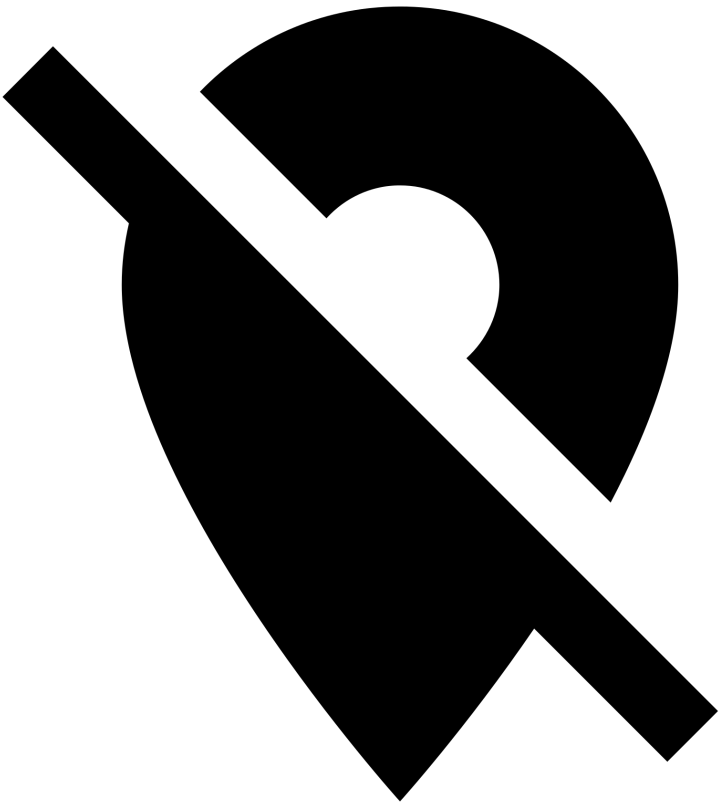
Onboard Visual Control of a Quadcopter MAV Performing a Landing Task on an Unknown Platform

J. Blom

19 February 2019

Delft University of Technology



TU Delft

Onboard Visual Control of a Quadcopter MAV Performing a Landing Task on an Unknown Platform

by

J. Blom

to obtain the degree of Master of Science
at the Faculty of Aerospace Engineering,
Department of Control & Simulation,
Delft University of Technology.

Student number:	4306473	
Project duration:	March 12, 2018 – March 6, 2019	
Thesis committee:	Dr. ir. G. C. H. E. de Croon	TU Delft, supervisor
	ir. K. Y. W. Scheper	TU Delft, supervisor
	Dr. ir. M. Snellen	TU Delft
	ir. C. de Wagter	TU Delft

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

TU Delft

Preface

This thesis is written as final part of a study to obtain a degree of Master of Science in Aerospace Engineering at Delft University of Technology.

I would like to take this space to thank Guido and Kirk for their guidance and the entirety of Sim 0.08 for their help throughout the process. Especially Matthijs for helping me with making proper drawings for my paper.

For any questions regarding the theory behind the control system or its implementation, feel free to reach out to me, I will be happy to answer any questions I can.

Contents

Introduction

This thesis is written as final part of a study to obtain a degree of Master of Science in Aerospace Engineering at Delft University of Technology.

In this thesis the problem of autonomously landing a Micro Air Vehicle (MAV) on a platform of a priori unknown dimensions at an unknown location is tackled. For this task the MAV will be without a position fix from for example GPS and depend largely on visual information.

This will be done using an adaptation of an optical flow divergence based landing algorithm, to allow for landing on a platform vertically displaced from the background, by replacing the noisy global divergence with the size increase of the platform in the virtual camera frame. Also the combination of the algorithm with visual servoing logic, allows the platform to be displaced w.r.t. the center of the image plane, as long as it is in view.

This report consists of two main parts: a scientific paper describing the algorithm, experiments and results (Part I), followed by a preliminary report describing the literature and initial investigations used to come to the mentioned approach (Part II). This last part goes into some depth on Event-Based Vision Sensors, which would be a good fit as a sensor in replacing the standard frame based camera. These and other recommendations are given in the last part of this report: Part III.

I

Scientific Paper

Onboard Visual Control of a Quadrotor MAV Performing a Landing Task on an Unknown Platform

Student: J. Blom
Supervisors: K.Y.W. Scheper, G.C.H.E. de Croon
Delft University of Technology, Kluyverweg 1, Netherlands

ABSTRACT

Vision based control allows Micro Air Vehicles (MAV) to move autonomously in GPS-denied environments, for example in indoor applications. An open issue in this field is landing on an unknown platform. The difficulty in visual control with respect to such an unknown platform, is a lack of scale. Without knowledge of the scale of offsets and object sizes (without height knowledge from GPS) it is difficult to determine an appropriate response from the controller. A control algorithm is designed to fit these requirements using an adaptation of a constant optical flow divergence based landing scheme, combined with an Image Based Visual Servoing approach applied to features in the Virtual Camera. The approach leads to satisfactory behavior in Gazebo simulations; it results in a robust controller for a range of starting heights and divergence settings.

1 INTRODUCTION

Vision based control allows Micro Air Vehicles (MAV) to move autonomously in GPS-denied environments, for example in indoor applications. An open issue in this field is landing on an unknown platform.

The difficulty in visual control w.r.t. such an unknown platform, is a lack of scale. Without knowledge of the scale of offsets and object sizes (without height knowledge from GPS) it is difficult to determine an appropriate response from the controller.

This distinction constitutes the division of Visual Servoing into Position Based Visual Servoing (PBVS) and Image Based Visual Servoing (IBVS), where it is the latter that lacks scale, meaning control is solely based on features in the image plane (Chaumette and Santos (1993)). In MAV applications however a part of the state ($[\theta, \phi]$) is usually known, allowing for an intermediate solution: Hybrid Visual Servoing (HVS) (Malis et al. (1999)). In HVS the image can be rotated to align a flat object with a horizontal "virtual camera" (Zheng et al. (2017)) & (Fink et al. (2015)), increasing the performance of IBVS algorithms applied to the problem.

For the vertical part of this problem: landing without scale or height knowledge, approaches have been developed. One is to keep the optical flow divergence $D = \frac{v_z}{Z}$ constant. This D can be measured from just a set of frames; it does not require any additional knowledge. When D is held constant, v_z reduces logarithmically over time, resulting in a smooth landing (De Croon et al. (2013)). There is however a known problem with this approach: a controller with error signal: $e_d = D_m - D$ will be unstable near the singularity at $Z = 0$. This problem was solved, using variable gains however by de Croon (2016) and applied to a drone performing a successful smooth landing by Ho et al. (2016).

Combining this vertical control with the previously discussed horizontal control, leaves only the problem of tracking the platform over time in the image plane. This can be done by tracking features belonging to the object over time and taking their average position and distance to each other as a reference for the center of the object and its size. Corners will be used as features in this approach, since these are strong features with contrast in two directions, they will be extracted from the image using a FAST corner detection approach as first introduced by Trajković and Hedley (1998). This tracking approach however does reduce the applicability of the solution to textured objects; a landing on a smooth surface can not be performed with this approach.

The main contribution of this research is an adaptation of the optical flow divergence based landing algorithm from (Ho et al., 2016), to allow for landing on a platform vertically displaced from the background, by replacing the noisy global divergence with the size increase of the platform in the virtual camera frame. Also the combination of the algorithm with visual servoing logic, allows the platform to be displaced w.r.t. the center of the image plane, as long as it is in view.

Section 2 provides a more elaborate explanation of the theories and paradigms mentioned above, while section 3 describes how this theory is translated into algorithms. Section 4 describes the simulation experiments performed to test the algorithms. The results of these experiments can be found in section 5. These results are discussed in section 6 and finally conclusions and recommendations are posed in section 7.

2 THEORY

A functional solution to the problem posed in section 1 consists of three partial solutions: tracking the landing platform, servoing towards it without altitude knowledge and to perform a smooth landing on it with the same constraint. These partial solutions define the layout of this section, where section 2.1 handles the tracking, 2.2 visual servoing and 2.3 the controlled landing strategy.

2.1 Tracking

In a computer vision context, tracking can be defined as: "Following an object over time in the camera view". An object can be described as a set of features. One way to extract features from an image, is using its gradients in grey-scale intensity. Optical flow can be estimated from these features, enabling the prediction of the next location of these same features, with an accuracy that depends on the quality of the chosen features. Corners are strong features, because of their intensity gradient in two directions in the image plane. Hence the tracking algorithm developed in this research uses FAST corner detection (Trajković and Hedley (1998)) to provide features from which optical flow is calculated to predict the next position of these same corners, around which to redetect, as schematically depicted in Figure 1.

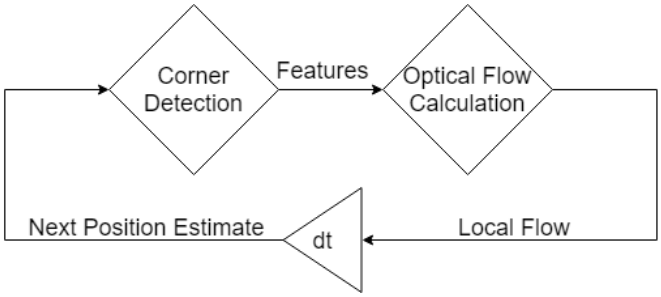


Figure 1: Tracking philosophy

Keeping track of the change in width of the landing platform is used in the control algorithm for landing explained in section 2.3. The relative change in width can be estimated using the relative change in average distance of the tracked features to their center of gravity, of which a proof is written out below.

Theorem 1. *The relative size increase of a planar object in the virtual camera frame is linearly related to the average distance of tracked features, belonging to that object, to their combined center of gravity, as long as the object is sufficiently far away*

Proof. Equation 1 describes one of the basic laws of optics (Halliday et al. (2013)), where g and b are the object and image distances respectively and f is the focal length, as shown in Fig. 2.

$$\frac{1}{g} + \frac{1}{b} = \frac{1}{f} \tag{1}$$

Clearly then, $b \approx f$ when $g >> f$. Figure 2 also shows that the sizes of the object and its image (s_g & s_b) are related by:

$$s_b = \frac{b}{g} s_g \approx \frac{f}{g} s_g$$

If we then describe points in two virtual camera frames v_1 & v_2 (explained in section 2.2) for MAV's located at the points v_1 & v_2, observing the planar points X_1 & X_2 and their center of gravity X_{cg} for an object or Region of Interest (ROI) described by its center of gravity at X_{roicg} and its edges at X_{roi1} & X_{roi2} as shown in Fig. 3. The distances of these points in the two reference frames can then be described by the following relations, where capital letters indicate coordinates in the world frame, while superscripts indicate the virtual camera frame points belong to:

$$||x_1 - x_{cg}||^{v1} = \frac{f}{h_{v1}} ||X_1 - X_{cg}||$$

$$||X_1 - X_{cg}||^{v2} = \frac{f}{h_{v2}} ||X_1 - X_{cg}||$$

$$||x_{roi1} - x_{roicg}||^{v1} = \frac{f}{h_{v1}} ||X_{roi1} - X_{roicg}||$$

$$||x_{roi1} - x_{roicg}||^{v2} = \frac{f}{h_{v2}} ||X_{roi1} - X_{roicg}||$$

Hence when an MAV has moved from point v_1 to point v_2 the size of the ROI will have changed as following:

$$\frac{||x_1 - x_{cg}||^{v1}}{||x_1 - x_{cg}||^{v2}} = \frac{||x_{roi1} - x_{roicg}||^{v1}}{||x_{roi1} - x_{roicg}||^{v2}}$$

□

The proof of the 2D problem above can be used to show the approach works for 3D cases too, given the center of the object is located at the Focus of Expansion (FOE). This assumption can be made for this research, since the goal is for the observer to land on the tracked object. Figure 4 shows such a 3D problem, where the black dots indicate a static feature as viewed in the two virtual camera frames v_1 and v_2. Now when $x_{cg} = x_{FOE}$, as mentioned above, points located on the edge of the ROI (depicted as red dots in the figure) move along the same line as the feature between two frames, thereby reducing the 3D problem to the 2D problem proven above. Other than this Fig. 4 also shows the two image plane directions can be considered separately, since:

$$\frac{d_{v2}}{d_{v1}} = \frac{s_{v2}}{s_{v1}}$$

and

$$dx_{v1} = d_{v1} \cos\theta$$

$$dx_{v2} = d_{v2} \cos\theta$$

Hence

$$\frac{dx_{v2}}{dx_{v1}} = \frac{s_{v2}}{s_{v1}}$$

and assuming a constant shape

$$\frac{w_{v2}}{w_{v1}} = \frac{h_{v2}}{h_{v1}} = \frac{s_{v2}}{s_{v1}} = \frac{dx_{v2}}{dx_{v1}}$$

The same holds for the y-direction. The proof uses the location of only one point, but the same holds when the spread of a group of N_c corners is used, as in eq. 2 leading to a more stable way to relate the old and new widths of the object in the virtual camera frame w_{v1} and w_{v2}.

$$\frac{w_{v2}}{w_{v1}} = \frac{1}{N_c} \sum_{i=0}^{N_c} \frac{||x_i - x_{cg}||^{v2}}{||x_i - x_{cg}||^{v1}} \tag{2}$$

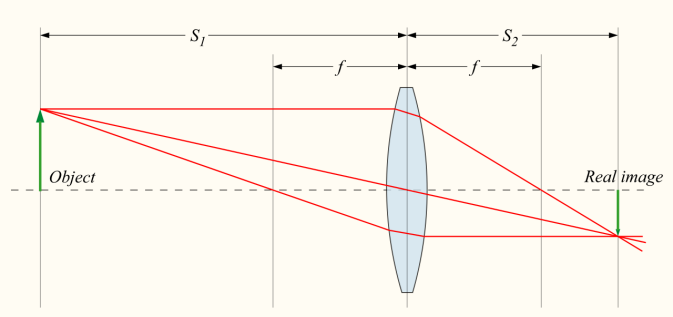


Figure 2: Standard convex lens optics (Commons (2006))

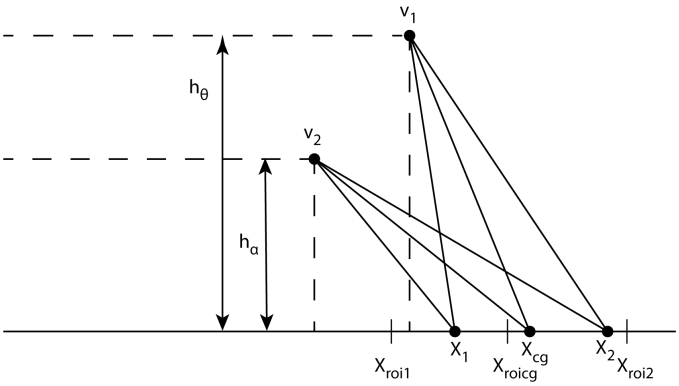


Figure 3: Points 1 and 2 and their center of gravity belonging to object (ROI) and the observation location virtual camera frames v_1 & v_2

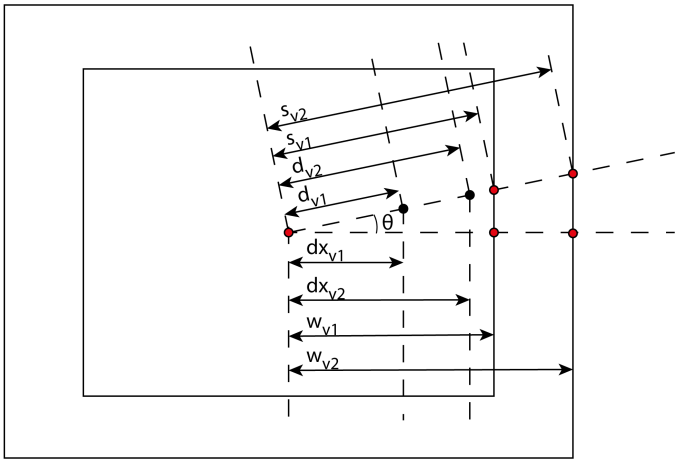


Figure 4: A static feature (black dot) and rectangular object as viewed in two virtual camera frames (v_1 & v_2), for an observer at two different heights

This way of estimating the growth or shrinking of the object in the virtual camera frame requires tracking the same features over time. However, features will inevitably be lost for a number of reasons, one of which is that they might move out of view when the MAV moves closer to the platform. Such problems are solved in section 3.1.

