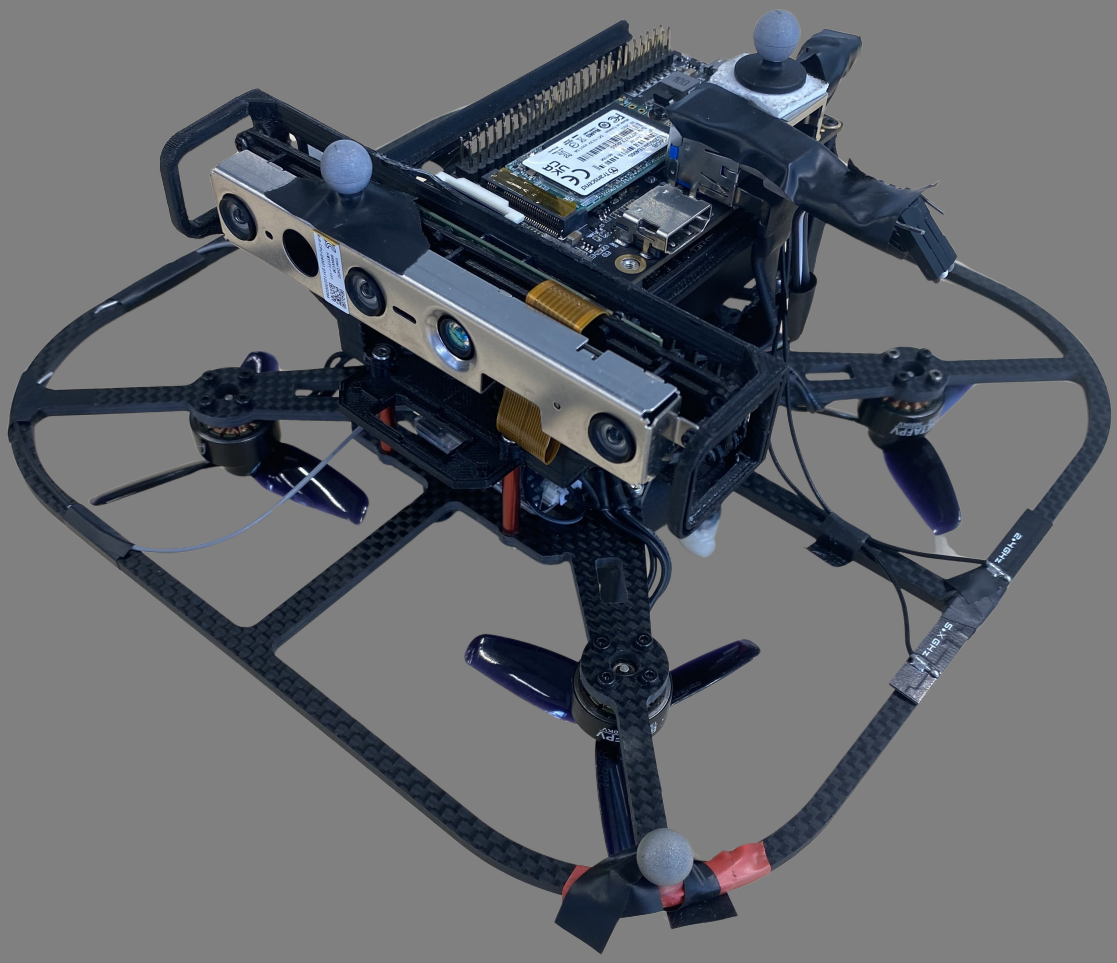
Master Thesis

Inclined quadrotor landing using on-board sensors and computing

Master Thesis Robotics

Ernst Cancrinus



Delft University of Technology

TU Delft

Master Thesis

Inclined quadrotor landing using on-board sensors and computing

by

Ernst Cancrinus

Student number:	4915577
Project duration:	September 4, 2023 – May 30, 2024
Thesis supervisor:	Prof. dr. R. Babuska
Thesis committee:	Dr. L. Ferranti
	Dr. G. Li
	Dr. S. Sun

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

TU Delft

Abstract

Achieving autonomous inclined landing would be an important step towards quadrotors which are able to land anywhere and under any conditions. If a quadrotor is able to safely land anywhere, even when a landing platform is not available, this would open up many useful applications. Firstly, the quadrotor could safely land whenever its battery levels get low or when contact with an operator is lost. Secondly, the quadrotor could be used for different applications, such as reconnaissance, search and rescue, infrastructure or delivery services.

There are many challenges to the field of autonomous inclined quadrotor landing. Firstly, landing safely on an inclined surface without the use of a perching mechanism requires that the quadrotor has a low approach velocity. The quadrotor should also land at the same angle as the inclination of the landing surface. To meet these constraints, the quadrotor will be required to perform an agile landing maneuver. Furthermore, the quadrotor will also have to estimate its attitude, position and velocity towards the platform during the landing maneuver. Due to the agile landing maneuver, landmark-based localization of the quadrotor will be more difficult, since these methods require the quadrotor's on-board camera to be directed at a certain landmark which can be used for guidance. Current methods for autonomous inclined landing either use external sensors or a landing mechanism to deal with the presented challenges.

During this project, an algorithm is developed which can estimate the quadrotor's attitude, position and velocity during the landing maneuver, while using only on-board sensors. State estimations are generated using two sources: a landmark-based localization algorithm and a Visual-Inertial Odometry (VIO) algorithm. The landmark-based localization algorithm uses markers placed near the landing surface to determine the quadrotor's attitude and position relative to the landing platform. Estimations from these two systems are fused by an Extended Kalman Filter (EKF). Furthermore, we train a policy network in a deep reinforcement learning approach for control of the quadrotor during the landing maneuver. We use a field-of-view constraint during the training of this policy network to keep markers used by the localization algorithm in sight of the quadrotor's on-board camera sensor during the landing maneuver.

During a series of experiments in the Gazebo simulator, we validate performance of the state estimation system during the inclined landing maneuver. We show that the marker localization algorithm's performance is improved by implementing a field-of-view constraint during the training of the policy network. We also show that state estimation by the EKF outperforms the two individual state estimation algorithms. In the Gazebo simulator, the quadrotor is able to use the state estimation system to land without the use of external sensors.

Preface

This thesis was written as finalization of the Master Robotics at TU Delft. I would especially like to thank my supervisor, Prof. dr. R. Babuska, for all his support and extensive help during the project. I would also like to thank my daily supervisor, Bas van der Heijden for his input. Furthermore, I would like to thank Dennis Benders and Thijs Niesten for their technical help.

Contents

1 Introduction

To achieve fully autonomous landing in a versatile range of environments, on-board state estimation is an important step. Outside of controlled lab environments, external sensors will often not be available. By relying solely on on-board sensors, the quadrotor will be able to land anywhere, increasing utility and safety. Autonomous landing ensures the quadrotor can execute safe landings when battery levels are low or when operator connection is lost. Additionally, the quadrotor could be used for a variety of autonomous tasks, such as reconnaissance, search and rescue, infrastructure inspection or delivery services. These areas would greatly benefit from the quadrotor being able to land autonomously on any surface.

Using on-board state estimation during the landing maneuver means that the quadrotor will not rely on any external sensors, such as an external motion capture system, during the landing maneuver. The quadrotor will only use on-board sensors to find its attitude, position and velocity relative to the landing surface. Although external sensors are often used for precise localization of the quadrotor in controlled lab environments, their usage limits utility of the quadrotor, as the quadrotor can only be localized whenever external sensors are available.

There are challenging aspects to autonomous inclined quadrotor landing. To land safely, the quadrotor can only approach at slow velocity and at a landing angle similar to the inclination of the landing surface. These constraints will require the quadrotor to perform a complicated flight maneuver. This agile flight maneuver will increase the difficulty of on-board state estimation, since sensor measurements will be affected by the extreme movement during flight.

For our application, the state estimation has to be accurate under extreme flight behavior. Although some methods, such as VIO or some landmark-based localization methods, notably used in drone racing, are able to provide accurate estimations under extreme flight behavior, the application of drone racing requires less accurate state estimation than inclined quadrotor landing. In this thesis, we develop a system able to provide accurate localization during agile flight behavior.

Apart from on-board state estimation, the quadrotor will also need a control system to perform the landing maneuver. The control method used during this thesis to control the quadrotor during the landing maneuver is based on prior research (Kooi and Babuška 2021). A policy network is trained to control the quadrotor during the landing maneuver using a reinforcement learning approach. The benefit of using a policy network to control the quadrotor over other control methods such as MPC or PID controllers is that these methods fall short when more complicated flight maneuvers are required.

1.1. Thesis objective

The objective of this thesis is thus to develop an on-board state estimation system capable of estimating the quadrotor's position, attitude and velocity relative to the landing surface. This system should only

use sensors which are mounted on the quadrotor and be robust enough to allow for landing under varying conditions. The information provided by the state estimation system should be usable by the on-board control system to conduct the landing maneuver. This goal is considered achieved if the on-board state estimation system is able to consistently and accurately measure the quadrotor's state during the landing maneuver, allowing the quadrotor to safely land on the landing surface.

1.2. Contributions

- Firstly, partial system identification for the Mantis quadrotor is performed. Several experiments are conducted for which the results can be used to model the quadrotor's behavior during the landing maneuver in the simulated environments.
- Secondly, a state estimation system is created which can estimate the quadrotor's position, velocity and attitude relative to the landing surface. These estimations can then be used by the trained policy network to control the quadrotor during the landing maneuver.
- Thirdly, the existing reinforcement learning approach for training a policy network for control of the quadrotor is adapted to work with the Mantis quadrotor and to work in combination with the state estimation system. The largest contribution to the existing policy training framework is the addition of a field-of-view constraint, which keeps markers near the landing surface in the field-of-view of the quadrotor's camera sensor for as long as possible, improving the accuracy of state estimations.
- Furthermore, a method is developed to integrate the PX4 controller software with the policy network used for control of the quadrotor. This is an important step, since the PX4 controller software has a large effect on flight behavior.
- Finally, experiments are conducted in the Gazebo simulator to validate the state estimation system's performance during the landing maneuver.

1.3. Related works

This section will give an overview of research related to the autonomous inclined landing task of this thesis.

Autonomous inclined quadrotor landing is a widely researched subject. State estimation during the inclined landing maneuver has been performed using different on-board sensors. Kim et al. (2021) use a depth camera to estimate the slope and position of the landing platform. To land the quadrotor on the platform, they use extendable landing skids. This simplifies the landing task as the quadrotor does not have to perform an agile flight maneuver and is able to keep the landing platform within the field-of-view of the quadrotor's depth camera at all times. Lesak et al. (2022) use multiple radar sensors mounted on the quadrotor to estimate the landing platform's slope and position. Extendable landing skids are also used during this research. Dougherty, D. Lee, and T. Lee (2014) use lasers attached to the quadrotor to find the inclination of the landing platform. The quadrotor's CMOS camera detects the intersection of the lasers with the landing platform. Using three different lasers, Dougherty, D. Lee, and T. Lee (2014) are able to identify the inclination angle and direction of the landing platform.

