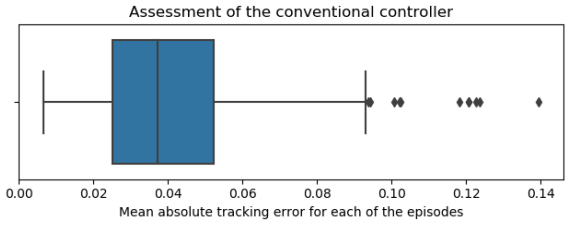


Figure 5-8: Mean absolute tracking error in episodes with no obstacle present, for 10,000 episodes.

The random-action agent, like its name suggests, picks a random action each timestep. This could be a force from the discrete force-options or the *None* option.

Even though neither of the two agents has a learning capability, the same procedure has been followed as for the reinforcement learning agents, to allow for comparison. These trivial agents thus went through a 'training' phase of 5,000 episodes. The results of this can be seen in figure 5-10. These training results can be used as a baseline to compare the training performance of reinforcement learning agents with.

Additionally, both agents went through a greedy evaluation phase of 50 episodes. This is equal to the evaluation approach used for all other agents and resulted in the performance indicators shown in table 5-2.

What is interesting to note from these results is that not all episodes necessarily end with a crash or a save. As noted in section 5-2-3, it is possible for an episode to reach the end of the

episode with the ball never colliding with a wall, nor satisfying the requirements for a save. The reason this can happen is that the reference signal is randomly generated each episode, and its path might not necessarily come close to the wall within the set number of time steps. Examples of episodes in which this is the case can be seen in figure 5-9. To quantify this effect and establish a baseline, 10,000 episodes were run with the no-action controller. From these episodes, 12.8% resulted in a timeout. Considering that only 50 episodes were run in the initial evaluation phase, the difference between this number and the percentage timed out shown in table 5-2 for the no-action controller is unsurprising.

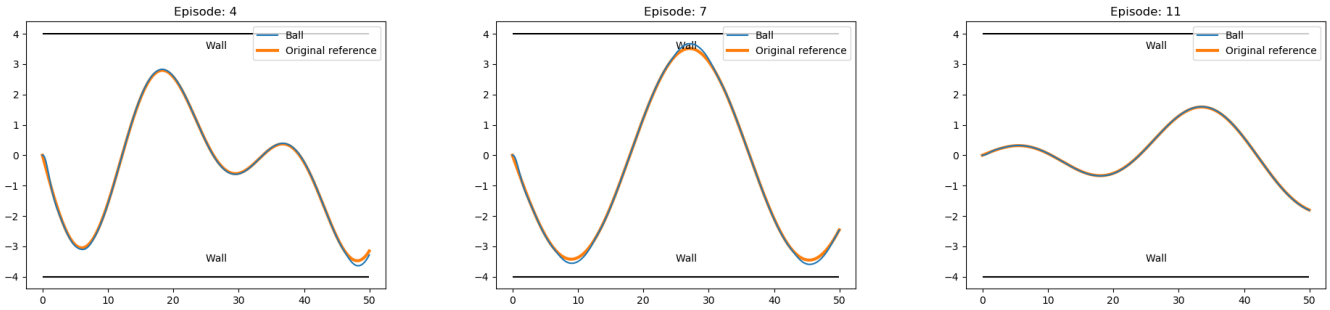


Figure 5-9: Three sample episodes resulting not in a crash or save, but in a timeout, when no-action is performed by the agent.

Overall, looking at figure 5-10 and table 5-2, it can be concluded that neither of the agents shows any type of learning, as expected, and neither achieves a desirable performance. Furthermore, it can even be argued that the random action agent has a negative influence because it results in a larger percentage of episodes resulting in a crash than if no action were to be taken by the agent at all. In other words, it results in the ball colliding with the wall even though the reference signal does not.

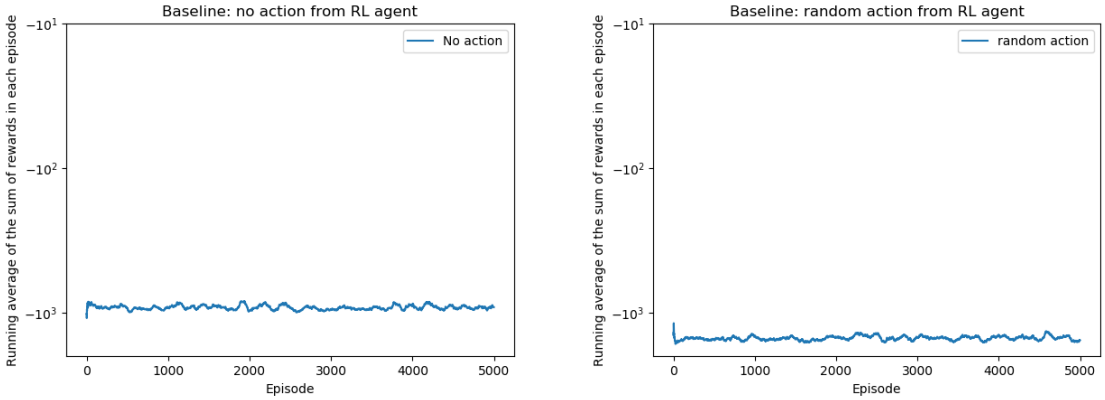


Figure 5-10: Training performance of a no-action and a random-action agent

	Average reward	% crashed	% saved	% timed out
No-action	-918	86%	0%	14%
Random-action	-1558	96%	4%	0%

Table 5-2: Performance of the baseline agents during 50 evaluation episodes.

5-4-2 Training performance

Now that the baseline has been set, the actual reinforcement learning agents can be trained. This has been done for all 657 agents, following the approach described in 5-3, for 5,000 episodes each.

Due to the large number of agents involved, it would be impractical to investigate the learning of each of these agents. Instead, the learning performance of the best and worst performing agents might be considered to give an idea of the learning ability of a specific method.

Learning performance of the best agents for each method

The performance during learning for the best agents is what is shown in figure 5-11, where best is defined as the agent with the least negative average reward during the successive greedy evaluation phase. A method is for practical purposes defined as a combination of an algorithm (Monte-Carlo, Q-learning or SARSA) and a choice of eligibility trace (λ). There are thus a total of 9 methods; 1 Monte-Carlo method, 4 Q-learning methods and 4 SARSA methods.

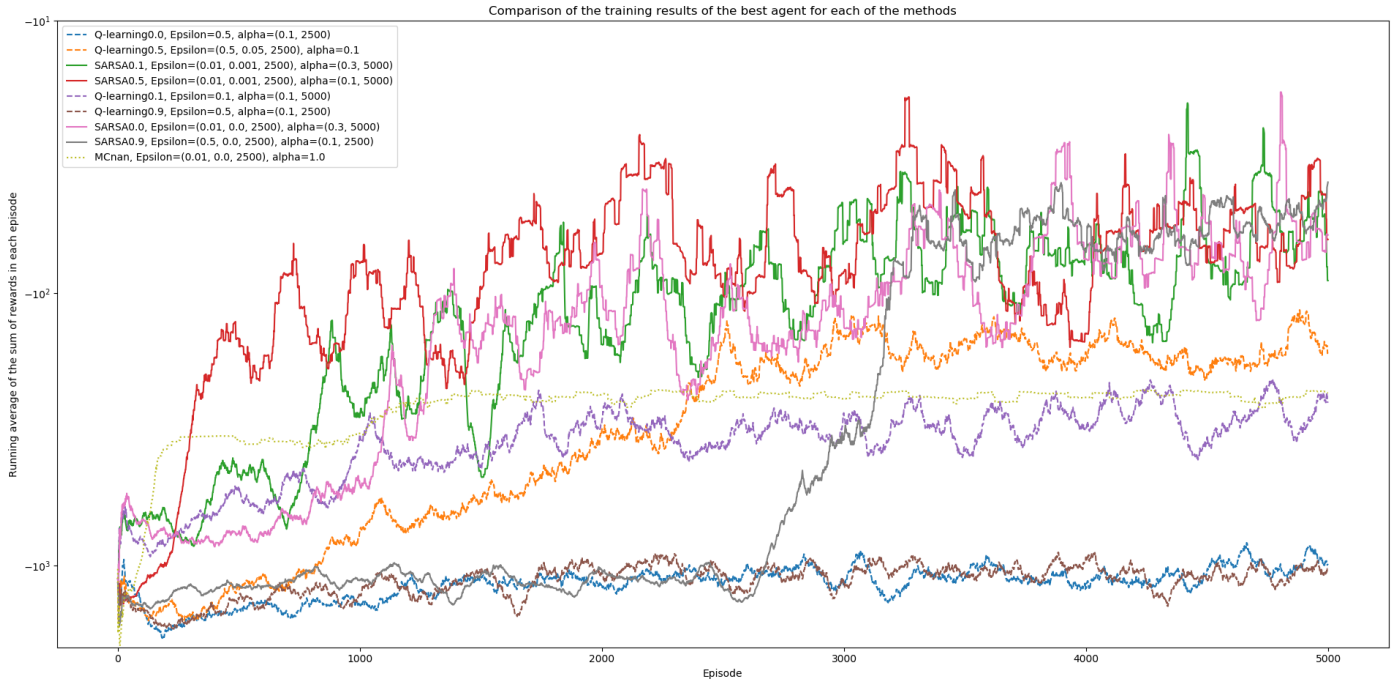


Figure 5-11: Comparison of the training results of the best performing agents for each of the reinforcement learning methods. *Note: a running average of 100 episodes is used to make the graph more readable.*

When studying the graph above it can immediately be noted that, in contrast to the baseline, most of the methods clearly show improvement in training performance during the training phase. Each of these agents increases its average total reward in training, except for Q-learning(0.9) and Q-learning(0.0). A clear indication that at least most of the reinforcement learning methods are working.

What is important to note here is that the performance in training can be an indicator of the agent learning, but the lack of an increase in training performance does not necessarily mean that the agent has not learned anything.

The reason for this is that the agent could be using an off-policy method, in other words, the agent is not following its current estimate for the optimal policy, *the estimation policy*, but instead relies on a *behavior policy* for exploration.

Furthermore, the training performance of a method might look worse than its evaluation performance if the method relies on a large exploration rate ϵ.

For the two methods that do not show improvement in performance during the training phase, this is both the case. Both methods use the off-policy Q-learning algorithm and rely on a large constant exploration rate $\epsilon = 0.5$. Therefore it is perfectly possible that these agents have learned as well, this will need to be checked during the greedy evaluation.

Similarly, regarding the learning rate of each of the methods, it might be tempting to make observations like '*the training performance of most of the SARSA methods increases faster than the training performance of the Q-learning methods*'. This does however not mean that the SARSA methods necessarily learn faster than the Q-learning methods. The Q-learning methods are off-policy methods and the agents shown here rely on a significantly larger (initial) exploration rate.

Finally, when looking at some of the specific lines, three interesting things can be noticed. First of all the learning performance of the Monte Carlo method seems much more constant than that of the other methods. Secondly, the training performance reached by the best Monte Carlo agent is lower than that of the best SARSA and Q-learning methods. Third, and finally, when looking at the learning performance of SARSA(0.9) it can be seen that the learning performance goes up drastically between episode 2500 and 3500. This is especially interesting when noting that this agent started off with quite a high epsilon ($\epsilon = 0.5$), but during training, this decreases, and after episode 2500 this was set to be constant at $\epsilon = 0.0$. This could, of course, be coincidental, but considering the previous discussion about the influence of the exploration rate on the training performance, expecting this to be causal would not be unreasonable.

Learning performance of the worst agents for each method

By investigating not only the best but also the worst performing agents for each method, as shown in figure 5-12, the robustness with respect to hyperparameters can be tested. The worst agent for each method is defined as the agent with the most negative average reward during the successive greedy evaluation. Four observations can be made from this graph and the comparison with figure 5-11:

- While the best agents from the SARSA methods had the highest performance training, compared to other methods, the worst SARSA agents show no training improvement at all.
- All 4 of the worst agents trained using Q-learning show some increase in performance during the learning phase.

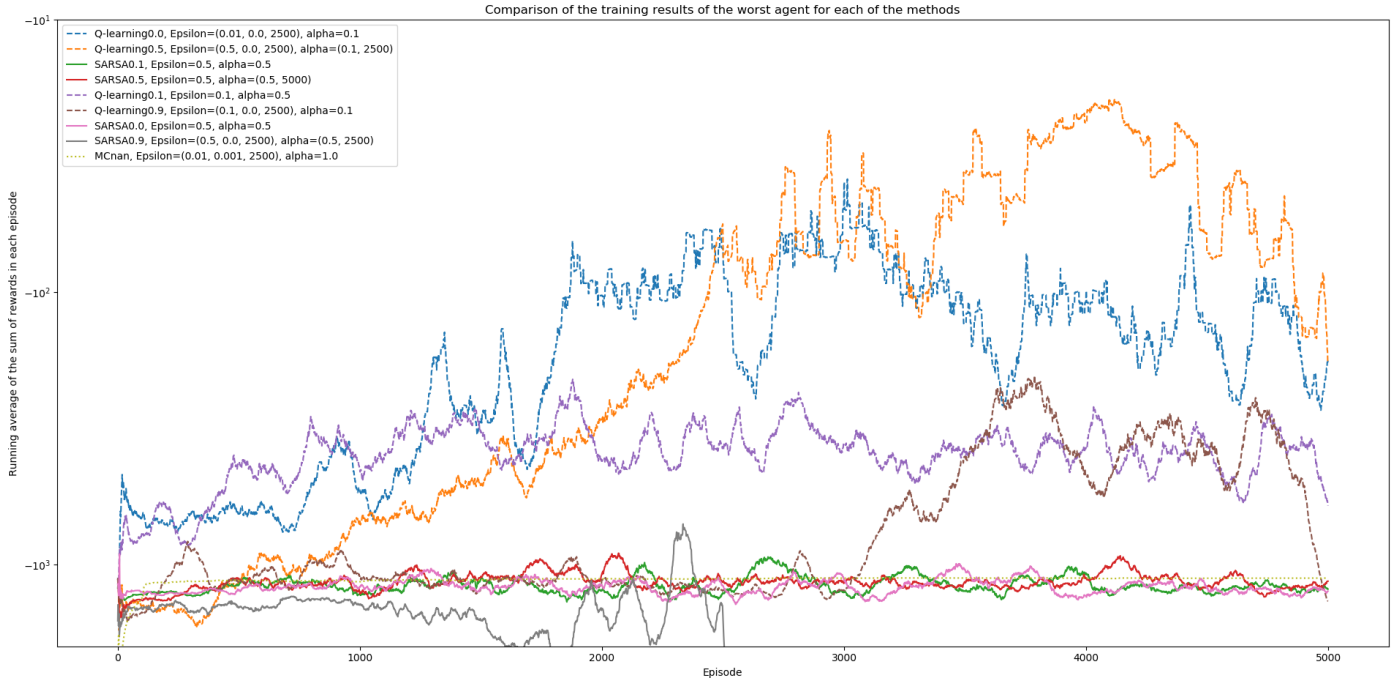


Figure 5-12: Comparison of the training results of the worst performing agents for each of the reinforcement learning methods. *Note: a running average of 100 episodes is used to make the graph more readable.*

- The average sum of rewards in training, after 5,000 episodes, for the best agent trained with Q-learning(0.5) is comparable to that of the worst agent trained with Q-learning(0.5).
- The average sum of rewards in training, after 5,000 episodes, for the worst agent trained with Q-learning(0.0) is higher than that of the best agent trained with Q-learning(0.0).

The combination of these observations and discussions above lead to the following sub conclusions, based on the results from the training phase:

SC2 For all 9 tested methods there is some set of hyperparameters that leads to a better performance in training than the baseline methods.

SC3 For Q-learning agents, the average total reward during training is not a good predictor of performance during evaluation. This is most likely due to the higher exploration rate used by the top Q-learning agents.

5-4-3 Initial greedy evaluation

While the training performance can be an indicator of learning, the leading measure of performance for this case was chosen to be the performance during a fully greedy evaluation. This was chosen because this enables an equal comparison between the performance of on-policy and off-policy algorithms, and among methods with different exploration rates ϵ.

One might argue that a fully greedy, non-learning policy, might not be the desired policy to be used in a real-world application; it might be desirable to have an agent that always keeps on learning and adjusting to its environment. However, the goal of this preliminary investigation is more on getting an understanding of different algorithms and sets of hyperparameters, and not as much on developing a real-world application. Therefore, having an equal comparison, without the random component introduced by ϵ, is preferred.

As mentioned in section 5-3, a greedy evaluation of 50 episodes was performed for each of the 657 agents. The average total rewards from these evaluation episodes were then compared and used to find the 'best' and 'worst' agents shown in figures 5-11 and 5-12. Their actual evaluation performance can be seen in tables 5-3 and 5-4.

			Performance during 50 greedy evaluation episodes			
Algorithm	ϵ	α	**Average reward**	**% crashed**	**% saved**	**% timed out**
Q-learning(0.0)	0.5	(0.1, 2500)	-16.62	0%	98%	2%
Q-learning(0.1)	0.1	(0.1, 5000)	-21.74	0%	98%	2%
Q-learning(0.5)	(0.5, 0.05, 2500)	0.1	-16.88	0%	88%	12%
Q-learning(0.9)	0.5	(0.1, 2500)	-33.96	0%	98%	2%
SARSA(0.0)	(0.01, 0.0, 2500)	(0.3, 5000)	-34.22	0%	92%	8%
SARSA(0.1)	(0.01, 0.001, 2500)	(0.3, 5000)	-19.10	0%	92%	8%
SARSA(0.5)	(0.01, 0.001, 2500)	(0.1, 5000)	-19.59	0%	92%	8%
SARSA(0.9)	(0.5, 0.0, 2500)	(0.1, 2500)	-52.89	0%	88%	12%
MC	(0.01, 0.0, 2500)	1.0	-225.75	0%	100%	0%

Table 5-3: Evaluation scores of the best performing agent for each of the 9 methods.

			Performance during 50 greedy evaluation episodes			
Algorithm	ϵ	α	**Average reward**	**% crashed**	**% saved**	**% timed out**
Q-learning(0.0)	(0.01, 0.0, 2500)	0.1	-769	62%	32%	6%
Q-learning(0.1)	0.1	0.5	-816	78%	4%	18%
Q-learning(0.5)	(0.5, 0.0, 2500)	(0.1, 2500)	-1,262	78%	18%	4%
Q-learning(0.9)	(0.1, 0.0, 2500)	0.1	-5,792	64%	4%	32%
SARSA(0.0)	0.5	0.5	-1,371	78%	22%	0%
SARSA(0.1)	0.5	0.5	-1,911	98%	0%	2%
SARSA(0.5)	0.5	(0.5, 5000)	-1,285	100%	0%	0%
SARSA(0.9)	(0.5, 0.0, 2500)	0.1	-10,000	0%	0%	100%
MC	(0.01, 0.001, 2500)	1.0	-1,124	100%	0%	0%

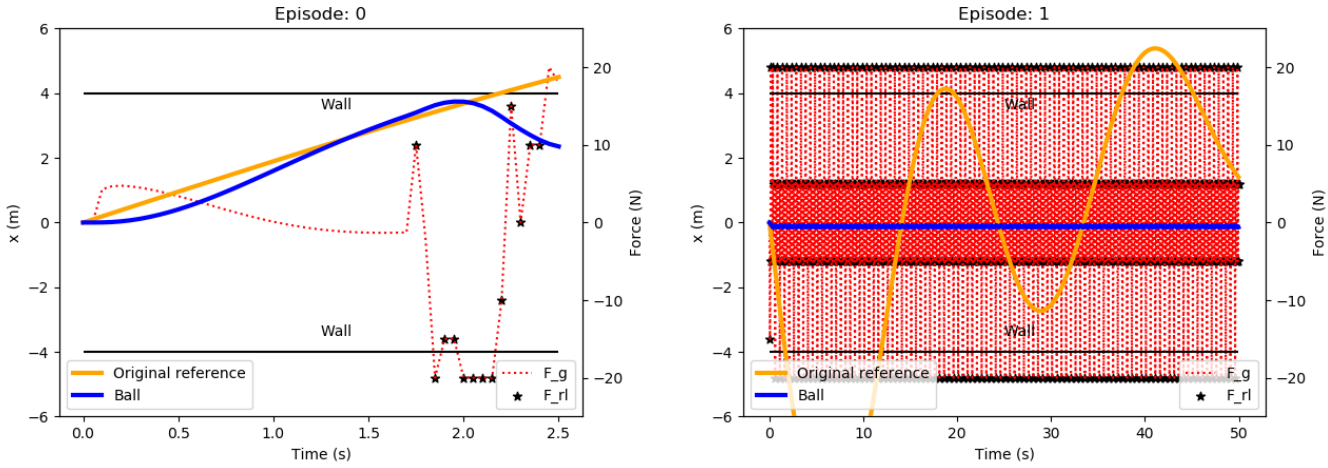
Table 5-4: Evaluation scores of the worst performing agent for each of the 9 methods.

From these tables the following interesting observations can be made:

- For each of the 9 methods there exists at least one set of hyperparameters that enable the agent to learn a policy that performs very well, showing 0% crashes over 50 episodes of evaluation. An example of a save, preventing a crash with the wall, can be seen in figure 5-13a .
- Over 50 episodes, the percentage of timed out episodes, which should theoretically be around 12.8%, differs a lot. Among the different evaluations, there is thus still a factor

of randomness involved, this is likely due to the randomly generated reference signal.

- There exists a policy that successfully prevents crashes, but at the cost of a large negative reward. This can be seen from the evaluation performance of the worst SARSA(0.9) agent. Further investigation shows that this agent does so by not allowing the ball to follow the reference signal, but instead go straight ahead, as can be seen in figure 5-13b.



(a) One episode of the best Q-learning(0.5) agent.