2.2 Visual Servoing

For this research an In-Hand Hybrid Visual Servoing scheme will be used based on the Virtual Camera approach, where In-Hand means the camera moves when the robot moves i.e. the camera is attached to the robot (Chaumette et al. (2016)). This approach uses the known Euler angles from the MAV's state, to correct for misrepresentations of a flat object in a rotated camera view, as shown in Fig. 5. When the object is represented in this horizontal Virtual Camera frame, traditional Image Based Visual Servoing algorithms can be applied, provided the object is flat and aligned with the local world horizontal plane (Popova (2015)). These algorithms are of the form as described in eq. 3, where $\boldsymbol{s}$ is a vector of features, which can be chosen to match the control tasks. Such features might include for example: orientation, object size and object center of gravity (Tahri and Chaumette (2005)). The interaction matrix $\boldsymbol{L}$ describes the transformation of velocities in the body frame to changes of object features in the 2D camera frame.

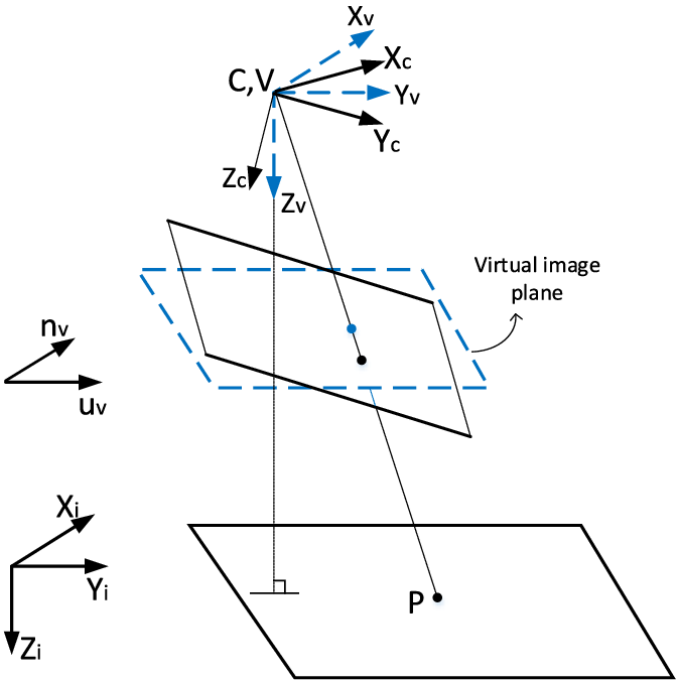


Figure 5: Virtual Camera Plane from Zheng et al. (2017)

$$\boldsymbol{s} = \boldsymbol{L}\boldsymbol{v} \tag{3}$$

The system can then be controlled based on the inverse of this equation. Tuning of the gains for such a control algorithm and the selection of the features in $\boldsymbol{s}$ are explained in section 3.

2.3 Landing

Optical flow divergence can be used to perform vision based landings. Optical flow can be mathematically described using eq. 4, for movement above a flat surface after correcting for rotational rate effects (Waxman and Wohn (1985)). Positive optical flow divergence is defined as going away from the center, towards the edges of the camera frame, as caused by a downward relative velocity. From this equation it can be deduced that keeping optical flow divergence constant will cause the downward velocity to reduce logarithmically, thus leading to a smooth landing as pointed out by De Croon et al. (2013).

$$\vartheta_x = \frac{v_x}{Z} \qquad \vartheta_y = \frac{v_y}{Z} \qquad D = \frac{v_z}{Z} \tag{4}$$

There is however a known problem with this approach: it results in an unstable system at low altitudes, because the control signal goes to infinity when approaching $Z = 0$ as pointed out by de Croon (2016). A solution to this problem is to perform an initial gain tuning, starting from a known to be stable gain as done by Ho et al. (2016). Use the oscillations to tune the gain, when oscillations are noticed, slightly reduce the gain and use that as starting gain for the landing. Then exponentially reduce the gain over time depending on the divergence setting, according to eq. 5.

$$K(t) = K_{min} + (K_{start} - K_{min})e^{-Dt} \tag{5}$$

As mentioned in section 2.1, corners are reliable features, especially when their local detection is based on the predicted next position from optical flow estimations. It thus makes sense to use these features for an estimation of the divergence, rather than the estimated local optical flow, which is used only as a prediction for their next position. Using the tracked corners for estimating the local optical flow divergence at the object location, would thus be more reliable. The relative increase in size of the object ($\frac{dw}{w\,dt}$) seems a natural candidate as a replacement for the divergence, which basically describes the number of pixels a feature moves outward divided by its distance from the center of the image in pixels. In the rest of this thesis $\frac{dw}{w\,dt}$ will be referenced as D_b.

A filtered D_b is expected to be more accurate in approximating D than a filtered D_{LK} from a Lukas-Kanade divergence approximation. Another reason to prefer D_b over global optical flow divergence, is that (as the name suggests) the latter is an average of the flow away from the center of the entire image, including background, while D_b represents the same type of information for the object only. This would allow the platform to be at a different height than the rest of the objects in view, or even move w.r.t. them.

3 Methodology

TU Delft uses Paparazzi to create and test control algorithms for UAVs. In this framework, modules can be created and called from an airframe file. In this research work was mainly performed on the "opticflow" module and a new module called "ibvs_sim". This latter module handles the servoing part as will be explained in section 3.2 and the first handles the tracking of the desired landing location as explained in section 3.1.

3.1 Tracking

The object to be tracked for a run, will be chosen by the user at initialisation. The user defines a location in the camera view and the size of the window (ROI) around that point, which includes the object and preferably nothing else. This ROI will then be propagated over subsequent frames and can be represented in the virtual camera frame by two variables: the center $(x_{ROIcg}, y_{ROIcg})^v$ and the size $(w_{ROI}, h_{ROI})^v$. These values are estimated using the center of gravity of the tracked features (x_{ccg}, y_{ccg}) and their average distance to it $(\bar{dx}, \bar{dy})$ and by keeping track of their offsets to the variable they reflect (x_{corr}, y_{corr}) and $(\frac{w}{d}, \frac{h}{d})$ respectively, where $x_{corr} = x_{ROIcg} - x_{ccg}$ and $\frac{w}{d} = \frac{w_{ROI}}{dx}$. These offsets are first calculated at the initialisation of the features in their window and updated when corners are lost over time and when new corners are added to replace them. While descending,

the part of the screen taken up by the ROI increases. Once it takes up more than 75 % of the height or width of the screen, it is reinitialized to take up 50 %.

3.2 Servoing

The feature vector $\boldsymbol{s}$ chosen for this research includes the object size and center of gravity. Control is directly based on these features, using a proportional controller around the nominal thrust for vertical control and zero velocity for the two horizontal directions. The gains used to calculate the control signal, are logarithmically reduced over time to counter the instability problem mentioned in section 2.3. A successful landing consists of the 4 phases shown in Fig. 6, where it should be noted that the logic in phase 0 is also part of phase 1. In this approach control gains are initially set to a low value, known to cause stable hovering behavior in all considered situations. In this research, the horizontal gains for the x & y-directions are always the same and linearly related to the vertical gain for control in the z-direction. The stability of both horizontal and vertical control depends on the height w.r.t. the object, as shown by 4, it thus makes sense to make one depend on the other.

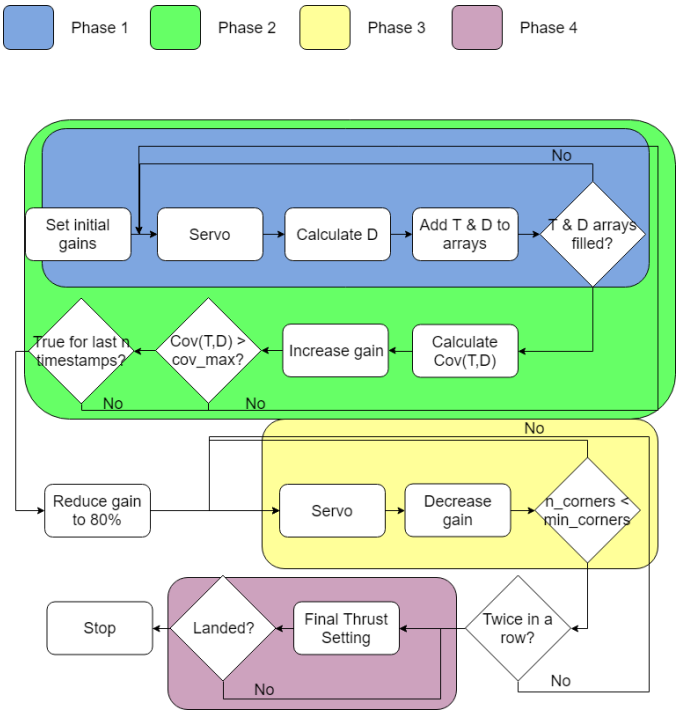


Figure 6: Flowchart depicting the different flight phases

Figure 6 shows how a Divergence (D_b) array is filled in phase 1, which is used in phase 2 to calculate the covariance of D_b with Thrust (T). When this covariance is positive for 2 seconds, the MAV is oscillating and the gain has been increased above its optimum value. The control gain is then reduced to 80% of its last value, after which the actual landing is started in phase 3, according to the constant optical flow divergence paradigm. When eventually the number of corners detected is lower than half of the desired number of corners for two consecutive timestamps, the MAV is assumed to be close enough to the object to start the final landing phase, phase 4, where the thrust is held constant at a value just below the thrust setting for hover. If the tracking algorithm should malfunction due to unexpected circumstances, the MAV will go into this mode as well, leading to a safe landing, even though the system could not perform its imagined tasks.

Before using it in a constant optical flow landing, D_b is filtered using the low-pass filter described in eq. 6, where lpf is the low-pass factor, or the weight of the new measurement D_{bi} w.r.t. the old estimate D_{bi-1}. As long as $D_{bi} - D_{bi-1} < (\Delta D)_{max}$, because large changes are not likely to occur between two timestamps.

$$D_b = D_{bi-1}(1 - lpf) + lpf\, D_{bi} \tag{6}$$

4 Experimental Setup

To test the algorithm, the MAV is to autonomously servo to a location specified in the camera frame and land there. This is done in simulation, using the Gazebo plugin for paparazzi. The results of these experiments can be found in section 5. Through the simulation experiments the performance of D_b as a linear approximation of D is tested first (section 4.1) and after that the system as a whole is tested (section 4.2).

A simulation using the Gazebo plugin from paparazzi has realistic 3D motion and physics. To get an idea of what it is, Fig. 7 shows a simulation of a Bebop drone flying around in the TU Delft Cyberzoo. Essential for our research is the fact that camera footage is also simulated; images from the bottom-"camera" come in at the specified frame rate. An example of such a frame is shown in Fig. 8.

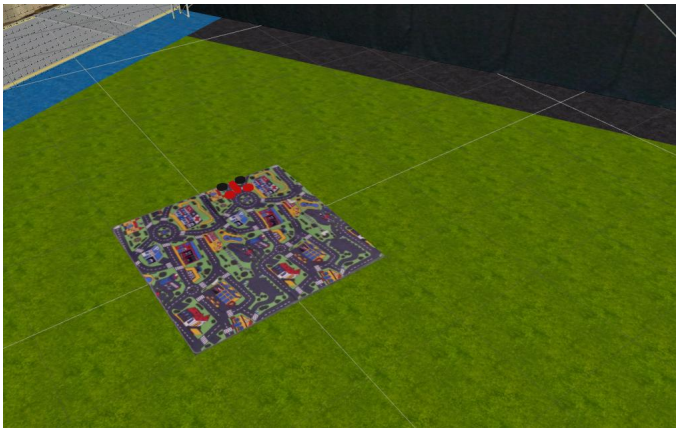


Figure 7: Gazebo simulation of Bebop drone in Cyberzoo

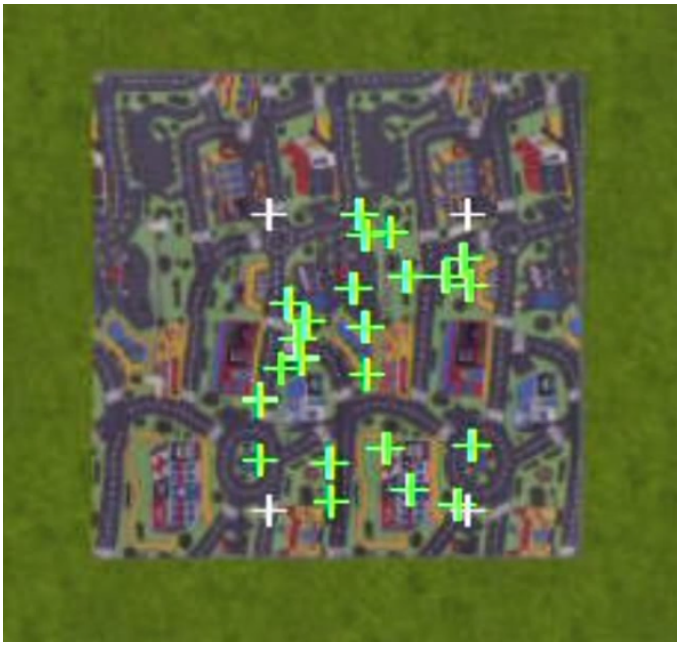


Figure 8: Footage from the simulated bottom camera, where white crosses indicate corners of the ROI and green crosses indicate tracked features

Both for testing the performance of D_b as a replacement variable, and that of the system as a whole, the MAV's position and velocity should be known over the course of the experiment. In simulation these values can be determined for any point in time, because they are the result of a mathematical model of the system. In real life this is can be handled by the Cyberzoo's Opti-track system, which triangulates the MAV's position based on the position of markers belonging to the MAV as perceived by a grid of cameras hanging from the ceiling.

4.1 D_b experiment

To test how well D can be approximated using D_b, the MAV descends with a constant v_z, thus creating a dataset with a spread in ground truth values for $D = \frac{v_z}{Z}$. This value and the calculated D_b are logged and used to create a linear relationship between the two and create error curves for $\hat{D} - D$. The starting altitude for these experiments is $h_0 = 6m$, while v_z is varied to be $v_z = [0.5, 0.8, 1.0, 1.2, 1.5] ms^{-1}$.

4.2 System test

To test the performance of the system, Z, v_z and the horizontal error ϵ are tracked and plotted over time. The desired landing location (from which to calculate the offset ϵ) is calculated from the angle it makes with the center of the camera and the height of the MAV. This way a dataset is created with all possible combinations of the settings: $\boldsymbol{D_b} = [0.02, 0.04, 0.08, 0.16, 0.24]$ and $\boldsymbol{h_0} = [2, 3, 4, 5, 6]$.

5 Simulation Results

The results for the D_b test plots are presented first (section 5.1) after which, results are presented for tests of the system in its entirety (section 5.2).