Another subject very relevant to the research topic of this thesis is the field of autonomous drone racing. In autonomous drone racing, the goal is to fly through a series of racing gates as fast as possible. To fly through the racing gates safely, the quadrotor will have to detect the relative pose of the racing gates. This is a similar task to finding the relative pose of a landing platform. Different techniques are used to detect the pose of the racing gates. Kaufmann, Gehrig, et al. (2018) directly estimate the pose of the racing gate using a neural network which is trained using a combination of simulated and real world images of the racing gate. Kaufmann, Bauersfeld, et al. (2023) use an algorithm which only detects the corner image coordinates of the racing gate. A plane fitting algorithm is then used to detect the pose of the racing gate relative to the quadrotor. Furthermore, to improve state estimations, these estimations are fused with state estimations from a Visual-Inertial Odometry (VIO) algorithm using an Extended Kalman Filter (EKF).

Drone gap traversal is the task of flying a quadrotor through a narrow gap. This maneuver often requires agile flight behavior. To control a quadrotor during the gap traversal maneuver, Lin et al. (2019) use a policy network trained in a dynamically simplified environment. The policy network outputs a thrust and attitude command to the quadrotor's PX4 control software. The policy network used during this thesis works in the same manner. To deal with sim2real transfer, Gaussian noise is added to the quadrotor's state for every timestep during training. During real world flight, the quadrotor's state is provided by an external motion capture system. Xie et al. (2023) also perform gap traversal using partial on-board sensing. They detect the gap's location through depth images. During real-world experiments, the quadrotor's position is still given by an external motion capture system.

Autonomous drone landing is also often performed through the use of a perching mechanism. An additional mechanism is mounted to the quadrotor which can attach itself to the landing surface. This increases the range of final states the quadrotor can have for a successful landing, simplifying the landing maneuver and allowing for landing in extreme environments. Mao et al. (2023) use a perching mechanism to land a quadrotor on an inclined surface of up to 90 degrees. They use a field-of-view constraint to keep the landing target within sight of the quadrotor's camera. The landing is performed using only the on-board camera and IMU for state estimation.

To conclude, a lot of research has been done on autonomous inclined quadrotor landing, on-board state estimation and reinforcement learning for quadrotor control, although these have not been combined yet to achieve the goal of this thesis. Most of the research done on autonomous inclined quadrotor landing still uses either an external motion capture system for state estimation, or uses a landing mechanism to simplify the required landing maneuver. Although on-board state estimation is performed in a similar manner in autonomous drone racing, flying through a racing gate is a less constrained maneuver compared to inclined quadrotor landing, as the quadrotor can fly through a racing gate at high velocity and different angles. This also holds for the research done on gap traversal. Research done on drone gap traversal is also often done using an external motion capture system for quadrotor localization. Although research on drone perching has achieved autonomous landing on an inclined surface using only on-board sensors and computing, the perching mechanism allows for more extreme approach velocities and angles since the quadrotor can attach itself to the landing surface.

2

On-board state estimation algorithm

This section will explain the on-board state estimation algorithm used to estimate the quadrotor's position during the landing maneuver. The state estimations which this system provides are used by the trained policy network to control the quadrotor during landing. The control policy network is further described in Chapter 3.

2.1. System overview

As described in Chapter 1, the state estimation algorithm provides an estimate of the quadrotor's position relative to the landing surface during the landing maneuver. The pitch and velocity of the quadrotor are also estimated. To accomplish this, the state estimation system uses the quadrotor's on-board sensors, such as the camera and inertial-measurement unit (IMU). Based on the estimated position, velocity and pitch of the quadrotor, the policy network will then send a commanded thrust and a commanded pitch value to the quadrotor's control software. The PX4 controller software will change this thrust and pitch command into direct actuation of the quadrotor's motors. Figure 2.1 provides an overview of this system.

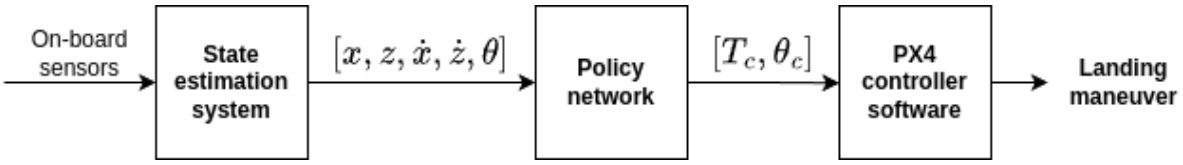


Figure 2.1: General system overview.

The state estimation system created during this thesis consists of three main parts. Two perception subsystems, a marker localization algorithm and a Visual-Inertial Odometry algorithm (VIO), estimate the state of the quadrotor using the quadrotor's on-board sensors. State estimations of these perception subsystems are then fused using an Extended Kalman Filter (EKF).

Two perception systems are used for state estimation instead of one because both systems are not able to provide sufficiently accurate state estimation individually. The marker localization algorithm provides accurate position and attitude measurements but at irregular intervals. The VIO algorithm provides state measurements at high frequency but suffers from accumulating errors over time. By fusing the estimations of the two perception systems with an EKF, we can combine their strengths for more accurate state estimation.

2.2. Marker localization algorithm

The first of the two perception-based state estimation algorithms works by detecting markers placed at known locations. These are used to localize the quadrotor relative to the landing platform. This provides an accurate estimate of the quadrotor's position and orientation. As can be seen in Figure 2.2, the marker localization algorithm uses camera images to find the transformation from the quadrotor to the marker, ${}^{Q}\mathbf{T}_{M}$. The transformation from marker to landing surface, ${}^{M}\mathbf{T}_{P}$, is given as ground-truth information. With these two transformations, the transformation from platform to quadrotor ${}^{P}\mathbf{T}_{Q}$ is found. Technical implementation details of the algorithm can be found in Section 6.2.1.

As the quadrotor flies towards the landing platform in an agile flight maneuver, the camera loses sight of the marker. This was solved by implementing a field-of-view constraint as described in Section 3.3.

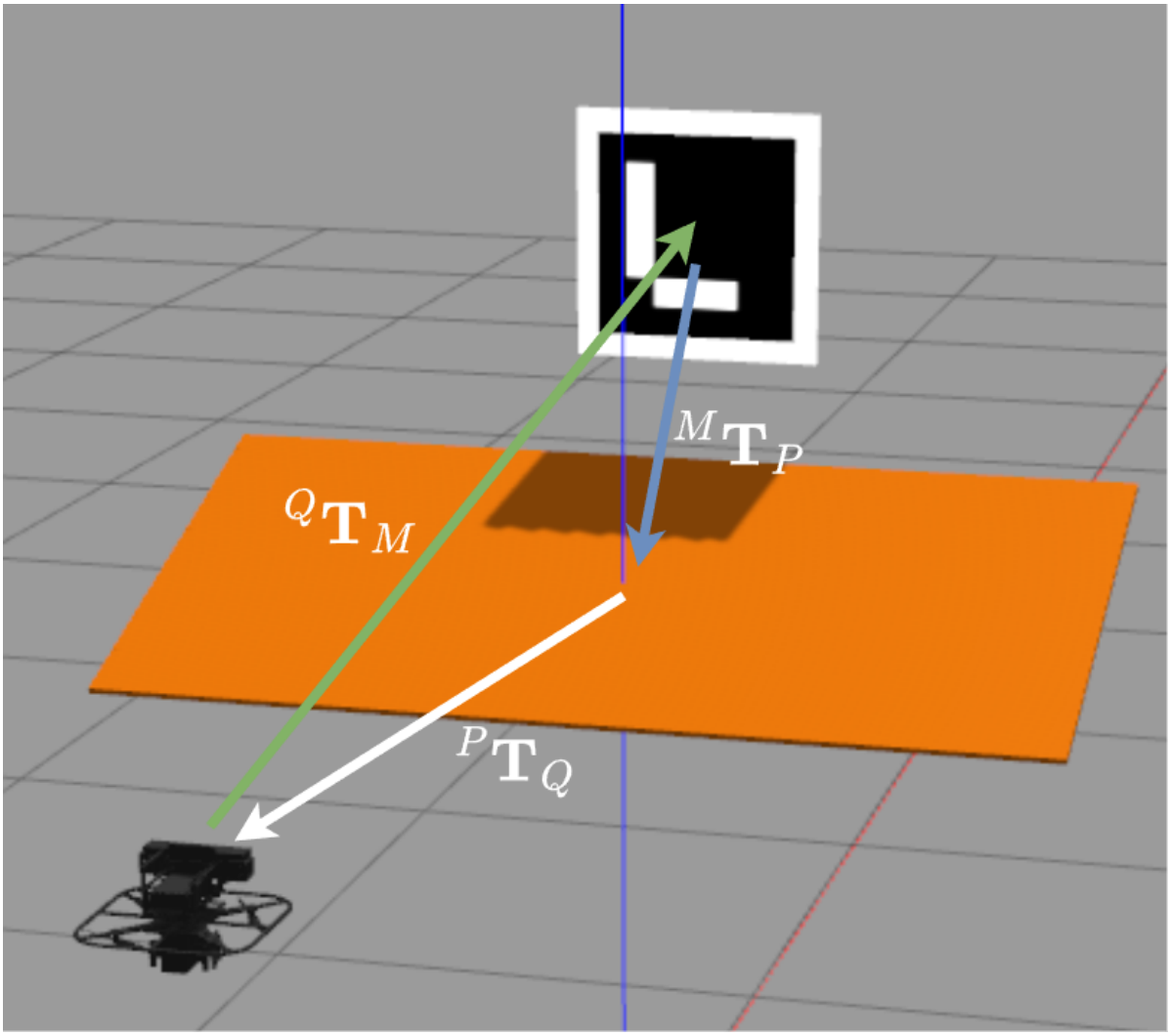


Figure 2.2: Overview of the marker localization algorithm.

2.3. Visual-Inertial Odometry

The second perception subsystem is a VIO algorithm, which is widely used for on-board state estimation of quadrotors. VIO combines camera and IMU measurements to provide accurate state estimation. By tracking image features across consecutive images, a camera can detect changes in the quadrotor's position. Because the state estimation by feature tracking is dependent on the quality of images, poor lighting conditions or motion blur will decrease performance of the algorithm. By fusing the estimations from feature tracking with those of the Inertial Measurement Unit (IMU), which provides state

estimations at a much higher frequency and does not depend on camera images, VIO algorithms can provide accurate state estimations. Technical implementation details of the VIO algorithm can be found in Section 6.2.2.

2.4. Extended Kalman Filter

The state estimations have to be fused in such a way that they give a continuous state estimate to the policy network. Without accurate and continuous state estimations, the policy network will not be able to successfully land the quadrotor.

To fuse the state estimations from the VIO and marker localization algorithms an Extended Kalman Filter (EKF) is used. The fused state estimations from the EKF are sent to the control policy network. As shown in Figure 2.3, the VIO algorithm provides a more detailed state estimation than the marker localization algorithm. The VIO algorithm provides position, linear and angular velocity, and pitch estimations. The marker localization algorithm only provides position and pitch estimations.