(b) One episode of the worst SARSA(0.9) agent.

Figure 5-13: Two episodes not resulting in a crash, one agent following a desirable policy (a), one agent following an undesirable policy (b).

Another interesting observation can be made by studying figure 5-13a. Interestingly the first action taken by the Q-learning(0.5) agent in the episode shown in figure 5-13a is a force upward, towards the wall. This seems counter-intuitive on first sight. However, further investigation shows that this is quite common in the episodes of this agent.

An explanation for this has not yet been found but could go in one of two directions. Either the policy has not (yet) converged to the optimal solution, more training episodes might help this, or there is some benefit to performing this counter-intuitive action. A possible cause for this benefit might have something to do with the termination conditions as defined for the reinforcement learning conditions. These require the ball to have been close to the wall at some point. As such, this might have unwillingly caused this incentive for the reinforcement learning agent. It is thus recommended that the termination conditions in the full research are re-investigated to ensure these do not provide undesired incentives to the RL agents.

From these observations and results the following sub conclusions can be drawn:

SC4 For all 9 tested methods there is some set of hyperparameters that leads to a policy in which the ball is successfully saved from crashing into the wall, in most of the episodes.

SC5 To accurately compare the performance of the best agents, an evaluation with 50 episodes and randomly generated reference signals is not adequate.

SC6 The current setup of the reinforcement learning problem, with small negative rewards on all actions that are not *None*, successfully counteracts policies that prevent crashes by not letting the ball follow the reference signal at all.

5-4-4 Extended greedy evaluation

As mentioned in the sub conclusions of section 5-4-3, to adequately compare the trained agents, the 50 episodes with randomly generated reference signals are not enough. Furthermore, since the evaluation results, as shown in table 5-3 are quite comparable, it's difficult to determine for certain which one is actually the est.

To tackle both issues, and enable an accurate comparison of the final trained methods, a second greedy evaluation has been performed. In these second evaluation phase, for each of the 9 methods, the 5 best performing agents (from the initial evaluation phase) were selected. These 45 agents were then evaluated during 500 greedy evaluation episodes. However, contrary to the initial evaluation, all agents were evaluated on the same 500 reference signals. Therefore, the evaluation conditions are now truly equal for all agents.

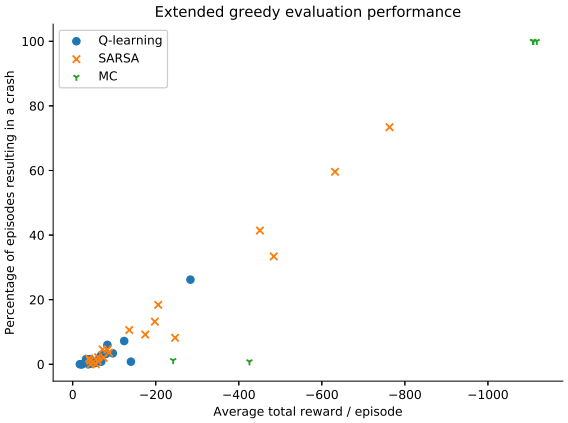


Figure 5-14: Key performance indicators during the extended greedy evaluation of the top 5 agents for each of the 9 methods.

The results of this extend evaluation evaluation can be seen in figure 5-14. Furthermore, the exact performance of the best agent for each of the 9 methods, including the underlying set of hyperparameters, is given in table 5-5. The data of this extended greedy evaluation confirms the conclusion from the first evaluation that for each of the 9 methods some set of hyperparameters exists that lead to the desired performance.

Furthermore, the following interesting observations can be made by studying the graph above, the table below and by comparison with the results of the initial greedy evaluation:

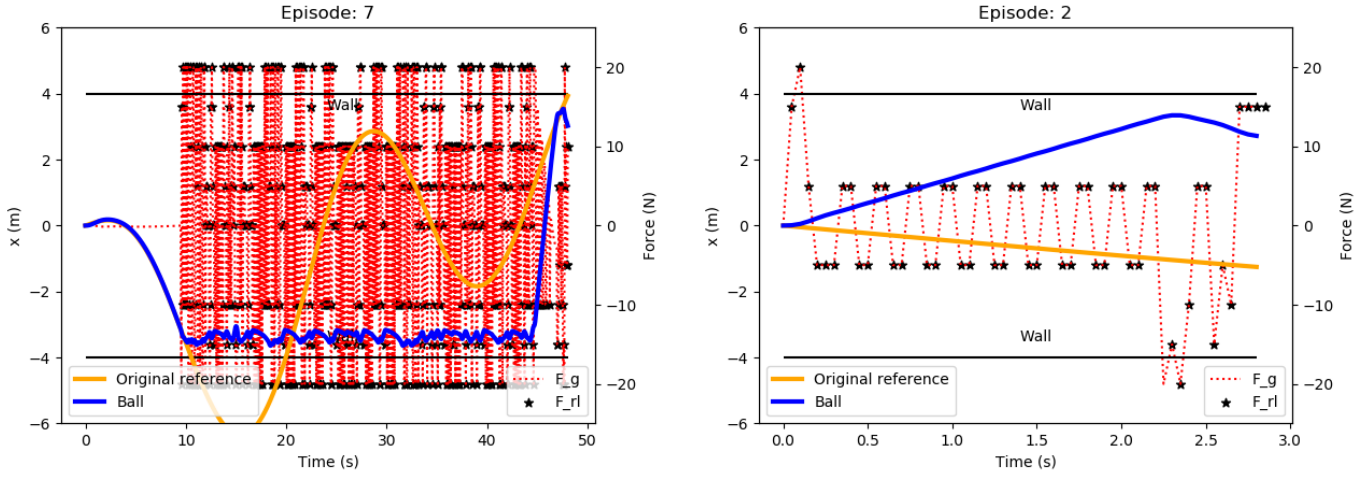
- Now that all agents have been evaluated on the same set of reference signals, the difference in the percentage of episodes ending in a timeout is significantly less. Between 7.2% and 7.6% of episodes result in a timeout. This is notably lower than the 12.8% predicted as the baseline. This difference can be explained by considering that while 12.8% of the reference signals do not intersect the wall, some of these signals might come close enough to one of the walls to experience the force of the wall, thereby triggering a save or collision.

- From the evaluation episodes with the best Monte Carlo agent, 98.8% of the episodes result in a save, only 1.2% in a crash and not in a timeout. This goes however together with the worst performance in terms of average reward. Investigation of some of the episodes show that this agent relies on a policy where the reference signal is not followed at all, instead, the ball is immediately forced towards the upper wall, after which the ball is 'saved' from a collision. This can be seen in figure 5-15b and is clearly undesired behavior.

- The set of hyperparameters that gave the best performances in the initial greedy evaluation, is not for every method the set that gives the best performance in this extended evaluation.
- All top Q-learning agents have a high initial exploration rate.
- All top SARSA agents have a low, decreasing, exploration rate.
- The best Monte Carlo agent performs worse than the top Q-learning and SARSA agents.
- The agent with arguably the best performance in this evaluation is the Q-learning(0.5) agent, with a learning rate of 0.1, and an exploration rate that starts at 0.5, but decreases to 0.05 in the first 2500 episodes.

			Performance during 500 greedy evaluation episodes			
Algorithm	ϵ	α	**Average reward**	**% crashed**	**% saved**	**% timed out**
Q-learning(0.0)	0.5	0.1	-32	1.6%	91.2%	7.2%
Q-learning(0.1)	0.5	(0.5, 2500)	-38	0%	92.6%	7.4%
Q-learning(0.5)	(0.5, 0.05, 2500)	0.1	-17	0%	92.8%	7.2%
Q-learning(0.9)	0.5	(0.3, 2500)	-46	0.4%	92.0%	7.6%
SARSA(0.0)	(0.01, 0.0, 2500)	(0.5, 5000)	-72	4.6%	88.0%	7.4%
SARSA(0.1)	(0.01, 0.001, 2500)	(0.3, 5000)	-39	1.8%	91.0%	7.2%
SARSA(0.5)	(0.01, 0.0, 2500)	(0.3, 5000)	-42	1.2%	91.2%	7.6%
SARSA(0.9)	(0.01, 0.0, 2500)	(0.1, 2500)	-43	0%	92.4%	7.6%
MC	(0.01, 0.0, 2500)	1	-242	1.2%	98.8%	0%

Table 5-5: Evaluation scores for the best performing agents for each of the 9 methods, in 500 extended evaluation episodes.



(a) Agent preventing a collision, but not meeting the requirements for a save.

(b) Best Monte Carlo agent forcing the ball to the upper wall, to perform a save.

Figure 5-15: Policies that result in a higher or lower percentage of timeouts than expected.

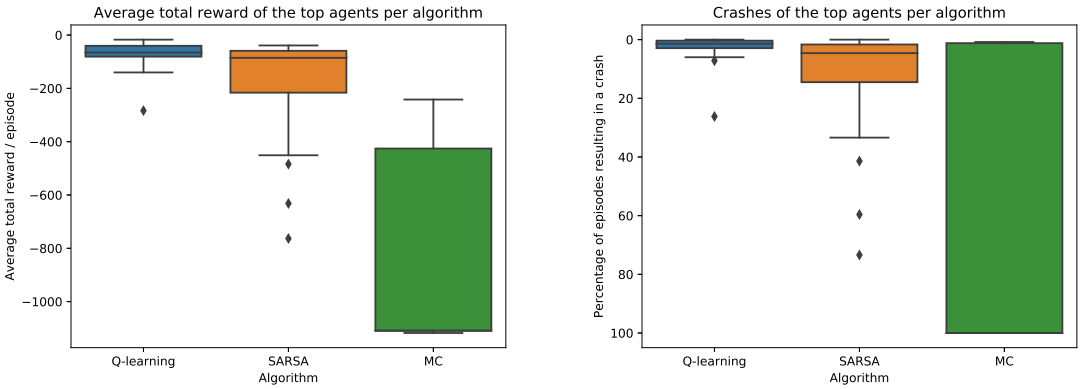


Figure 5-16: Comparison of performance of Q-learning, SARSA and MC agents.

5-4-5 Influence of the RL algorithm

A comparison of the distribution of average total reward and percentage of crashes among Q-learning, SARSA and Monte Carlo agents, is shown in figure 5-16. By combining the information from this graph with the observations from the extended greedy evaluation, the following conclusions can be drawn with respect to the RL-algorithm:

SC7 Monte Carlo methods seem least suited for this reinforcement problem, based on the low performance of the Monte Carlo agents (see figure 5-16) and undesirable policies (see figure 5-15b).

SC8 Q-learning is best suited to train agents in this reinforcement learning problem, because for this specific problem:

(a) Agents trained using a temporal difference method achieve on average a higher performance than those trained using the Monte Carlo method.

(b) The top agents trained using Q-learning are able to achieve a higher performance than the top agents trained using SARSA.

(c) Q-learning is more robust with respect to the hyperparameters.

This might be surprising when looking back to the training performance, where it looked like SARSA was performing better. This is, however, as discussed, caused by Q-learning being an off-policy method where the top agents use a large exploration rate during training. This clouds the measured performance in training, so the evaluation performance is significantly better than the training performance. This can also be seen in figure 5-17, where the average reward during training is compared to that during the fully greedy evaluation for all 657 agents.

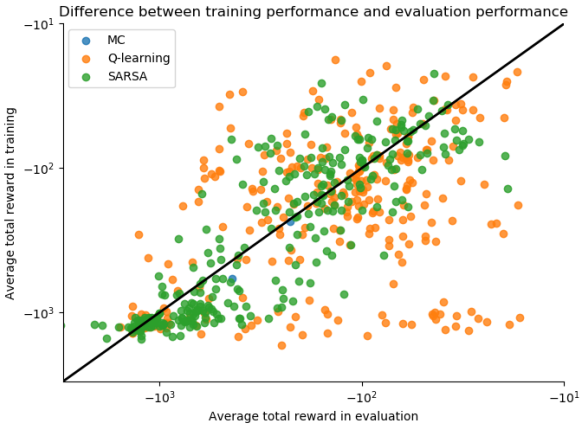


Figure 5-17: Difference between performance in training and evaluation.

5-4-6 Influence of hyperparameters

As the goal of this preliminary research is, among others, to gather an understanding of the influence of hyperparameters on training results, this influence needs to be investigated. The key challenge here is that the influences of the hyperparameters are not independent of each other. As such, the direct influence of a hyperparameter on the final performance is in many cases not obvious. However, when looking at the performance for different values of ϵ, α and λ still some observations can be made and conclusions can be drawn.

Exploration rate (ϵ)

First of all, the average total reward for different values of the exploration rate is shown in figure 5-18. For this overview all 648 temporal difference agents where used. After studying this figure, the following sub conclusions are reached, for this specific reinforcement learning problem:

SC9 Q-learning agents, in general, benefit from a high, non-decreasing exploration rate.

SC10 For most SARSA agents, a low, decreasing exploration rate is beneficial to the performance.

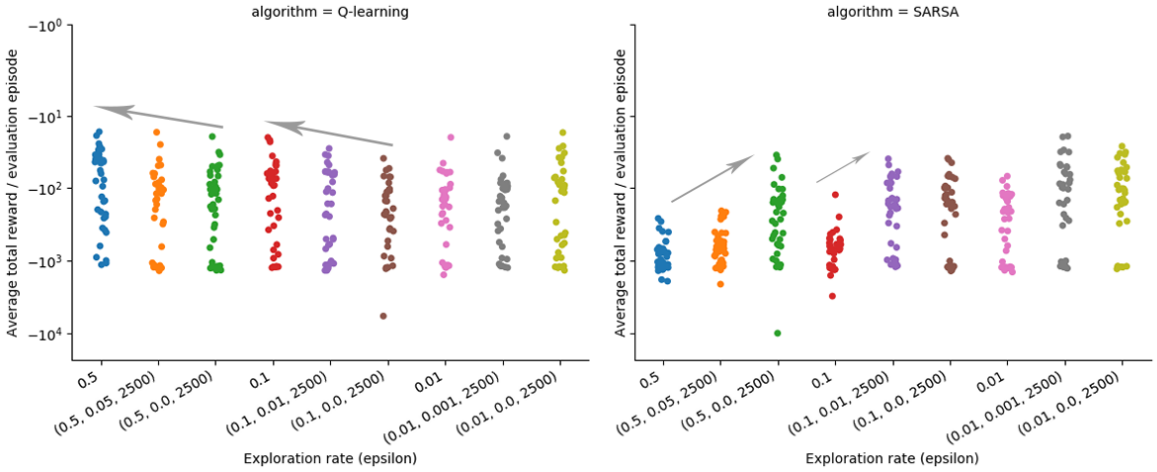


Figure 5-18: Average total reward in evaluation of the 648 temporal difference agents, for different values of the exploration rate (ϵ).

Not only the temporal difference methods make use of an exploration rate. For the Monte Carlo methods, the same 9 options for the exploration rate were used, resulting in 9 unique agents. Their results can be seen in figure 5-19. It can, however, be argued that these are not enough data points to assess the exact influence of the exploration rate on performance. Therefore it is recommended that if Monte Carlo methods are chosen for the full research, more agents need to be trained and evaluated to assess this influence.

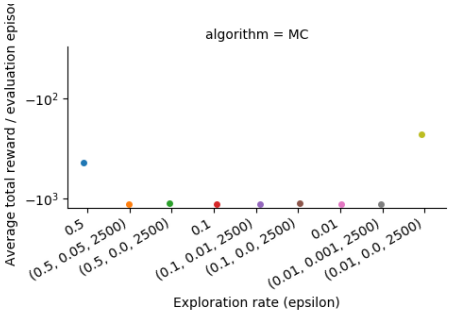


Figure 5-19: Average total reward in evaluation of the 9 Monte Carlo methods, for different values of the exploration rate (ϵ).

Learning rate (α)

The average total reward in evaluation for different values of the learning rate (α) can be seen in figure 5-20. The following sub conclusions are posed regarding the influence of the learning rate, in this specific reinforcement learning problem:

SC11 Q-learning agents, in general, benefit from a decreasing learning rate, unless the learning rate is already small (≤ 0.1), then the effect is less.

SC12 For Q-learning agents, a learning rate that decreases to a quarter of its initial value after the first half of the episodes seems most promising.

SC13 For SARSA agents, a learning rate that decreases to half its initial value after the first half of the episodes looks most promising.

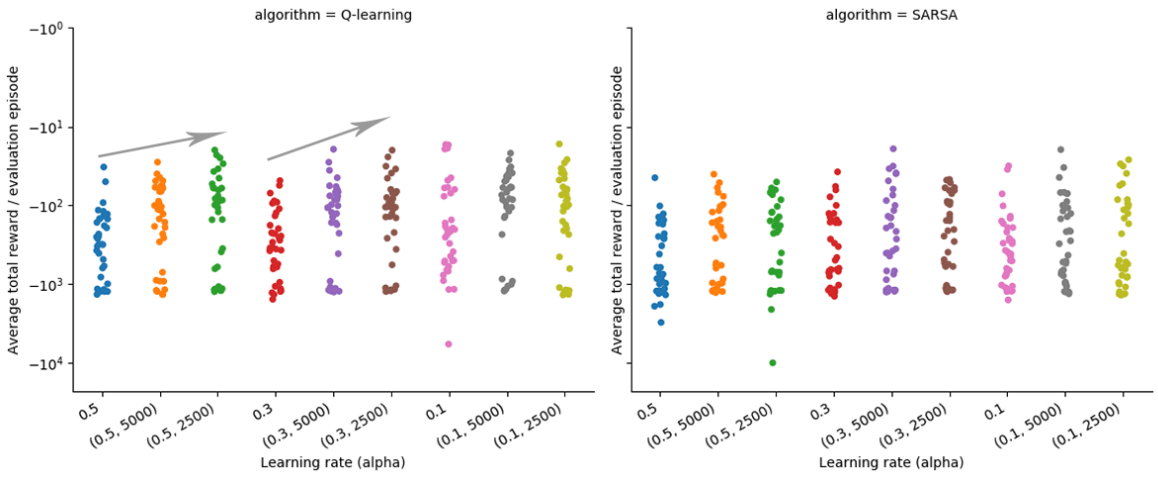


Figure 5-20: Average total reward in evaluation of the 648 temporal difference agents, for different values of the learning rate (α).

Eligibility traces(λ)

The average total reward in evaluation with different eligibility traces (λ) can be seen in figure 5-21. Based on this graph the following sub conclusions are formulated, with respect to this specific reinforcement learning problem:

SC14 A λ between 0.1 and 0.5 seems most beneficial for this problem when using either Q-learning or SARSA.

SC15 Some agents without eligibility traces ($\lambda = 0$) are able to achieve the same level of performance as the top agents with eligibility traces. The bulk, however, has a slightly lower performance.

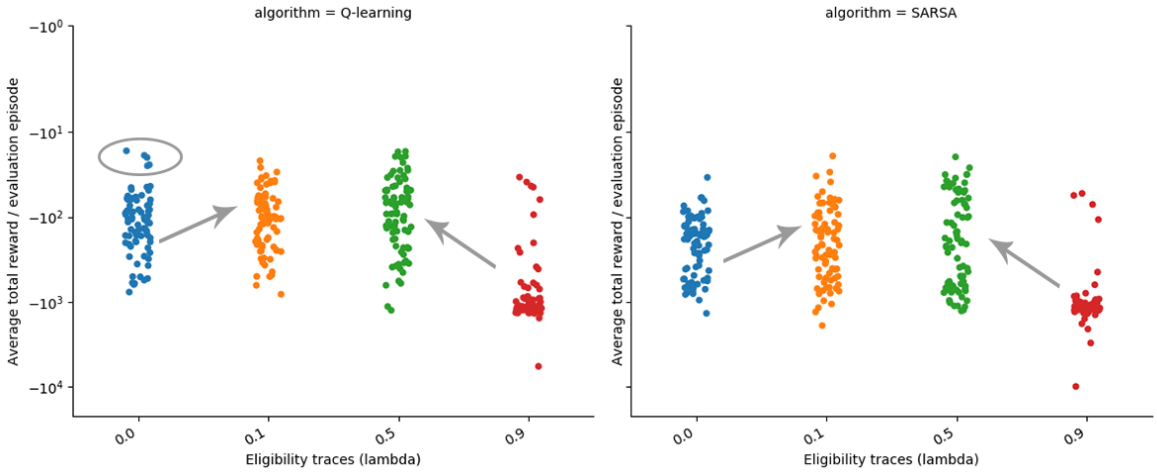


Figure 5-21: Average total reward in evaluation of the 648 temporal difference agents, for different eligibility traces (λ).

It must be noted that all the sub conclusions following from this investigation are for this specific reinforcement learning problem, not necessarily for RL-problems in general. However, these working observations and implied influences can be used as a good starting point for picking the sets of hyperparameters to be used in the full research.

5-4-7 Investigation of best performing agent

Previously, the performance of all agents has been discussed in terms of average total reward and the percentage of episodes resulting in a crash. It is, however, useful to assess the best performing agents also within terms of the larger goal; obstacle avoidance. To do so, three things will be investigated; independence with respect to the wall position, independence with respect to the reference signal and the final agent policy.

The agent that will be evaluated is the best Q-learning(0.5) agent. This agent had 0 crashes and the lowest average total reward (-17) in the extended greedy evaluation. It had been trained using an initial exploration rate of 0.5, which decreased to 0.05 in the first 2500 episodes, after which it was constant. The learning rate was set to be constant at 0.1. From here on this agent will be referred to as the *top agent*.