5.1 D_b test results

The results of the experiments described in section 4.1 are presented in this section. First a linear mapping is created from D_b to the ground truth $D = \frac{v_z}{Z}$, of which the results are shown in Figs. 9 & 10.

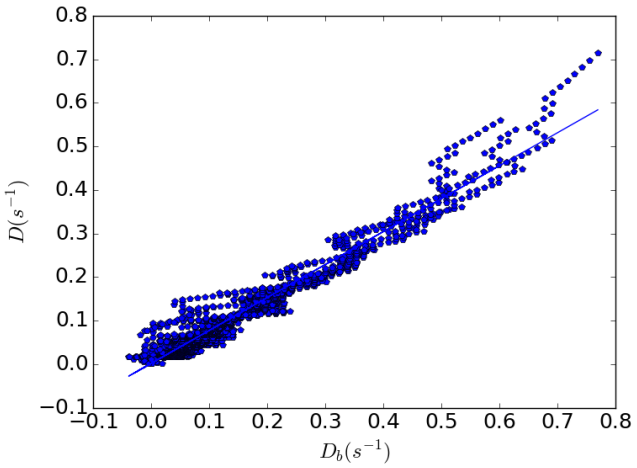


Figure 9: D vs D_b with linear relation

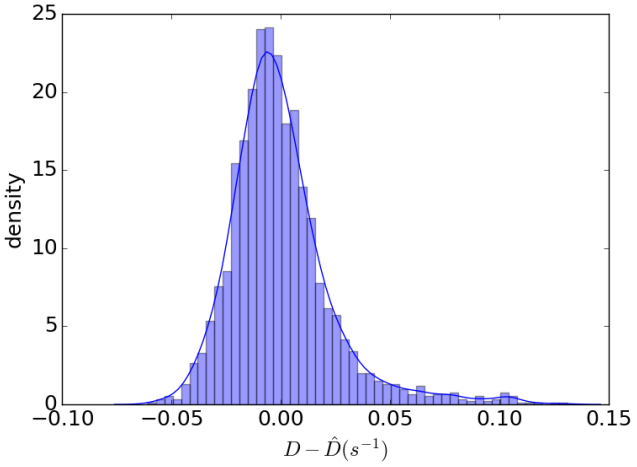


Figure 10: Approximation error $D - \hat{D}$, using D_b to determine $\hat{D}$

The performance of D_b as a linear indicator of D is also plotted along the axes: h, D & t, to see if any of these pa-

rameters is of significant influence on the accuracy of the approximation. These plots can be found in Figs. 11, 12 & 13. Where Fig. 11 shows the approximation using D_b is worse for high and low altitudes, Fig. 12 shows a similar effect for high and low values of D. Because of the experiment setup and the way data is gathered (using a constant downward velocity) these two parameters can not be completely separated and might well be measuring the same effect.

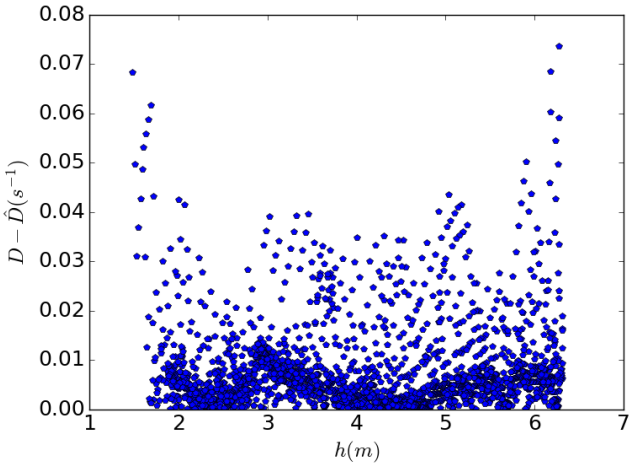


Figure 11: $D - \hat{D}$ vs height, using D_b to determine $\hat{D}$

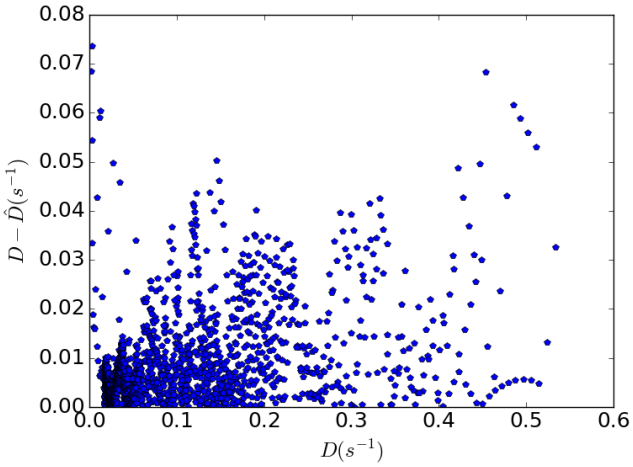


Figure 12: $D - \hat{D}$ vs D, using D_b to determine $\hat{D}$

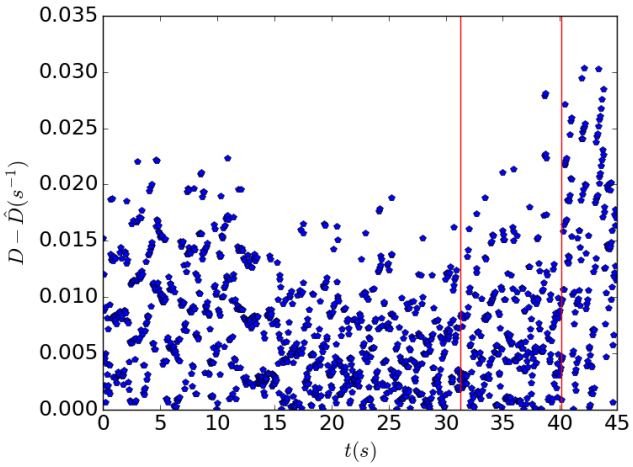


Figure 13: $D - \hat{D}$ vs t, using D_b to determine $\hat{D}$, with reinitialization timestamps as vertical lines for one run

Figures 9 & 12 show D_b can be used to approximate D quite well in the range $0 < D < 0.5$. Figure 9 however, seems to suggest a linear relationship can be found for every run separately. More specifically good fits can be created for every phase between two window reinitialisations as shown by Fig. 14. Figure 15 shows the error distribution when D is approximated using the $D = f(D_b)$ relation for that window reinitialization (the process of window (re)initialization is described in section 3.1). This would explain the satisfactory performance of the algorithm in achieving a smooth landing as shown in section 5.2.

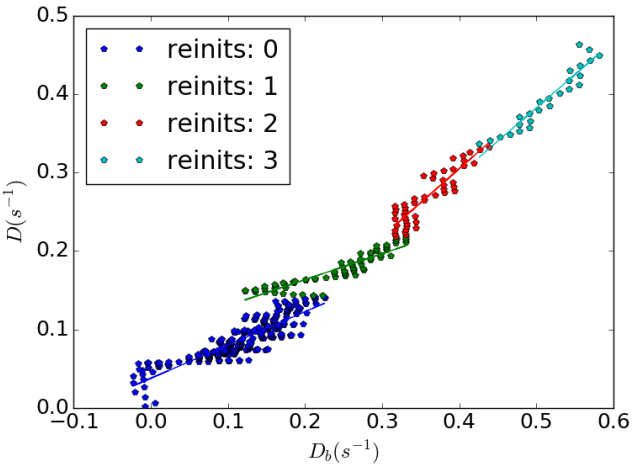


Figure 14: D vs D_b with linear relation between every two window reinits

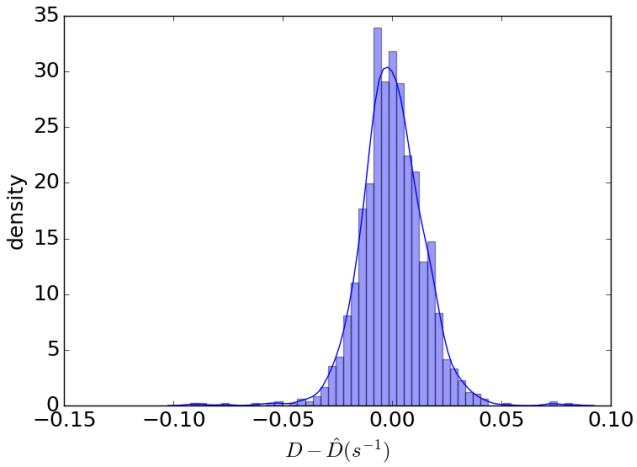


Figure 15: D vs $\hat{D}$ as approximated from D_b and the linear relation specifically for every window reinitialization

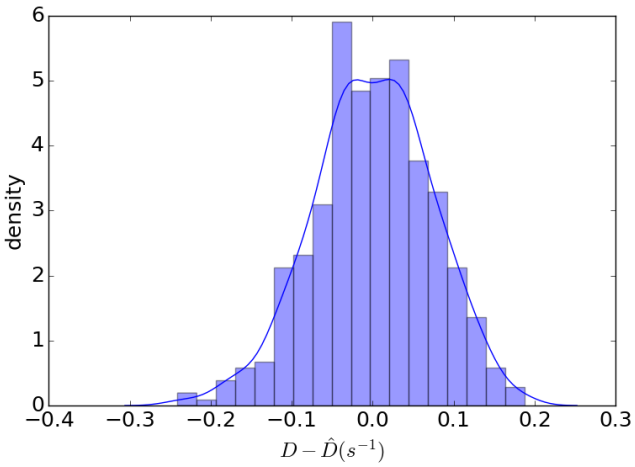


Figure 17: $D - \hat{D}$, using D_{LK} to determine $\hat{D}$

The overall approximation $\hat{D}$ based on D_b is compared to D_{LK} as retrieved by a Lukas-Kanade approach as applied by Ho et al. (2016). To achieve this, Figs. 16 & 17 show the approximation $\hat{D}$ based on D_{LK} and its error with the ground truth D respectively, like Figs. 9 & 10 show for D_b.

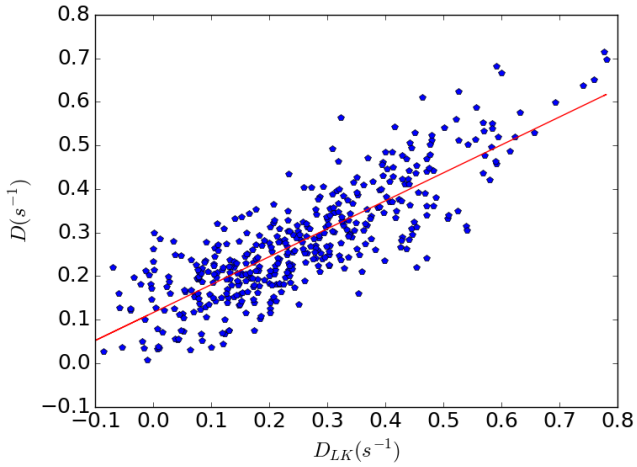


Figure 16: D vs D_{LK} with linear relation

Lastly Fig. 18 shows the number of timestamps between two consecutive divergence approximations from a Lukas-Kanade scheme as used by Ho et al. (2016). When no good fit can be created from the available flow vectors, for example due to a lack of features or a single concentration of features on a small part of the image plane, the algorithm does not generate a result. Especially the latter is frequently the case for the application considered in this paper, since an object, displaced from the center of the image plane is tracked. The average number of frames between approximations from this Lukas-Kanade algorithm is only $\Delta\bar{t}_{LK} \approx 2T_{frame}$, however Fig. 18 shows that it is not unlikely to be without an estimation for 4 frames over the course of a landing.

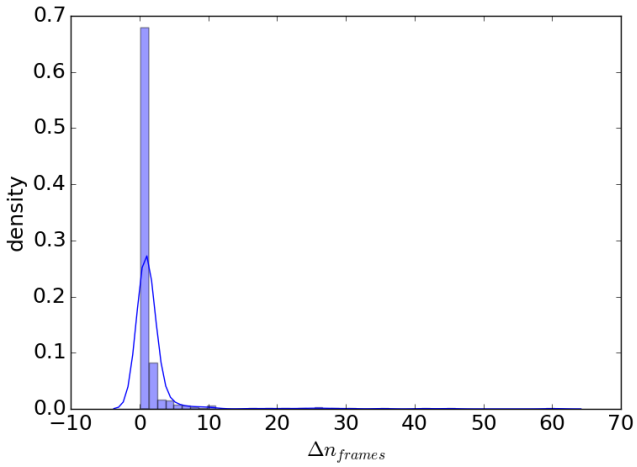


Figure 18: Number of frames (n_{frames}) between two consecutive D approximations from Lukas-Kanade algorithm as used by Ho et al. (2016)

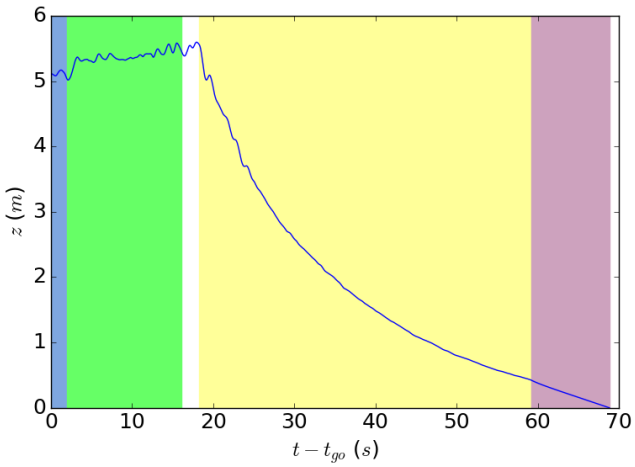


Figure 19: Height plot with shading per flight phase as in Fig. 6

5.2 System test results

Successful landings were achieved for a range of starting heights $2 < h_0 < 5$ and Divergence settings $0.02 < D < 0.24$. For all these settings, plots similar to those in this section were created.

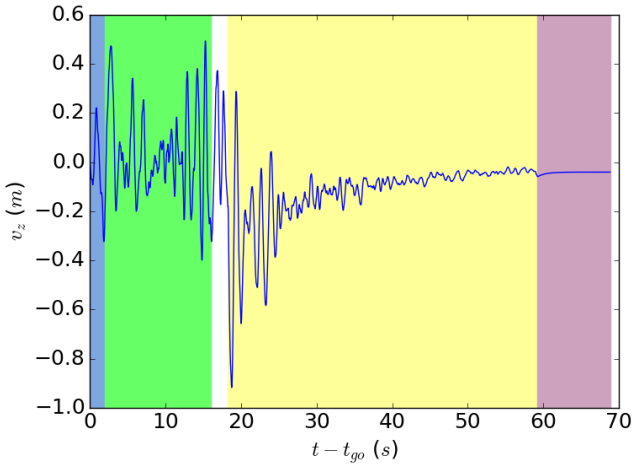


Figure 20: Vertical velocity plot with shading per flight phase as in Fig. 6

Height plots for a successful landing look like those in Fig. 19, with the shaded areas representing the flight phases described in Fig. 6. They clearly show the expected behavior, starting with oscillations increasing in amplitude in phase 2. Then the oscillations get weaker, which is where the gain is reduced to 80 % of its last value. After that the landing is started, with a downward velocity reducing in size, as is also shown by Fig. 20. In the last phase a constant final thrust setting is kept leading to the straight line for both velocity and altitude in the last phase.