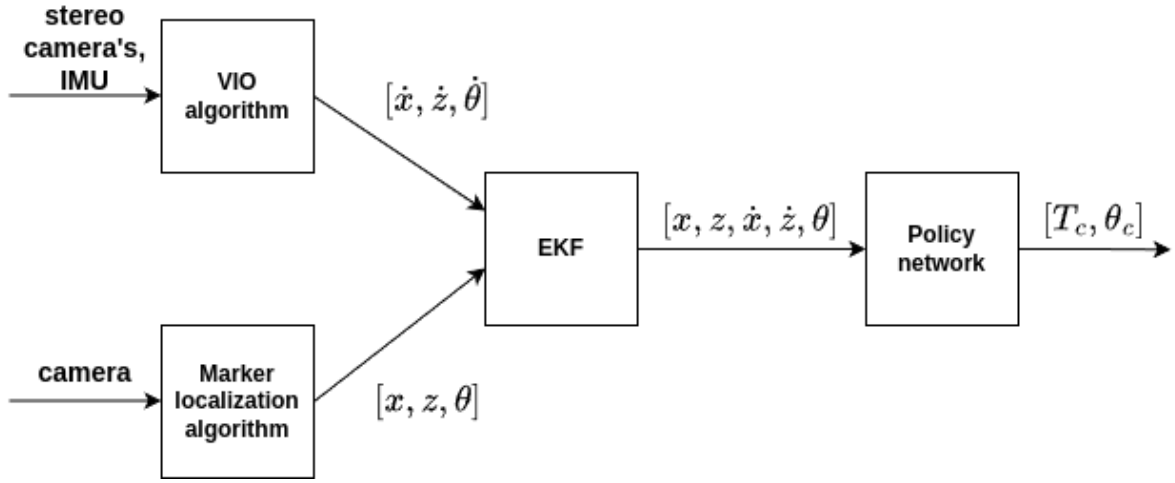


Figure 2.3: Overview of the state estimation system.

The EKF can handle varying amounts of estimated states from different input sources. We can therefore selectively use the most reliable state estimations from each source. Because the marker localization algorithm gives very accurate position and attitude estimations, its position and pitch estimations are fed into the EKF. Using two different sources with direct position estimates causes the EKF to oscillate between the positions provided by each source. Therefore, only the velocity estimates from the VIO algorithm are used. The VIO estimations are continuous, in contrast to the marker localization algorithm's state estimations. Therefore, the EKF uses them to accurately estimate the quadrotor's state when there are no markers in sight of the quadrotor's camera. The EKF uses a standard kinematic model to estimate the quadrotor's position using velocity data when the marker localization algorithm is not sending estimations. Technical implementation details of the EKF can be found in Section 6.2.3.

3

Control policy network

This section will explain how a trained policy network is used to control the quadrotor during the landing maneuver. Adaptations to the existing framework by Kooi and Babuška (2021) will also be described.

3.1. Overview of the policy network training approach

To train the quadrotor control policy network, a simplified simulation environment is used which simulates the quadrotor's response to actions from the policy network using a model of the quadrotor's dynamics.

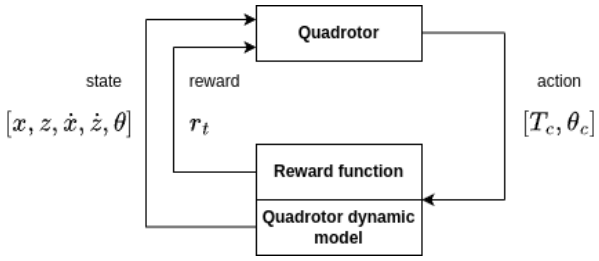


Figure 3.1: Overview of the policy network training process.

Figure 3.1 shows the reinforcement learning approach used to train the policy network. The control policy network outputs a commanded thrust and pitch which are used to update the quadrotor's state through the dynamic model. A reward is provided based on the state to guide learning. Each training episode stops when the quadrotor reaches the goal state or the maximum timesteps are reached. This process repeats until the policy network reliably lands the quadrotor on the inclined surface.

The environment used to train the policy network was adapted from research by Kooi and Babuška (2021). The Crazyflie nano quadrotor was used in their research. This project uses the Mantis quadrotor instead, as it is equipped with better sensors for on-board state estimation. Because we use the Mantis quadrotor, a different dynamical model has to be used to train the policy network. This is further explained in Section 3.2.1. Technical implementation details of the reinforcement learning approach can be found in Section 6.3.1.

When flying the real Mantis quadrotor, the system works differently from Figure 3.1. Because the control policy network does not output direct individual motor actuation commands, quadrotor controller software is used to map the thrust and pitch commands to actuation of the quadrotor motors. An overview of this is given in Figure 3.2. Although the PX4 controller software seems like an arbitrary part of the control loop, it plays an important role when identifying a dynamic model of the quadrotor. The PX4 controller software is very complex and cannot be modeled directly into the training environment.

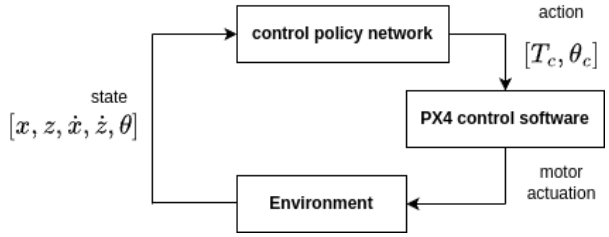


Figure 3.2: Real world control loop.

A more detailed explanation of how the policy network was integrated with the PX4 controller software is given in Section 6.3.4.

3.2. Training process

This section will describe the techniques used to train the policy network.

3.2.1. Simulation model

During the research by Kooi and Babuška (2021), the dynamic model used to model the CrazyFlie was identified in different research by Förster (2015). For the Mantis quadrotor a new dynamic model has to be identified. Because the Mantis quadrotor works with PX4 software, the steps by Förster (2015) for system identification cannot be repeated directly as the PX4 controller software influences the results of these experiments. Section 6.3.4 explains how we were able to identify a dynamic model which can be used in combination with PX4. For details on the system identification process used for the Gazebo simulator, see Section 6.6.3.

The quadrotor state inside the training environment is defined as follows:

$$s = \begin{bmatrix} x & z & v_x & v_z & \theta \end{bmatrix}^T \tag{3.1}$$

The quadrotor dynamics are modeled as follows:

$$\begin{bmatrix} \ddot{x} \\ \ddot{z} \end{bmatrix} = R \cdot \begin{bmatrix} 0 \\ T \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \tag{3.2}$$

Where R is the rotation matrix from body to world frame and T is the acceleration thrust in body frame. The change in thrust acceleration is given by:

$$\dot{T} = A_t \cdot T + B_t \cdot T_c \tag{3.3}$$

The change in pitch is given by:

$$\dot{\theta} = A_\theta \cdot \theta + B_\theta \cdot \theta_c \tag{3.4}$$

Since the landing maneuver is performed in a 2D plane, we do not model yaw and roll behavior or movement in the Y plane. The coefficients for the thrust and pitch equations are derived by simple flight experiments as explained in Section 6.6.1.

3.2.2. Reward function

The reward function used during training is defined as follows:

$$r_t = \begin{cases} 0 & \text{if} \quad s \in S_g \\ -6 & \text{if} \quad s \in S_o \\ -2 & \text{if} \quad s \in S_b \\ -1 & \text{otherwise.} \end{cases} \tag{3.5}$$

Where S_g is the set of goal states, S_o the set of obstacle states belonging to the landing platform, and S_b is a set of boundary states.

The set of goal states is defined as follows:

$$S_g = \{s || s_i - s_{g,i} | < \delta_{g,i}, \forall i\} \tag{3.6}$$

Where $\delta_{g,i}$ is a set threshold distance.

A simple euclidean distance based reward function cannot be used because the quadrotor does not follow a straight trajectory towards the landing goal due to the complicated landing maneuver.

3.2.3. Curriculum learning approach

Due to the sparse reward function, extensive exploration is required for successful training. A curriculum learning approach is therefore used to improve the training process. During training, the difficulty of the task is incrementally increased. The inclination of the landing platform and the starting distance from the goal are increased with small increments after each episode or timestep. The range of goal states is incrementally decreased. The field-of-view constraint discussed in Section 3.3 only actives after a large amount of timesteps.

3.3. Adaptations to existing framework

While the existing framework remains largely unchanged, several adaptations were made. The dynamic model used to model the quadrotor's behavior was changed, since a different quadrotor was used during this research. Another addition to the existing framework is the field-of-view constraint, which will be explained in this section.

Because the marker localization algorithm requires markers to be in the camera's field-of-view, it is important that the quadrotor's camera is directed towards the markers for as long as possible. To facilitate this, a field-of-view constraint was added to the existing reward function. This field-of-view constraint rewards behavior where the quadrotor's axis perpendicular to its rotors is directed towards the marker. When the policy network is trained in this manner, during inference, the quadrotor's camera will be directed for a longer period towards the marker, allowing for improved state estimation. An explanation for the implementation of this constraint is given below.

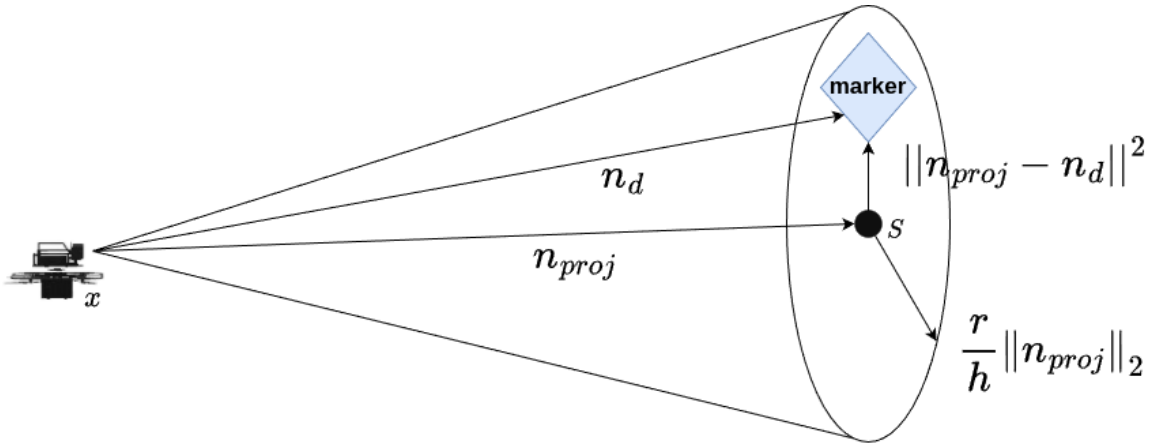


Figure 3.3: Field-of-view constraint.