Independence with respect to the wall position

To test the independence of the top agent with respect to the wall position, greedy evaluations were conducted at different wall positions. The wall position was varied in two different ways, symmetrical and asymmetrical. In the symmetrical variation, the two walls were placed at equal distance from $x = 0$, the starting point of the ball, the distance was however varied. While in the training the walls were always placed at $x = -4$ and $x = 4$, the walls where now placed at $x = (0.5, 0.675, ..., 8)$ and $x = (-0.5, -0.675, ..., -8)$, for a total of 41 unique variations. In the asymmetrical variation, the lower wall stayed at $x = -1$, however the upper wall was placed at $x = (0.5, 0.675, ..., 8)$. Both variations can be seen in figure 5-22a

This created a total of 82 unique environment setups. The agent was evaluated within each environment during 500 fully greedy evaluation episodes each. This leads to the results shown in figure 5-22b.

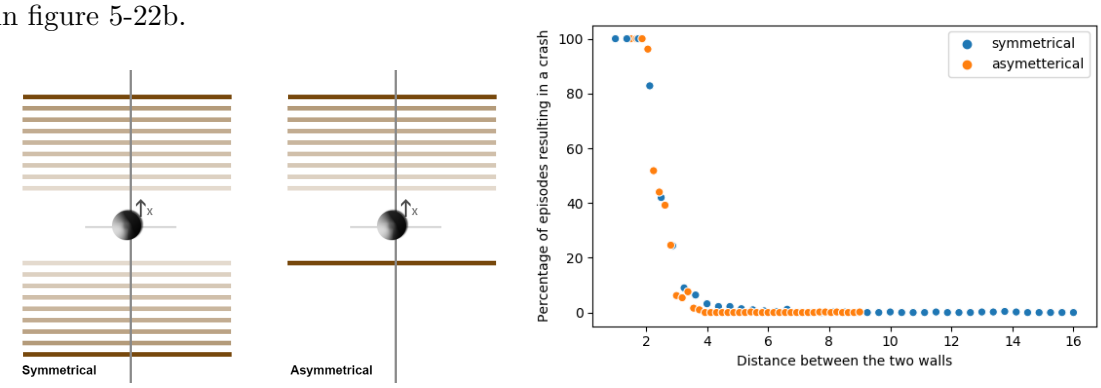


Figure 5-22: Performance of the top agent at different wall positions during 500 fully greedy evaluation episodes.

From these results, it can clearly be seen that the performance of the agent stays at roughly the same level, as long as the distance between the two walls does not become too small. Furthermore, it can be seen that the performance does not differ significantly among the symmetrical and asymmetrical variant.

If one considers that the wall effect distance is currently set at one meter, it is obvious that an environment with the walls placed $\leq 2m$ away from each other is significantly different from the training environment. The ball would always be within the influence zone of at least one wall. Similarly when the distance between the two walls is between 2 and 3 meters, the area where the ball is not within the influence of a wall is quite small, less than 1 meter. This likely explains the drop in performance seen at these distances.

The following subconclusion can thus be drawn:

SC16 The performance of the top agent is independent of the wall position, as long as the distance between the walls is larger than 4 meters.

Independence with respect to the reference signal

Similar to the approach above, simulations were performed to test independence with respect to the reference signal. However, because the reference signal is already randomly generated each episode, the parameters used to generate this reference signal were changed instead. Overall, the parameters where changed such that the range of potential reference signals was increased. This was done by:

- Varying the number of sine waves N_{sin}, from 1 sine wave up till 20 combined sine waves.
- Doubling the bandwidth of the randomly generated amplitude a_i, from $a_i \in [-4, 4]$ to $a_i \in [-8, 8]$.
- Increasing the bandwidth of the randomly generated period b_i, from $a_i \in [0.01, 0.05]$ to $a_i \in [0.005, 0.1]$.

- Allowing the deviation c_i to be randomly generated $c_i \in [-0.5, 0.5]$. As a result the reference signal is now not necessarily 0 at $t = 0$, the ball thus does not start at the reference signal, but already with some tracking error.

Once again 500 greedy evaluation episodes were performed, once with the extended parameter bandwidth and once with the normal bandwidth, and the number of sine waves varying from 1 to 20. The results of this evaluation can be seen in figure 5-23.

From this figure, two interesting observations can be made. Firstly, increasing the number of sine waves, in other words, making the reference signal more complex, decreases the performance. This can be seen both when the parameters (a_i, b_i, c_i) are generated from within the original and the extended bandwidths. Secondly, the extension of the parameter bandwidth leads to a further decrease in performance.

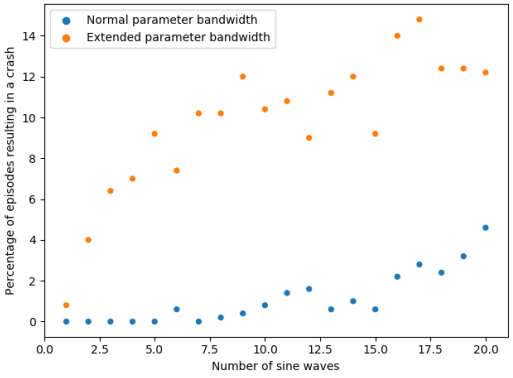


Figure 5-23: Performance of the top agent when varying the parameters of the ranomly generated reference signal.

By studying the episodes in which a crash occurs it can be seen that such a crash usually happens when the reference signal, and thus the ball, approaches the wall at high speeds. An example of this can be seen in figure 5-24a.

It can thus be concluded that:

SC17 An increase in either the number of sine waves or the extension of the bandwidth of the parameters, can lead to the ball approaching the wall at higher speeds, which results in a decrease in performance.

This means that the agent is not truly independent of the reference signal. However, from a practical perspective, it can be argued that preventing a crash in 80% of the episodes, with reference signals that cross the wall at higher speeds then the agent was trained on, is still a reasonable performance.

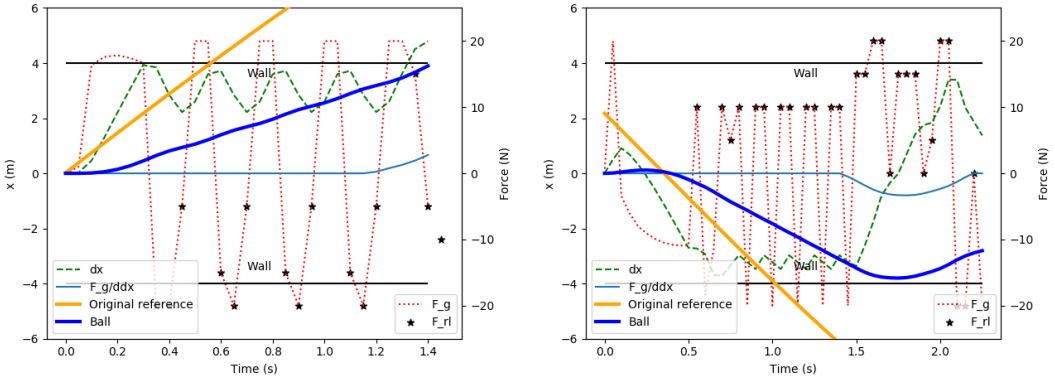


Figure 5-24: Two examples of episodes with a more complex reference signal that is faster and does not necessarily start at $x = 0$.

Final agent policy

As the final part of the investigation of the top agent, the policy of this agent will be checked. This can be used as a sort of reality check, to make sure that the agent indeed uses its internal policy representation and that actions taken during an episode correspond with the policy.

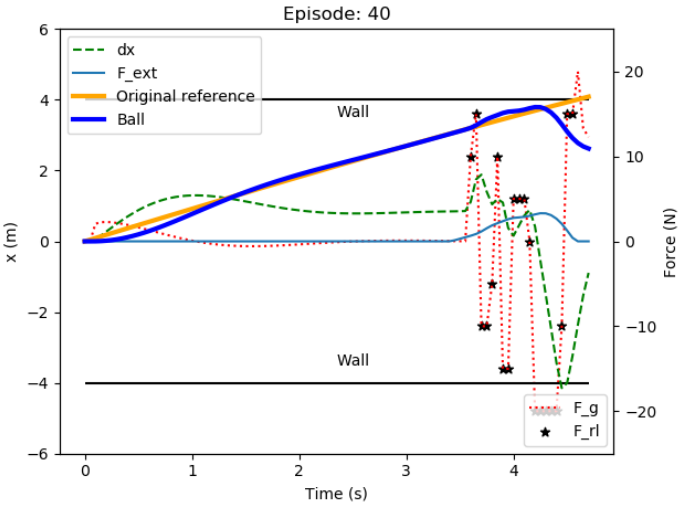
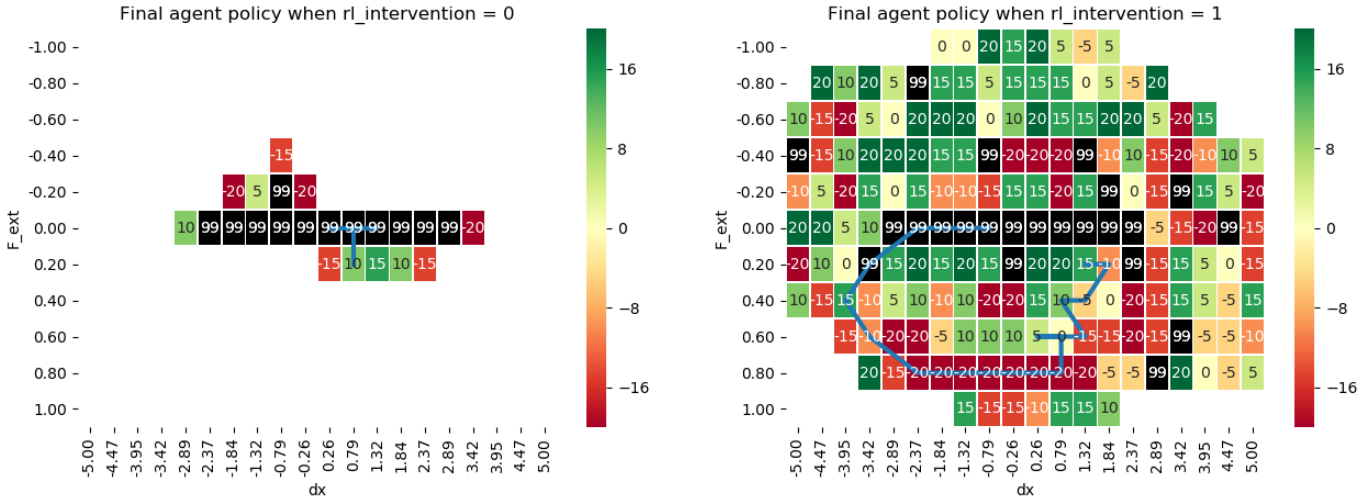


Figure 5-25: Visualization of an episode in which the top agent successfully saves the ball from colliding with the wall.

The final policy of the top agent is shown in figure 5-26. This 3-dimensional matrix is split up into two 2-dimensional visualizations, figures 5-26a and 5-26b. These respectively describe the policy when the agent has not yet intervened (executed an action which was not *None*) and the policy when it has. The *None* action is indicated by the value '99'. All other numbers indicate the force in Newton, where a force upwards is defined as positive. Only the policy for states that are visited in one or more percent of the episodes is shown.



(a) Policy when the agent has not yet executed its first action.

(b) Policy after the agent has intervened at least once.

Figure 5-26: The state-space, policy and transitions during an episode in which the top agent successfully saves the ball from colliding with the wall.

Furthermore, the transition within the state-space during the episode shown in figure 5-25, is indicated by the blue line. By following this line and comparing it with the actions shown in figure 5-25, it can be confirmed that the agent indeed exactly follows its learned policy.

5-5 Conclusions

During the preliminary investigation, a 1-dimensional simplified version of the problem at hand was investigated. A setup was chosen such that it replicated some of the challenges expected to be faced in the full research, including the novel placement of the reinforcement controller within the control loop. This setup was then turned into a reinforcement learning problem, after which a total of 657 reinforcement learning agents were trained during a grid search. Within this grid search not only different reinforcement learning algorithms were considered (Monte Carlo, on-policy SARSA and off-policy Q-learning), but also different values for the exploration rate (ϵ), learning rate (α) and eligibility traces (λ).

After the learning phase, each of these agents was then evaluated during a fully greedy evaluation phase of 50 episodes. The most promising agents were then evaluated further on an equal set of 500 evaluation episodes. Based on these results, a qualitative analysis on the influence of both the reinforcement learning algorithm and the influence of hyperparameters could then be conducted. Furthermore, the best agent, an agent trained with Q-learning(0.5), was then evaluated in depth.

The goal of the preliminary research was threefold: learning how to do such an experiment, investigating a novel RL-control scheme and gathering an understanding of hyperparameters and there influence on the training & results. The sub conclusions 1 through 15 from the results discussion, as well as the other key observations and conclusions, can be summarized around these same three goals.

Learning how to do such an experiment

1. For Q-learning agents, the average total reward during training is not a good predictor of performance during evaluation. This is most likely due to the higher exploration rate used by the top Q-learning agents. (SC3)

2. To accurately compare the performance of the best agents, an evaluation with 50 episodes and randomly generated reference signals is not adequate. (SC5)

Investigate novel RL-control scheme

3. This novel RL-control scheme can work, as is demonstrated by the performance of the trained agents (SC2, SC4). Especially the top Q-learning(0.5) agent which, in every of the 500 evaluation episodes, let's the ball follow the reference signal uninterrupted, until a wall is detected, then the ball is successfully saved from hitting the wall.

4. The current setup of the reinforcement learning problem, with small negative rewards on all actions that are not *None*, successfully counteracts policies that prevent crashes by not letting the ball follow the reference signal at all. (SC6)

5. However, to ensure that this novel RL-control scheme indeed works as desired it is important that oscillations between the conventional controller and the reinforcement agent are not encouraged (SC1) and during training, the agent is exposed to the full set of situations it will have to operate in. (SC16, SC17)

Gather an understanding of hyperparameters and there influence on the training & result.

6. Monte Carlo methods are least suited for this reinforcement problem, based on the low performance of the Monte Carlo agents and undesirable policies. (SC7)

7. When using SARSA, a low, decreasing, exploration rate (SC10), a λ between 0.1 and 0.5 (SC14, SC15), and a learning rate that decreases to half of its initial value during the first half of the episodes, is expected to produce the best performing agents. (SC13)

8. When using Q-learning, a high, non-decreasing, exploration rate (SC9), a λ between 0.1 and 0.5 (SC14, SC15), and a learning rate that decreases to a quarter of its initial value during the first half of the episodes, are expected to produce the best performing agents. (SC11, SC12)

9. Q-learning is best suited to train agents in this reinforcement learning problem, because for this specific problem these agents achieve a higher performance (SC8a, SC8b) and are more robust with respect to the hyperparameters. (SC8c)

These combined conclusions do not only show that the three goals of the preliminary investigation have been fulfilled but also partly answer subquestion 2.1, posed in the problem definition: *How can the reinforcement learning-based control system best be setup?*.

Chapter 6

Additional Results

In this chapter the additional results of the thesis research will be presented. First, the filtering of the onboard IMU measurements on the Parrot Bebop 1 quadrotor will be discussed in section 6-1. Then, the system identification experiments and their results will be presented in section 6-2. Thirdly, the determination of the hyperparameters will be discussed in section 6-3. The top agent that was trained with these hyperparameters, and its training, will then be investigated in section 6-4. Then, the measurements of the wall effect, used to assess the potential of extending this method to the avoidance of obstacles on the same level as the quadrotor, will be presented in section 6-5. Finally, the preliminary results of a potential improvement to the reward structure of the RL setup will be discussed in section 6-6.

6-1 Filtering the onboard IMU measurements

Key to the estimation of the external forces are the acceleration measurements provided by the onboard accelerometer. However, MicroElectroMechanical Systems (MEMS) accelerometers like the one on the Parrot Bebop 1 are known to contain quite a lot of noise. This was also visible upon inspection of the raw acceleration measurements on the Bebop, as can be seen from figures 6-1 and 6-2.

What also can be seen from these figures is the fact that the accelerometer also measures the gravity factor, as discussed in section III.B of the article. This explains the mean of -9.81 for the body acceleration in vertical direction, even when the quadrotor is still standing on the ground.

Furthermore, an increase in the noise can be seen after the quadrotor takes off, around 10 seconds after the start of the measurements. An indication that the measured deviations of the mean acceleration are not just sensor noise.

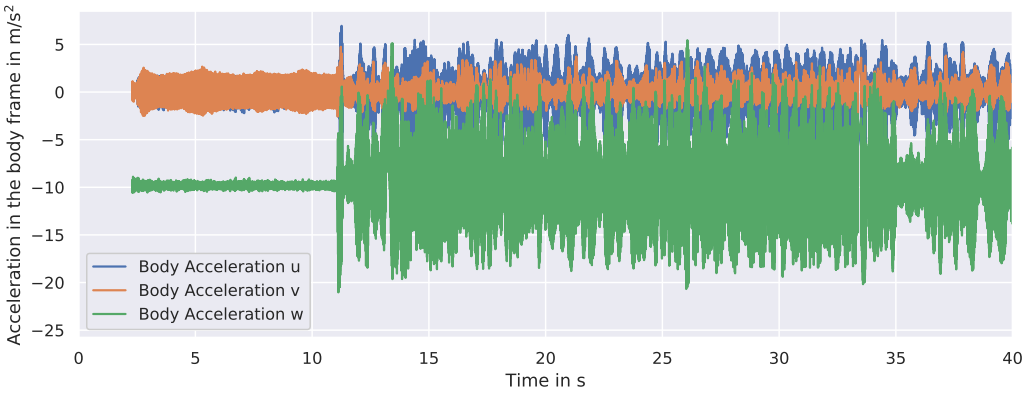


Figure 6-1: Accelerations in the body frame as measured by the accelerometer during a test flight.

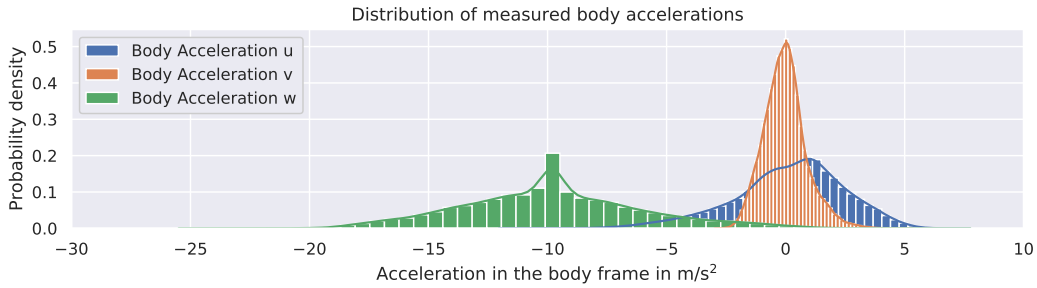


Figure 6-2: Distribution of the accelerations in the body frame, as measured by the accelerometer.

This is confirmed by a frequency spectrum analysis of the body accelerations, as shown in figure 6-3. This analysis shows a lot of noise on the speeds at which the rotors spin during flight: 6000 - 9000 RPM, so 100 - 150Hz. While these accelerations are referred to as noise, it can actually be expected that these measurements are true measurements of vibrations, caused by the rotors. Nevertheless, for the purposes of this research, they are problematic. As such, a 4th order Butterworth filter was designed to remove these vibrations from the measurements. This low-pass filter has a cutoff frequency of 3Hz and the gain it applies at the different frequencies is depicted in the figure below as well.

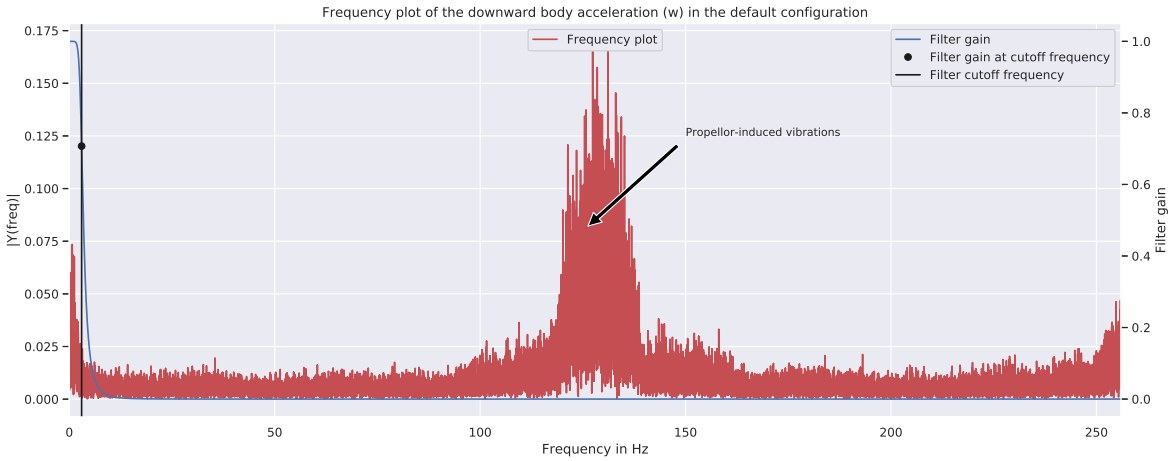


Figure 6-3: Frequency spectrum analysis of the body accelerations.

The result of applying this filter on the measured acceleration in the vertical body axis can be seen in figure 6-4 and 6-5. The filter removes most of the vibrations and noise from the measurements. When relating the shown accelerations to the displayed vertical position of the quadrotor it becomes clear that the remaining accelerations correspond to the changes in vertical speed.