The part that is of interest to this research, is the logarithmic part of the z-plot in Fig. 19. It shows small deviations from the expected slope, which can be explained when we plot the window reinitialization timestamps with it, as in Fig. 21. As explained in section 2.3, the philosophy is to keep $D_b = \frac{dw}{w\,dt}$ constant. When the window is reinitialized, meaning w is discretely changed from 75% to 50% of the image width, this leads to a stepwise change in the reference signal which leads to an instantaneous change in the controller output.

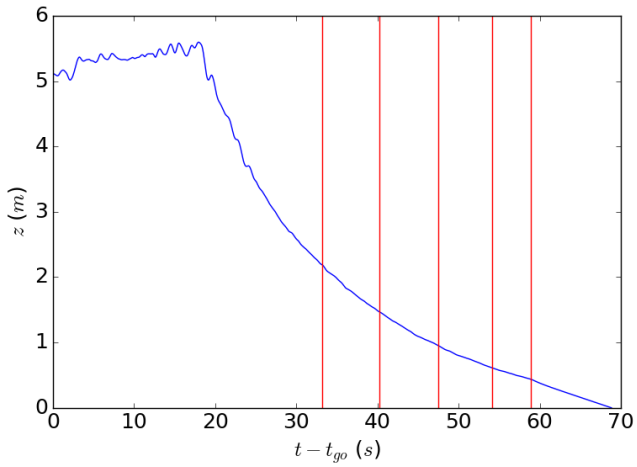


Figure 21: Height plot with vertical lines at window reinitialization

When landings for different divergence settings are compared as in Fig. 22, it can be seen that the higher settings lead to smoother behavior. It is apparently more difficult to keep a low relative size increase, for which small control inputs are required, which are more sensitive to discrepancies.

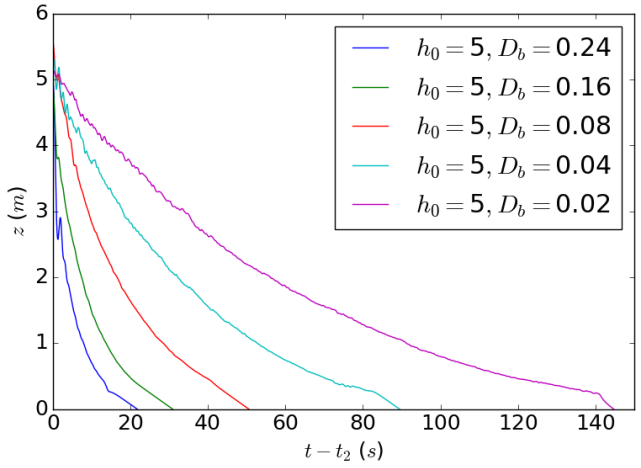


Figure 22: Height plot for a range of divergence settings starting from a height of $5m$, from start of landing phase 2

The horizontal error over time for the same run as those in Figs. 19,20 & 21 is shown in Fig. 23.

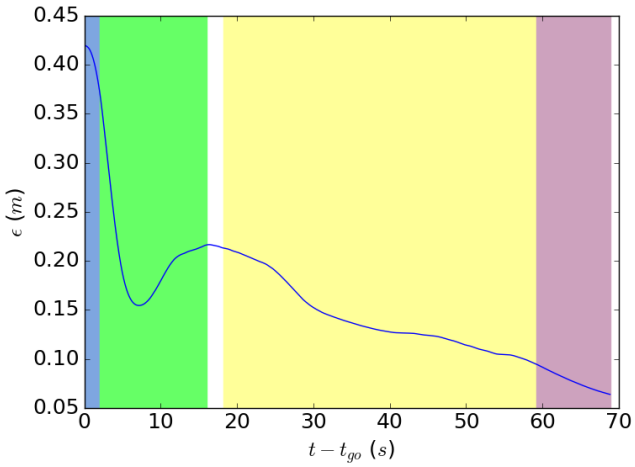


Figure 23: Horizontal error plot with shading per flight phase as in Fig. 6

6 Discussion

The effectiveness of D_b for approximating D is discussed first in section 6.1, after which the performance of the entire system, based partly on this approximation is discussed in section 6.2.

6.1 D_b to approximate D

As mentioned in section 2.3 a positive aspect of using D_b over D is that an object (a platform) can be selected with a vertical offset to the background i.e. the algorithm is insensitive to background motion and only reacts to offsets belonging to the object.

Another benefit of D_b over D_{LK} is pointed out in Fig. 18, showing the number of frames skipped between two consecutive calculations of D_{LK}. Clearly over the course of a landing it is not unlikely to be without an estimation of D for more than 4 frames, while an approximation of D_b is made for every frame.

When Figs. 10 & 17 are compared, one can see that using D_b to approximate D indeed leads to a more accurate representation for these experiments. It should be noted however that the D_{LK} approximator developed by Ho and de Croon (2016) was designed for situations where features are evenly distributed over the image, which is not the case when one is focusing on one object, as in the experiments performed in this research. However for such situations where features are confined to only a part of the image, the D_b approach is significantly more accurate than the D_{LK} approach.

The effect of modelling the camera as a convex lens, as described in 2.1, can be spotted in Fig. 14, where the effect of a certain ground truth D on D_b, i.e. the slope of the linear approximation, depends on the distribution of the features for that window initialization. Features close to the edge of the

image plane move more than would be expected using the linear relation mentioned in the same section:

$$||x_1 - x_{cg}||^{v1} = \frac{f}{h_{v1}}||X_1 - X_{cg}||$$

This leads to an overestimation of D, i.e. a more flat linear relation, for window initialisations where features are more concentrated at the edge of the image plane.

Figure 11 shows the approximation using D_b is worse for high and low altitudes, Fig. 12 shows a similar effect for high and low values of D. Because of the experiment setup and the way data is gathered (using a constant downward velocity) these two parameters can not be completely separated and might well be measuring the same effect as mentioned in section 5. Where one possible explanation is that a linear approximation of D from D_b is simply fitted best around the average D of all the runs.

A stronger explanation however, is that at a large height a feature is more easily mistaken for a neighbouring feature that was tracked in the previous image, because the two features are only a few pixels away for highly textured objects. While at small heights, the effect of the large distance assumption of Theorem 1 becomes measurable. Non-linearities can occur when this assumption is violated. This hypothesis can be tested by performing experiments where the amount of texture on the object to be tracked is varied. This would likely affect the performance at large heights, but not for the small ones.

The expected negative effect on accuracy of the window reinitializations as explained in section 3.1 is not observable from the data; Fig. 13 shows no sudden increase in the approximation error after window reinitialization.

6.2 Controller performance

From the plots for different starting heights and $D_b = \frac{dw}{w\,dt}$ settings it can be seen that D_b is a fitting replacement for the noisy optical flow divergence in a constant divergence landing approach. The even spacing of the window reinitializations as seen in plots such as the one in Fig. 21 is a second indication that the algorithm performs well, since it means the window size increases linearly over time, as it is supposed to.

It should be noted that the offset $D - \hat{D}$ using D_b, which varies per reinit as shown by Fig. 14, means the MAV is not actually landing with an exactly constant divergence, but because it is constant between every 2 window reinitializations, the strategy does lead to a logarithmically reducing vertical velocity, as shown in Fig. 22. The effect of this change in relation in $D = f(D_b)$ is a constant divergence landing with a slightly different D for every window reinitialization, as is visible from the change in slope after window reinitializations in Fig. 21.

The Virtual Camera IBVS approach further leads to satisfactory results when combined with the proposed algorithm, as can be seen from the plot for the horizontal error in Fig. 23. The error approaches zero for every run, being slightly more stable for higher starting altitudes.

When trying to validate these results in real life however a new problem was encountered, where the number of tracked corners would suddenly drop drastically between two subsequent frames. Which requires a reinitialization in the last properly tracked window, meaning a clean detection and calculation of the offsets (x_{corr},y_{corr},$\frac{w}{d}$,$\frac{h}{d}$) w.r.t. the last accurate approximation of the center of gravity and size of the object. While the loss of half the corners twice in a row triggers the controller to go into the last flight phase as mentioned in section 3.2. This in combination with a weak battery made it impossible to validate the entire control scheme in a real world experiment.

7 Conclusions and Recommendations

In this research, successful landings were performed using IBVS logic applied to features in the virtual camera frame, combined with an adaptation of the optical flow divergence based landing rationale from Ho et al. (2016). The algorithm was applied to a range of starting heights and divergence settings in Gazebo simulations in Paparazzi.

These landings were performed solely on textured objects, because the algorithm requires the tracking of at least 2 corners and preferably more, because more corners lead to a more stable approximation of the size increase and translation of the object in the camera view. If not enough texture can be found in the region specified in the camera view, the MAV directly goes into the final landing phase, which means it will descend with a constant thrust setting, slightly lower than what would be required for hover.

From the experiments it is clear that D_b can be used to replace D_{LK} which is the result of global optical flow estimations. Thanks to the small errors on the linear approximation and the overall availability, in contrast to the outcome of the Lukas-Kanade scheme, a well functioning system, robustly servoing and landing on a specified location in the screen, can be created based on D_b.

The results of the D_b experiments would probably be even better when $\frac{v_{z,i}+v_{z,i-1}}{2}$ is used as ground truth D, since that is the value $D_b = \frac{dw}{w\,dt} = \frac{w_i - w_{i-1}}{w_i\,dt}$ actually approximates; the approximation is shifted by $\frac{1}{2}T_{frame}$. Another way to improve the approximation power of D_b as $\hat{D}$ would be applying a filter to them based on data from the accelerometer. The effect of assuming the camera to be a perfect convex lens, mentioned in section 6.1 can further be countered by initializing the window more gradually, by for example only reinitializing features from an outer sub-window in an inner sub-window. When a feature for example moves to a location at 70 % of the image width it can be reinitialized between 0 and 50 %, avoiding a difference in accuracy for different window initializations.

From the plots it is evident that the controller becomes less stable for lower starting altitudes. This is probably due

to the fact that the controller gains are already too high at the start of the tuning process, which seems to agree with the fact that landing phases 1 & 2 are quite short at these low altitudes, meaning oscillations are found after only a slight increase of the controller gains. Reducing the starting gain and the step-wise increase, are therefore likely to increase the performance.

Improving the horizontal performance could also be achieved by putting effort into finding a better fit for the ratio of the horizontal and vertical gain, as the horizontal control seems to be more aggressive than the vertical control. For now the horizontal gain is tuned based on the vertical gain, for reasons mentioned in section 3.2. Probably the controller performance would be better when a similar type of gain tuning is applied to the horizontal gain as is done for the vertical gain, in a separate second oscillation detection phase. This phase could for example have as a goal to keep the object centered in the virtual camera frame. Similar to the vertical gain tuning, the control gain for this task would then be increased until the MAV is clearly oscillating over the object.

As mentioned in section 6, a real world experiment was attempted to validate the results acquired from Gazebo simulation. It was however unsuccessful, due to the occasional complete loss of tracked corners between two frames, twice in a row, triggering flight phase 4 e.g. a constant thrust descend.

An increase in performance can also be achieved by a change in hardware: using an event-based camera instead of a regular frame based camera. The Dynamic Vision Sensor (DVS) developed by IniLabs is such an event-based camera, sending the pixel location and timestamp of a pixel changing by more than a threshold in greyscale intensity. A tracking algorithm, similar to the one used throughout this research, was created for event data from such a camera and could be used to replace the current object tracking module. Upsides of using the DVS as a vision sensor are its exceptionally high update rate and its natural edge indication, since edges cause brightness changes in a dynamic environment. This natural edge indication enables corner detection at low processing cost, while the high update rate enables fast corrections for instantaneous changes.

Next steps for application of this new control logic would be to apply it to landing on a moving platform, or a platform with a vertical offset from the background. Because the information taken from the images is restricted to the tracked object, the algorithm should also work for those situations in which the visual queues from the object differ from those belonging to the background.

The DVS seems to simply be a better fit for a computer vision task such as the one described in this paper. If a step was made to apply the strategy to landing on a moving platform, the benefits become even more evident, especially since the DVS does not suffer from motion blur and is less sensitive to changes in lighting conditions.

REFERENCES

F. Chaumette and A. Santos. Tracking a moving object by visual servoing. *IFAC Proceedings Volumes*, 26(2):643–648, 1993.

F. Chaumette, S. Hutchinson, and P. Corke. *Visual Servoing*, pages 841–866. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1_34. URL https://doi.org/10.1007/978-3-319-32552-1_34.

W. Commons. Lens 3, 2006. URL https://commons.wikimedia.org/wiki/File:Lens3.svg.

G. C. H. E. de Croon. Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. *Bioinspiration & biomimetics*, 11(1): 016004, 2016.

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

G. Fink, H. Xie, A. F. Lynch, and M. Jagersand. Experimental validation of dynamic visual servoing for a quadrotor using a virtual camera. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1231–1240. IEEE, 2015.

D. Halliday, R. Resnick, and J. Walker. *Fundamentals of physics*. John Wiley & Sons, 2013.

H. Ho, G. C. H. E. de Croon, E. van Kampen, Q. P. Chu, and M. Mulder. Adaptive control strategy for constant optical flow divergence landing. *arXiv preprint arXiv:1609.06767*, 2016.

H. W. Ho and G. C. H. E. de Croon. Characterization of flow field divergence for mavs vertical control landing. In *AIAA Guidance, Navigation, and Control Conference*, page 0106, 2016.

E. Malis, F. Chaumette, and S. Boudet. 2 1/2 d visual servoing. *IEEE Transactions on Robotics and Automation*, 15 (2):238–250, 1999.

M. G. Popova. *Visual Servoing for a Quadrotor UAV in Target Tracking Applications*. PhD thesis, 2015.

O. Tahri and F. Chaumette. Point-based and region-based image moments for visual servoing of planar objects. *IEEE Transactions on Robotics*, 21(6):1116–1127, 2005.

M. Trajković and M. Hedley. Fast corner detection. *Image and vision computing*, 16(2):75–87, 1998.

A. M. Waxman and K. Wohn. Contour evolution, neighborhood deformation, and global image flow: Planar surfaces in motion. *The International journal of robotics research*, 4(3):95–108, 1985.

D. Zheng, H. Wang, J. Wang, S. Chen, W. Chen, and X. Liang. Image-based visual servoing of a quadrotor using virtual camera approach. *IEEE/ASME Trans. Mechatronics*, 22(2):972–982, 2017.

II

Preliminary Thesis

Faculty of Aerospace Engineering

Event-based Onboard Visual Control of a Quadrotor MAV Performing a Landing Task

on a Platform of Unknown Size and Location

Jari Blom

12 September 2018

TU Delft Delft University of Technology

Challenge the future

Event-based Onboard Visual Control of a Quadrotor MAV Performing a Landing Task

on a Platform of Unknown Size and Location

Preliminary Master of Science Thesis

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

Jari Blom

12 September 2018

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

Dated: 12 September 2018

Readers:

dr.ir. G.C.H.E. de Croon

ir. K.Y.W. Scheper

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous Micro Aerial Vehicle (MAV) control is a growing research area, with applications ranging from engineering, such as bridge inspection [24] to leisure, for example DJI's Phantom 4 with "follow me" capability. Landing such an autonomous quadcopter MAV, where all sensing and processing takes place on board, on an unknown platform at an unknown location is still an open issue.

Consider the situation where a remote operator selects a landing site. The easiest way to create the situational awareness required for the operator is by using a vision sensor. Now in order to be applicable to indoor situations, the navigation should be independent of GPS. Preferably no additional sensor is used at all, except to determine the Euler angles and rotational rates, which have to be known at a high rate in order to assure stability in any operating mode. When the platform is selected by a remote user, without any knowledge of that platform or the height, it is impossible to determine the scale of the environment, severely increasing the difficulty of any control task.