In Figure 3.3, S is the position of the landing platform marker and x the position of the quadrotor. $\mathbf{n}_d$ is the vector from the quadrotor to the target. $\mathbf{n}_{\text{proj}}$ is the projection of $\mathbf{n}_d$ on the unit vector projected from the camera's center. The constraint is formulated as $\|\mathbf{n}_d - \mathbf{n}_{\text{proj}}\|_2 \leq \frac{r}{h} \|\mathbf{n}_{\text{proj}}\|_2$, where $\frac{r}{h} \|\mathbf{n}_{\text{proj}}\|_2$ is the radius of a cone at distance $\mathbf{n}_{\text{proj}}$ from the camera. During training, the policy network receives a negative reward when the constraint is not satisfied. The field-of-view constraint is relaxed when the quadrotor nears the target. This is done to prevent the constraint from hindering the landing maneuver.

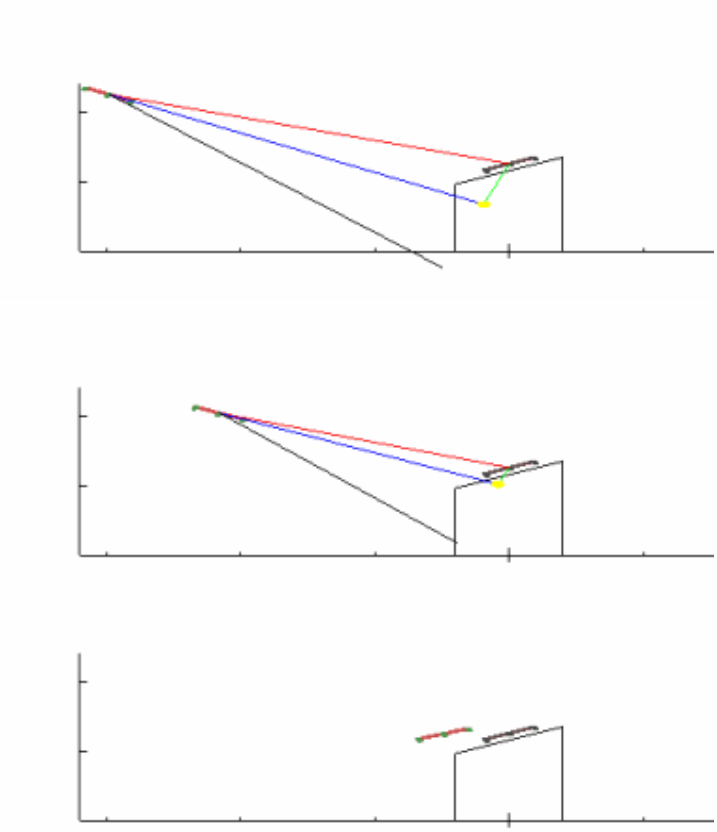


Figure 3.4: Rendering of the policy network being trained while the field-of-view constraint is active. As the quadrotor nears the landing target, the field-of-view constraint is relaxed in order to not hinder the landing movement.

Figure 3.4 shows a rendering of the quadrotor in the training environment while the field-of-view constraint is active during training. The red quadrotor is the actual quadrotor being controlled, the black quadrotor resembles the goal position. The field-of-view constraint is also rendered during training. The blue line resembles $\mathbf{n}_{\text{proj}}$. The red line resembles $\mathbf{n}_d$. The grey line resembles the maximum pitch the quadrotor can have before breaking the constraint. The green line resembles the distance from the projected center of the quadrotor's camera to the goal $||n_{proj} - n_d||^2$.

4

Results

This section presents the different experiments conducted to validate the performance of the state estimation system. The goal of these experiments is to validate the performance of the full state estimation algorithm described in Chapter 2.

Several flight experiments are conducted. Firstly, we show the difference in marker detection when the policy network is trained with the field-of-view constraint and without in Figure 4.2. To show the effect of the field-of-view constraint on the quadrotor's flight behavior and to give a general overview of how the quadrotor flies during the landing maneuver, we also plot the quadrotor's pitch and velocity during the landing in Figure 4.3. This flight is performed with ground-truth positional data. Secondly, we show the state estimations by the VIO algorithm. The results of this experiment can be seen in Figure 4.4. This flight is also performed with ground truth positional data. Thirdly, we show the fused state estimations by the EKF during the landing maneuver. During this experiment, the policy network only receives state estimations from the EKF and is not provided with ground truth positional data. The results of this experiment can be seen in Figure 4.5.

All experiments are conducted inside the Gazebo Software-In-The-Loop (SITL) simulator for PX4. More information on the simulation environment can be found in Section 6.5. The experiment setup can be seen in Figure 4.1. During flight experiments where the VIO algorithm is used, a background is added to facilitate feature detection by the VIO algorithm.

As explained in Section 3.2.2, the policy network is trained to land the quadrotor within the goal range. During these experiments and during training of the policy network, the goal range is set to 0.1. This means that the quadrotor will try to land the quadrotor within 0.1 meters in X and Z direction from the landing goal. Furthermore, the pitch constraint is set to 0.05 rad from the goal inclination. The policy network will thus try to land the quadrotor within these constraints in the Gazebo simulator. Once the quadrotor is within the goal range, the experiment is stopped.

All flights are performed while the quadrotor is in a hovering position, 3.5 meters behind the landing platform (X = -3.5) and 2 meters above ground level (Z = 2). For the marker localization algorithm, a single marker is placed on the opposite side of the landing platform, at a height of 1.8 meters (Z = 1.8) and 0.5 meters behind the platform (X = 0.5). The landing platform has a height of 1.15 meters (Z = 1.15) in the center, and is placed at X = 0. The landing platform has an inclination of 0.448 rad. The marker is placed at an inclination as this was found to improve detection.

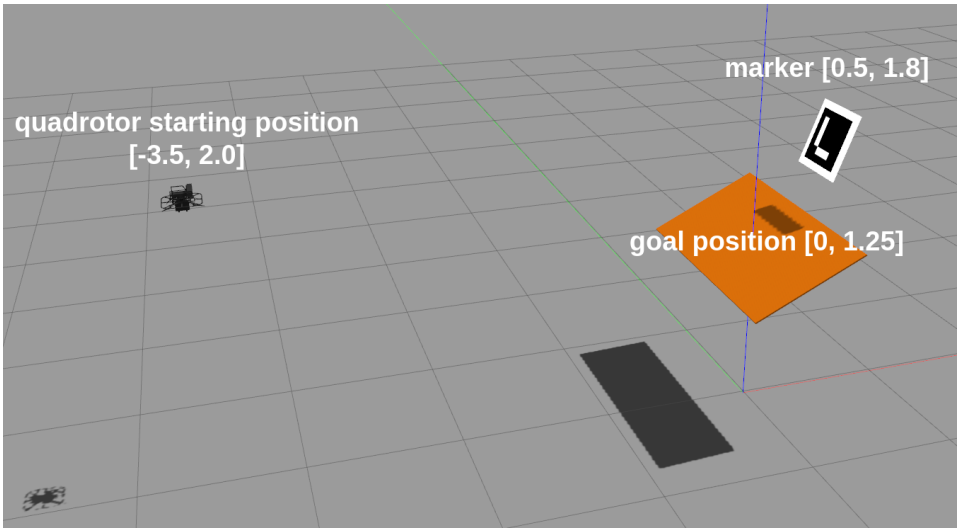
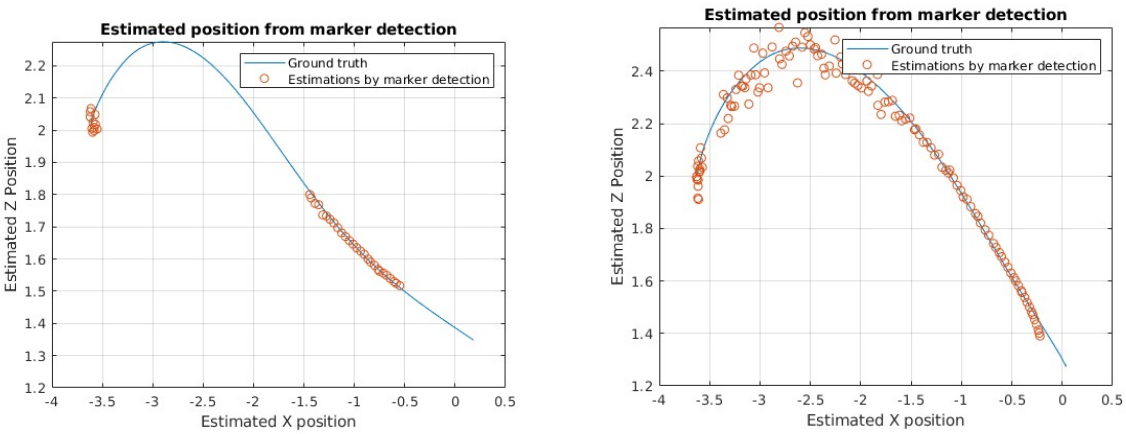


Figure 4.1: Overview of the experiment setup. X and Z coordinates are shown for the quadrotor starting position, goal position and marker.



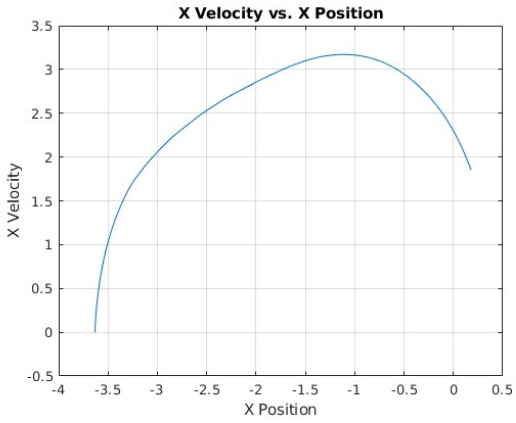
(a) Ground truth position (no field-of-view constraint).

(b) Ground truth position (with field-of-view constraint).

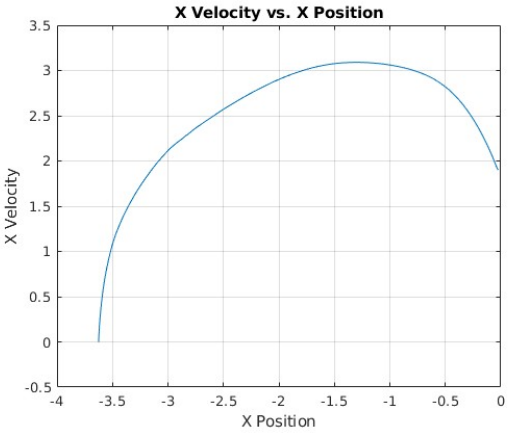
Figure 4.2: Localization by marker detection during the landing maneuver. One policy network is trained with field-of-view constraint, the other without. Landing maneuver is performed with ground truth data.

Figure 4.2 shows the increase in the frequency of marker detections when the field of view constraint is used during training of the policy network. The marker is now detected more often during the flight and not only near the beginning and end of the landing maneuver. Figure 4.2 shows that the quadrotor flies higher when controlled by the policy network which was trained with the field of view constraint. The increased height during the flight causes the marker, which is placed relatively high as well, to be in sight of the quadrotor's camera sensor for longer. Furthermore, comparing Figure 4.3e and Figure 4.3f shows that the policy network which is trained with the field of view constraint increases the quadrotor's pitch value slower towards the maximum pitch. This enables the marker to stay in sight of the field-of-view of the quadrotor's camera for longer as well.