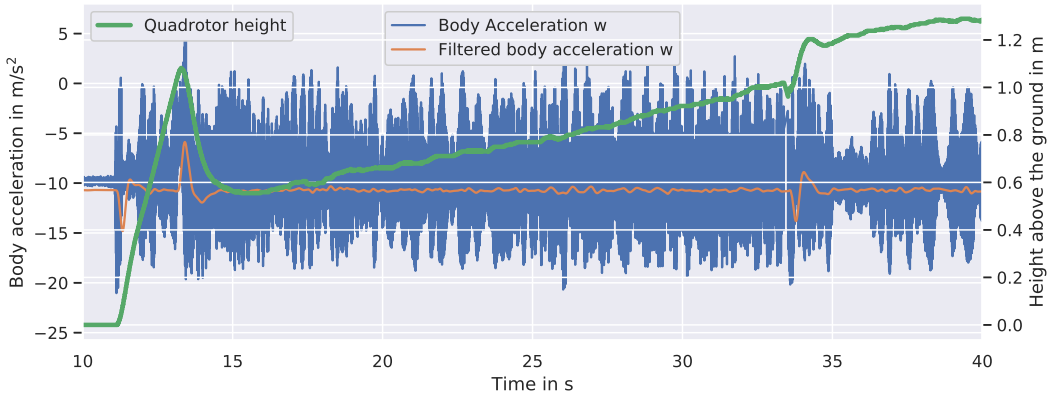


Figure 6-4: Filtered and unfiltered accelerations in the body z-axis.

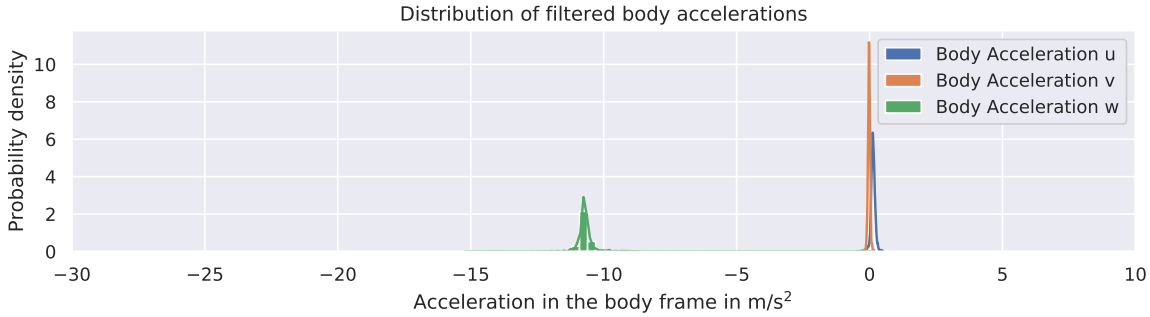


Figure 6-5: Distribution of the filtered accelerations in the body frame.

Unfortunately, as with any low-pass filter, this filter will introduce some delay into the filtered signal. For a 4th order Butterworth filter, the introduced delay is approximately $0.416/f_c$, with f_c the cutoff frequency. [34] With the chosen cutoff frequency of 3Hz this thus results in an expected delay of 140 milliseconds.

Because the accelerations now contain a 140-millisecond delay, the same filter is also applied on the motor speeds and angular rates to keep them synchronized in time. While their effect on the motor speeds mainly introduces a delay, the angular rates also contain a lot of noise, so they benefit from the actual filtering as well. This can be seen in figure 6-6.

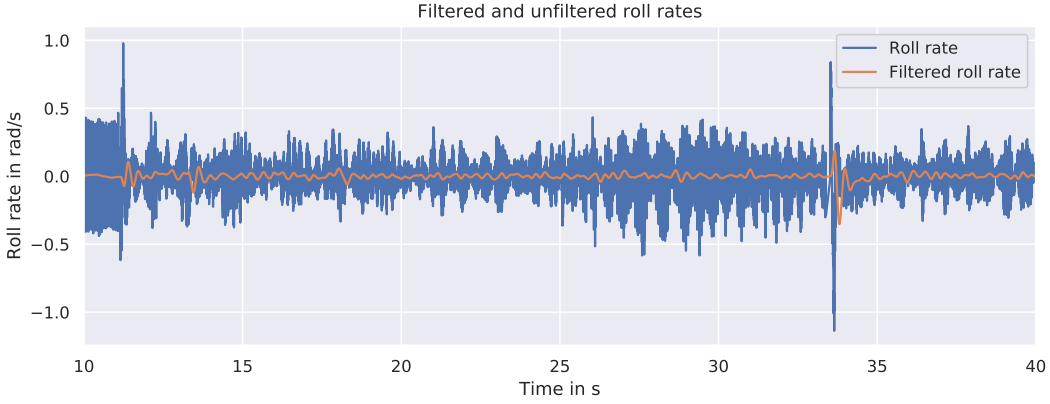


Figure 6-6: Filtered and unfiltered body roll rates.

6-2 System identification of the Parrot Bebop 1

In order to create a simulation environment that resembles the actual flight environment as much as possible, two system identification experiments were performed. The first experiment served to determine the actuator dynamics, inner loop gains of the quadrotor and drag coefficient in the body z-axis. In the second experiment the other drag coefficients, the moments of inertia about the x and y-axis, and the torque gains of the quadrotor were determined.

In the first system identification experiment, the quadrotor descended towards the ground with 0.2, 0.3 and 0.4 m/s. For each commanded descend speed, two flights were carried out, bringing the total to 6 measurement flights. Each measurement flight took about 30 seconds.

In the second system identification experiment the quadrotor was first commanded to consecutively fly forward, backward, to the right and to the left, with speeds ranging up to 1.5m/s. After this, consecutive attitude commands were given. First on the roll angle: $\psi = \{5°, -5°, 10°, -10°, -15°, 15°\}$ and then on the pitch angle $\theta = \{5°, -5°, 10°, -10°, -15°, 15°\}$. Finally, the quadrotor was commanded to perform rotations around its body z-axis, by instructing the following heading change commands: $\psi = \{0°, 90°, 270°, 90°, 0°\}$.

6-2-1 Actuator dynamics

The actuator dynamics were approximated based on the measurements from the first system identification experiment. A second order Butterworth low-pass filter with a cut-off frequency of 15Hz was used to mimic the relationship between commanded and actual motor speed. In the figure below, the commanded motor speed, as well as the actual and the simulated motor speeds are shown. What is important to note here is that the actual and commanded motor speeds are logged onboard the quadrotor and thus have been discretized.

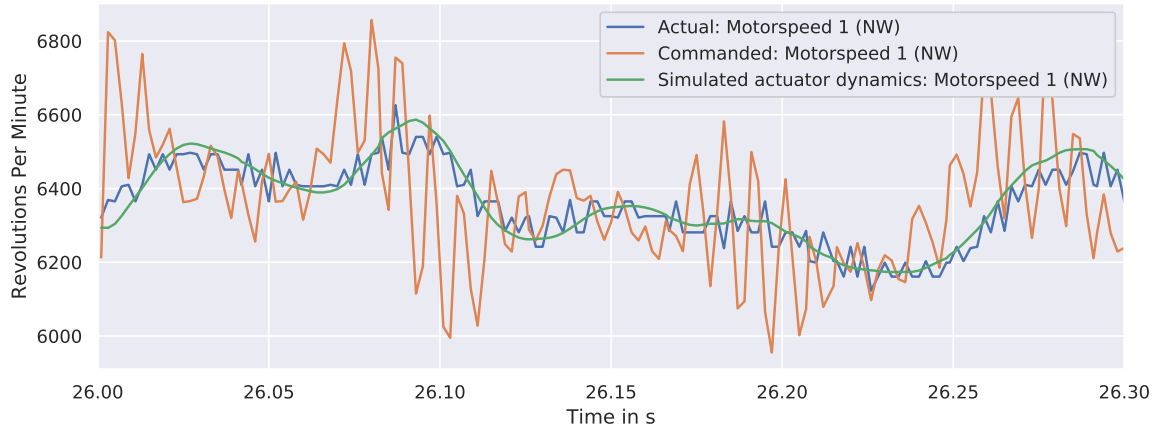


Figure 6-7: Actual, commanded and simulated motor speeds.

6-2-2 Inner vertical loop

The Paparazzi inner loop for vertical control, as shown below in figures 6-8 and 6-9 was recreated in Python. The initial gains were based on the gains that were found in the source code of this Paparazzi inner vertical control loop for rotorcraft.

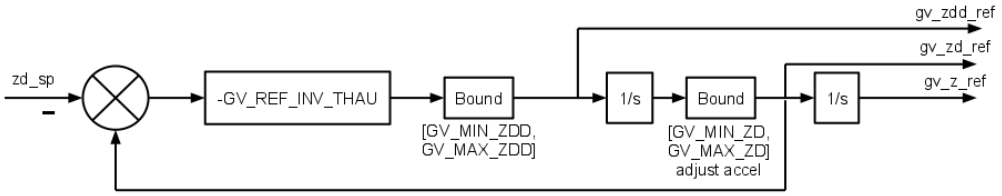


Figure 6-8: Paparazzi reference generator of the inner vertical control loop [42].

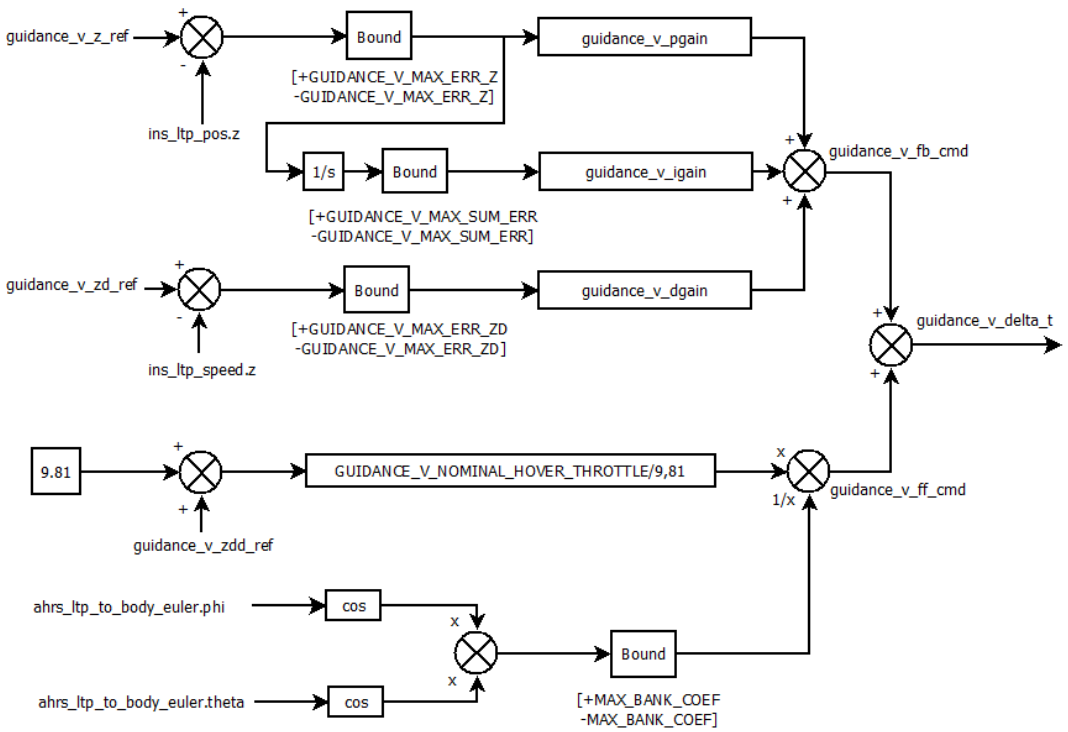


Figure 6-9: Paparazzi inner loop for vertical control [42].

The same system identification experiment described in the introduction was then carried out in simulating, commanding the quadrotor to descend with 0.2, 0.3 and 0.4m/s. The results were compared with the results from the real flights and used to adjust the gains. This was done until a similar response was achieved for all three commanded speeds.

The result of this can be seen in figures 6-10 and 6-11. These figures compare the simulated response to a 0.3 m/s descend command with that of an actual flight. As can be seen from these figures, the simulation response resembles the actual flight quite well.

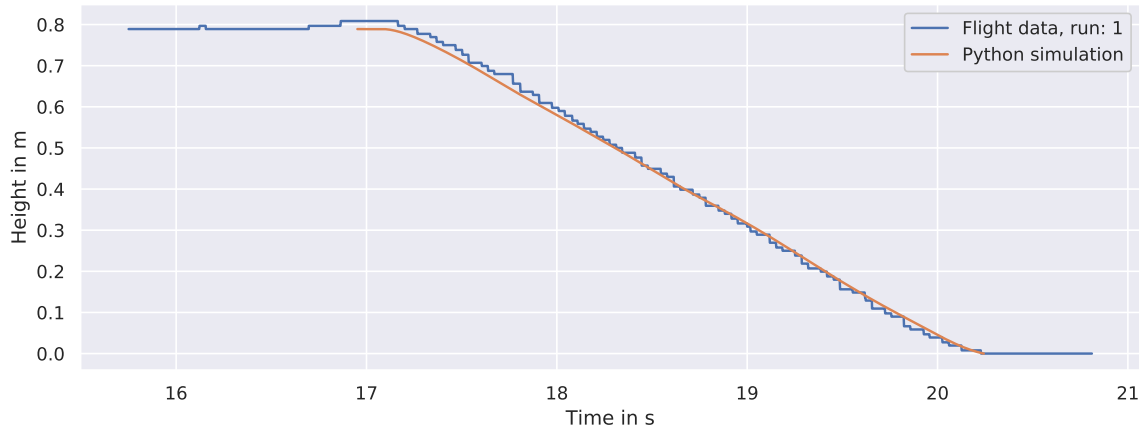


Figure 6-10: Simulated and measured height after receiving a 0.3m/s descend command.

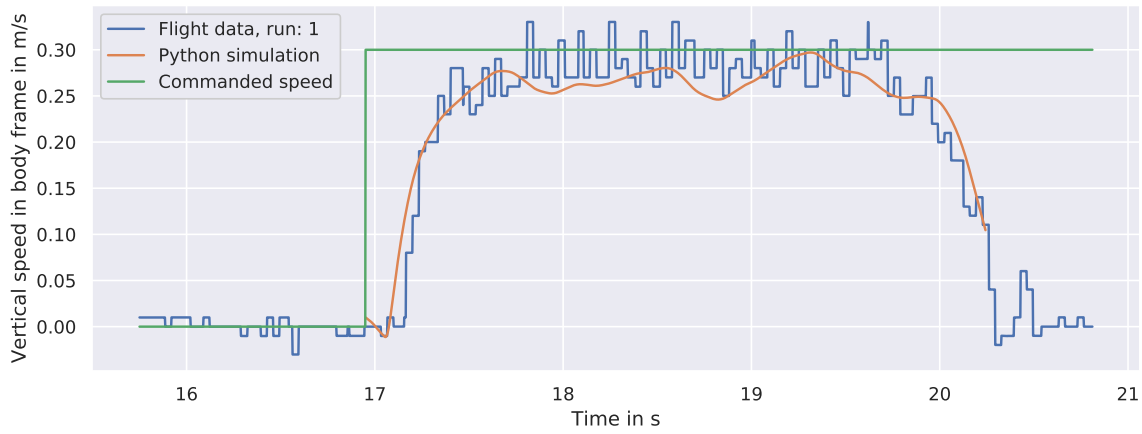


Figure 6-11: Simulated and measured speed after receiving a 0.3m/s descend command.

6-2-3 Estimation of the drag coefficients

As discussed in section III of the article, the drag forces at different body speeds can be estimated by performing experiments in free flight, outside of the presence of obstacles. All external forces are then assumed to be zero. If the drag in a certain direction is then assumed to be some drag coefficient times the speed squared, these drag coefficients can be estimated.

This was done for the Parrot Bebop 1 drone, using the data from both system identification experiments. The estimation of the drag coefficient in vertical direction $k_{D,z}$, will be described below as an example.

To estimate $k_{D,z}$, first the equation for estimating the external torque, equation 6-1, is rewritten to equation 6-2.

$$\frac{F_{ext,z}}{m} = \underbrace{\hat{\dot{w}} + pv - qu}_{\text{from IMU}} + \sum_i \frac{k_i}{m} \underbrace{\omega_i^2}_{\text{from motors}} - \frac{1}{m} F_{drag,z} \tag{6-1}$$

$$\frac{F_{drag,z}}{m} = \pm\hat{\dot{w}} + pv - qu + \sum_i \frac{k_i}{m}\omega_i^2 \tag{6-2}$$

The drag force is estimated for each measurement point in the previously described flight experiment. The drag is then assumed to be of the form: $|F_{drag,z}/m| = k_{D,z}w^2$, and an equation of this type is fitted to the measurement data using linear least squares regression. The result can be seen in figure 6-12. This resulted in an estimated drag coefficient of $k_{D,z} = 0.271$.

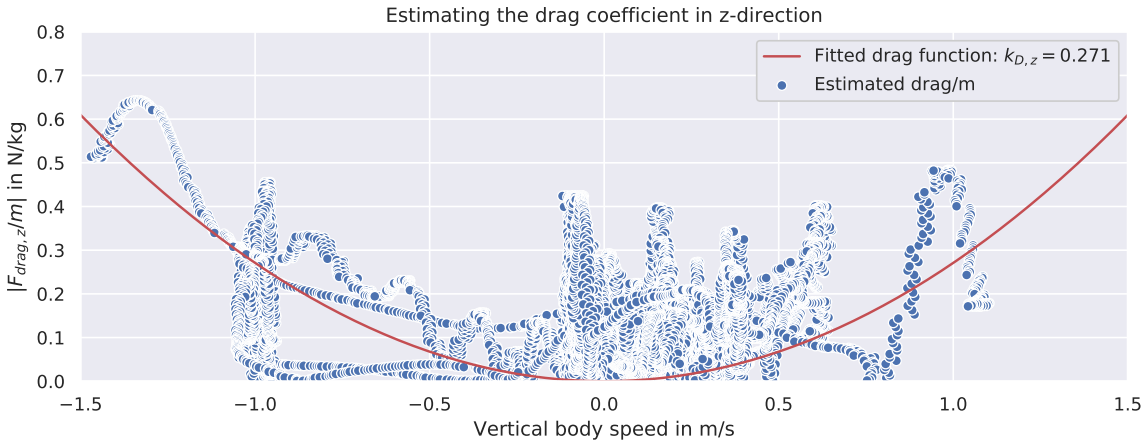
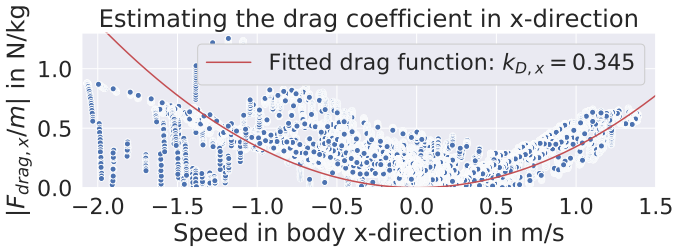
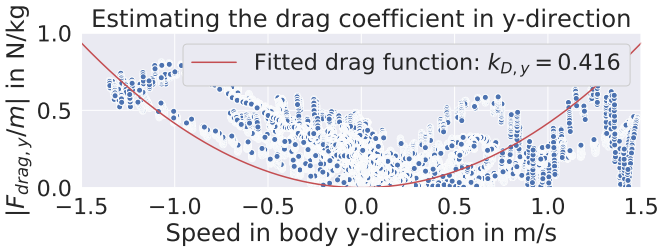


Figure 6-12: Fitted drag model and estimated drag forces in z-direction.

The same process was followed for estimating the drag coefficients in the x and y direction of the quadrotor body frame. The results of this are shown in figures 4-16 and . They led to the following estimates for the drag coefficients in these directions: $k_{D,x} = 0.345$ and $k_{D,y} = 0.416$.



(a) Fitted drag model and estimated drag forces in x-direction.



(b) Fitted drag model and estimated drag forces in y-direction.

Figure 6-13: Fitted drag models and estimated drag forces in x and y-direction.

6-2-4 Moments of Inertia

Finally, the moments of inertia about the x and y-axis of the quadrotor were estimated. This was done by following the procedure described in section III.B of the article, using the measurement data gathered in the second system identification experiment. The estimation of the moment of inertia about the body y-axis will be described below as an example.

As mentioned in the article, equation 6-3 can be used to estimate the external torque around the y-axis, assuming angular drag to be negligible. Since the measurements were carried out with no nearby obstacles, the external torque is assumed to be zero. This equation can then be rewritten to equation 6-4 to estimate I_{yy}.

$$\frac{\tau_{ext,y}}{I_{yy}} = \dot{q} - \frac{1}{I_{yy}}\left[\frac{k_1}{m}\omega_1^2 d_{1,y} + \frac{k_2}{m}\omega_2^2 d_{2,y} - \frac{k_3}{m}\omega_3^2 d_{3,y} - \frac{k_4}{m}\omega_4^2 d_{4,y}\right] \tag{6-3}$$

$$I_{yy} = \frac{\frac{k_1}{m}\omega_1^2 d_{1,y} + \frac{k_2}{m}\omega_2^2 d_{2,y} - \frac{k_3}{m}\omega_3^2 d_{3,y} - \frac{k_4}{m}\omega_4^2 d_{4,y}}{\dot{q}} \tag{6-4}$$

The moment of inertia about the y-axis can be estimated for every time step of the performed pitch maneuvers. The results of which are shown in figure 6-15. From the large distribution, it can be concluded that there is quite some noise present in the estimation. There are even some time steps for which the estimated I_{yy} is negative.

A potential explanation for this would be that both equations 6-3 and 6-4 make use of the angular acceleration $\dot{q}$. The angular acceleration is not actually measured by the quadrotor, so it was estimated by taking the derivative of the filtered body rate q. This introduces inaccuracies into both the external torque estimate and the estimated moment of inertia. For this research, the mean of all estimations is taken as the estimate of the moment of inertia about the y-axis: $I_{yy} = 0.00282$ kg·m^2.