Currently such a solution does not exist. However in [29] a vision based autonomous landing on a moving platform of known dimensions is achieved. A vision based autonomous landing on the ground was achieved in [25] using a variable gain approach, compensating for the unknown height, as will be described in the literature review. The Simultaneous Localisation And Mapping (SLAM) approach used in [30] is one solution to the servoing problem, but this requires extra processing power and memory, rendering it an unfit approach for the smaller MAV's which are the scope of this research.

One of the major issues in this field of research is the low update rate of the onboard cameras (typically 5-20 Hz), the delay thus induced makes it difficult to ensure stability, especially in a dynamic environment. Another major issue is the processing power that is required to distil useful information out of the data produced by a frame-based camera. These frames contain a lot of data which is not useful in any way, but has to be processed either way.

Recently research has been performed to overcome these issues using event-based cameras, such as a Dynamic Vision Sensor (DVS), which is able to achieve an update rate of 10kHz [1], which is even higher than a typical update rate of an IMU. Another distinctive advantage of using an Event-based sensor is that it naturally indicates edges, saving computational power in object recognition and optical flow estimation. The high update rate of the DVS

opens up possibilities for high precision tasks and aggressive manoeuvres in a dynamic environment. In part also because it does not suffer from motion blur and has a high dynamic range. Another reason to use event cameras is because tracking solutions based on events are less sensitive to lighting conditions than those based on frame based cameras [32].

Based on the preceding, the following research objective is formulated:

> **Research Objective**
>
> To design a controller for a quadcopter MAV, based solely on angular information from an IMU and visual data from a Dynamic Vision Sensor, enabling autonomous landing on a flat platform of unknown dimensions at an unknown location without any knowledge of the environment.

Where possible, proven methods for frame based cameras will be adapted for usage with a DVS and combined into a controller able to reach the research objective. These methods will be investigated in a literature research, focusing on the three fields that together constitute the problem space:

- Visual Servoing
- Tracking
- Controlled Landing Strategies

This literature research is documented in the first part of this preliminary thesis in Chapter 2. The discussion in Chapter 3 builds up to choosing an algorithm from the ones presented in Chapter 2, of which some are tested in Chapter 4. Finally a research plan, including the proposed algorithms, experiments, required facilities and a planning can be found in Chapter 5.

Chapter 2

Literature Review

As mentioned in Chapter 1, three research fields are of interest for this thesis: Visual Servoing, Tracking and Controlled Landing Strategies. In this chapter an overview of frame based research in these fields is given as well as contemporary contributions to these fields with event-based cameras.

2-1 Visual Servoing

Visual Servoing with a camera onboard a UAV belongs to the category of eye-in-hand Visual Servoing, meaning a motion of the vehicle will cause motion of the camera [12]. This category can be split up into new categories as shown in Fig. 2-1. The layered layout of this figure defines the structure of the larger part of this section and the division of the studied literature in Table 2-1. After reporting on literature from these categories, two seperate subjects will be discussed in short: research on an MAV following a moving platform and Visual Homing.

2-1-1 Layer 1

Position based (3D): Image data is used to find the pose of the robot w.r.t. a known object. Computing the pose (relative rotation and translation) w.r.t. some reference coordinate requires knowledge of the camera intrinsic parameters and a 3D model of an observed object. The research in [29] uses known object sizes to estimate depth, to autonomously land an MAV on a static platform, with approximately the same approach as in [9].

Image based (2D) : Only image data is used to control a robot in 3D space. Using a pure 2D approach is unusual in UAV visual servoing, because high frequency angular readings are usually already given to assure hover stability, rendering it logical to use this information as input to the controller. Still some research has been performed in this direction even on UAV's such as by [26], where an adaptive controller is created. This controller uses image moments μ_{ij}, which is a frequently used approach as section subsection

Table 2-1: Distribution of Visual Servoing literature over the categories defined in Fig. 2-1, where H = Homography, I = Interaction matrix, SS = State Space PE = Pose Estimation and VC = Virtual Camera

Work	VS Type	Layer 2	Layer 3	Application	Implemented	Environment	Camera
[12]	All	-	-	-	Theory	-	-
[44]	All	-	-	UAV	Simulation	Dynamic	Frame
[29]	Position	-	-	UAV	Real World	Static	Frame
[22]	Image	-	-	-	Theory	-	-
[8]	Image	H	-	UAV	Real World	Dynamic	Frame
[5]	Image	H	-	Simulation	-	Static	Frame
[17]	Image	H	-	UAV	Simulation	Static	Frame
[23]	Image	I	-	Robotic Arm	Real World	Dynamic	Event
[45]	Hybrid	All	-	UAV	Theory	Static	Frame
[36]	Hybrid	RL	Depth	Robot Car	Real World	Static	Frame
[2]	Hybrid	RL	SS	UAV	Real World	Dynamic	Frame
[3]	Hybrid	RL	PE	Robotic Arm	Real World	Dynamic	Frame
[28]	Hybrid	RL	PE	UAV	Simulation	Dynamic	Frame
[34]	Hybrid	H	-	-	Theory	-	-
[38]	Hybrid	H	-	UAV	Theory	Static	Frame
[26]	Hybrid	I	VC	UAV	Simulation	Static	Frame
[20]	Hybrid	I	VC	UAV	Real World	Static	Frame
[53]	Hybrid	I	VC	UAV	Real World	Dynamic	Frame
[21]	Hybrid	I	VC	Robotic Arm	Real World	Static	Event

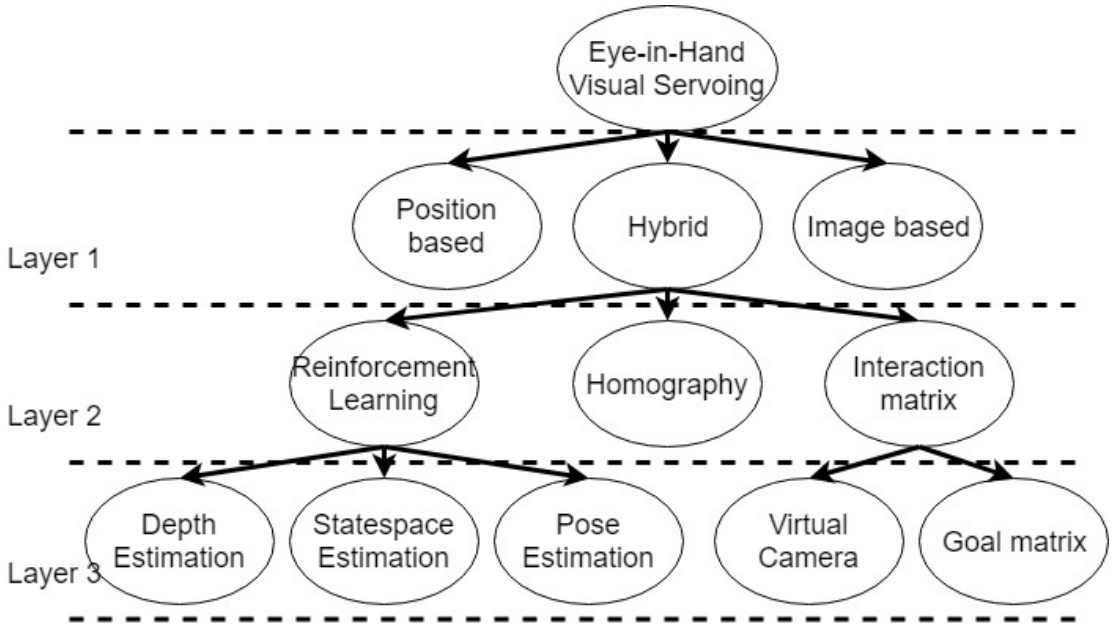


Figure 2-1: Visual Servoing for MAV's split up into categories

2-1-3 will show. Most IBVS research however is performed on robotic arms, for which we know the velocity and rotational rates, in MAV control however only the rotational rates are usually known at a high rate with little noise. The fact that these velocities and rotations are required for IBVS, makes that full image based visual servoing is an uncommon approach for UAV control. An exception to this is [40] , which runs into problems when the UAV is close to a large object or above a slanted surface. Mainly because of the attitude control based on visual inputs, which also causes problems in calculating optical flow divergence.

Hybrid (2D 1/2) Some knowledge on the observed scene, such as the MAV's Euler angles and rates, is used in combination with the image data to provide a positional error signal to a robot. For reasons already explained in this section, this is a common approach to Visual Servoing for MAV's, making use of fast angular rate measurements to compensate for the lack of knowledge on the Cartesian velocities. Different ways to use this type of knowledge are described in section 2-1-2.

2-1-2 Layer 2

As described in subsection 2-1-1, the most used type of VS for MAV's is Hybrid Visual Servoing (HVS). Other than IBVS it does not require high update rates for stabilization and other than PBVS it requires no knowledge of the observed scene, which is required by the definition of the Research Objective. The three most common HVS approaches are: Reinforcement Learning (RL), Homography and Interaction matrix based solutions. Their specifics are described in this subsection.

Reinforcement Learning

In reinforcement learning the goal is to find the optimal state-action value function $Q^{\star}_{w}(\boldsymbol{x}, \boldsymbol{a})$. Where $Q_w(\boldsymbol{x}, \boldsymbol{a})$ is of the form as defined in eq. 2-1, where $\boldsymbol{r}(\boldsymbol{x})$ are unit activities and the weights $\boldsymbol{w}$ are updated using eq. 2-2, with δ_t as defined by eq. 2-3. $\boldsymbol{e_t}$ is the eligibility trace vector, which depends on whether exploring (with probability ϵ) or exploiting behavior (1-ϵ) is adopted for this step.

$$Q_w(\boldsymbol{x}, \boldsymbol{a}) = \sum_{i=1}^{n} w_{ai} \cdot r_i(\boldsymbol{x}) \tag{2-1}$$

$$\Delta \boldsymbol{w}_a = \alpha \cdot \delta_t \cdot \boldsymbol{e}_t \tag{2-2}$$

$$\delta_t = R_{t+1} + \gamma \cdot max_a Q_t(\boldsymbol{x}_{t+1}, \boldsymbol{a}) - Q_t(\boldsymbol{x}_t, \boldsymbol{a_t}) \tag{2-3}$$

In this approach $\boldsymbol{a}$ could be the vector $\boldsymbol{q}$ and $\boldsymbol{x}$ could be the image coordinates and velocities of multiple points on the landing platform, combined with IMU data. For example in [28] the state used in the Q-function is a tuple of the current observation and the goal observation.

The downside of this approach is that it requires extensive training, which is a tedious exercise for MAVs, because recovering from a crash takes time and human effort. Also this type of control is not as much a proven concept as the other two branches in this layer.

Homography matrix

A homography matrix relates object coordinates in the current camera frame to the goal state. Research in this category is done by [34] respectively [38], also describing the process of gain tuning for noise rejection. [8] presents a homography based approach to pose estimation. Relating the projection of the i^{th} image to the world frame using $H^i_w = H^i_0 H^0_w$, requires the initial homography H^0_w, which can be obtained by using 4 or more points in the image with the same Z, so for example a square on the ground. The homography H^0_w is then found using eq. 2-4, where h_{ij} is the value in row i and column j of H. Where all h_{ij} can be solved for using a pseudo-inverse method, note that no knowledge of Z is required to determine H^0_w.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 1 & x_1 & y_1 & 1 & -x_1x'_1 & -y_1y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx'_n & -y_nx'_n \\ 0 & 0 & 1 & x_n & y_n & 1 & -x_nx'_n & -y_ny'_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix} \tag{2-4}$$

To ensure stability and non-singularity of this matrix, the camera coordinates $[x, y]$ are normalised such that $0 < x, y < 1$. This process provides a unique solution, scaled to give $H = I$ when the MAV is in the goal pose. Another way to solve for the Homography, without

knowledge of the scene is presented in [5] where 3 points are used in combination with IMU data to determine a unique solution for the homography matrix and the scaling factor α_i. If nothing is known about any point in view, meaning the platform is not necessarily flat, at least eight points are required to solve for the homography.

Using H the position of any point on the current image can be related to that object's position in the goal reference frame. The task function $\boldsymbol{e} = [\boldsymbol{e}_v, \boldsymbol{e}_\omega]^T$ is then defined by eq. 2-5 as is done in [5]. $\boldsymbol{m}^\star$ can be found using the camera intrinsic parameter matrix K_c as shown in eqs. 2-6 & 2-7, where $\boldsymbol{p}^\star$ is the position of the point to be tracked in the reference image.

$$\begin{bmatrix} \boldsymbol{e}_v \\ \boldsymbol{e}_\omega \end{bmatrix} = \begin{bmatrix} (H - I)\boldsymbol{m}^\star \\ H - H^T \end{bmatrix} \tag{2-5}$$

$$\boldsymbol{p}^\star = K_c \boldsymbol{m}^\star \tag{2-6}$$

$$K_c = \begin{bmatrix} f & f_s & u_0 \\ 0 & f_r & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-7}$$

According to [17] a gain matrix K exists such that this system is exponentially stable, which is only tested in simulation. Usually however, the control gains in such a Homography based controller are scaled with an approximation of the distance to the object in the reference pose d^*. It is important to realize that this approach requires camera calibration parameters K_c. A Homography based on a calibrated camera is called a Euclidean Homography, which is different from the projective Homography that has been discussed in this section. For such a homography eq. 2-6 is not required, since its values are already scaled.

Interaction Matrix

In IBVS and HVS research an interaction matrix L can be used to relate translations in the camera frame to velocities in the world frame, as shown in eq. 2-8.

$$\dot{\boldsymbol{x}} = L_x \boldsymbol{v}_c^b \tag{2-8}$$

In HVS, the interaction matrix can be split into two separate matrices L_v & L_ω describing the influence of the camera's Cartesian and rotational velocities respectively. These matrices can be defined in different ways: as a $[3 \times 3]$ or a $[2 \times 3]$ matrix.

The $[3 \times 3]$ approach is used in [34] & [12] and is defined by eqs. 2-10 & 2-11. These matrices can be used in a control law of the form in eq. 2-9 but require an approximation of z^b. Note that all values without a superscript are defined in the camera frame in the equations in this section.

$$\boldsymbol{v}_c = -L_{\boldsymbol{v}}^+ (\lambda \boldsymbol{e}_t + L_{\boldsymbol{\omega}} \boldsymbol{\omega}_c) \tag{2-9}$$

$$L_v = \frac{1}{z^b} \begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \\ 0 & 0 & -1 \end{bmatrix} \tag{2-10}$$

$$L_\omega = \begin{bmatrix} xy & -(1+x^2) & y \\ 1+y^2 & -xy & -x \\ -y & x & 0 \end{bmatrix} \tag{2-11}$$

Approximating z^b can be done in a number of ways:

1. perform a pose or depth estimation.
2. use $z^b = z^{\star b}$.
3. finding $\rho = \frac{z^b}{z^{\star b}}$

[12] and [34] describe a way to determine the interaction matrices using this last approach. An extended state vector $s_t = (\boldsymbol{x}, log(z^b))$ is defined of which the error is then $e_t = (x - x^\star, log(\rho_z))$. ρ_z in this equation can be found using 2-12, where $r = det(H)$, thus requiring an approximation of the Homography matrix and $\boldsymbol{n}$, which is the normal vector of the platform, to be determined using eq. 2-13.

$$\rho = \frac{r}{\boldsymbol{n}^T[x, y, 1]^T} \tag{2-12}$$

$$\boldsymbol{n} = R\boldsymbol{n}^\star \tag{2-13}$$

The $[2 \times 3]$ approach is used in [20] in combination with the virtual camera approach described in subsection 2-1-3, in an MAV application, using image moments. This approach does not require an approximation of z^b, only the calibration parameters of the camera, the performance is however dependent on the size of the platform and the height above it.