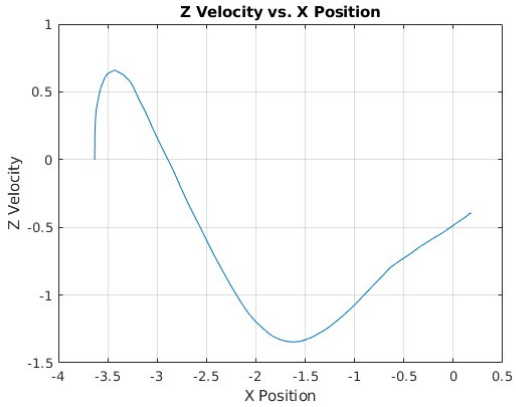
Figure 4.2 and Figure 4.3 also clarify that the transfer of the control policy network from the training environment to the Gazebo simulator and from the CrazyFlie to the Mantis quadrotor has not impacted performance of the policy network negatively. As can be seen, the control policy network lands the quadrotor within the goal range when trained with or without the field of view constraint. This also shows that the implementation of the field-of-view constraint has no negative effect on the ability of the policy network to land the quadrotor.



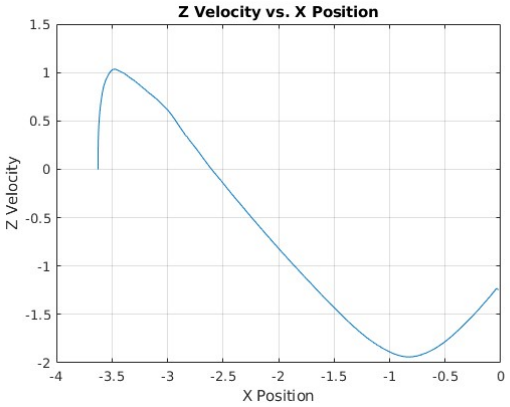
(a) Ground truth X velocity (no field-of-view constraint).



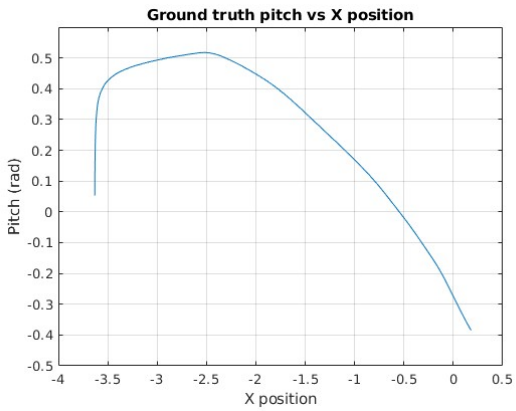
(b) Ground truth X velocity (with field-of-view constraint).



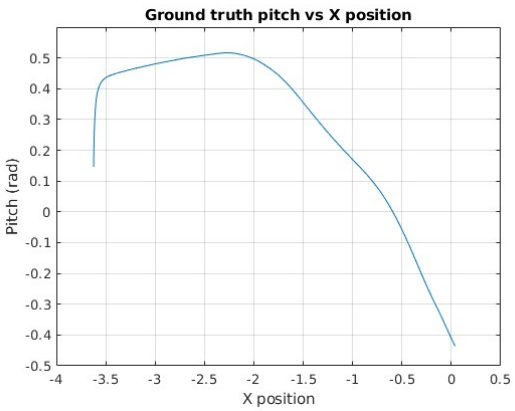
(c) Ground truth Z velocity (no field-of-view constraint).



(d) Ground truth Z velocity (with field-of-view constraint).

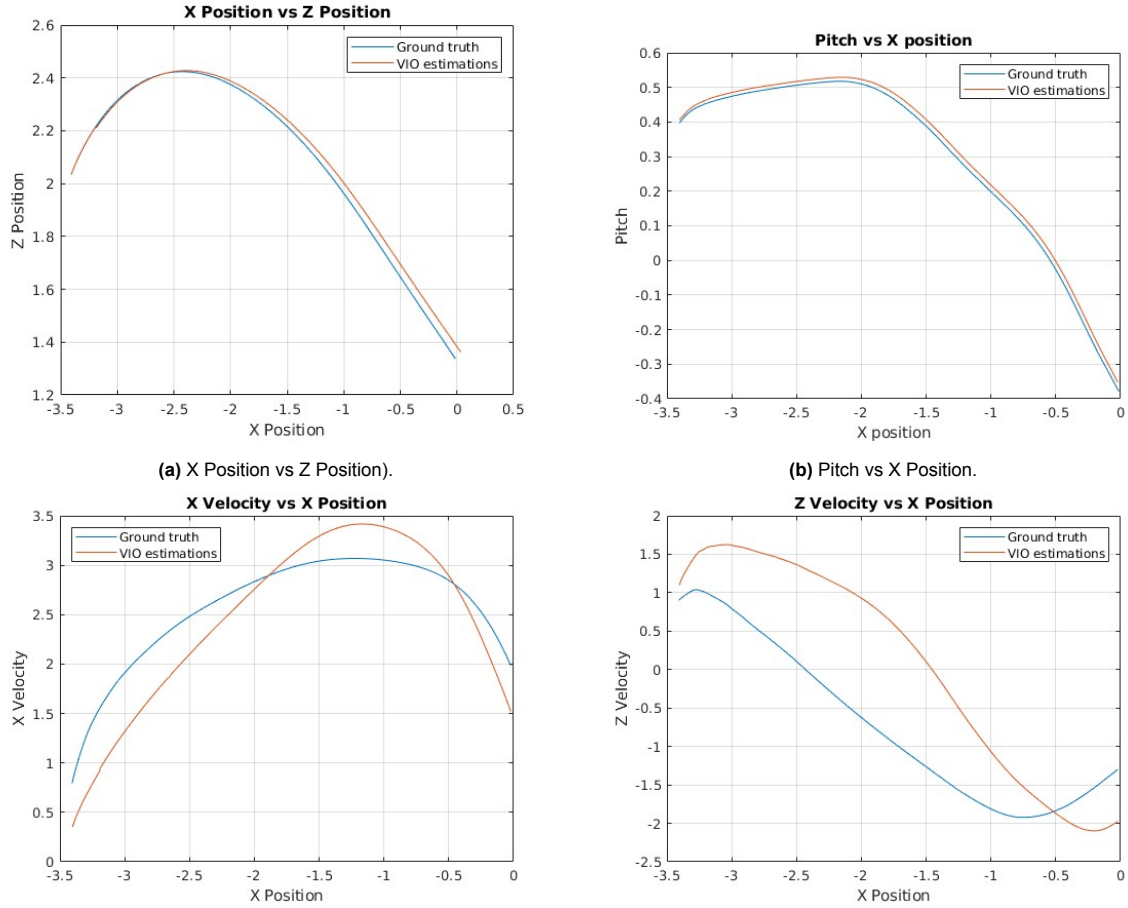


(e) Ground truth pitch (no field-of-view constraint).



(f) Ground truth pitch (with field-of-view constraint).

Figure 4.3: Velocity and pitch ground truth data during the landing maneuver of model trained with and without field-of-view constraint. Landing maneuver is performed with ground truth data.



(a) X Position vs Z Position).

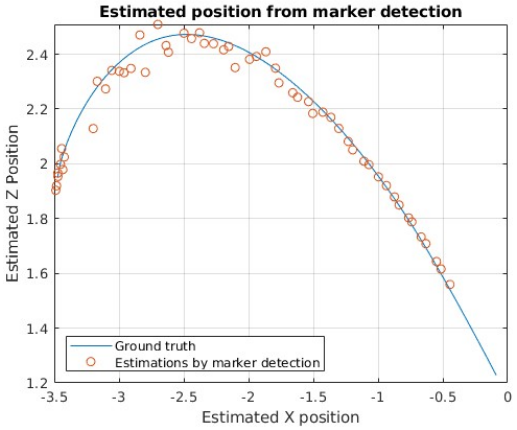
(b) Pitch vs X Position.

(c) X Velocity vs X Position.

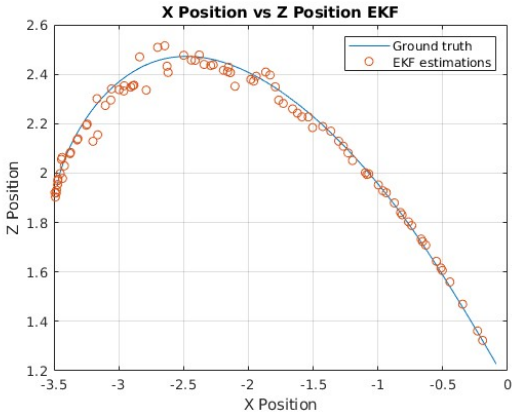
(d) Z Velocity vs X Position.

Figure 4.4: VIO state estimation during landing maneuver. Landing maneuver is performed with ground truth data.

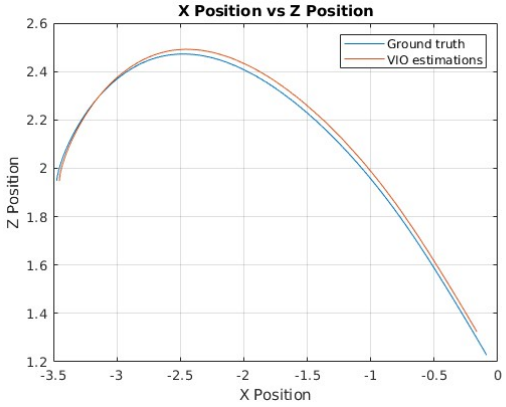
Figure 4.4 shows the state estimations by the VIO algorithm during the landing maneuver. As seen in Figure 4.4a, the VIO algorithm provides very accurate position estimations. A slight drift in position estimations of a few centimeters is seen as the landing maneuver progresses. Figure 4.4b demonstrates that the pitch estimations are also highly accurate. The velocity estimations in Figure 4.4c and Figure 4.4d are less accurate. Although the VIO position estimates seem accurate enough for landing the quadrotor, they often do not suffice during landing experiments. The control policy network adjust its output actions based on the VIO state estimates. However, as these estimates become increasingly inaccurate over time, the policy network outputs incorrect actions. These erroneous actions extend the time required for the quadrotor to reach its goal, causing the error in the VIO estimates to accumulate further. The marker localization algorithm does not suffer from increasing state estimation errors since the state estimations are not relative measurements. Furthermore, only using the VIO algorithm's state estimations decreases reliability, as the VIO algorithm's performance varies over multiple flight experiments.