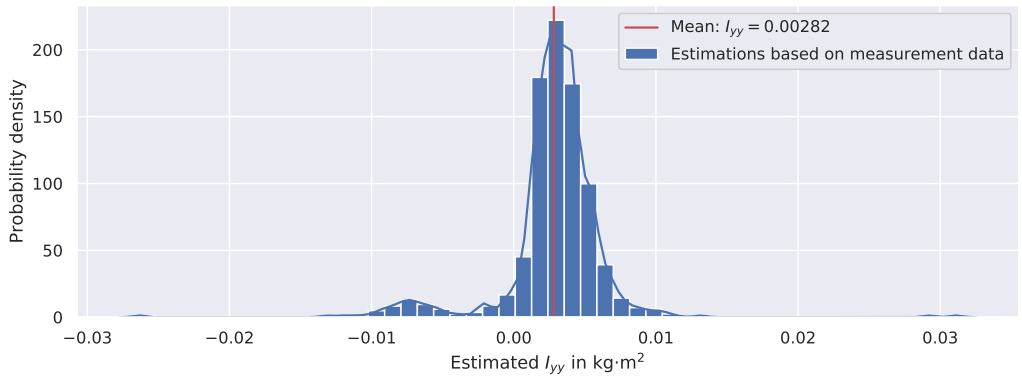


Figure 6-14: Estimated moment of inertia about the y-axis during the pitch maneuvers.

Similarly, the estimation of the moment of inertia about the x-axis uses the angular acceleration $\dot{p}$ and large distribution can be seen for the estimated I_{xx} in figure 6-15. The mean of the distribution is then taken as the estimate of the moment of inertia about the x-axis: $I_{xx} = 0.00242$ kg·m^2.

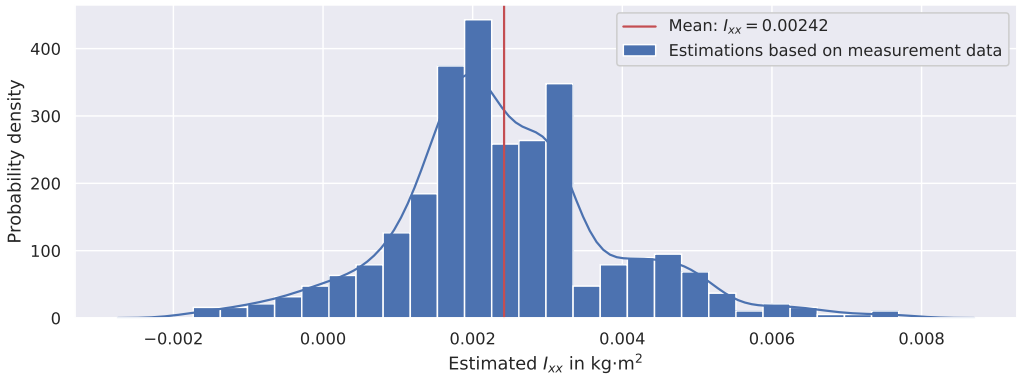


Figure 6-15: Estimated moment of inertia about the x-axis during the roll maneuvers.

6-2-5 Torque model

As discussed in section III.B of the article, it is assumed that the torque produced by one rotor can be approximated by $T(\omega_i) \approx l_i \omega_i^2$ [20]. The torque gains l_i/I_{zz} were then estimated from the heading change maneuvers in the second system identification experiment, where $\tau_{ext,z}$ is assumed to be zero. It was then assumed that l_i was equal for all rotors, allowing equation 6-5 to be rewritten to estimate the torque gains l_i/I_{zz}, as shown in equation 6-6.

$$\frac{\tau_{ext,z}}{I_{zz}} = \dot{r} + \frac{l_1}{I_{zz}}\omega_1^2 - \frac{l_2}{I_{zz}}\omega_2^2 + \frac{l_3}{I_{zz}}\omega_3^2 - \frac{l_4}{I_{zz}}\omega_4^2 \tag{6-5}$$

$$\frac{l_i}{I_{zz}} = \frac{\dot{r}}{-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2} \tag{6-6}$$

Using this equation the torque gains can be estimated for every time step during the heading change maneuvers. By taking the mean, this resulted in the following estimation for the torque gains: $l_i/I_{zz} = 3.904 * 10^{-6}$/rad.

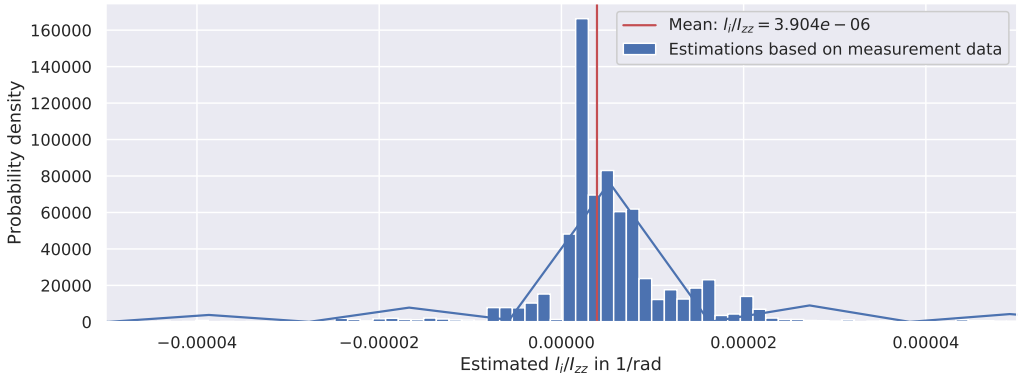


Figure 6-16: Estimated torque gains for each time step in the heading change maneuvers.

6-3 Determining the hyperparameters

As mentioned in section IV.B of the article, two grid searches were carried out in the simulation environment to determine the best sets of hyperparameters for training. The setup and results of these grid searches will be discussed below.

6-3-1 Setup

There are four key hyperparameters that determine the behavior of the RL agent; the learning rate (α), exploration rate at each step (ϵ_{step}), episode-long exploration ($\epsilon_{episode}$) and decay of eligibility traces (λ). To determine a good set, two grid searches were carried out in the simulation. The first grid search was used to determine the best set of hyperparameters for training an agent from scratch. Furthermore, the best performing agent was selected from these results. The second grid search helped determine the set of hyperparameters that are used when a previously trained agent is placed in a new, slightly different, environment.

For the learning rate, three starting points were considered: $\alpha_0 = \{0.1, 0.3, 0.5\}$. The learning rate was decreasing so at episode i, the learning rate is given by $\alpha = \alpha_0 k/(k+i)$, with $k = N_{episodes}/2$ and $N_{episodes}$ the number of episodes.

The episode-long exploration, as determined by $\epsilon_{episode}$, was varied in starting value, as well as decay rate. There were three startling values $\epsilon_{episode,0} = \{0.01, 0.1, 0.5\}$ and three methods of decay: constant $\epsilon_{episode}$, linearly decreasing to a tenth of the value in the first half of the episodes, and linearly decreasing to 0 in the first half of the episodes. This thus resulted in 9 options for $\epsilon_{episode}$.

Four different values for the decay of eligibility traces were considered: $\lambda = \{0.0, 0.1, 0.5, 0.9\}$. In every case λ was kept constant throughout the episodes. Similarly, the two following constant values for the step-wise exploration rate were considered: $\epsilon_{step} = \{0.0, 0.01\}$.

In the initial grid search, a total of 216 different combinations of these parameters were thus evaluated. For each set, 10 agents were trained in the simulation environment for 500 episodes, after which they were evaluated during 100 fully greedy episodes. In this greedy episodes, there was no exploration and no learning. These 2160 agents were then compared, based on the average total reward during these evaluation episodes, a measure of agent performance, and the number of episodes since the agent's policy last changed, a measure of agent stability. Based on this comparison the best set of hyperparameters was determined. From the 10 agents trained with this set, the top performing agent was then selected as the *top agent*. This selection of the top agent is described in section V.B.2 of the article.

In the secondary grid search, 2160 copies of the top agent, 10 for every combination of hyperparameters, were placed in a simulation environment. This simulation environment was slightly modified, a +0.1 bias was added to the $F_{ext,z}/m$ state, the agent thus had to learn to adjust its policy accordingly. Each agent was trained for 100 episodes, using the same exploration strategies and hyperparameter options as in the initial grid search, only with $N_{episodes}$ now 100 instead of 500. Each agent was then evaluation during 100 fully greedy episodes. To determine the best set of hyperparameters, the same procedure was used as for the previously discussed grid search, looking both at the performance of the agent and the stability.

It is important to note here that *best* refers to the local optimum, the best available set from the 216 analyzed sets of hyperparameters. While these 216 options were selected carefully, based on both literature [62] and a preliminary investigation, there might exist a better set of hyperparameters globally.

6-3-2 Hyperparameters for initial training

In the first grid search, 10 agents were trained from scratch for each of the 216 hyperparameter sets. An overview of the performance and stability metrics of each of the 2160 trained agents can be seen in Fig. 6-17a. From this figure, it can be seen that there exist a number of agents with similar top performance. Further inspection showed that the similar performance of these agents is due to the fact that they have converged to a policy that is, for all practical purposes, equal. From this point forward, this policy shall be referred to as the *optimal policy*.

(a) Performance and stability of all 2160 agents trained in the initial grid search

(b) Performance and stability of the 10 agents trained with the best set of hyperparameters

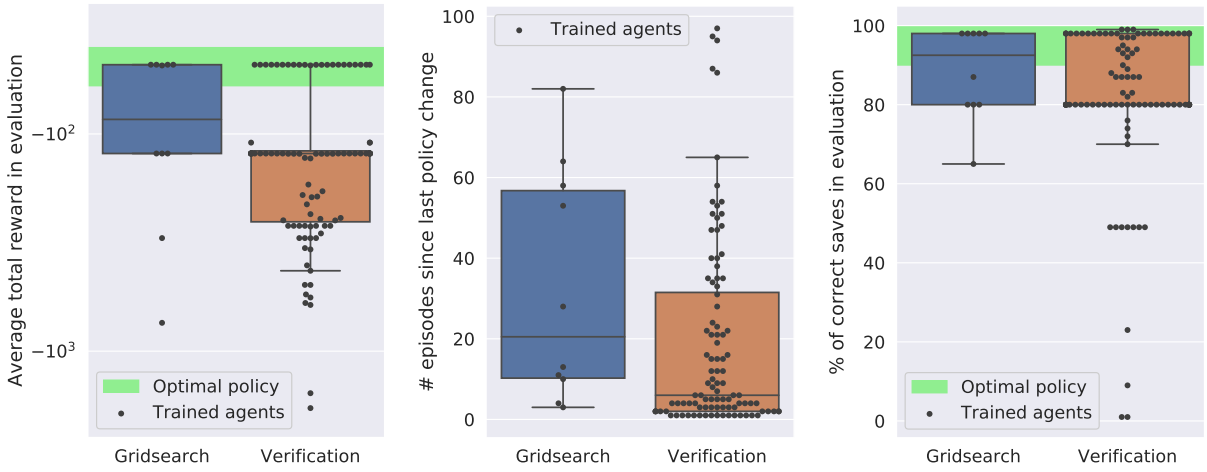
Figure 6-17: Performance and stability of the best and all hyperparameter sets in the initial grid search.

In the initial grid search the following set of hyperparameters was found to be best for training an agent from scratch: $\lambda = 0.1, \epsilon_{step} = 0.01, \alpha_0 = 0.5$, and $\epsilon_{episode,0} = 0.5$ with $\epsilon_{episode}$ linearly decreasing to zero during the first half of the episodes. Of the 10 agents that were trained with this set, 5 have the optimal policy, as can be seen from Fig. 6-17b. When looking at the last policy change, it can be observed that all agents have found the current policy on average 33 episodes before the end of training. Combined, these two metrics made this set of hyperparameters clearly stand out above other sets.

One must, however, be aware of the selection bias; this set of hyperparameters was selected because of its good performance (50% of agents have the optimal policy) and stability. As such, these 10 agents might not be representative of the population intended to be analyzed. Therefore 100 verification agents were trained with this set of hyperparameters. The training

was performed for 500 episodes, under exactly the same conditions as in the initial grid search. The evaluation was once again done during 100 fully greedy episodes.

From the results, shown in Fig. 6-18, the selection bias can clearly be seen. Of the 100 agents only 22%, instead of 50% have the optimal policy. Additionally, the agents have on average found their current policy more recently, an indication that the agents might be less stable. While these figures give a more accurate estimate of agent performance and stability, the selected set of hyperparameters can still be considered a good set to use in the subsequent experiments. Especially when considering that also other sets that performed well in the initial grid search are subject to this selection bias. Furthermore, when considering the performance in terms of obstacle avoidance, one could argue that also the performance of most trained agents that do not have the optimal policy is quite good. As can best be seen from Fig. 6-18c, 85% of the trained agents save the quadrotor successfully in ≥80% of the episodes.



(a) Boxplot showing the average total reward in the evaluation episodes.

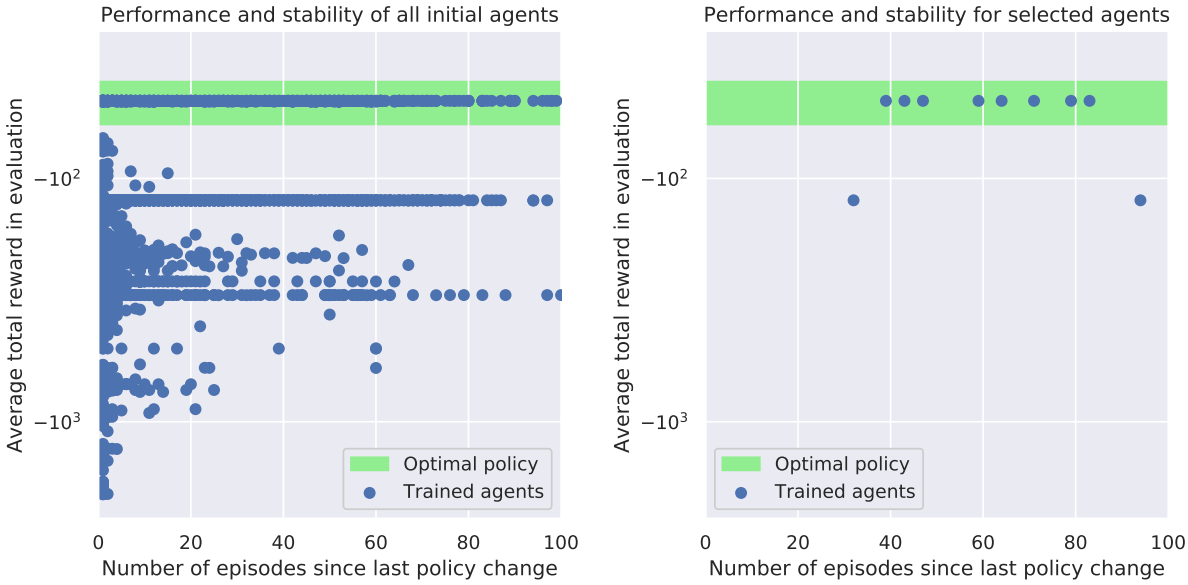
(b) Boxplot of the % of evaluation episodes resulting in a correct save.

(c) Boxplot showing the number of episodes since the last policy change.

Figure 6-18: Selection bias, as seen in the performance and stability metrics for the first grid search.

6-3-3 Hyperparameters for the continuation of training

The results of the secondary grid search are shown in figure 6-19a. In this grid search, the following set of hyperparameters was found to be best for letting the top agent adopt to a slightly different environment: $\lambda = 0.1, \epsilon_{step} = 0.01, \alpha_0 = 0.1$, and $\epsilon_{episode,0} = 0.01$ with $\epsilon_{episode}$ linearly decreasing to zero during the first half of the episodes. Of the 10 agents that continued training with these hyperparameters, 8 found the optimal policy of the modified environment, as can be seen from figure 6-19b. Furthermore, they were quite stable in doing so, the last policy change was on average 61 episodes before the end of training. Considering they were only trained for 100 episodes this shows that the agents are quick to adapt to a change in environment when using these hyperparameters.

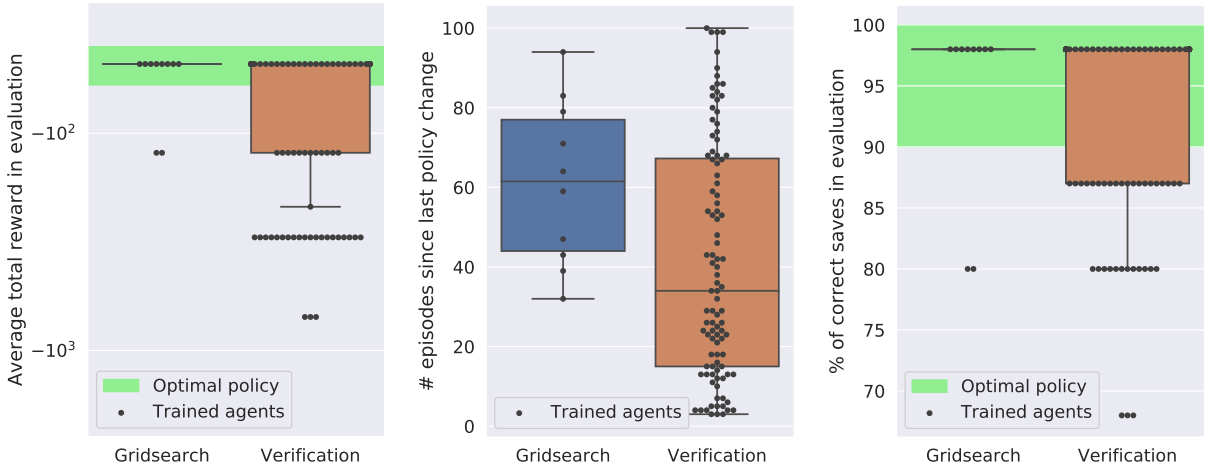


(a) Performance and stability of all 2160 agents for which training was continued in the secondary grid search.

(b) Performance and stability of the 10 agents that continued training with the best set of hyperparameters.

Figure 6-19: Performance and stability of the best and all hyperparameter sets in the continuation grid search.

To correct for the selection bias, the experiment was repeated with 100 agents, now all using the set of hyperparameters mentioned above. The results are shown in figure 6-20. From the 100 agents, 64% learned the optimal policy of the modified environment, with the mean last episode change being 41 episodes before the end of training.



(a) Boxplot showing the average total reward in the evaluation episodes.

(b) Boxplot of the % of evaluation episodes resulting in a correct save.

(c) Boxplot showing the number of episodes since the last policy change.

Figure 6-20: Selection bias, as seen in the performance and stability metrics for the second grid search.

6-4 Investigation of the top agent

In the section below, both the training and evaluation results of the top agent will be discussed in some more detail. This is an extension of the results discussed in section V.B.2 of the article.

6-4-1 Training of the top agent

The training of the selected top agent is shown in figures 6-21 and 6-22. These figures depict respectively the rewards received during training and the absolute changes to the Q-value function. From these figures, several interesting observations can be made:

First of all, the agent only experiences six crashes and three timeouts during its training. Most of the episodes thus end with a correct save. Secondly, while the reward per episode is seen to vary quite a bit, an upward trend can be seen in the rolling reward, especially between episode 0 and 250. Finally, changes to the Q-function keep happening, but the frequency of large changes decreases over time.

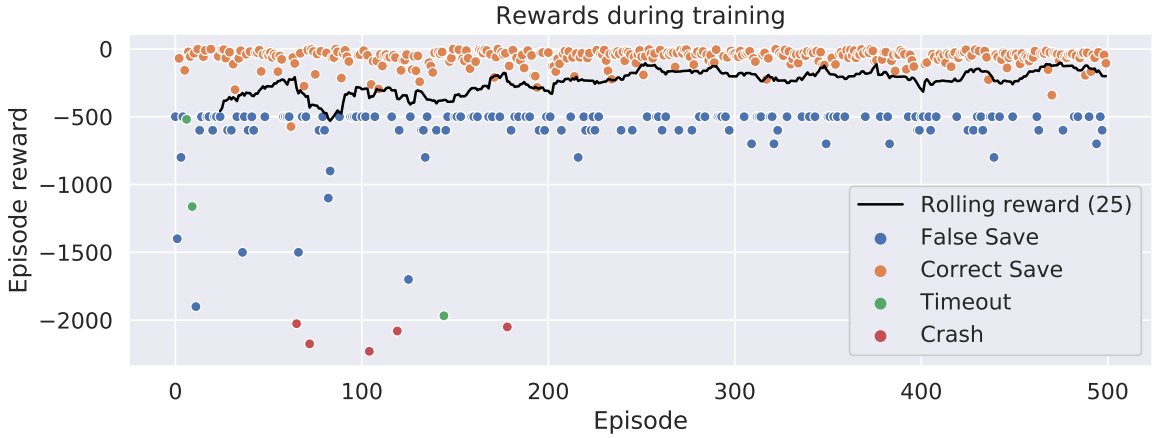


Figure 6-21: Rewards received during training by the selected top agent.

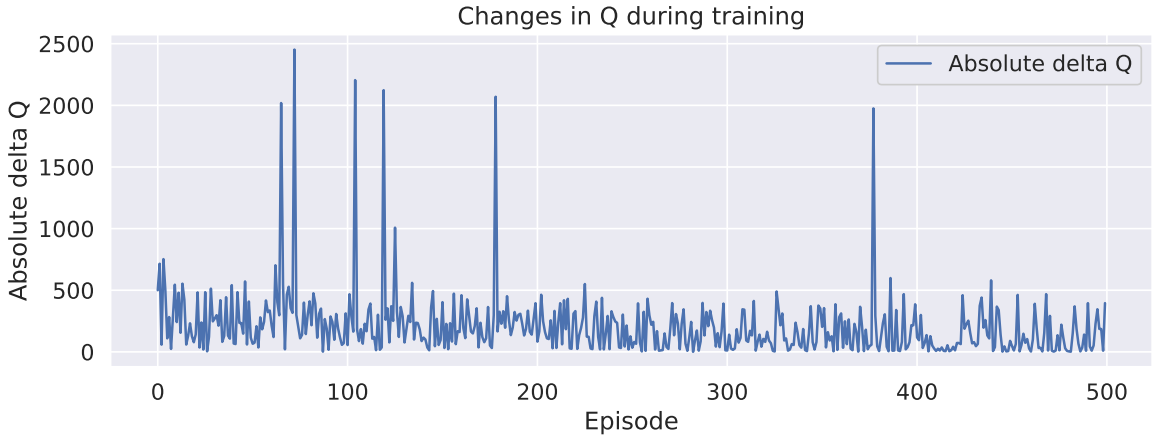


Figure 6-22: Changes in Q during training of the top agent.