A common way of updating the interaction matrix is by using Broyden's method, shown in eq. 2-14, however this algorithm assumes all velocities and rates to be known. Other than with for example a robotic arm visual servoing task, this kind of knowledge is not naturally present in UAV control, requiring extra sensors, such as GPS or requiring integration of the accelerometer data.

$$L_n = L_{n-1} + \frac{(x_n - x_{n-1} - L_{n-1}v_n)v_n}{v_n^T v_n} \tag{2-14}$$

2-1-3 Layer 3

The final subdivision of the literature is described in this section, divided over the two branches to be split up: Reinforcement Learning and Interaction Matrix approaches.

Deep Learning Pose Estimation

The relative pose to an object can be found using a deep neural network trained on image data such as in [3] and [28]. Both use a pre-trained neural network for object recognition as the first layer of their network. The networks converge well and generate accurate estimations, even for noisy images, occlusion and variable lighting conditions. Worth mentioning is the fact that the controller in [3] is trained on one image, which is rotated and displaced into a range of configurations, creating a set of relative poses to be determined from it. However as the research objective describes, this thesis focusses on visual servoing towards an unknown target, meaning no such training will be possible for the task.

Deep Learning State Space Estimation

In [2] the outcome of a State Space estimation from Hebbian learning is fed into the RL controller. Based on known self-motion of the robot and visual signals, a space coding (binary representation) of the environment is created. The approach is biologically inspired; the algorithm presents the environment in a similar fashion as the hippocampal place cells in rats. The network grows incrementally with every interaction of the robot with the environment. The robot takes 4 views rotated 90^o w.r.t. the last, at a set of locations, thus creating a panoramic view from those points, whose locations are estimated from integrating the velocity measurements of the robot. The robot has an exploration policy, varying the exploratory behaviour depending on the familiarity of the robot with the current position.

Deep Learning Monocular Depth Estimation

In [36] the output of a supervised learner for depth estimation is fed into a reinforcement learning controller on a remote control car, using only one camera. The image is divided into windows, for which texture energies and gradients are calculated, to then be combined with the windows they form a vertical stripe with. The texture gradient of every stripe then gives an indication of the proximity of objects in that section of the field of view.

Virtual Camera approach

In [20] a PID controller is used to make a drone servo to two markers on the ground, using the "virtual camera approach". The virtual camera approach assumes small Euler angles and a flat surface and projects the images on an image plane parallel to the surface using knowledge of $[\theta, \phi]$ as shown in Fig. 2-2. This approach is also applied in [53] to create a backstepping controller for a hover task above a plane at different heights, while the objects in view move slightly. It should be noted that in most of these researches the UAV only has to hover above the markers, without a change in Z, rendering the task slightly simpler. Using a servoed camera always pointing parallel to the inertial z-axis, it is possible to hover above a moving platform with this approach as is done in [44].

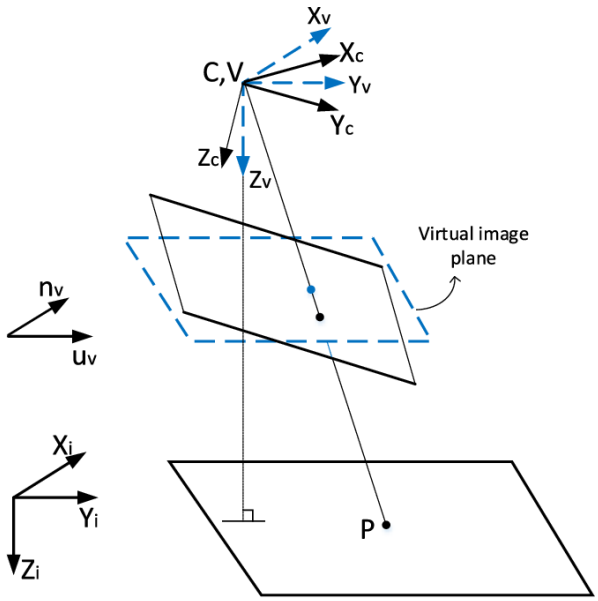


Figure 2-2: Virtual Camera Plane from [53]

Where in [53], θ & ϕ are assumed to be negligible, [20] represents a more robust control algorithm, where the image is corrected for its rotation around both these angles first. This research uses image moments as mentioned in subsection 2-1-1. Image moments originally come from the field of object recognition, because they can be used to define the shape of an object independent of shape or rotation around the camera's optical axis. [11] introduces the use of image moments in visual servoing, for exactly those reasons. From K points in this coordinate system, all centered image moments μ_{ij} are found using eqs. 2-15 & 2-16, where this last one can also be applied to y^v to get $\bar{y^v}$. These μ_{ij}'s can either be used to construct L as proposed in [47] & [11] or to use as the to be controlled features $\boldsymbol{s}$ using eqs. 2-17, 2-18 & 2-19, where the λ's are the focal lengths in pixels. On this $\boldsymbol{s}$, the control inputs can be based as shown in eqs. 2-20, 2-21 & 2-22 as is done by [20], or the higher order control described by eqs. 2-23, 2-24, 2-25 & 2-26 as proposed by [47] & [11], which is only to be used on symmetric objects. An advantage of this last approach is that it does not require an approximation of the MAV's velocity in the virtual camera frame or PD controller on $\boldsymbol{s}$.

$$\mu_{ij} = \sum_{k=1}^{K} (x_k^v - \bar{x}^v)^i (y_k^v - \bar{y}^v)^j \tag{2-15}$$

$$\bar{x}^v = \frac{1}{K} \sum_{K} x_k^v \tag{2-16}$$

$$s_1 = s_3 \bar{x}^v \qquad s_2 = s_3 \bar{y}^v \tag{2-17}$$

$$s_3 = \sqrt{\frac{\mu_{20}^\star + \mu_{02}^\star}{\mu_{20} + \mu_{02}}} \tag{2-18}$$

$$s_4 = \frac{1}{2} tan^{-1}\left(\frac{2\mu_{11}}{\frac{\lambda_2}{\lambda_1}\mu_{20} - \frac{\lambda_1}{\lambda_2}\mu_{02}}\right) \tag{2-19}$$

$$u_T = -k_{h2}(v_3^v/k_{h1} - s_3 + 1) \tag{2-20}$$

$$u_\theta = -k_{l2}(v_x^v/k_{l1} - s_1) \qquad u_\phi = -k_{l2}(v_y^v/k_{l1} - s_2) \tag{2-21}$$

$$u_{psi} = k_{psi}/(J_3 s_4) \tag{2-22}$$

Where J_3 is the inertia around z and the gains k are positive constant gains, with requirements for stability as described in [20]. A positive point about this approach is that it does not require the platform to be flat, as long as its tilt angle is known.

$$s_x = (c_2c_3 + s_2s_3)/K \qquad s_y = (s_2c_3 + c_2s_3)/K \tag{2-23}$$

$$c_3 = c_1^2 - s_1^2 \qquad s_3 = 2s_1c_1 \qquad K = \frac{I_1 I_3^{\frac{3}{2}}}{\sqrt{S}} \tag{2-24}$$

$$c_1 = \mu_{20} - \mu_{02} \qquad c_2 = \mu_{03} - 3\mu_{21} \qquad s_1 = 2\mu_{11} \qquad s_2 = \mu_{30} - 3\mu_{12} \tag{2-25}$$

$$I_1 = c_1^2 + s_1^2 \qquad I_2 = c_2^2 + s_2^2 \qquad I_3 = \mu_{20} + \mu_{02} \tag{2-26}$$

Where S is an approximation of the surface occupied by the object in the camera view.

[11] describes how pure IBVS can be performed using high order features based on image moments. These high order terms are added because low order features $s = [x_g, y_g, a, \alpha]^T$ (center of gravity, object area and object orientation; s_4 from eq. 2-19), induce a coupling of ω_x and v_y, as well as ω_y and v_x. With ω_x and ω_y known however, these high order features might not be necessary, reducing noise and computation time. In fact, $\omega_x^\star$ and $\omega_y^\star$ can be taken out of the equation (eq. 2-9) entirely, since a velocity command will be sent to the velocity controller in paparazzi and the desired angular rate will be deduced from this desired cartesian velocity, which is of course different for other types of robots which do not have a coupling between rotations and translations in the body frame. Image moments are only meaningful when they are applied to planar objects oriented parallel to the camera frame, hence it can not be considered seperate from the virtual camera approach for MAVs.

Reference Interaction matrix

In order to use the interaction matrix approach, as described in subsection 2-1-2, an approximation of d is required, which can be circumvented by finding $\frac{v_z}{z}$ from optical flow measurements as done by [40], or by simply taking $d = Z = d^\star$. This approach is of course only stable for a small range of d around $d^\star$.

2-1-4 Event Based VS

Recently, event based cameras have been used for visual servoing tasks in robotic arm applications [21] & [23]. Usually the AER event representation system, also used by [43], is employed as a bridge between the sensor and the controller. The same software is used by [14] to balance a pencil on its tip with feedback from two DVS sensors in a range of light conditions. The cameras are used to create a 3D model of the pencil in the environment, based on which a PD controller outputs its control signals.

2-1-5 Following a moving platform

In [35] an MAV uses an IBVS approach to follow a moving platform. It uses visual features s_i, belonging to an object to be tracked, to define a visual error signal δ_1. Now by tuning the control gains, a specific task can be performed in an accurate and stable manner. A platform on the ground with 4 distinguishable points are used in this research, where one of those points is displaced vertically to test the applicability of this control scheme to a non-planar target. The gains are however tuned for a platform of a certain size, the controller works for a range of platform sizes around this calibration size, but the solutions is far from scale independent.

2-1-6 Visual Homing

An interesting area of research related to Visual Servoing is that of Visual Homing; an insect inspired approach to vision based navigation. The navigation solutions presented in this research area are based on the concept of snapshot matching, where the position of features in the current image are compared to those in the reference image, with the help of a compass. The compass can be used to prevent the robot from accepting an almost identical image in the opposite configuration to the reference configuration [49]. If for example a robot car is supposed to drive along a road into a certain configuration as in [37], the image created by the robot rotated 180^o around its z-axis is more similar to the reference image than it is for the larger part of the rotations around this axis. The compass then prevents the navigation algorithm from accepting this local optimum, which is a typical problem encountered especially in Root Mean Square (RMS) methods for the gradient of the image difference as for example implemented in [52].

2-2 Tracking

The research objective states the MAV is to land on an a priori unknown platform, which rules out the possibility of using active LED markers as for example used in [10]. More generalizable approaches, making use of the fact that contours are readily available from the DVS output, will be investigated in this section. Starting with approaches used on conventional frame based cameras in subsection 2-2-1, followed by current methods used with event based cameras in subsection 2-2-2.

2-2-1 Frame Based Contour Tracking

In frame based tracking, color is usually an important discriminator. Since color data is not available when one is using a DVS, these color based algorithms are excluded from the literature review. Instead contour tracking algorithms are investigated, for which the DVS, with its natural edge detection, would be a good fit as a sensor.

The Hidden Markov model used in [13] uses a simple Bayesian network to recognize the patterns it has been trained on. As a Markov process, it uses knowledge of the last state to find the probability of the next observed state. Using a profile model, the Active Shape

Model in [4] alters the suggested shape to match strong edges. In [42] a feature matching approach combined with a trajectory coherence check is used to track the movement of objects. This requires modelling the target's movement and filtering the measured motion based on that model.

In [46] an MAV is flown through a gap in a wall. Through the gap the background shows, with a lower camera frame velocity than the wall, making it possible to track the gap based on a difference in camera reference frame velocity.

2-2-2 Event Based Tracking

The tracking algorithms used in the Visual Servoing research mentioned in subsection 2-1-4 perform event clustering based on:

- Spatial location
- Time stamp

This means a 3D map has to be saved to compare the time stamps of events, though such an approach does not necessarily require a lot of memory, as shown by [32]. A similar clustering approach to object detection is presented in [50], where the velocities of the corners, found using the algorithm presented in [51], are compared to see which corner belongs to an independently moving object. This way corners are associated to an object, however this requires the object to always be moving w.r.t. the environment, the same goes for the clustering algorithm presented in [19]. The software used and described by [51] can be found on github and run on a simulated dataset as described in [39], where two corner detection algorithms for event cameras are described and their performance is compared in a range of environments: FAST and Harris corner detection.

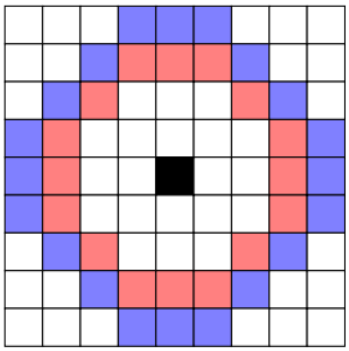
For the event based Harris corner detector a binary surface Σ_b is defined, where $\Sigma_b(\boldsymbol{p}) = 1$ if from the last N events an event occurred at $\boldsymbol{p}$ and $\Sigma_b(\boldsymbol{p}) = 0$ otherwise. Then using the gradient of this surface in a window W around the current event, one is able to find those $\boldsymbol{p}$'s for which a contrast exists in both the x and y direction using eq. 2-28, where M is defined in eq. 2-27.

$$M(\boldsymbol{e_i}) = \sum_{\boldsymbol{e} \in W} g(\boldsymbol{e}) \begin{bmatrix} \frac{\Sigma_b(\boldsymbol{e})}{dx}^2 & \frac{\Sigma_b(\boldsymbol{e})}{dx}\frac{\Sigma_b(\boldsymbol{e})}{dy} \\ \frac{\Sigma_b(\boldsymbol{e})}{dy}\frac{\Sigma_b(\boldsymbol{e})}{dx} & \frac{\Sigma_b(\boldsymbol{e})}{dy}^2 \end{bmatrix} \tag{2-27}$$

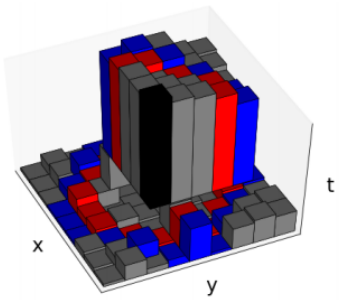
$$R(\boldsymbol{e_i}) = det(M) - k \cdot trace^2(M) = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{2-28}$$

Where λ_1 & λ_2 are the eigenvalues of M. Other than with traditional Harris corner detection however, Σ_b is only defined around recently detected corners, which saves storage and processing of unnecessary data, but also requires the corners to stay in view.

The adaptation of the FAST corner detector uses the Surface of Active Events (SAE) whose value is the timestamp of the most recent event at each pixel as described by eq. 2-29. Two rings of pixels are defined around the current event as shown in Fig. 2-3a & 2-3b, one with radius $r = 3$ (red) and one with $r = 4$ (blue). On each ring the algorithm searches



(a) Pixel grid with the two rings in red and blue



(b) Visualisation of SAE for a corner in the upper right corner

Figure 2-3: Event-based FAST Corner detection from [39]

for a continuous segment (arc) where the $SAE(\boldsymbol{p})$ is significantly higher, meaning an event occurred more recently, than on the rest of the ring. For the inner ring, this arc must have a length between 3 and 6 pixels, for the outer ring it must be between 4 and 8. When this condition holds for both rings, the event is classified as a corner.

$$SAE(x, y) = t_i \tag{2-29}$$

An advantage of the FAST method over the Harris corner detector is that it does not require the calculation of a derivative, which is both computationally expensive and noisy. This is the main reason why it processes events around 15 times faster than the Harris method, which on the other hand performs better in highly textured environments.