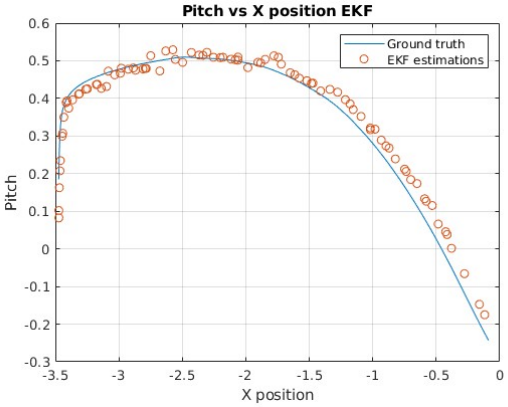
(a) Estimated Position from Marker Detection.



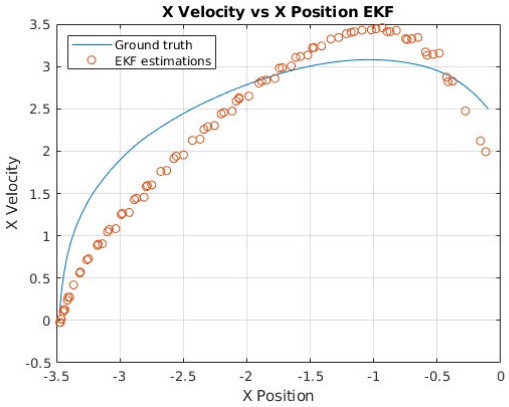
(b) Estimated Position from EKF.



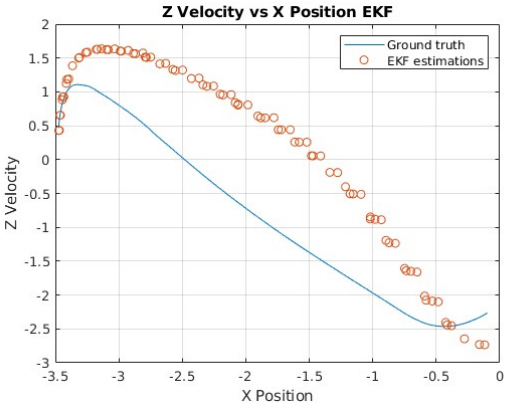
(c) Estimated Position from VIO.



(d) Estimated Pitch by EKF.



(e) Estimated X Velocity by EKF



(f) Estimated Z Velocity by EKF.

Figure 4.5: Localization during landing by EKF. No ground truth data is used during the landing maneuver.

Figure 4.5 shows the estimations from the marker localization algorithm, VIO algorithm and EKF during a flight where the policy network is not provided with ground-truth positional data but only with state estimations from the EKF. Comparing Figure 4.5a and Figure 4.5b shows that the position estimations by the EKF are more numerous than those by the marker localization algorithm. The EKF can send state estimations at a higher frequency because it also relies on measurements from the VIO algorithm. The VIO algorithm can send estimations at a higher frequency because it also integrates measurements from the IMU, which operates at a much higher frequency than the camera sensor. The EKF continuously publishes state estimations during parts of the flight when the marker localization algorithm is inactive, such as near the beginning and end of the landing maneuver, further demonstrating the improved state estimation. Figure 4.5e and 4.5f are the output velocity estimations by the EKF, which closely resemble Figure 4.4c and Figure 4.4d. This similarity is logical since only the VIO algorithm sends direct velocity estimations to the EKF. When the EKF receives no direct position updates, it updates the state estimations solely based on the velocity estimations through a kinematic model (Moore and Stouch 2016). As shown in Figure 4.5b, the velocity estimations are sufficiently accurate for the EKF to provide reliable position estimations for the policy network when direct position estimations are not available from the marker localization algorithm. Furthermore, the intervals where the EKF is dependent on only velocity estimations are relatively short.

To conclude, these experiments show that the EKF provides reliable estimations which allow the policy network to land the quadrotor on the platform. Individual perception subsystems do not provide sufficiently accurate estimations or at a high enough frequency. The EKF especially outperforms both individual systems near the end of the landing maneuver, which is the most critical phase.

5

Conclusions

5.1. Summary

The goal of this research was to develop a state estimation system which could be used to autonomously land a quadrotor on an inclined platform. To achieve this, we first performed partial system identification for the Mantis quadrotor. Using the quadrotor's CAD model, testbench experiment results, PX4 controller gains, and information from the component manufacturers, we created a dynamic model of the quadrotor for the Gazebo simulator.

This model was used to develop a state estimation system that uses the quadrotor's on-board sensors to estimate the quadrotor's position, attitude and velocity during the landing maneuver. By fusing the measurements from a marker localization and VIO algorithm through an EKF, accurate state estimation was possible.

Several adaptations to the existing policy network training framework were made. Firstly, because the Mantis quadrotor utilizes different controller software compared to the CrazyFlie nano quadrotor used during the previous research, a method was developed to integrate the control policy network with the PX4 controller software. Secondly, to improve state estimation accuracy, a field-of-view constraint was implemented into the existing reinforcement learning approach.

After integrating the state estimation system and the policy network with the PX4 controller software, a set of experiments were performed inside the Gazebo PX4 SITL simulator to validate the system's performance. These flight experiments demonstrated that the state estimation system outperforms both single perception algorithms and enables the control policy network to succesfully land the quadrotor without relying on external sensors.

5.2. Future work

There are several potential improvements for the current system.

Firstly, the marker localization algorithm could be replaced with an alternative landmark-based localization algorithm. Although the marker localization algorithm provides accurate localization, it requires markers to be placed near every landing surface. Using an algorithm which can identify possible landing surfaces without requiring pre-placed markers would significantly increase utility. Other research has used the depth camera sensor to detect the slope and position of a landing platform (Kim et al. 2021).

Secondly, the policy network can be improved upon. The reinforcement learning environment could be transferred to JAX (Bradbury et al. 2018) using Gymnax (Lange 2022). This transition might increase the training speed significantly. Additionally, the training method could be adapted to work with quadrotors that are less agile. This will require changes to the curriculum learning method. For example, if the

quadrotor has a slower pitch response, a succesful landing maneuver would require a longer approach distance. This would not work with the current curriculum learning approach, which gradually expands the initial horizontal distance to the landing platform. The quadrotor would not be able to reach the goal state if its initial state is close to the landing platform.

Thirdly, transferring the existing framework to the real world involves several steps. To complete the system identification process, real world flight experiments will have to be conducted to determine the coefficients for the thrust and pitch equations for the dynamic model. Additionally, to use the perception state estimation algorithms in the real world, the quadrotor's camera and IMU sensors will have to be calibrated. Because the entire framework was created to work on the PX4 SITL simulator, no major software changes are required to transfer the state estimation and control system to the real world.

6

References

6.1. Mantis quadrotor technical specifications

6.1.1. Physical attributes

Table 6.1: Physical Attributes

Attribute	Value
Mass	0.580 kg
Height	0.22 m
Width	0.18 m
Motor axis to quadrotor center distance	0.085 m

6.1.2. Quadrotor components

Table 6.2: Components List

Component	Description
Motors	BetaFPV 1506 3000KV Brushless Motors
Flight Controller	Holybro PX4 KakuteH7v2
Camera	RealSense D435i
Propellers	Gemfan 3052
Companion Computer	Jetson Nano / Xavier NX/TX2 NX
Carrier Board	Jetson A203 V2
Battery	TATTU R-LINE 1300MAH 22.2V 95C 6S LIPO ACCU

6.2. State estimation Algorithms

6.2.1. Implementation of marker localization algorithm

The marker localization algorithm is implemented as a ROS node. To detect the markers, the Aruco ROS library is used (Pal Robotics 2014). This library takes as input images from the quadrotor's camera and outputs marker detections in image frame. The marker localization node then uses the detected markers' coordinates in image frame and the known transforms from marker to platform ${}^{M}\mathbf{T}_{P}$ and uses these to estimate the quadrotor's position with regards to the landing platform ${}^{Q}\mathbf{T}_{P}$. Placing multiple markers could improve state estimation as the marker localization algorithm is able to detect multiple

markers at the same time. For this project, a single marker was sufficient due to the implementation of the field-of-view constraint. The input image used for the Aruco ROS package is the image given by

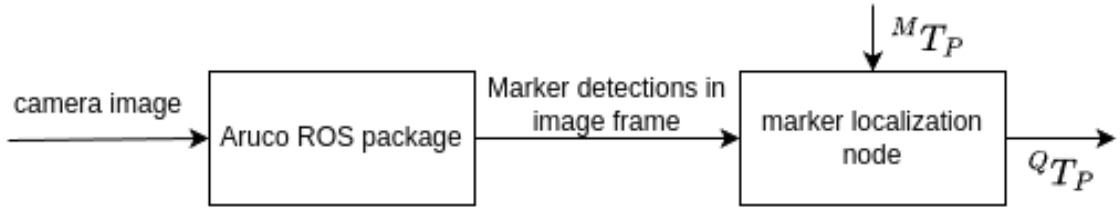


Figure 6.1: Overview of the marker localization system.

the left imager of the RealSense D435 camera. The maximum streaming rate is 90 Hz at a resolution of 848x480 pixels (Intel Corporation 2024). While simulating all sensors and algorithms in the Gazebo SITL simulation, hardware limitations can cause performance to decrease. To limit the computational power required for the marker detection algorithm, we use the same image as is already used by the VIO algorithm, and disable the streaming of depth and RGB images by the RealSense camera. When the software is used on the real Mantis quadrotor in the real world and computing is done on-board the Nvidia Jetson computing device, computational power will also be limited. Streaming images at a higher rate to the Aruco ROS package also allows for more accurate predictions. Although the RGB camera of the Realsense D435 can stream at higher resolutions, it does so at a much lower framerate of at most 30 Hz (Intel Corporation 2024).

6.2.2. Implementation of VIO algorithm

The OpenVINS open-source VIO algorithm (Geneva et al. 2020) is used to give state estimations through Visual-Inertial Odometry. As explained in section 6.2.1, limited computing power requires efficient use of the camera's streamed images. The OpenVINS algorithm expects sensor measurements from the IMU and camera images, which can be from a single camera or from multiple cameras. The most accurate localization performance was found by using the left and right imager cameras from the RealSense camera and giving these as stereo camera input to the OpenVINS algorithm. Although these stream at a lower resolution than the RealSense's RGB camera, they can be streamed at a much higher framerate, which was found to have a larger effect on the algorithm's performance. The images from the RealSense camera's left and right imager are streamed to the VIO algorithm at a frequency of 90 Hz. The VIO algorithm receives IMU measurements at a frequency of 250 Hz.

6.2.3. Implementation of Extended Kalman Filter

To fuse the state estimations from the marker localization and VIO algorithms, the Robot Localization package (Moore and Stouch 2016) is used. This is an existing software package often used for state estimation of robots. The package contains an implementation of an EKF which is easily tunable for specific applications such as quadrotor state estimation. Before measurements are sent from the perception subsystems to the EKF, a ROS node transforms them into the same coordinate frame.

Tuning the EKF is important for achieving accurate final state estimations. The EKF can be tuned by adjusting the process noise covariance matrix, initial estimate covariance matrix and the covariance matrices of the input state estimations from the two perception subsystems. Additionaly, The EKF can be tuned by only using a part of the state estimations provided by each perception subsystem.