Since the Q-value function is still changing, it is important to check whether the agent has indeed converged and is stable in its policy. In other words, if training had continued for a couple more episodes, would the agent still have had the same policy?

To check this, four different continuations of the training were investigated, each for 100 episodes:

A Continue training with exploring starts, using the hyperparameters as they were after 500 episodes of initial training. ($\lambda = 0.1, \epsilon_{actor} = 0.01, \epsilon_{critic} = 0.0, \alpha_{critic} = 0.5 * \frac{250}{250+500} \approx 0.1667$).

B Continue training without exploring starts, using the hyperparameters as they were after 500 episodes of initial training.

C Continue training with a fresh set of the 'retrain' hyperparameters, with exploring starts.

D Continue training with a fresh set of the 'retrain' hyperparameters, but without exploring starts.

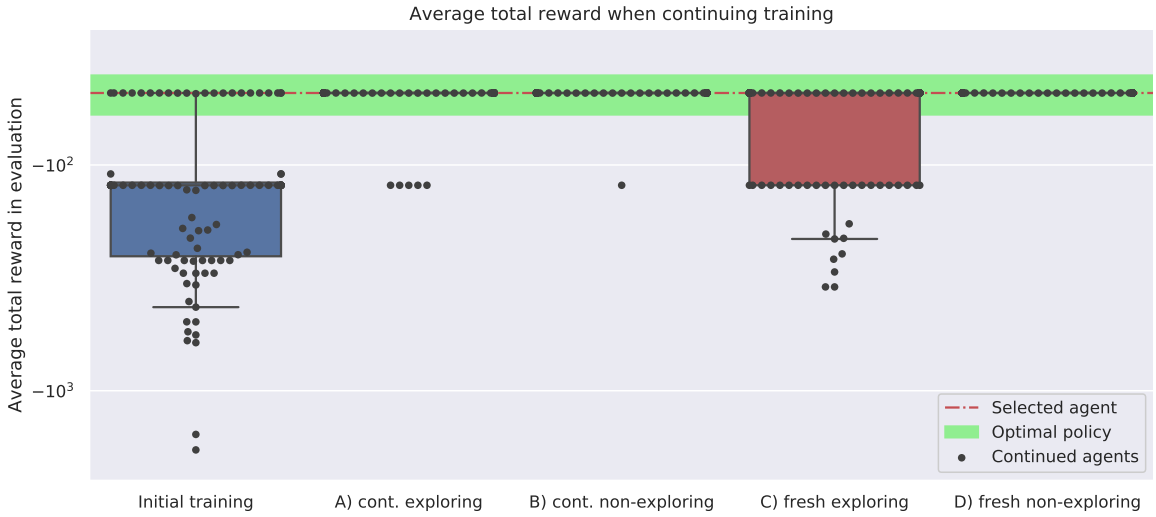


Figure 6-23: Average total reward for 100 copies of the top agent, after training has been continued in four different manners.

The results can be seen in figures 6-23 and 6-24. From the results the following observations can be made.

First of all when considering whether the agent would have had the same policy when training had continued for a couple more episodes, continuation A is the most relevant. Its results show that after 100 additional episodes 95% of the agents had the same policy. Thus indicating a large chance that the agent would have had the same policy.

Secondly, it can be observed that when continuing to use exploring starts, a large percentage of the agents forget the previously found optimal policy. As such, reducing the exploring starts range throughout the training episodes might be a beneficial improvement to the RL setup.

Overall, of the 100 copies of the selected top agent, most are still performing excellently after training has been continued in either of the four ways. Only those for which training was continued with a fresh set of parameters and using exploring starts, saw a significant portion of agents 'forget' the top policy.

Finally, it is interesting to note that even though most agents end up with the same policy they started with (figure 6-23), almost all experience policy changes throughout the continued training (figure 6-24).

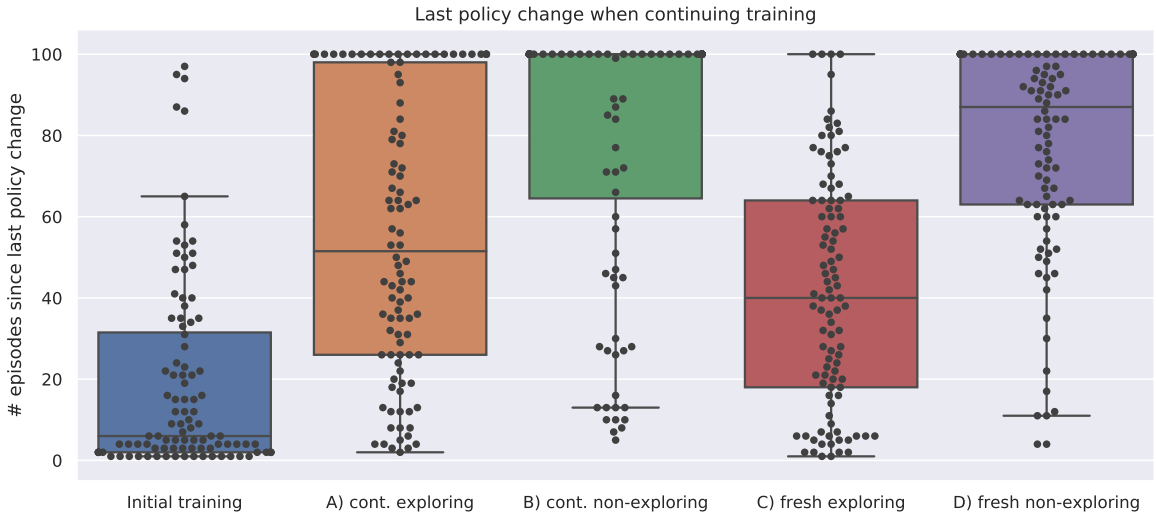


Figure 6-24: Number of episodes since last policy change for 100 copies of the top agent, after training has been continued in four different manners.

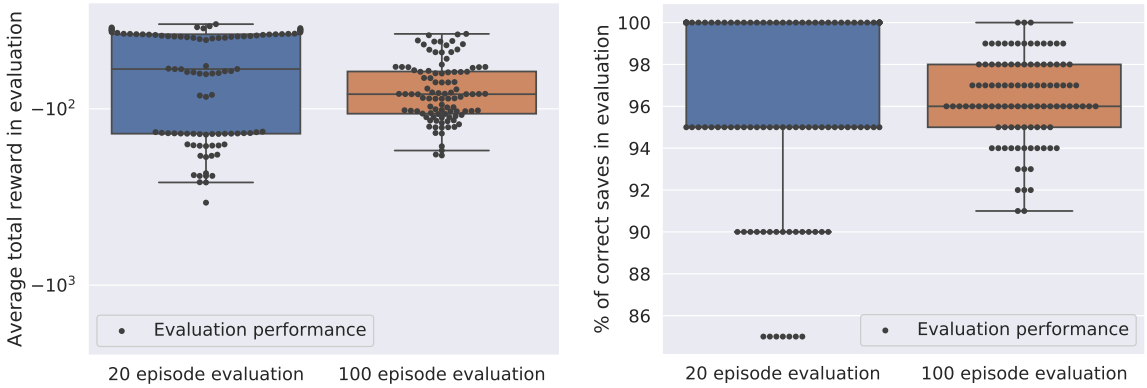
6-4-2 Evaluation of the top agent

In addition to the 100-episode long evaluation conducted directly after training of the top agent, two additional evaluations were performed to gather an understanding of the performance distribution of this agent. In the first evaluation, the agent was evaluated 100 times, with each evaluation consisting of 20 fully greedy evaluation episodes. In the second evaluation, the agent was evaluated once again 100 times, but now each evaluation consisted of 100 fully greedy evaluation episode. In both cases, each evaluation episode was conducted with uniquely randomly generated noise on the $F_{ext,z}/m$ estimator.

The results are shown in figure 6-25 and table 6-1. They can be used to assess whether the performance metrics of policies evaluated in simulation (100 episodes) or real flight (20 episodes), are significantly different from the performance distribution of this top agent in simulation.

	# episodes	Mean reward	σ_{reward}	Mean % correct save	σ_{saved}
20 episode evaluation	2,000	-92.01	69.59	95.90%	4.66%
100 episode evaluation	10,000	-86.50	32.06	96.36%	2.02%

Table 6-1: Key performance metrics after evaluating the top agent for 20 and 100 episodes.



(a) Boxplot showing the average reward in evaluation when evaluating the top agent for 20 and 100 episodes.

(b) Boxplot showing the percentage of evaluation episodes resulting in a correct save when evaluating the top agent for 20 and 100 episodes.

Figure 6-25: Distribution of performance when evaluating the top agent for 20 and 100 episodes.

6-5 Using obstacle-airflow interactions to detect a wall

To estimate the feasibility of detecting large vertical surfaces on the same level as a quadrotor, like walls, an experiment was conducted using the Parrot Bebop 1 drone. This experiment was once again carried out in the Cyberzoo at Delft University of Technology, Within this testing area, a 1m wide, 2.05m high screen was placed that functioned as the wall. The quadrotor then approached the wall at a height of 1.5m, while staying on the center line of the wall with its (forward) body x-axis perpendicular to the wall. It approached the wall starting from 1-meter to up to 1-centimeter distance, where the distance is taken from the front of the quadrotor.

Following the procedure described in section III of the article, the 3 external forces and 3 external torques were then estimated at each timestep. The results are set out against the distance to the wall in the figures below.

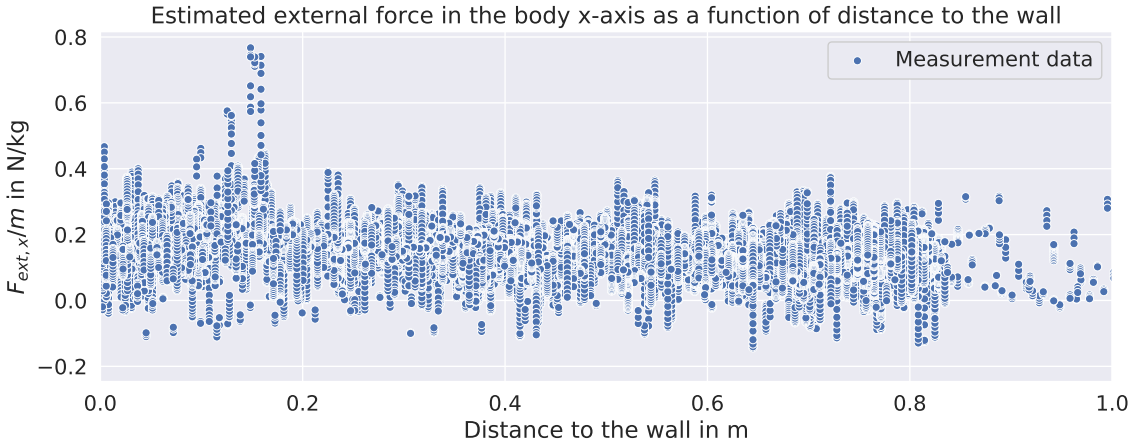


Figure 6-26: Estimated external force in the body x-axis as a function of distance to the wall.

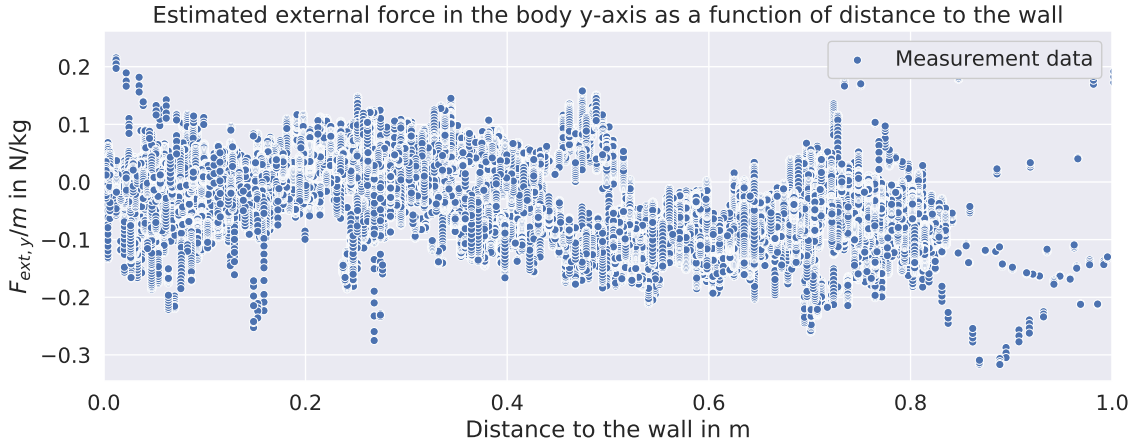


Figure 6-27: Estimated external force in the body y-axis as a function of distance to the wall.

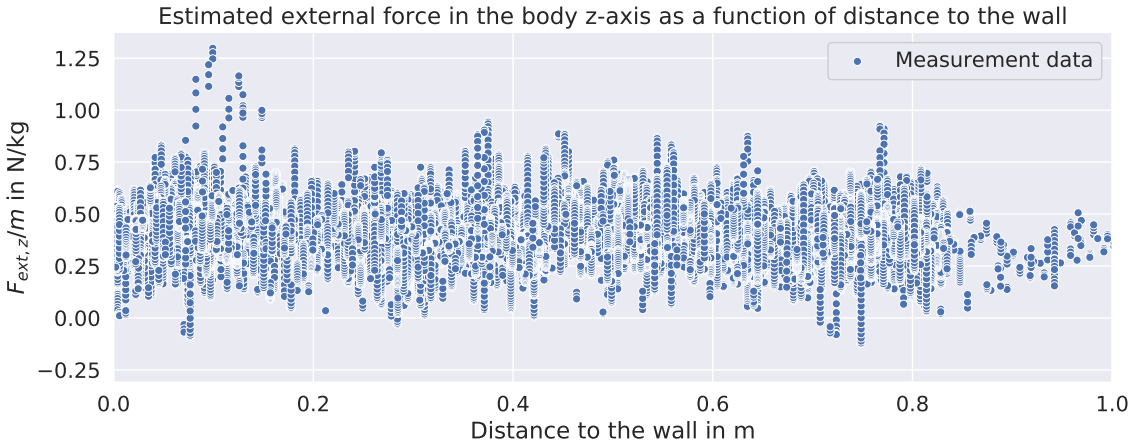


Figure 6-28: Estimated external force in the body z-axis as a function of distance to the wall.

From the estimated external forces in the body x, y and z-axis, as shown in figures 6-26, 6-27 and 6-28, no clear influence of the wall can be seen. An explanation for this can be that influences are smaller than the noise, or other disturbances present in the estimations.

In the measurement data from Mckinnon [37] external forces in the x-y plane between 0.06 and 0.1N were estimated at several centimeters from the wall. Considering the mass of the Bebop 1, similar forces would result in in an $F_{ext,x}/m$ or $F_{ext,y}/m$ between 0.14N/kg and 0.24N/kg. As can be seen from the figures 6-26 and 6-27, the noise in the estimated forces in the x and y-axis has a similar magnitude.

When investigating the external force in the body z-axis, the measurements from Mckinnon [37] suggest a downward force (decrease in lift) between 0.05N and 0.15N. The effect on the Parrot Bebop 1 is not likely to be exactly the same, as the used quadrotors differ in many characteristics, including rotor radius. However, if the forces are assumed to be of a similar order of magnitude, a $F_{ext,z}/m$ between around 0.1N/kg and 0.5N/kg is expected for the Bebop 1. This would be similar to the magnitude of the noise, as seen in figure 6-28.

It can thus be expected that the signal-to-noise ratio in the estimation of external forces is too low to see the influence of the obstacle-airflow interactions caused by the wall.

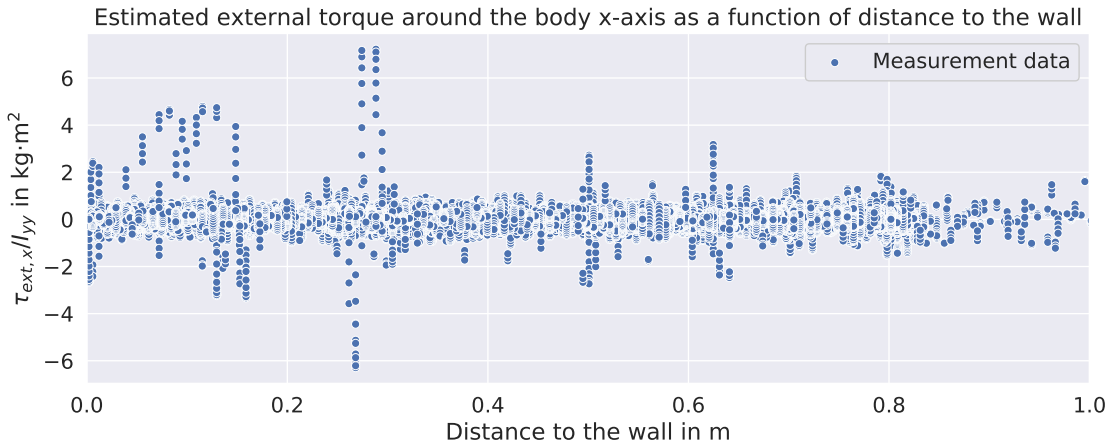


Figure 6-29: Estimated external torque around the body x-axis as a function of distance to the wall.

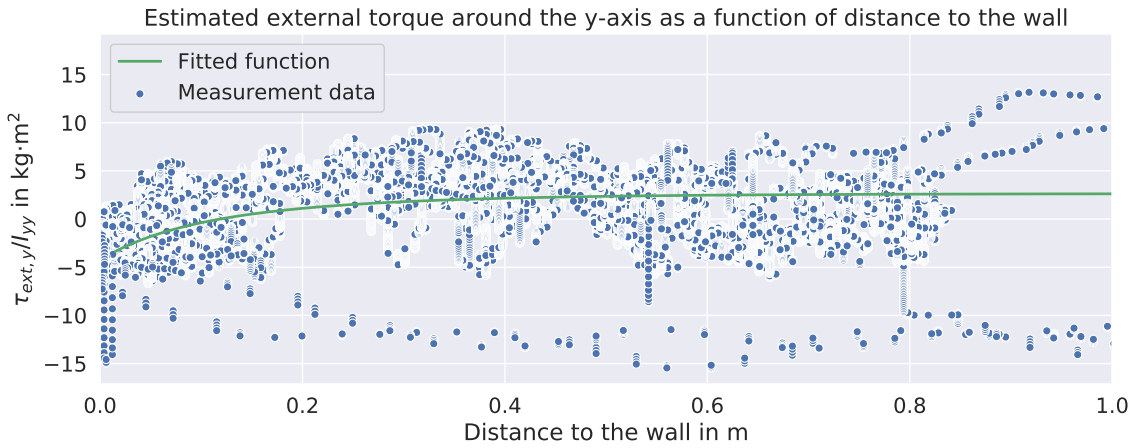


Figure 6-30: Estimated external torque around the body y-axis as a function of distance to the wall.

The estimated external torques around the body x, y and z-axis are shown in figures 6-29, 6-30 and 6-31. In the external torque around the body x (pitch) and z-axis (heading), no clear influence of the wall can be seen. The estimated external torque around the y-axis does show an effect that correlates with the distance to the wall. From 20cm distance an increasing external pitch down moment can be seen in figure 6-30. This could correspond to a decrease in thrust of the front rotors.

To get an estimate of the SNR of this effect, equation 6-7 was fitted to this dataset using Non-linear least squares and the Trust Region Reflective (TRF) algorithm. The distance to the wall is denoted by o, and k_1 to k_5 are general coefficients.

$$\frac{\tau_{ext,y}}{I_{zz}} = k_1 \frac{1}{o + k_2} + k_3 \Big(\frac{1}{o + k_4}\Big)^2 + k_5 \tag{6-7}$$

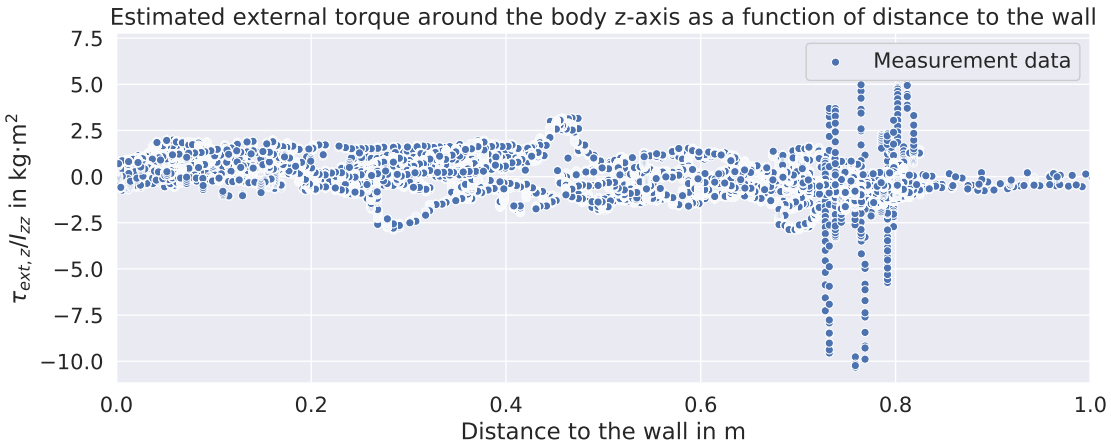


Figure 6-31: Estimated external torque around the body z-axis as a function of distance to the wall.