The Iterative Closest Point algorithm presented in [41] links pixels to an object based on the distance between a point on a surface of active pixels and that of the average position of the model representing the object to be tracked. Such an approach however, requires the size of an object to stay approximately the same. A more adaptive approach w.r.t. scale is the multi-Kernel approach described in [27], where using one Kernel would be more appropriate for real-time application onboard an MAV, considering processing power limitations. Creating a Kernel after an approximation of the shape of the object, might cause a single Kernel tracker to perform as well as a multi-Kernel tracker, which is able to track any kind of geometry. The principle is the same: the mean of an object is moved over the camera coordinate system and the approximated size of the object is updated based on the last approximation of the mean, the locations of new events and the expected geometry of the object.

In [54] an iterative approach is used, finding optical flow and image plane position estimates of object features from corners detected from the events. Worth noting is the variable window size, based on the optical flow measurements of the object to be tracked.

[7] presents a simple particle tracking approach, where the probability of an event belonging to a certain object tracker, is modelled as a bivariate normal distribution as in eq. 2-30. When the probability P is above a certain threshold the tracker position μ is updated based on the update law in eq. 2-31. To see if a tracker is still useful or if it can be discarded,

an activity function $A_i(t)$ is created for every tracker.

$$P_i(\boldsymbol{p}) = \frac{1}{2\pi}\|\beta_i\|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{p}-\boldsymbol{\mu_i})^T\beta^{-1}(\boldsymbol{p}-\boldsymbol{\mu_i})} \tag{2-30}$$

$$\mu = \alpha\mu + (1-\alpha)\boldsymbol{p} \tag{2-31}$$

The tracker in [31] moves around the screen based on a calculation of the center of mass of a detected object. Similar to the variable window size in [54], both the cluster size and the event number threshold N_{ev} are changed during runtime for this algorithm, based on event detections in the extension of the cluster and the number of events detected within the cluster per period of time respectively. This requires background events to be excluded first, using the spatial and temporal correlation of the events.

A cluster based object tracking approach to traffic tracking is presented in [33], the position of the cluster is moved around based on incoming events, identified as belonging to that cluster. This approach is shown to work well when a set of objects move w.r.t. a static camera. Which is also the application type of the similar tracking algorithm presented in [18].

Another option is to combine the high speed tracking ability of the DVS with conventional object recognition methods on the frames from the accompanying frame based camera, as is done by [48].

2-3 Controlled Landing Strategies

Performing a controlled landing can be achieved by reducing the (downward) vertical velocity with height. A smooth landing is when height decreases logarithmically, which can be achieved by keeping $\frac{v_z}{Z}$ constant. PBVS and other servoing approaches for which depth is known, typically use different strategies, because direct height control is possible.

For other approaches however this relation can be approximated by finding the optical flow divergence. A downward facing camera is usually chosen, when the target is located below the MAV. With a downward facing camera the relations in eq. 2-32 regarding optical flow are used to describe an MAV's movement above a flat surface, after correcting for rotational rate effects [15].

$$\vartheta_x = \frac{v_x}{Z_0} \qquad \vartheta_y = \frac{v_y}{Z_0} \qquad \vartheta_z = \frac{v_z}{Z_0} \tag{2-32}$$

A controller based on this philosophy is created in [25], where the problem of instabilities caused by a delay and noise in the calculation of the divergence D is pointed out. To counter this effect, the control gain k is a logarithmic function of time, as h is supposed to be.

In [30] an approach is presented to perform SLAM using distance estimation based on these oscillations, described in [16]. The adaptive control pipeline from this thesis is shown in Fig. 2-4, where the horizontal control is scaled using the pseudo-scale λ_i' as described by eq. 2-33, where α is a positive constant.

$$\lambda_i' = \frac{K_i}{\alpha} \tag{2-33}$$

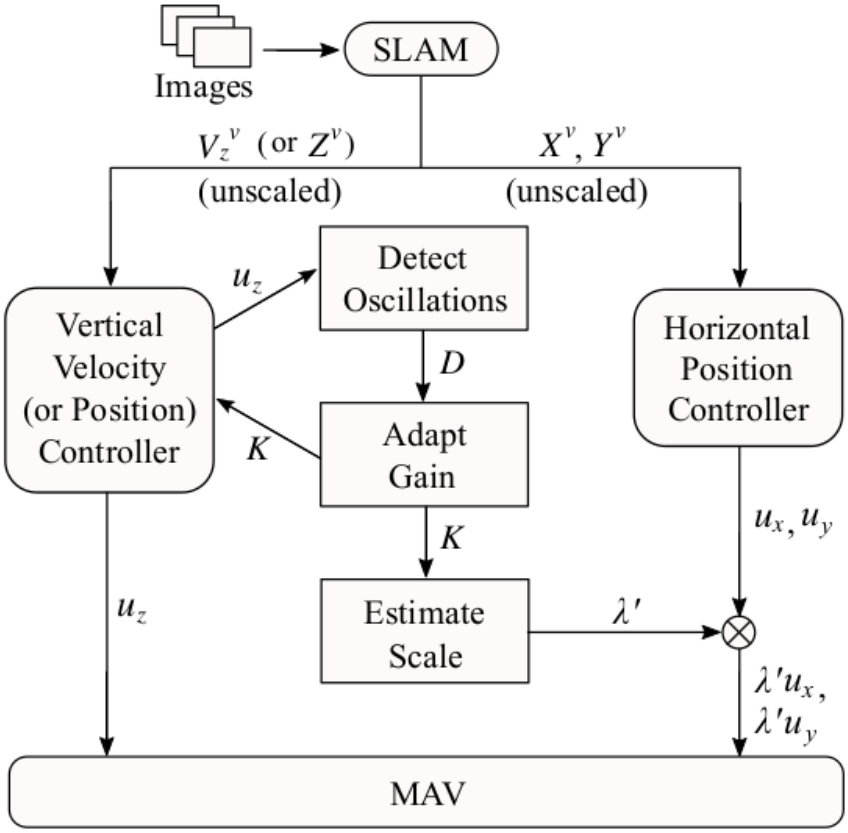


Figure 2-4: Control pipeline from [30]

2-3-1 Event Based Optical Flow

In [43] an event based optical flow estimation algorithm is created and used to perform a smooth landing with an MAV, based on local gradients of Σ_e, which is the same as the surface of active events described in section 2-2-2. The method is first introduced in [6] to track a line on a rotating disk using a static camera and involves the creation of a surface Σ_e, which is defined in eq. 2-34, where t is the last time stamp at which an event occurred at the pixel located at $[x, y]$. Now using the gradient of this surface, the local flow velocity can be found using eq. 2-35, which is locally regularized to compensate for missing events.

$$\Sigma_e(x, y) = t \tag{2-34}$$

$$\nabla\Sigma_e = (\frac{1}{v_x}, \frac{1}{v_y})^T \tag{2-35}$$

In [43] the approach is slightly altered to express the flow velocity as the normal vector to Σ_e in eq. 2-37 expressed in function parameters of the surface $\Pi = [p_x, p_y, p_t, p_0]^T$ around the pixel $[x_i, y_i]$ described in eqs. 2-36. Strong edges sometimes cause multiple events in quick succession, leading to an overestimation of the flow velocity. To prevent this, a refractory period $t_R = 0.3s$ after a pixel's last emitted event is introduced, when events are not accepted

from that location.

$$p_x x_i + p_y y_i + p_t t_i + p_0 = 0 \tag{2-36}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\|\nabla \Sigma_e\|^2} \nabla \Sigma_e = -\frac{p_t}{p_x^2 + p_y^2} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{2-37}$$

Chapter 3

Discussion

As mentioned in the introduction a visual servoing approach will be used to achieve the research objective. Now since the objective states the MAV is to land on a platform of unknown dimensions, a PBVS approach will not suffice. An HVS approach is selected because, as mentioned in section 2-1, the angular readings are provided at a high rate anyway and using them creates a more stable controller.

Performing deep learning pose, state space or depth estimation is out of the scope of this thesis, also there are few working examples of this approach on MAV's. The projects in which visual servoing is done based on the outcome of one of these algorithms also only show performance in a single environment. The fact that this type of control requires training in every new environment renders it a less attractive approach.

For this thesis an HVS Interaction Matrix approach using image moments will be used, because it is a proven concept for MAV control. Also it is possible to test this approach using the Visual Servoing Platform from Inria.

For tracking, it makes sense to use an existing event-based algorithm, because creating a more efficient one from current frame based solutions is out of the scope of this thesis.

The simplicity of the approach in [7] makes it attractive, however a bivariate normal distribution does not seem like the proper way to represent a moving object as seen by an event based camera. Mainly object edges are visible by a DVS, so the center of a flat object is unlikely to be visible, while a bivariate normal distribution gives the highest probability at its center. The optimal value of β depends on the width of the object in view, meaning this value should change with height. To do this an approach such as in [31] can be used, where the tracker window size depends on event detections in the extension of the cluster and the number of events detected within the cluster per period of time. When such an approach is used, the object should be probabilistically represented as a line rather than a plane. Preferably, this line should depend on the outline of the object, as viewed by the DVS, so the events induced by a textured background can be excluded. The

activity function to be used for such a tracker can simply be a threshold; when a tracker is inactive for more than two seconds it is reinitialized near the average position of all trackers. Also instead of calculating the probability P of an event belonging to a tracker for every event, a threshold distance can be calculated from the threshold probability and the current window size value β. That way only the distance between the event and the tracker has to be calculated, saving processing power w.r.t. calculating the probability for every event.

The algorithm to be tested in this thesis is to use a FAST corner detection algorithm on the dataset, to find the location of a set of corners and use their average as an approximation of the middle of the platform. A window will be defined around every corner to perform corner detection on, thus reducing the calculation effort. It should be noted that the initial corner detection is not necessarily performed on event data, but could be performed on images from the frame based camera that is included in the DVS.

Since this project will not make use of a PBVS approach a constant optical flow divergence strategy is chosen for performing a controlled landing using vision. To prevent instabilities, the gain for this constant divergence controller is varied with height, which is estimated based on oscillations, as described in section 2-3. Using a logarithmic k as a function of time as proposed by [25], will not be done, because the MAV might be required to stop descending for a period of time, in order to accurately track the platform. Such an approach can however be used in between estimations of h.

Chapter 4

Initial Investigations

This chapter describes the findings of the initial investigations performed parallel to the literature review laid down in Ch. 2. The three fields briefly looked into first in this preliminary phase are: the DVS data, control architecture and corner detection.

After creating the DVS, the University of Zurich also created a DVS simulator. The simulator can be used to create an event dataset, compressed into a '.bag' file, from a Blender scene and a camera trajectory. This simulator can be used to get an idea of the type of data generated by the DVS and the '.bag' files can be used as inputs to test the suggested corner tracking algorithm, depth estimation and visual servoing control loop. For all of these applications, this simulator is of great use, because other than with real world data, the relative and absolute positions of every object in view is precisely known at every timestamp. An example of what the output of the simulator looks like, can be found in the appendices.

A Homography based controller, with the task to servo into a configuration where a detected object is in the center of the camera view, was created in Matlab's Simulink and can be found in the appendices. Most important is that the HVS controller is occasionally fed with a delayed height estimate. This delay is introduced because height will either be: asynchronously updated based on the oscillatory behavior of the MAV or be updated based on a calculation of the derivative of the divergence; which in any case requires some time to update.

Also an attempt was made to create a Harris corner detection algorithm in Python on a dataset from the simulator described above. However the detector described next was found early on in the process, this detector already includes a working interface with '.bag' files, visualization tools and two working detectors in $C++$: a Harris and a FAST corner detector.

More elaborate investigations are performed into converting the University of Zurich's corner detection algorithm into a corner tracking algorithm and using Lagadic's ViSP to simulate a vehicle controlling based on an IBVS approach.

The event corner detection algorithms described in Ch. 2 have been created and integrated with the DVS simulator by the Scaramuzza group at the University of Zurich. This corner detection algorithm is altered in this preliminary thesis in order to track the corners of an

object initiated at the center of an object in view. Ideally this initialisation is done by the user by a click and drag approach to define a window in which to look for corners of the object to track, but for now this initialisation is done by defining an initialisation location for the centroid and the radius to search around it for the also predefined number of corners.

The working principle of this tracker is visually explained in Fig. 4-1, which shows how step by step the events are excluded from further processing, ultimately excluding 97% of the events because they are either not a corner or do not belong to the object to be tracked. Clearly the size of the search window, which is initialised by the user, should be updated based on the estimated height. A separate loop makes sure that all inactive trackers are reinitialized at a random location close to the center of the tracked object, because in a dynamic situation, unused trackers make it possible for other corners, that appear close to the initial center of the object at a later timestamp, to be identified as a corner to be tracked. Another measure implemented to prevent this is that a corner is discarded when it is further away from the centroid than twice the distance of the second most distant corner belonging to the object. Figure 4-2 shows the corner tracker applied to a dataset with several figures displayed on a table, with the tracked corners in blue and red.

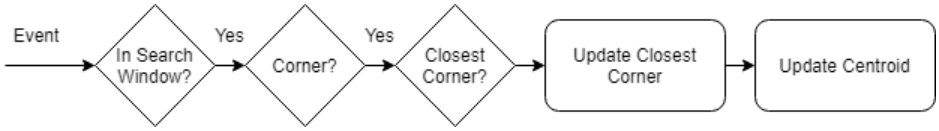


Figure 4-1: Corner tracker logic

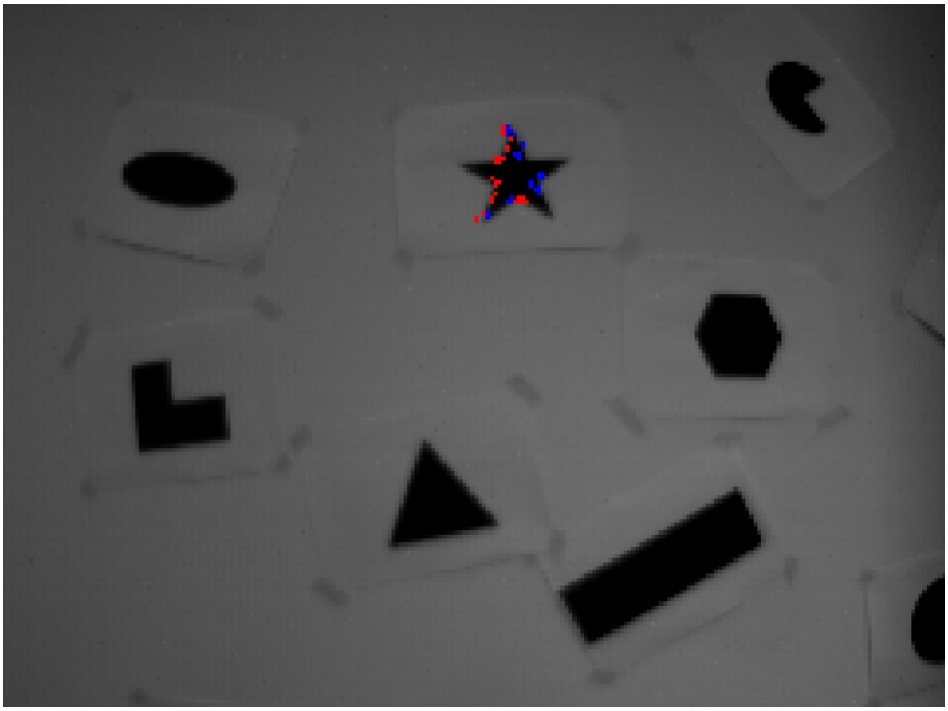


Figure 4-2: Corner tracker applied to dataset with various static shapes and a moving camera

Another area of investigation is Lagadic's Visual Servoing Platform (ViSP), for which the

use as a simulation tool and onboard application on a robot is inquired. The pre-made IBVS simulation, shown in Fig. 4-3, of a robot servoing to center four points in the camera reference frame, can be used as a baseline for this thesis. In the figure, the green lines are the trajectories of the points in the camera view (on the left) and the world view (on the right).