The covariance matrix values from the state estimations from the marker localization algorithm are tuned to be much lower than those from the VIO algorithm's. This ensures that the EKF considers the marker localization algorithm's state estimations as more reliable.

For the process noise covariance matrix, values relating to the acceleration are set relatively low, since those are not being measured directly. The process noise covariance matrix is further fine-tuned through trial and error.

As mentioned in section 2, we only use part of the estimations made from each perception subsystem. Directly merging absolute pose measurements from both sources caused oscillations in the estimated position. The policy network is very sensitive to these oscillations, which causes failure during the landing maneuver. To prevent these oscillations, we only use the position and attitude estimations from the marker localization algorithm. From the VIO algorithm, we only use the angular and linear velocity components. The EKF relies on these measurements when the marker localization algorithm is not providing estimations.

The estimations from the marker localization algorithm and VIO algorithm are streamed at frequencies of around 90 Hz and 150 Hz respectively. The EKF outputs estimations at a frequency of 90 Hz. The implementation of the EKF for our application can be found on the cor-drone-dev Github repository.

6.3. Policy network

6.3.1. Implementation of reinforcement learning approach

The simulation environment used for the training of the policy network was developed during previous research by Kooi and Babuška (2021). The simulation environment is created in Python using the Python Gym[1] library. The original code for the simulation environment can be found on Github[2]. Training the policy network in the simulation environment takes around 90 minutes on a HP Zbook Studio G5 laptop with a Intel I7-8750H CPU and Nvidia Quadro P1000 GPU.

The Reinforcement learning approach was not directly implemented into the Gazebo SITL PX4 simulator because this simulator is too complex. The computational cost for simulating a single landing is much higher and there is no existing framework for running large amounts of training episodes. Furthermore, the added complexity of fully simulating the PX4 controller software will make training of the policy network much more difficult.

6.3.2. Implementation of field-of-view constraint

The field-of-view constraint is implemented into the existing simulation environment developed by Kooi and Babuška (2021). Code for the field-of-view constraint can be found on Github[3].

6.3.3. Implementation of the dynamic model

As explained in Section 6.6.1, we perform simple flight experiments to obtain the coefficients for the dynamic model used during the training of the policy network. After several flight experiments in the Gazebo simulator, the following coefficients are obtained for the dynamic model of the Mantis quadrotor:

$$
\begin{aligned}
A_t &= -7.4639 \\
B_t &= 7.4186 \\
A_\theta &= -8.3424 \\
B_\theta &= 8.444
\end{aligned}
$$

The code for the implementation of the dynamic model can be found on Github[3].

6.3.4. PX4 integration of policy network

As described in Chapter 3, the PX4 controller software is used to control the quadrotor during flight using the policy network. In the previous research on this subject by Kooi and Babuška (2021), the commanded thrust and pitch values from the policy network were sent directly to the CrazyFlie's onboard controller. This approach does not work with the PX4 controller.

A commanded thrust and pitch value can be sent to the PX4 controller by publishing an AttitudeTarget[4] message to MAVROS. The commanded thrust value is a value between 0 and 1. These values are

[1] https://github.com/openai/gym/
[2] https://github.com/Jacobkooi/InclinedDroneLander
[3] https://github.com/cor-drone-dev
[4] https://docs.ros.org/en/noetic/api/mavros_msgs/html/msg/AttitudeTarget.html

then sent to the PX4 controller software's Attitude Controller[5]. The PX4 controller software eventually uses the commanded pitch and thrust values to send individual motor commands.

The issue with sending the AttitudeTarget message is that the PX4 software does not provide a consistent thrust output for a given thrust input command. This is because the output thrust is also dependent on the quadrotor's battery levels. In order to succesfully land the quadrotor and create a dynamic model which can be used to train the policy network, A mapping is required to ensure a constant thrust output from an input thrust command.

To accomplish this, the reinforcement learning approach is changed such that the policy network outputs an acceleration setpoint in the body frame of the quadrotor. The PX4's acceleration controller is then modeled to get the normalized thrust command from the acceleration setpoint. The normalized thrust command can then be sent through the AttitudeTarget message to MAVROS, which in turn sends the commanded pitch and thrust to the PX4 controller. The modeled PX4's acceleration controller uses the "hover thrust estimate", an estimate of the normalized input thrust command which is required for the quadrotor to hover. PX4 continuously calculates this value based on battery levels.

By using the "hover thrust estimate" to map an acceleration setpoint to the normalized thrust setpoint which can be sent to PX4, we can ensure that the PX4 controller always follows the commanded acceleration setpoint.

6.4. Software setup

this section will explain the complete software setup used to control the quadrotor.

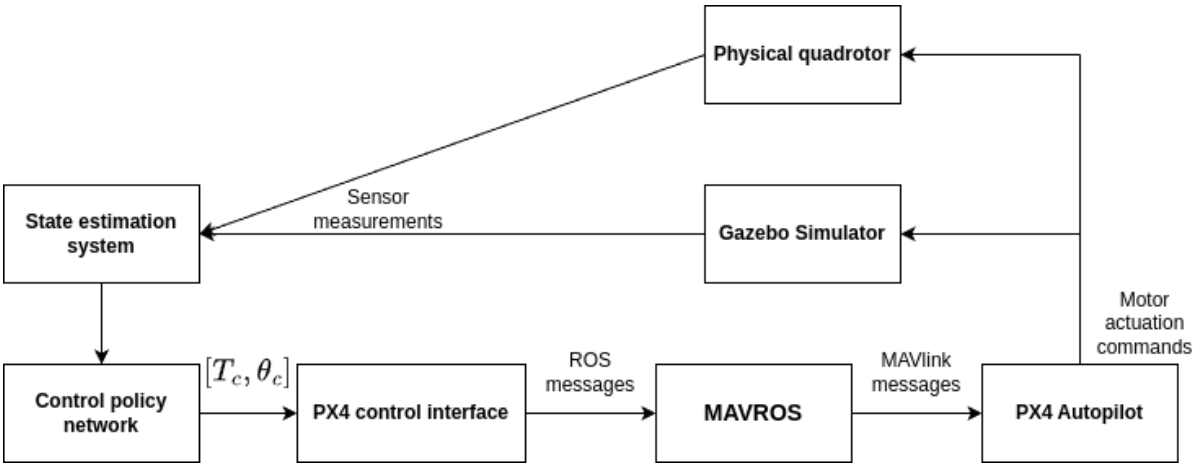


Figure 6.2: Overview of the complete software setup.

Figure 6.2 shows the complete software setup used during the landing maneuver. The state estimation system consists of multiple ROS nodes which send state estimations in the form of ROS messages to the control policy network.

The control policy network is also implemented as a ROS node. Because the entire code is written in C++, we have to convert the trained policy network from the training environment to a TorchScript model[6]. The TorchScript file can be used directly with C++ through LibTorch, which is a Torch distribution for C++.

[5] https://docs.px4.io/main/en/flight_stack/controller_diagrams.html
[6] https://g-airborne.com/bringing-your-deep-learning-model-to-production-with-libtorch-part-2-tracing-your-pytorch-model

The PX4 control interface[7] provides a control interface which can interface any control algorithm with the PX4 controller software through MAVROS. The PX4 control interface is used here to provide an interface between the control policy network and MAVROS.

MAVROS[8] provides an interface between ROS and the PX4 autopilot controller software. MAVROS converts ROS messages to MAVLink messages which can be sent to the PX4 controller software.

6.5. Gazebo SITL simulation

Gazebo is a widely used simulator for robotics, featuring a realistic physics engine. During this thesis Gazebo was primarily used for two reasons: the ability to simulate sensors such as the IMU and RealSense camera on the quadrotor, and the ability to fully simulate the quadrotor's PX4 Controller Software using Software-In-The-Loop (SITL) for PX4. Real world flight testing is very time-consuming. By using SITL for PX4, accurately simulating the complete software used on the real Mantis quadrotor is possible. This allows for realistic simulations without expensive real-world testing. In the Gazebo simulator, we use the realsense-gazebo-plugin[9] to fully simulate the RealSense camera inside the Gazebo environment. To simulate the quadrotor dynamics, we use the widely used Gazebo RotorS model (Furrer et al. 2016). All code for the simulation is available on the cor-drone-dev repository.

6.6. System identification

6.6.1. Flight experiments

To find the coefficients of the thrust and pitch equations, we conduct simple step experiments where a setpoint thrust or pitch is sent to the PX4 control software. We then fit the pitch and thrust equations on the flight data using Matlab's nlgreyest function. For the real Mantis quadrotor, the same flight experiments can be conducted. The Matlab scripts used to analyze the flight data rosbags can be found on Github[7].

[7] https://github.com/cor-drone-dev
[8] https://github.com/mavlink/mavros
[9] https://github.com/pal-robotics/realsense_gazebo_plugin

6.6.2. Testbench experiments

As part of the system identification process, two experiments were conducted where the quadrotor was mounted on a testbench. One experiment measured the quadrotor's thrust in Newton based on the input AtttitudeTarget message to the PX4 controller software. The other experiment determined the rotational velocity of the quadrotor's motors based on the input AttitudeTarget message. Figure 6.3 and 6.4 show that for an input command of 0, the output thrust is not equal to zero. This is because the quadrotor's motor's will activate to a low initial rotational velocity if the PX4 control software is set to "offboard control" mode. During the experiments, the normalized output thrust sent to the PX4 controller software is increased in intervals of 0.04 every seven seconds. This is done to limit the effect of the settling time for reaching a certain rotational velocity on the measured output thrust and rotational velocity of the motors. The thrust measured is for all four motors combined.

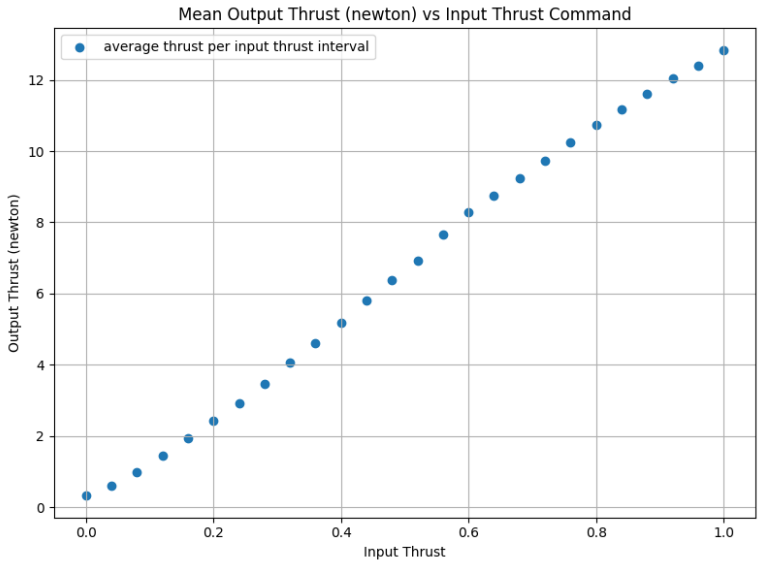


Figure 6.3: Thrust experiment.