This resulted in the following coefficients $k_1 = 1.00, k_2 = 0.262, k_3 = -0.709, k_4 = 0.262, k_5 = 2.271$. The deviations with respect to the fitted function were then calculated and the standard deviation was found to be $\sigma = 3.38$. Using equation 22 of the article (section V.A.3), and correcting for the bias in the $\tau_{ext,y}/I_{zz}$ estimation, the SNR at 0.15cm from the wall was estimated to be -4.15dB.

As the effect becomes stronger when nearing the wall, the SNR will also increase. Detection and avoidance closer to the wall might thus still be possible, perhaps requiring to fly at lower speeds. The expected SNR of the wall effect seen in $\tau_{ext,y}/I_{zz}$ as a function of the distance to the wall is shown in figure 6-32.

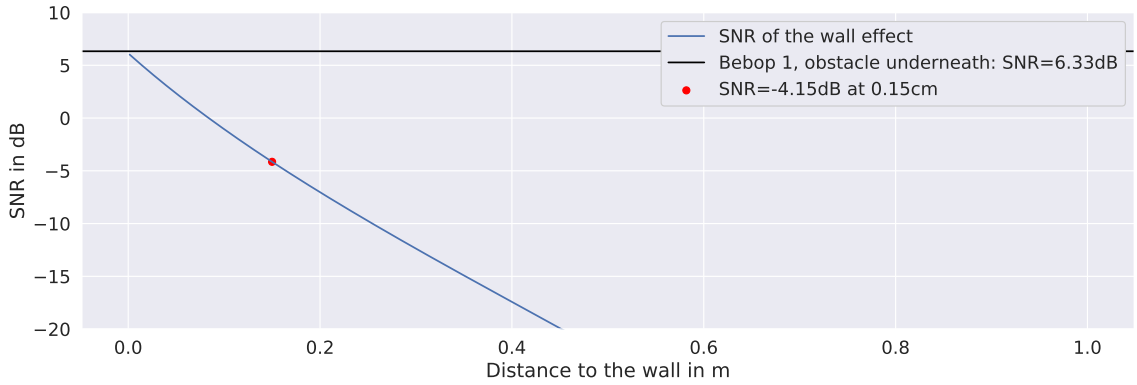


Figure 6-32: Expected SNR of the $\tau_{ext,y}/I_{zz}$ estimation, as a function of the distance to the wall.

Based on the results discussed above, combined with figure 20 from the article, it can be expected that none of the RL agents will find a policy better than the trivial *always-save* policy when the current $\tau_{ext,y}/I_{zz}$ estimation is used as the (only) indicator for the presence of a wall.

However, by analyzing the effects in more detail, using better sensors, or using more than 1 external force or torque as the state, the performance might be improved. Furthermore, the performance could also be improved by implementing some of the recommendations mentioned in section IX of the article, and chapter 8 of the thesis.

6-6 Positive rewards for performing a correct save

Even with exploring starts the agent seldom experiences a crash during training. This can be seen from figure 6-21. As such, the fact that performing a 'no-action' might increase the likelihood of such a crash, is often not fully accounted for in the value function. The agent simply hasn't experienced this often enough. Based on this insight an experiment was carried out with an alternative reward structure.

Previously all rewards were negative: -2,000 for a crash, -500 for a false save, a small negative reward for a hover and the following negative reward for a save:

$$R(s,a) = \begin{cases} -500 & \text{if } z_s > 0.25 \\ -50 * \frac{0.25 - z_s}{0.25 - z_{termination}} & \text{if } z_s \leq 0.25 \end{cases} \tag{6-8}$$

The reward for a save would thus be zero when saved exactly at 0.25m and becoming negative when saved closer to the ground. The idea behind this is to stimulate the quadrotor being saved farther away from the ground while staying with only negative rewards.

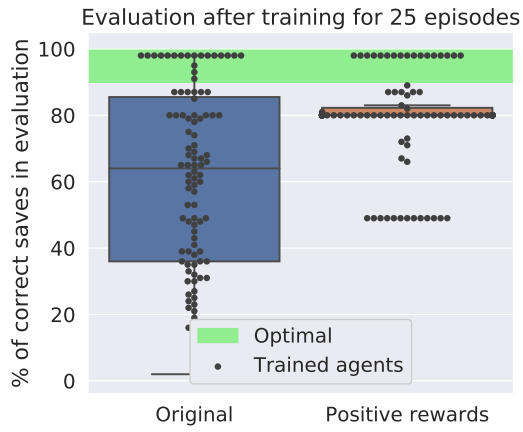
In the experiment, the rewards for a save were changed, such that they are positive for a correct save. When saved at the termination boundary (0.05m) it will be zero, and when saved at the maximum height (0.25m) it will be +50.

$$R(s,a) = \begin{cases} -500 & \text{if } z_s > 0.25 \\ 50 * \frac{z_s - 0.05}{0.25 - z_{termination}} & \text{if } z_s \leq 0.25 \end{cases} \tag{6-9}$$

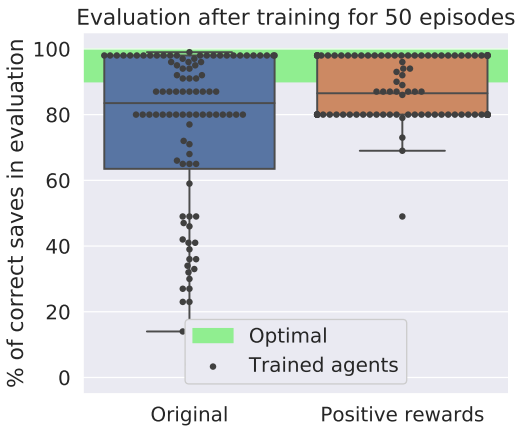
Using this new reward structure 100 agents were trained for 25, 50, 100 and 500 episodes. The training was performed in the created simulation environment, using the hyperparameters deemed best for initial training, as provided in section IV.E of the article. The results are shown in figure 6-33 and table 6-2. As the reward structure has been altered, the percentage of episodes resulting in a correct save is now used as the key metric of performance.

	Correct save in $\geq$ 80% of episodes		**Optimal policy**	
	Original	Positive rewards	Original	Positive rewards
25 episodes	33%	82%	16%	20%
50 episodes	67%	95%	25%	37%
100 episodes	92%	94%	19%	55%
500 episodes	85%	87%	22%	40%

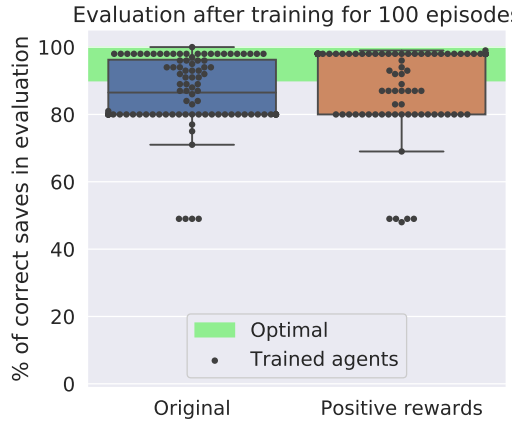
Table 6-2: Percentage of the trained agents with good or optimal performance when being trained with the original or altered reward structure.



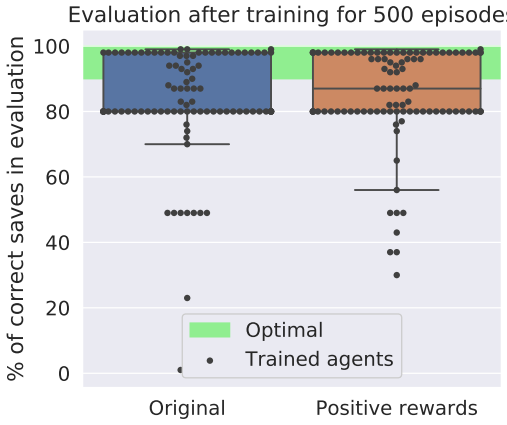
(a) Boxplot of performance when training for 25 episodes.



(b) Boxplot of performance when training for 50 episodes.



(c) Boxplot of performance when training for 100 episodes.



(d) Boxplot of performance when training for 500 episodes.

Figure 6-33: Comparison of performance, as measured by the percentage of correct saves, between agents trained using the original and the altered reward structure.

From these results it can be observed that the overall performance distribution for agents trained for 25 or 50 episodes is better when using the altered reward structure. Of the 100 agents trained for 50 episodes with the altered reward structure, 95% saved the quadrotor in $\geq$ 80% of the evaluation episodes. With the original reward structure this was only 67%. Furthermore, the chances of the agent finding the optimal policy are increased.The number of agents that found the optimal policy in 50 episodes increased from 20% to 37% by using this new reward structure. This increase is also present when training for 25, 100 or 500 episodes.

Finally, it was noticed that for the agents trained with the alternative reward structure, on average, the number of episodes since the last policy change was higher. This indicated that these agents converged sooner to there policy.

It can thus be concluded that rewarding correct saves with a positive reward can be beneficial to both the performance and stability of trained agents.

Chapter 7

Conclusions

This chapter will summarize the most important results and conclusions from the article (chapter 3), literature survey (chapter 4), preliminary investigation (chapter 5) and additional results (chapter 6).

The conclusions from the literature survey and preliminary investigation contain references to the underlying sub conclusions, indicated as SC#. An overview of these sub conclusions and their location within the thesis is provided in appendix B.

7-1 Conclusions

In the literature study, the state of the art in three fields was studied: obstacle avoidance, obstacle-airflow interactions between a quadcopter and obstacles and the field of reinforcement learning.

From the study into current obstacle avoidance methods, it was concluded that there is added value in a low-cost, obstacle avoidance method that does not require good lighting conditions or the addition of any sensors, especially for smaller quadcopters.

With respect to the obstacle-airflow interactions between a quadcopter and obstacles, it was noted that the effect due to ground and ceiling surfaces can be measured up to a distance of 5 times the rotor radius. Furthermore, simple models exist describing this relationship. About the effect near vertical walls, less is known, however, an experiment by Mckinnon [37] indicated that the effect can be measured up to 0.35m for a drone of similar size to the Parrot Bebop quadrotor.

The result of the study of reinforcement learning was twofold. First, by studying the basis of reinforcement learning, it allowed for a computational implementation of reinforcement learning, which was used in both the preliminary investigation and is the final thesis research. Secondly, by studying the state of the art, several challenges were identified that could be expected in the full research. Most notably the challenges of safety, robustness, online efficiency and sample efficiency. The literature study however also identified potential solutions, such as SHERPA [35], an algorithm that could potentially tackle the issue of safety.

In the preliminary research, a 1-dimensional simplified version of the challenge at hand was turned into a reinforcement learning problem on which 657 reinforcement learning agents were trained during a grid search. Among these agents were Monte Carlo, SARSA and Q-learning agents, with different values for the exploration rate (ϵ), learning rate (α) and eligibility traces (λ). After a fully greedy evaluation, investigation of the influence of these hyperparameters and an in-depth evaluation of the best performing agent, the following key conclusions were drawn:

- This novel RL-control scheme can work, as is demonstrated by the performance of the trained agents (SC2, SC4). Especially the top Q-learning(0.5) agent which, in every of the 500 evaluation episodes, let's the ball follow the reference signal uninterrupted, until a wall is detected, then the ball is successfully saved from hitting the wall.
- However, to ensure that this novel RL-control scheme indeed works as desired it is important that oscillations between the conventional controller and the reinforcement agent are not encouraged (SC1) and during training, the agent is exposed to the full set of situations it will have to operate in. (SC16, SC17)
- Q-learning is best suited, and Monte Carlo methods are least suited, to train agents in this reinforcement learning problem, based on the performance in this specific problem (SC8a, SC8b) and the robustness with respect to the hyperparameters. (SC8c)
- When using Q-learning, a high, non-decreasing, exploration rate (SC9), a λ between 0.1 and 0.5 (SC14, SC15), and a learning rate that decreases to a quarter of its initial value during the first half of the episodes, are expected to produce the best performing agents. (SC11, SC12)

While these conclusions are specific to the 1-dimensional simplified reinforcement learning problem, they constituted a good starting point for approaching the reinforcement learning problem of the final thesis research.

This final thesis research consisted of three phases. In the first phase, a large number of reinforcement learning agents were trained in simulation to detect and avoid obstacles underneath a simulated quadrotor. Using this setup several experiments were then executed in simulation. From these results, it can be concluded that:

- The estimated external force in the z-direction can be a good indicator for the presence of obstacles underneath.
- A reinforcement learning agent can be trained to use this estimated external force to detect and avoid obstacles, but due to the stochasticity involved in both the state signal and the exploration, not all agents being trained will find the optimal policy.
- The combination of episode-ending actions, multistep actions and a lot of timesteps compared to the number of states, make exploration challenging within this RL setup.
- Not many training episodes are required. Training for 100 episodes is sufficient when the purpose is to achieve the best performance for each of the trained agents. However, if one is only concerned with finding one agent with the optimal policy, it can be more efficient to train agents for less episodes.

In the second phase, several experiments were conducted in real flight. These flight experiments were conducted in the Cyberzoo of Delft University of Technology, using a Parrot Bebop 1 drone. From these results, it can be concluded that:

- The top agent trained in simulation is able to save the quadrotor from hitting the obstacle underneath in 80% of the episodes.

- The optimal policy found in simulation is also the optimal policy in real flight, its performance is however about 20% worse in real flight.
- It is possible for an agent trained fully from scratch in real flight, to achieve the same performance. This is shown by training 5 agents during 50 episodes each, without prior knowledge or training in simulation. Of these 5 agents, one found the optimal policy.

In the third and final phase, the extension of this method to other quadrotors and other obstacle types was investigated. The first was done by demonstrating that, following the procedure provided in the article, the obstacle avoidance method could successfully be extended to the Parrot Bebop 2 quadrotor. The extension to other obstacle types was investigated by evaluating the top agent with another obstacle underneath. Furthermore, several measurement flights were performed close to a wall, assessing the potential of using this method to detect and avoid obstacles on the same level. From these results, it can be concluded that:

- Application of this method to other quadrotors is possible and requires only limited time and resources. The performance of the obstacle avoidance method depends on the signal-to-noise ratio for that specific quadrotor.
- For surfaces above the quadrotor, e.g. ceilings, the method is expected to achieve a similar or even better performance as for surfaces underneath.
- The extension to surfaces on the same level as the quadrotor, e.g. walls, is expected to require some improvements to the SNR of the estimated forces and torques, or to the usage thereof, before a similar performance can be achieved.
- The current proof-of-concept is not only able to detect and avoid large surfaces, but also an 75cm x 53cm x 17.5cm box placed underneath the quadrotor.

Overall, it can thus be concluded that it is possible to use reinforcement learning and obstacle-airflow interactions for the detection and avoidance of obstacles underneath a quadrotor. Furthermore, it is expected that this can be extended to obstacles above, and perhaps obstacles on the same level. By doing so, not only the main research question and the subquestions have been answered. Also a new obstacle avoidance method has been developed. A method that does not require the addition of any sensors, and therefore with great potential to serve either as a primary obstacle avoidance method for lightweight quadrotors or as an extra check or redundancy to other obstacle avoidance methods, thereby increasing safety and reliability.

Chapter 8

Recommendations

There are several ways in which the developed obstacle avoidance method can be improved or extended. They can be categorized in improvements to the simulation environment, improvements to the developed method, the extension to other obstacles, and the extension to other UAVs.

Recommendations to improve the simulation environment

The key recommendation to improve the simulation environment is to include the delay introduced by the filtering of measurements. By doing so, it is expected that the performance in the simulation environment better represents the performance in the actual flight environment, as the difference in required reaction time will have been removed.

Recommendations to improve the developed method

The developed method could be improved by either improving the estimation of external forces and torques or by improving the way these estimated forces and torques are used in the reinforcement learning setup. When considering ways to improve the estimation of the external forces and torques several recommendations can be made:

- **More accurate thrust, torque and drag models** The estimations of external forces and torques can be improved by using more accurate models for the produced thrust, torque and drag of the quadrotor. This could be accomplished by using more advanced models, or by improving the initialization procedures and assumptions used for the current models. An example of the latter would be to improve the estimation of the rotor gains k_i/m by no longer assuming equal gains for of four rotors, instead, using an initialization procedure in which the gain for each rotor is estimated individually.
- **Reduction of the sensor noise** Much of the noise present in the estimations is caused by noise in the IMU measurements. By using more accurate accelerometer and gyroscope sensors, the torque and force estimations could be improved.
- **Improved filtering** The remaining sensor noise and the noise caused by actual vibrations in the body could perhaps be further reduced by improved filtering, for example by using a Kalman filter instead of the low-pass Butterworth filter [30]. Another improvement would be the reduction of delay introduced by the filter, as this would decrease the reaction time of the obstacle avoidance method.
- **Filtering out other disturbances** Other disturbances, like wind, turbulence or the airflow of other aerial vehicles, can cause external forces and torques on the quadrotor. Methods might be developed by which they can be identified and corrected.

Additionally, the performance of the developed method could be increased by implementing some of the following improvements to the reinforcement learning setup.

- **Extending the state-space** More of the information available could be used by the agent to detect obstacles. For obstacle-airflow interactions expected to result in more than one external force or torque, e.g. those caused by a wall, providing multiple estimators as a state to the RL agent could improve performance. Secondly, one could consider providing not only the latest estimation of external forces or torques but also that of several previous timesteps. Thirdly, the performance can be improved by increasing the discretization density.
- **Function approximators** Extending the state-space can be limited by the curse of dimensionality. As such, implementing a function approximator like a SVM or neural network, as described in section 4-3-9, to represent the action-value function, can be an improvement.
- **New reward structure** As shown in section 6-6 improvements to the reward structure, specifically rewarding correct saves with positive rewards, can improve the performance distribution of the RL agents.
- **Improved exploration strategies** As exploration was one of the key challenges for this RL agent, further research into improvements to the used exploration strategy could increase the stability and convergence of the RL agents. Two key areas that would be of specific interest are the limited number of crashes experienced by the RL agents, as mentioned in section 6-6 and reduction of the exploring starts range throughout the training episodes, as discussed in section 6-4.
- **State-of-the art reinforcement learning techniques** As the field of reinforcement learning is highly active, there are several state-of-the-art RL techniques that might be applied to improve the stability or convergence of the RL agents. Several of these techniques are described in section 4-3-9 of the article. One technique that might be of specific interest to future research would be the application of apprenticeship learning, as described by Abbeel and Ng [1].

Extension to other types of obstacles

The potential of extending this method to the detection and avoidance of other static obstacles is discussed in section VII.A of the article. In that section both the extension to smaller obstacles and to other obstacle locations, e.g. above or in front of the quadrotor, are assessed. An additional topic for future research is the detection and avoidance of dynamic obstacles using this method.

It has already been shown by Mckinnon [37] that the wake of a nearby quadrotor can result in small but measurable external forces and torques. As such, it would be interesting to investigate whether the proposed method might be used to avoid other aerial vehicles in flight.

Similarly, the extension of this method to avoid other dynamic obstacles could be investigated. A key challenge for the RL agent in both cases would be determining the action to take, as

the best avoidance action is likely to depend on the movement of the dynamic obstacle as well. As such, this application might be combined with the extension of the state space to include the estimations for multiple timesteps, thereby providing information about the rate of change to the RL agent.

Extension to other UAVs

While the potential of extending this method to other quadrotors is already discussed and demonstrated in section VII.B of the article, it is also recommended that the potential of extending this method to other types of UAVs is further investigated.

The extension to other multirotors, as well as single rotor UAVs is expected to be relatively straightforward. The equations of motion, and thus the derived external force and torque estimators, are expected to be of similar form. Similar assumptions and initialization procedures might be used as well. Furthermore, obstacle-airflow interactions are expected to be similar for most rotorcraft.

Additionally, the ground-effect is a well-known effect in fixed-wing aircraft. A such, this low-cost method could potentially be used to detect obstacles underneath a fixed-wing UAV. The potential of using this method to detect obstacles above a fixed-wing UAV would require further research. The potential for detection of obstacles on the same level is expected to be limited, considering the forward speed required by a fixed-wing aircraft.

Thirdly, one might consider the application of this method for flapping wing UAVs like the DelFly, an insect-flight inspired Micro Air Vehicle (MAV) developed at the Delft University of Technology [10]. As these UAVs are often designed to be extremely lightweight, the available onboard energy, sensors, and processing capabilities are usually limited. A low-cost method like this could thus be an interesting obstacle avoidance solution for these flapping wing UAVs.

Appendix A

Implemented RL algorithm

$\epsilon_{\text{episode}} \leftarrow \epsilon_{\text{episode},0}$, $\alpha \leftarrow \alpha_0$;
Initialize, for all $s \in S, a \in A(s)$:
    $Q(s,a) \leftarrow 0$, $E(s,a) \leftarrow 0$;
    $\pi(a|s) \leftarrow$ random policy;
for $i \leftarrow 1$ ***to*** $N_{episodes}$ **do**
    $s \leftarrow s_0$;
    repeat
        Take action $a_{\text{no-action}}$, observe r, s';
        $s \leftarrow s'$;
    until *finished exploring starts*;
    History$_i \leftarrow$ empty list;
    Actor: **repeat**
        Choose a, based on the current state s, using policy π, but with a chance ϵ_{step} of picking a random action instead;
        Take action a, observe r, s';
        History$_i \xleftarrow{+} (s,a,r,s')$, $s \leftarrow s'$;
    until *s is terminal*;
    Critic: **foreach** $s,a,r,s' \in History_i$ **do**
        $\delta \leftarrow r + \gamma \max_a Q(s',a) - Q(s,a)$;
        $a* \leftarrow \arg\max_a Q(s,a)$;
        if $a \neq a*$ **then**
            $E(s,a) \leftarrow 0$;
        $E(s,a) \leftarrow 1$;
        foreach $\hat{s} \in S, \hat{a} \in A(s)$ **do**
            $Q(\hat{s},\hat{a}) \leftarrow Q(\hat{s},\hat{a}) + \alpha\delta E(\hat{s},\hat{a})$;
            $E(\hat{s},\hat{a}) \leftarrow \gamma\lambda E(\hat{s},\hat{a})$;
        end
        foreach $\bar{s} \in S$ **do**
            if $\epsilon_{episode} < rand(0,1)$ **then**
                $\pi(a|\bar{s}) \leftarrow$ random action;
            else
                $\pi(a|\bar{s}) \leftarrow \arg\max_a Q(\bar{s},a)$;
            end
        end
    end
    $\epsilon_{\text{episode}} \leftarrow \max(\epsilon_{\text{episode}} - (\epsilon_{\text{episode},0} - \epsilon_{\text{episode,final}})/(N_{\text{episodes}}/2), 0.0)$;
    $\alpha \leftarrow \alpha_0 N_{episodes}/(N_{episodes} + i)$;
end

Algorithm 1: Implementation of Watkin's Q(λ) method with extended exploration.

Appendix B

List of sub conclusions

SC	Conclusion	Page
SC1	The reward structure must be set up such that it does not stimulate unwanted oscillatory behavior between the conventional controller and the reinforcement learning agent.	76
SC2	For all 9 tested methods there is some set of hyperparameters that leads to a better performance in training than the baseline methods.	86
SC3	For Q-learning agents, the average total reward during training is not a good predictor of performance during evaluation. This is most likely due to the higher exploration rate used by the top Q-learning agents.	86
SC4	For all 9 tested methods there is some set of hyperparameters that leads to a policy in which the ball is successfully saved from crashing into the wall, in most of the episodes.	88
SC5	To accurately compare the performance of the best agents, an evaluation with 50 episodes and randomly generated reference signals is not adequate.	88
SC6	The current setup of the reinforcement learning problem, with small negative rewards on all actions that are not *None*, successfully counteracts policies that prevent crashes by not letting the ball follow the reference signal at all.	88
SC7	Monte Carlo methods seem least suited for this reinforcement problem, based on the low performance of the Monte Carlo agents (see figure 5-16) and undesirable policies (see figure 5-15b).	91
SC8	Q-learning is best suited to train agents in this reinforcement learning problem	91
SC8a	Agents trained using a temporal difference method achieve on average a higher performance than those trained using the Monte Carlo method.	91
SC8b	The top agents trained using Q-learning are able to achieve a higher performance than the top agents trained using SARSA.	91
SC8c	Q-learning is more robust with respect to the hyperparameters.	91
SC9	Q-learning agents, in general, benefit from a high, non-decreasing exploration rate.	92
SC10	For most SARSA agents, a low, decreasing exploration rate is beneficial to the performance.	92
SC11	Q-learning agents, in general, benefit from a decreasing learning rate, unless the learning rate is already small (≤ 0.1), then the effect is less.	93
SC12	For Q-learning agents, a learning rate that decreases to a quarter of its initial value after the first half of the episodes seems most promising.	93
SC13	For SARSA agents, a learning rate that decreases to half its initial value after the first half of the episodes looks most promising.	93
SC14	A λ between 0.1 and 0.5 seems most beneficial for this problem when using either Q-learning or SARSA.	93
SC15	Some agents without eligibility traces ($\lambda = 0$) are able to achieve the same level of performance as the top agents with eligibility traces. The bulk however has a slightly lower performance.	93
SC16	The performance of the top agent is independent of the wall position, as long as the distance between the walls is larger than 4 meters.	95
SC17	An increase in either the number of sine waves or the extension of the bandwidth of the parameters, can lead to the ball approaching the wall at higher speeds, which results in a decrease in performance.	96

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[2] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29 (13):1608–1639, 2010. ISSN 02783649. doi: 10.1177/0278364910371999.

[3] Francisco Bonin-font Alberto and Ortiz Gabriel. Visual Navigation for Mobile Robots : A Survey. *Journal of intelligent and robotic systems*, 53(3):263–296, 2008. doi: 10.1007/s10846-008-9235-4.

[4] Amazon. Amazon.com: Prime air. `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`, 2018. URL `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`. [Online; accessed 9-Oct-2018].

[5] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6): 26–38, 2017. ISSN 10535888. doi: 10.1109/MSP.2017.2743240.

[6] Andrew G Barto. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications*, 13(1):41–77, 2003.

[7] I. C. Cheeseman and W. E. Bennett. The Effect of the Ground on a Helicopter Rotor in Forward Flight. *Aeronautical Research Council Reports and Memoranda*, 3021:12, 1955. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.226.6371`.

[8] Zhiyong Chen, Xiaoyuan Luo, and Bicheng Dai. Design of Obstacle Avoidance System for Micro-UAV Based on Binocular Vision. *Proceedings - 2017 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration, ICIICII 2017*, 2017-Decem:67–70, 2018. doi: 10.1109/ICIICII.2017.87.

[9] Joseph Conroy, Gregory Gremillion, Badri Ranganathan, and J. Sean Humbert. Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. *Autonomous Robots*, 27(3):189–198, 2009. ISSN 09295593. doi: 10.1007/s10514-009-9140-0.

[10] Guido De Croon, Mark A Groen, Christophe De Wagter, Bart Remes, G C H E De Croon, M A Groen, C De Wagter, B Remes, R Ruijsink, and B W Van Oudheusden. Design, Aerodynamics, and Autonomy of the DelFly. *Bioinspiration & biomimetics*, 7(2):025003, 2012. doi: 10.1088/1748-3182/7/2/025003. URL `https://www.researchgate.net/publication/225055859`.

[11] Thomas G Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000. ISSN 0038-0385. doi: 10.1177/0038038590430061311. URL `http://soc.sagepub.com/cgi/doi/10.1177/0038038590430061311`.

[12] DroneDeploy. Commercial Drone Industry Trends. Technical Report March, DroneDeploy, 2017.

[13] ANNE DINNOCENZIO. Wal-Mart testing drones in warehouses to manage inventory, 6 2016. URL https://apnews.com/ee9b77dba1a5460a91fef2842ccb8955/wal-mart-testing-drones-warehouses-manage-inventory.

[14] Milan Erdelj, Enrico Natalizio, Kaushik R Chowdhury, and Ian F Akyildiz. Help from the Sky: Leveraging UAVs for Disaster Management. *IEEE Pervasive Computing*, 1: 24–32, 2017.

[15] Aydin Eresen, Nevrez Imamolu, and Mehmet Önder Efe. Autonomous quadrotor flight with vision-based obstacle avoidance in virtual environment. *Expert Systems with Applications*, 39(1):894–905, 2012. ISSN 09574174. doi: 10.1016/j.eswa.2011.07.087.

[16] Eyesee-drone.com. Eyesee an inventory drone solution for your warehouses. https://eyesee-drone.com/#post-24. URL https://eyesee-drone.com/#post-24. [Online; accessed 9-Oct-2018].

[17] David Fofi, Tadeusz Sliwa, and Yvon Voisin. A comparative survey on invisible structured light. In *Machine vision applications in industrial inspection XII*, pages 90–99, 2004. doi: 10.1117/12.525369. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.525369.

[18] Nils Gageik, Thilo Müller, and Sergio Montenegro. Obstacle Detection and Collision Avoidance Using Ultrasonic Distance Sensors for an Autonomous Quadrocopter. *International Journal of Engineering Trends and Technology*, 17(2):1–6, 2012.

[19] Nils Gageik, Paul Benz, and Sergio Montenegro. Obstacle Detection and Collision Avoidance for a UAV With Complementary Low-Cost Sensors. *arXiv preprint arXiv:1704.05519*, pages 599–609, 2015.

[20] Luis Rodolfo García Carrillo, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard. Modeling the Quad-Rotor Mini-Rotorcraft. In *Quad Rotorcraft Control*, pages 23–34. Springer, London, 2013. doi: 10.1007/978-1-4471-4399-4. URL http://link.springer.com/10.1007/978-1-4471-4399-4_2.

[21] Jason Geng. Structured-light 3D surface imaging: a tutorial. *Adv. Opt. Photon.*, 3(2): 128–160, 6 2011. doi: 10.1364/AOP.3.000128. URL http://aop.osa.org/abstract.cfm?URI=aop-3-2-128.

[22] Daniel A Griffiths. A study of dual-rotor interference and ground effect using a free-vortex wake model. *American Helicopter Society 58th Annual Forum, Montreal, Canada, June 11-13*, 2002.

[23] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 1981.

[24] Ian Horswill. Visual Collision Avoidance by Segmentation. *Intelligent Robots and Systems*, pages 87–99, 1995.

[25] Chih-Min Hsiun and Cha'o-Kuang Chen. Aerodynamic characteristics of a two-dimensional airfoil with ground effect. *Journal of Aircraft*, 33(2):386–392, 1996. ISSN 0021-8669. doi: 10.2514/3.46949. URL http://arc.aiaa.org/doi/10.2514/3.46949.

[26] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):1–8, 2017. doi: 10.1109/LRA.2017.2720851. URL http://arxiv.org/abs/1707.05110.

[27] Tommi Jaakkola, Satinder P Singh, and Michale I Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. *Advances in Neural Information Processing Systems*, 7:345, 1994.

[28] Jameco.com. How it works: Xbox kinect. https://www.jameco.com/jameco/workshop/howitworks/xboxkinect.html. [Online; accessed 12-Oct-2018].

[29] Jaime Junell, Tommaso Mannucci, Ye Zhou, and Erik Kampen. Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning. *AIAA Guidance, Navigation, and Control Conference*, pages 1–16, 2016. doi: 10.2514/6.2016-1387. URL http://dx.doi.org/10.2514/6.2016-1387.

[30] Cezary Kownacki. Digital Signal Processing Optimization approach to adapt Kalman filters for the real-time application of accelerometer and gyroscope signals' filtering. *Digital Signal Processing*, 21:131–140, 2011. doi: 10.1016/j.dsp.2010.09.001. URL www.elsevier.com/locate/dsp.

[31] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. *Proceedings of Robotics: Science and Systems IV*, page w/o page numbers, 2008. ISSN 2330765X. URL https://www1.comp.nus.edu.sg/~leews/publications/rss08.pdf%5Cnhttp://www.roboticsproceedings.org/rss04/p9.html.

[32] Saurabh Ladha, Deepan Kishore Kumar, Pavitra Bhalla, Aditya Jain, and R. K. Mittal. Use of LIDAR for Obstacle Avoidance by an Autonomous Aerial Vehicle. *Paper presentation at IARC-2012 (2012)*, pages 1–6, 2012.

[33] Daewon Lee, Asad Awan, Suseong Kim, and H. Jin Kim. Adaptive Control for a VTOL UAV Operating Near a Wall. In *AIAA Guidance, Navigation, and Control Conference*, page 4835, 2012. ISBN 978-1-60086-938-9. doi: 10.2514/6.2012-4835. URL http://arc.aiaa.org/doi/10.2514/6.2012-4835.

[34] Kurt Manal and William Rose. A general solution for the time delay introduced by a low-pass Butterworth digital filter: An application to musculoskeletal modeling. *Journal of Biomechanics*, 40:678–681, 2007. doi: 10.1016/j.jbiomech.2006.02.001. URL www.elsevier.com/locate/jbiomech.

[35] Tommaso Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control*. PhD thesis, Delft University of Technology, 2017.

[36] Siegfried Martens, Paolo Gaudiano, and Gail A. Carpenter. Mobile robot sensor integration with fuzzy ARTMAP. *Proc. 1998 IEEE ISIC/CIRA/ISAS Joint Conference*, pages 307–312, 1998. ISSN 2158-9860. doi: 10.1109/ISIC.1998.713679. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=713679.

[37] Christopher D Mckinnon. *Data Driven , Force Based Interaction for Quadrotors by Data Driven , Force Based Interaction for Quadrotors*. PhD thesis, University of Toronto (Canada), 2015.

[38] C Goerzen Z Kong B Mettler. A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. *Journal of Intelligent and Robotic Systems*, 57 (1-4):65–100, 2010. doi: 10.1007/s10846-009-9383-1.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. ISSN 10450823. doi: 10.1038/nature14236.

[40] H. Morita, M. Hild, J. Miura, and Y. Shirai. View-based localization in outdoor environments based on support vector learning. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2302–2307, 2005. doi: 10.1109/IROS.2005.1545445.

[41] Allistair Moses, Matthew J. Rutherford, Michail Kontitsis, and Kimon P. Valavanis. UAV-borne X-band radar for collision avoidance. *Robotica*, 32(1):97–114, 2014. ISSN 02635747. doi: 10.1017/S0263574713000659.

[42] Paparazzi UAV. PaparazziUAV - Control Loops, 2019. URL `http://wiki.paparazziuav.org/wiki/Control_Loops`.

[43] Ronald Edward Parr. *Hierarchical Control and learning for Markov decision processes*. University of California, Berkeley Berkeley, CA, 1998. URL `citeseer.nj.nec.com/parr98hierarchical.html`.

[44] Carolyn Pearce, Margaret Guckenberg, Bobby Holden, Andrew Leach, Ryan Hughes, Connie Xie, Meredith Hassett, Andrew Adderley, Laura E Barnes, Mark Sherriff, et al. Designing a spatially aware, automated quadcopter using an Android control system. In *Systems and Information Engineering Design Symposium (SIEDS), 2014*, pages 23–28, 2014. ISBN 0101150644. doi: 10.1145/2963145. URL `http://dl.acm.org/citation.cfm?doid=2966278.2963145`.

[45] Caitlin Powers, Daniel Mellinger, Aleksandr Kushleyev, Bruce Kothmann, and Vijay Kumar. Influence of Aerodynamics and Proximity Effects in Quadrotor Flight. In Jaydev P Desai, Gregory Dudek, Oussama Khatib, and Vijay Kumar, editors, *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, pages 289–302. Springer International Publishing, Heidelberg, 2013. ISBN 978-3-319-00065-7. doi: 10.1007/978-3-319-00065-7{_}21. URL `https://doi.org/10.1007/978-3-319-00065-7_21`.

[46] pwc. Clarity from above : transport infrastructure The commercial applications of drone technology in the road and rail sectors. Technical Report January, PwC, 2017.

[47] Vernon J Rossow. Effect of Ground and/or Ceiling Planes on Thrust of Rotors in Hover. *Nasa Technical Memorandum*, 86754, 1985.

[48] Kirill V. Rozhdestvensky. Wing-in-ground effect vehicles. *Progress in Aerospace Sciences*, 42(3):211–283, 2006. ISSN 03760421. doi: 10.1016/j.paerosci.2006.10.001.

[49] Martin Russ, Michael Vohla, Peter Stütz, and S OYoung. LIDAR-based object detection on small UAV: Integration, Experimentation and Results. In *Infotech@ Aerospace 2012*, page 2479. American Institute of Aeronautics and Astronautics, 2012. ISBN 9781600869396. doi: 10.2514/6.2012-2479. URL `http://arc.aiaa.org/doi/pdf/10.2514/6.2012-2479`.

[50] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and Alexander C Berg. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, pages 211–252, 2015. ISSN 0920-5691. doi: 10.1007/s11263-015-0816-y. URL `http://dx.doi.org/10.1007/s11263-015-0816-y`.

[51] Roberto Sabatini, Alessandro Gardi, and Mark A Richardson. LIDAR Obstacle Warning and Avoidance System for Unmanned Aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):718–729, 2014.

[52] P. J. Sanchez-Cuevas, G. Heredia, and A. Ollero. Multirotor UAS for bridge inspection by contact using the ceiling effect. *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, pages 767–774, 2017. doi: 10.1109/ICUAS.2017.7991412.

[53] Pedro Sanchez-Cuevas, Guillermo Heredia, and Anibal Ollero. Characterization of the aerodynamic ground effect and its influence in multirotor control. *International Journal of Aerospace Engineering*, 2017, 2017. ISSN 16875974. doi: 10.1155/2017/1823056.

[54] Florian Schroff and James Philbin. FaceNet : A Unified Embedding for Face Recognition and Clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[55] Jinwoo Seo, Byoung-eon Lee, Beom-soo Kang, Sejong Oh, and Kwanjung Yee. Experimental Study on the Small-Scale Rotor Hover Performance in Partial Ground Conditions. *Journal of the Korean Society for Aeronautical & Space Sciences*, 38(1):12–21, 2010. doi: 10.5139/JKSAS.2010.38.1.012.

[56] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013. ISSN 13872532. doi: 10.1007/s10458-012-9200-2.

[57] I. Sharf, M. Nahon, A. Harmat, W. Khan, M. Michini, N. Speal, M. Trentini, T. Tsadok, and T. Wang. Ground effect experiments and model validation with Draganflyer X8 rotorcraft. In *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014*, pages 1158–1166, 2014. ISBN 9781479923762. doi: 10.1109/ICUAS.2014.6842370.

[58] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550 (7676):354–359, 2017. ISSN 14764687. doi: 10.1038/nature24270. URL `http://dx.doi.org/10.1038/nature24270`.

[59] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning Without State-Estimation in Partially Observable Markovian Decision Processes. In *Machine Learning Proceedings 1994*, pages 284–292. Elsevier, 1994. ISBN 9781558603356. doi: 10.1016/B978-1-55860-335-6.50042-8. URL http://linkinghub.elsevier.com/retrieve/pii/B9781558603356500428.

[60] Joshuah K Stolaroff, Constantine Samaras, Alexandra S Mitchell, Daniel Ceperley, Emma R O Neill, and Alia Lubers. Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery. *Nature Communications*, 9(1):409, 2018. ISSN 2041-1723. doi: 10.1038/s41467-017-02411-5. URL http://dx.doi.org/10.1038/s41467-017-02411-5.

[61] John Stowers, Michael Hayes, and Andrew Bainbridge-Smith. Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor. *2011 IEEE International Conference on Mechatronics*, pages 358–362, 2011. ISSN 978-1-61284-982-9. doi: 10.1109/ICMECH.2011.5971311.

[62] R S Sutton, D Precup, and S Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[63] Richard S Sutton and Andrew G Barto. *Reinforcement Learning : An Introduction*. MIT press, 2015 edition, 2015. ISBN 0262193981.

[64] Matthew E Taylor and Peter Stone. Transfer Learning for Reinforcement Learning Domains : A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009. ISSN 15324435. doi: 10.1007/978-3-642-27645-3. URL http://portal.acm.org/citation.cfm?id=1755839.

[65] Matthew E Taylor and Peter Stone. A survey on transfer learning. *Journal of Machine Learning Research*, 10:1633–1685, 2009. ISSN 18788750. doi: 10.1016/j.wneu.2011.09.036.

[66] Teodor Tomic and Sami Haddadin. A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots. In *IEEE International Conference on Intelligent Robots and Systems*, pages 4197–4204, 2014. ISBN 9781479969340. doi: 10.1109/IROS.2014.6943154.

[67] Teodor Tomić and Sami Haddadin. Simultaneous estimation of aerodynamic and contact forces in flying robots: Applications to metric wind estimation and collision detection. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June (June):5290–5296, 2015. ISSN 10504729. doi: 10.1109/ICRA.2015.7139937.

[68] Iwan Ulrich and Illah Nourbakhsh. Appearance-Based Obstacle Detection with Monocular Color Vision. In *American Association for Artificial Intelligence*, page 866871, 2000. ISBN 0262511126. doi: 10.1.1.43.2596. URL http://www.aaai.org/Papers/AAAI/2000/AAAI00-133.pdf.

[69] Stephen Verbist, Tommaso Mannucci, and Erik-Jan Van Kampen. The Actor-Judge Method: safe state exploration for Hierarchical Reinforcement Learning Controllers. *2018*

AIAA Information Systems-AIAA Infotech @ Aerospace, pages 1–21, 2018. doi: 10.2514/6.2018-1634. URL https://arc.aiaa.org/doi/10.2514/6.2018-1634.

[70] Voyage.auto. An introduction to lidar: The key self-driving car sensor. https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff. [Online; accessed 10-Oct-2018].

[71] Jon Walker. Industrial Uses of Drones 5 Current Business Applications, 7 2017. URL https://www.techemergence.com/industrial-uses-of-drones-applications/.

[72] Christopher J C H Watkins. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[73] Boyu Wei Boyu Wei, Ying Fan Ying Fan, and Baoquan Gao Baoquan Gao. Mobile robot vision system based on linear structured light and DSP. *2009 International Conference on Mechatronics and Automation*, pages 834–839, 2009. doi: 10.1109/ICMA.2009.5246792.

[74] Wikipedia. Optical flow - wikipedia. https://en.wikipedia.org/wiki/Optical_flow. URL https://en.wikipedia.org/wiki/Optical_flow. [Online; accessed 15-Oct-2018].

[75] Qingtao Yu, Fenghua He, Yu Zhang, and Jie Ma. Stereo-vision based obstacle motion information acquisition algorithm for quadrotors. *Chinese Control Conference, CCC*, pages 5590–5595, 2017. ISSN 21612927. doi: 10.23919/ChiCC.2017.8028244.

[76] Yang Yu. Towards Sample Efficient Reinforcement Learning *. Technical report, 2017. URL https://www.ijcai.org/proceedings/2018/0820.pdf.

[77] Yang Yu, Chen Lont, Zhang Weiwei, and Chen Long. Stereo vision based obstacle avoidance strategy for quadcopter UAV. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 490–494, 2018. ISBN 9781538612439.

[78] Nathan Zimmerman, Kellen Carey, and Cristinel Ababei. On aerial indoor position control and system integration for quadcopters using lidars and inertial measurement units. In *Proceedings of ASME 2016 Dynamic Systems and Control Conference*, 2016.