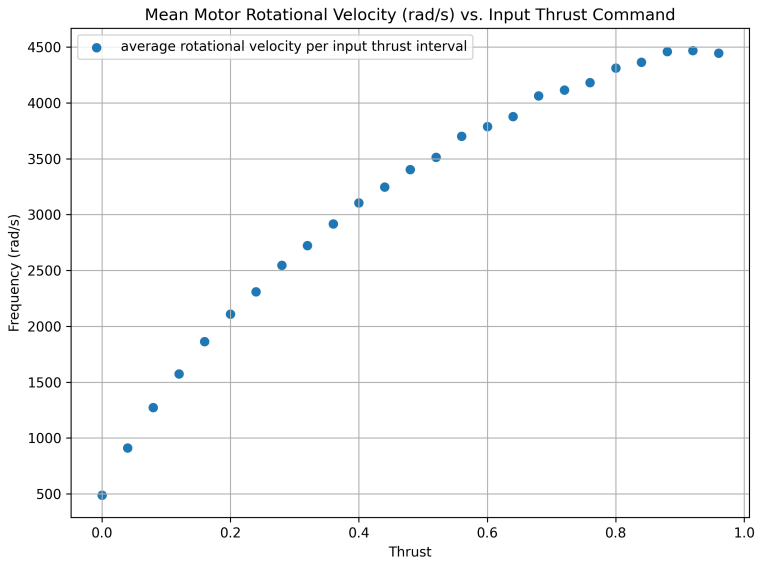


Figure 6.4: Motor rotational velocity experiment.

6.6.3. Gazebo SITL dynamic model

To simulate quadrotor dynamics inside Gazebo, the RotorS (Furrer et al. 2016) Gazebo Motor Model is used. This allows for the accurate simulation of quadrotor dynamics inside Gazebo. The RotorS model simulates the quadrotor dynamics by individually simulating dynamics for each rotor. The RotorS model requires a set of coefficients which are used to model the quadrotor's rotors. For the Mantis quadrotor, the following RotorS motor model coefficients were chosen:

Table 6.3: Quadrotor Dynamic Model Coefficients

Coefficient	Value
Time Constant Up	0.1
Time Constant Down	0.02
Max Rotational Velocity	4000
Motor Constant	2.5×10^{-7}
Moment Constant	3.18×10^{-3}
Rotor Drag Coefficient	4.11×10^{-5}
Rolling Moment Coefficient	1×10^{-6}

These coefficients were chosen based on available information from the propellers' and motors' manufacturer's datasheets. The moments of inertia which were used to model the quadrotor were determined by using the quadrotor's CAD model. To obtain an accurate estimation of the moments of inertia of the quadrotor, individual quadrotor parts were weighted and assigned these weights in the CAD model. Different autoCAD programs provide functionality to calculate moments of inertia from the CAD model with assigned weights per parts. The determined moments of inertia are as follows:

Table 6.4: Quadrotor Moment of Inertia Coefficients

Coefficient	Value
I_{xx}	0.0013218
I_{yy}	0.00083023
I_{zz}	0.0013443

The tuned controller gains from the real world Mantis quadrotor were also used inside the simulator. The PX4 controller gains for the Mantis quadrotor are as follows:

```
. ${R}etc/init.d/rc.mc_defaults

set MIXER quad_x
set PWM_OUT 1234

param set-default EKF2_AID_MASK 24
param set-default EKF2_MULTI_IMU 0
param set-default EKF2_HGT_MODE 3
param set-default EKF2_EV_DELAY 0

param set-default MC_ROLL_P 8
param set-default MC_ROLLRATE_P 0.08
param set-default MC_ROLLRATE_I 0.25
param set-default MC_ROLLRATE_D 0.001

param set-default MC_PITCH_P 8
param set-default MC_PITCHRATE_P 0.08
param set-default MC_PITCHRATE_I 0.25
param set-default MC_PITCHRATE_D 0.001
```

```
param set-default MC_YAW_P 4
param set-default MC_YAWRATE_P 0.2
param set-default MC_YAWRATE_I 0.1
param set-default MC_YAWRATE_D 0

param set-default MC_ROLLRATE_MAX 1600
param set-default MC_PITCHRATE_MAX 1600
param set-default MC_YAWRATE_MAX 1000

param set-default MPC_MANTHR_MIN 0
param set-default MPC_MAN_TILT_MAX 60

param set-default THR_MDL_FAC 0.3

param set-default PWM_MAIN_MIN 1075
param set-default PWM_MAIN_RATE 0

param set-default SDLOG_PROFILE 19

param set-default IMU_DGYRO_CUTOFF 50
param set-default IMU_GYRO_CUTOFF 90

param set-default CBRK_IO_SAFETY 22027
```

The full SDF file used to model the Mantis quadrotor can be found in the cor-drone-dev Github repository. A model for a much more agile quadrotor is also created for testing purposes. This model is created by changing the RotorS model parameters, moments of inertia and by tuning the controller gains. This model is not used during the thesis, as it is less realistic than the model for the Mantis quadrotor.

Table 6.5: Agile Quadrotor Dynamic Model Coefficients

Coefficient	Value
Time Constant Up	0.0001
Time Constant Down	0.0001
Max Rotational Velocity	4000
Motor Constant	2.5×10^{-7}
Moment Constant	3.18×10^{-3}
Rotor Drag Coefficient	4.11×10^{-5}
Rolling Moment Coefficient	1×10^{-8}

Table 6.6: Agile Quadrotor Moment of Inertia Coefficients

Coefficient	Value
I_{xx}	0.0013218
I_{yy}	0.00058023
I_{zz}	0.0013443

The controller parameters for the agile quadrotor are as follows:

```
. ${R}etc/init.d/rc.mc_defaults

set MIXER quad_x
set PWM_OUT 1234
```

```
param set-default EKF2_AID_MASK 24
param set-default EKF2_MULTI_IMU 0
param set-default EKF2_HGT_MODE 3
param set-default EKF2_EV_DELAY 0

param set-default MC_ROLL_P 8
param set-default MC_ROLLRATE_P 0.08
param set-default MC_ROLLRATE_I 0.25
param set-default MC_ROLLRATE_D 0.001

param set-default MC_PITCH_P 40
param set-default MC_PITCHRATE_P 0.039
param set-default MC_PITCHRATE_I 0.0
param set-default MC_PITCHRATE_D 0.0006

param set-default MC_YAW_P 4
param set-default MC_YAWRATE_P 0.2
param set-default MC_YAWRATE_I 0.1
param set-default MC_YAWRATE_D 0

param set-default MC_ROLLRATE_MAX 1600
param set-default MC_PITCHRATE_MAX 2600
param set-default MC_YAWRATE_MAX 1000

param set-default MPC_MANTHR_MIN 0
param set-default MPC_MAN_TILT_MAX 60

# use thrust curve factor (instead of TPA)
param set-default THR_MDL_FAC 0.3

param set-default PWM_MAIN_MIN 1075
# enable one-shot
param set-default PWM_MAIN_RATE 0
param set-default PWM_RATE 0

# enable high-rate logging profile (helps with tuning)
param set-default SDLOG_PROFILE 19

param set-default IMU_DGYRO_CUTOFF 50
param set-default IMU_GYRO_CUTOFF 90

param set-default CBRK_IO_SAFETY 22027
```

Bibliography

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: http://github.com/google/jax.

Dougherty, John, Daewon Lee, and Taeyoung Lee (2014). "Laser-based guidance of a quadrotor uav for precise landing on an inclined surface". In: *2014 American Control Conference*, pp. 1210–1215. DOI: 10.1109/ACC.2014.6859391.

Förster, Julian (2015). "System Identification of the Crazyflie 2.0 Nano Quadrocopter". In: URL: https://api.semanticscholar.org/CorpusID:139392646.

Furrer, Fadri et al. (2016). "Robot Operating System (ROS): The Complete Reference (Volume 1)". In: ed. by Anis Koubaa. Cham: Springer International Publishing. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

Geneva, Patrick et al. (2020). "OpenVINS: A Research Platform for Visual-Inertial Estimation". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4666–4672. DOI: 10.1109/ICRA40945.2020.9196524.

Intel Corporation (2024). *Intel RealSense D400 Series Datasheet*. URL: https://www.intelrealsense.com/depth-camera-d435/#:~:text=The%20Intel%C2%AE%20RealSense%E2%84%A2%20depth%20camera%20D435%20is%20a,as%20possible%20is%20vitally%20important..

Kaufmann, Elia, Leonard Bauersfeld, et al. (2023). "Champion-level drone racing using deep reinforcement learning". In: *Nature* 620.7976, pp. 982–987. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06419-4. URL: https://doi.org/10.1038/s41586-023-06419-4.

Kaufmann, Elia, Mathias Gehrig, et al. (2018). "Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing". In: *CoRR* abs/1810.06224. arXiv: 1810.06224. URL: http://arxiv.org/abs/1810.06224.

Kim, Jinho et al. (2021). "Autonomous Quadrotor Landing on Inclined Surfaces Using Perception-Guided Active Asymmetric Skids". In: *IEEE Robotics and Automation Letters* 6.4, pp. 7877–7877. DOI: 10.1109/LRA.2021.3101869.

Kooi, Jacob E. and Robert Babuška (2021). "Inclined Quadrotor Landing Using Deep Reinforcement Learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Prague, Czech Republic: IEEE Press, pp. 2361–2368. DOI: 10.1109/IROS51168.2021.9636096. URL: https://doi.org/10.1109/IROS51168.2021.9636096.

Lange, Robert Tjarko (2022). *gymnax: A JAX-based Reinforcement Learning Environment Library*. Version 0.0.4. URL: http://github.com/RobertTLange/gymnax.

Lesak, Mark C. et al. (2022). "Autonomous Quadrotor Landing on Inclined Surfaces in High Particle Environments Using Radar Sensor Perception". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12352–12358. DOI: 10.1109/IROS47612.2022.9981929.

Lin, Jiarong et al. (2019). *Flying through a narrow gap using neural network: an end-to-end planning and control approach*. arXiv: 1903.09088 [cs.RO].

Mao, Jeffrey et al. (June 2023). "Robust Active Visual Perching With Quadrotors on Inclined Surfaces". In: *IEEE Transactions on Robotics* 39.3, pp. 1836–1852. ISSN: 1941-0468. DOI: 10.1109/tro.2023.3238911. URL: http://dx.doi.org/10.1109/TRO.2023.3238911.

Moore, Thomas and Daniel Stouch (2016). "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: *Intelligent Autonomous Systems 13*. Ed. by Emanuele Menegatti et al. Cham: Springer International Publishing, pp. 335–348. ISBN: 978-3-319-08338-4.

Pal Robotics (2014). *aruco_ros*. https://github.com/pal-robotics/aruco_ros/tree/noetic-devel. Accessed: 1st february 2024.

Xie, Yuhan et al. (2023). *Learning Agile Flights through Narrow Gaps with Varying Angles using Onboard Sensing*. arXiv: 2302.11233 [cs.RO].