Figure 4-3: IBVS simulation in ViSP

However a lot still has to be done in order to simulate the situation where an MAV servos in order to center an object in view based on a number of detected corner points. For that a quadcopter dynamic model has to be inserted as a vehicle model, which is not readily available in ViSP and will have to be found elsewhere. The fact that this IBVS simulation is not based on an MAV model, introduces another issue: derotation of the image (the Virtual Camera approach) has to be implemented too; the simulated robot does not have to rotate in order to translate.

Finally when the approach has shown to perform in simulation, the controller can be implemented on board an MAV. For this it is beneficial that the ViSP architecture has shown to work for a range of robots in visual servoing tasks and that the source code for these projects is available on github.

Chapter 5

Research Plan

This chapter describes the technical and experimental work to be performed in this thesis.

The question to be answered by this research is the following:

Research Question
Is it possible to design a controller for a quadopter MAV, using only angular information from an IMU and visual data from a Dynamic Vision Sensor, performing an autonomous landing with constant optical flow divergence on a moving, flat platform of unknown dimensions, without any knowledge of the environment?

This question is constructed out of the smaller sub-questions:

Sub-Question 1
Can a smooth landing be performed on flat a platform of unknown dimensions, without any knowledge of the environment?

Sub-Question 2
Can a moving platform of unknown dimensions be tracked accurately using a Dynamic Vision Sensor?

Sub-Question 3
Can an MAV perform a 3D control task, solely based on angular information from an IMU and 2D image offsets?

Sub-Question 4
Can the solutions to Sub-Questions 1,2 & 3 be combined into one solution within their combined limitations?

The objective of this research follows logically from the research question:

Research Objective
To design a controller for a quadcopter MAV, based only on angular information from an IMU and visual data from a Dynamic Vision Sensor, enabling autonomous landing with constant optical flow divergence on a moving, flat platform of unknown dimensions without any knowledge of the environment.

Just as logically do the sub-goals follow from the sub-questions:

Sub-Goal 1
Design a constant optical flow controller, able to land an MAV on a moving flat platform of unknown dimensions, without any knowledge of the environment.

Sub-Goal 2
Design an object tracking algorithm, able to track a moving platform of unknown dimensions based on DVS events.

Sub-Goal 3
Design a controller able to servo into a certain configuration w.r.t. some object in view, based on angular information and 2D image offsets.

Sub-Goal 4
Create an onboard MAV controller with the combined functionalities described in Sub-Goals 1,2 & 3.

When sub-goal 4 is reached, the first onboard event-based visual servoing MAV, capable of landing on a moving platform, has been created.

Creating a working prototype of a new concept such as proposed in this project proposal is a time consuming exercise. Rather likely, problems will be met along the way, involving the implementation of the created algorithms onboard the MAV with a data stream from the DVS. However the controller as implemented in [43] will be available for use as a starting point in this project.

Rein de Vries provided the MAV-lab with a DVS from Insightness A.G. to be used for this project. His plan is to start his own company around a visual servoing MAV using a DVS. Which would potentially be a lighter and more accurate solution with a higher update rate and dynamic range than the same product using a frame-based camera for vision.

The tasks to be performed to meet the research objective are distributed over the following three categories: T_{Si}: involving the creation of a Simulink controller, T_{Ei}: converting Event data to corner coordinates and T_{Oi}: Onboard tasks, combining and implementing T_{Si} & T_{Ei}. The tasks are listed below:

Task (T_{S1})
Create a Simulink controller able to servo into the goal configuration for a static object

Task (T_{S2})
Create a Simulink controller able to keep within error range of the current configuration w.r.t a moving object

Task (T_{S3})
Create a Simulink controller able to perform constant divergence landing on a static platform using a decaying gain controller based on height approximation using observed oscillations

Task (T_{S4})
Combine the controllers created in T_{S1},T_{S2} & T_{S3} to come to a Simulink controller that meets the research objective

Task (T_{E1})
Create a corner detection algorithm which can be used on a simulated event dataset

Task (T_{E2})
Create an object tracking algorithm using the corners detected in a simulated event dataset

Task (T_{E3})
Edit the corner detection algorithm, such that it works on a dataset created using the DVS

Task (T_{E4})
Use the algorithms created in T_{E2} & T_{E3} to create an object tracking algorithm for DVS data

Task (T_{O1})
Implement the algorithm created in T_{E4} in the controller in paparazzi, to show the corner coordinates of the object to be tracked

Task (T_{O2})
Make the rotor's rpm depend on the corner coordinates

Task (T_{O3})
Convert the controller created in T_{S4} into C-code, integrated in Paparazzi

Task (T_{Final})
Tune the controller such that it meets the research objective

Clearly all T_{Oi} tasks build up to T_{final} which is the task to achieve Sub-goal 4 and with that the research objective. Sub-goal 1 & 3 are reached when all T_{Si} tasks are finished and the T_{Ei} tasks work towards reaching sub-goal 2.

All T_{Si} will be performed in Matlab's Simulink. For which a start has been made early on in the project. All T_{Ei} tasks will initially be performed in Python, after which the code will be transformed into C-code for use in the embedded tasks (T_{Oi}). Using TU Delft's Paparazzi, the code created in T_{E4} & T_{S4} will be gradually implemented on the MAV in the T_{Oi} tasks. Paparazzi contains libraries which can be used to test the code in small steps, using for example the OpenCV libraries.

The tests to conclude the T_{Oi}'s can be performed using several data visualization interfaces included in paparazzi. For example writing results in the terminal and showing corner coordinates on screen using the Open CV interface.

The conclusion of T_{final} is an experiment in TU Delft's CyberZoo, attempting a landing on an unknown platform at a random position. If this is completed successfully, the platform will be placed on a remote controlled car to test its ability to land on a moving platform.

The DVS event data to be worked with in this thesis is of the form $e = [x, y, t, p]$, where the polarity p indicates whether the pixel has gotten lighter or darker, the position $[x, y]$ is given as the pixel number to the right and below the left upper corner of the camera screen and time stamp t is nanosecond accurate. The simulator outputs these as a bag of events for every second, while the DVS outputs the events grouped into $100\mu s$ data packages. For the experiments linked to T_{Si}, the values $M_{Si,j}$ by which the performance will be measured are:

Measurable ($M_{S1,1}$)
Relative convergence speed $\frac{\Delta t}{\|\boldsymbol{x}^w_{MAV} - \bar{\boldsymbol{x}}^w_p\|}$ for a range of $\frac{S_p}{z^w_{MAV} - z^w_p}$

Measurable ($M_{S1,2}$)
Mean steady state error

Measurable ($M_{S2,1}$)
Mean position error for range of platform velocities $\|\dot{\boldsymbol{x}}_p\|$ and maximum acceleration $\|\ddot{\boldsymbol{x}}_{p,max}\|$

Measurable ($M_{S2,2}$)
Maximum position error for range of platform velocities $\|\dot{\boldsymbol{x}}_p\|$ and maximum acceleration $\|\ddot{\boldsymbol{x}}_{p,max}\|$

Measurable ($M_{S2,3}$)
Maximum platform velocity for stable controller

Measurable ($M_{S3,1}$)
Vertical velocity at landing

Measurable ($M_{S3,2}$)
Minimum platform size

Measurable ($M_{S4,1}$)
$M_{S2,3}$, $M_{S3,1}$ & $M_{S3,2}$

Measurable ($M_{S4,2}$)
Mean offset at landing

Measurable ($M_{S4,3}$)
Maximum offset at landing

Where S_p is the area of the platform. For the event-based experiments the performance will be measured using the following parameters:

Measurable ($M_{E1,1}$)
Maximum ratio of distance and corner separation size at which two corners can still be distinguished

Measurable ($M_{E1,2}$)
False positive ratio in proximity of an actual corner for a textured background

Measurable ($M_{E2,1}$)
Maximum flow velocity $\|\frac{\dot{\boldsymbol{x}}}{z}\|$ at which a corner can still be accurately tracked

Measurable ($M_{E2,2}$)
Maximum acceleration for which a corner can still be accurately tracked

Measurable ($M_{E2,3}$)
Average estimation error for a range of velocities and accelerations

Measurable (M_{E3})
$M_{E1,1}$ & $M_{E1,2}$

Measurable (M_{E4})
$M_{E2,1}$, $M_{E2,2}$ & $M_{E2,3}$

Finally the performance during the onboard experiments is evaluated using the following measurables:

Measurable ($M_{O1,1}$)
Maximum approximation error of corner coordinates

Measurable ($M_{O1,2}$)
Mean approximation error of corner coordinates

Measurable ($M_{O1,3}$)
Delay in approximation of corner coordinates

Measurable (M_{Final})
$\sum_{i,j} M_{Si,j}$

The values $\sum_j M_{O1,j}$ are found by comparing the output of the algorithm to OptiTrack data.

As can be seen from the Gantt chart in the appendices, implementation onboard the MAV in the first place requires working processing blocks for the DVS events ($M'_E s$). Since a start has already been made on the $M'_S s$ and an event dataset has been created, the first priority is to start working on the M_E milestones.

The thesis work is divided over two periods, with a three week holiday break in between. The first part consists of finishing the literature review and working on the T_{Si} & T_{Ei} tasks. After that the T_{Oi} & T_{Final} tasks are performed in combination with writing the Mid-Term report and describing the outcomes in the final thesis. To be able to follow this planning, a couple of issues should be tackled early on, such as finding the following values:

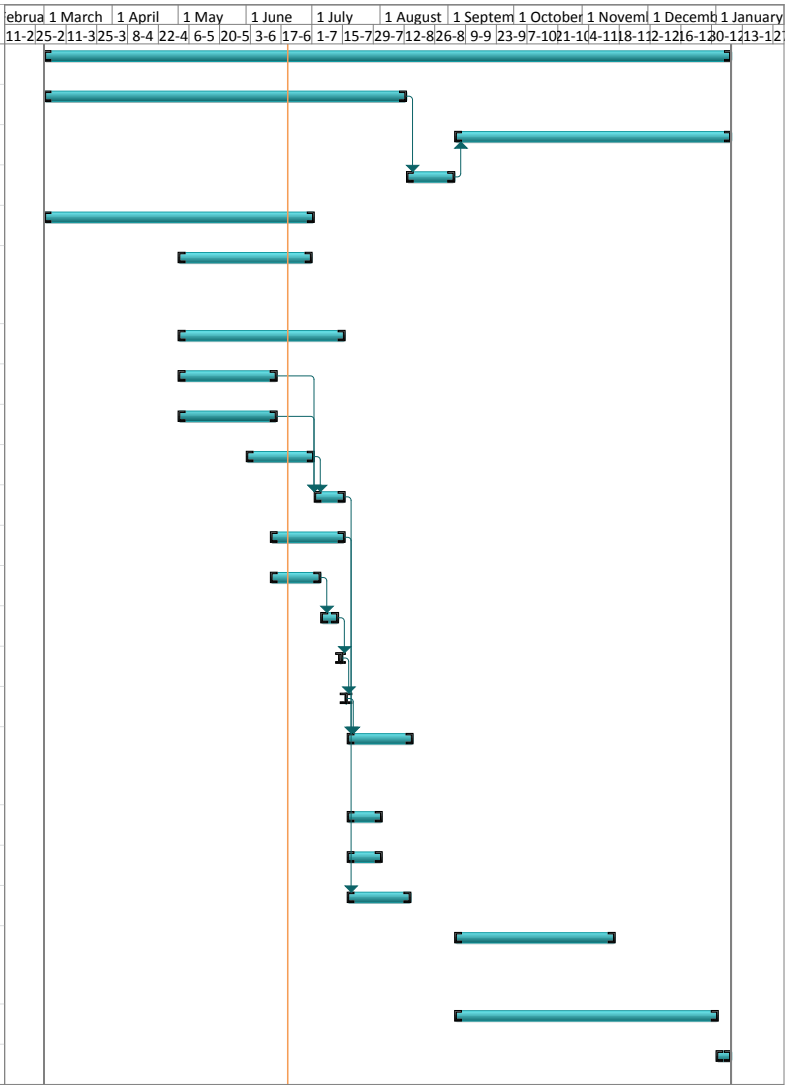
- Mass and Inertia
- Actuator and Measurement Delay
- Maximum Thrust and Moments τ

Which are required in order to properly model the control of the MAV, just as the following software, hardware and facilities are required to successfully perform all the tasks:

- ☑ Install Paparazzi on Ubuntu
- ☑ Install OpenCV and DAVIS simulator
- ☑ Access to CyberZoo, MAVLab
- ☐ Access to the DVS sensor
- ☐ Access to an MAV with DVS integrated with paparazzi

Appendices

ID	Task Mode	Task Name	Duration	Start	Finish
1		MSc. Thesis	11,15 mons	Thu 1-3-18	Mon 7-1-19
2		Thesis part 1	118 days	Thu 1-3-18	Sun 12-8-18
3		Thesis part 2	90 days	Tue 4-9-18	Mon 7-1-19
4		Summer Holiday	16 days	Mon 13-8-18	Mon 3-9-18
5		Literature Study	88 days	Thu 1-3-18	Sun 1-7-18
6		Research Methodologies	45 days	Tue 1-5-18	Sat 30-6-18
7		Simulink Simulations	55 days	Tue 1-5-18	Sun 15-7-18
8		TS1	33 days	Tue 1-5-18	Thu 14-6-18
9		TS2	33 days	Tue 1-5-18	Thu 14-6-18
10		TS3	22 days	Fri 1-6-18	Sun 1-7-18
11		TS4	11 days	Mon 2-7-18	Sun 15-7-18
12		Event data processing	25 days	Tue 12-6-18	Sun 15-7-18
13		TE1	17 days	Tue 12-6-18	Wed 4-7-18
14		TE2	6 days	Thu 5-7-18	Thu 12-7-18
15		TE3	2 days	Fri 13-7-18	Sat 14-7-18
16		TE4	1 day	Mon 16-7-18	Mon 16-7-18
17		Onboard Implementation	22 days	Tue 17-7-18	Wed 15-8-18
18		TO1	12 days	Tue 17-7-18	Wed 1-8-18
19		TO2	12 days	Tue 17-7-18	Wed 1-8-18
20		TO3	21 days	Tue 17-7-18	Tue 14-8-18
21		Final tuning and experiment (Tfinal)	53 days	Tue 4-9-18	Thu 15-11-18
22		Reporting	86 days	Tue 4-9-18	Tue 1-1-19
23		Prepare defence	5 days	Tue 1-1-19	Mon 7-1-19



```
BagMessage(topic='/dvs/events', message=header:
  seq: 1
  stamp:
    secs: 0
    nsecs:  18000000
  frame_id: ''
height: 180
width: 240
events:
  -
    x: 238
    y: 134
    ts:
      secs: 0
      nsecs:   1834204
    polarity: False
  -
    x: 168
    y: 151
    ts:
      secs: 0
      nsecs:   1852118
    polarity: True
  -
    x: 166
    y: 154
    ts:
      secs: 0
      nsecs:   1908270
    polarity: True
  -
    x: 234
    y: 155
    ts:
      secs: 0
      nsecs:   1915492
    polarity: False
  -
```

Figure 1: Example dataset created by the DVS simulator

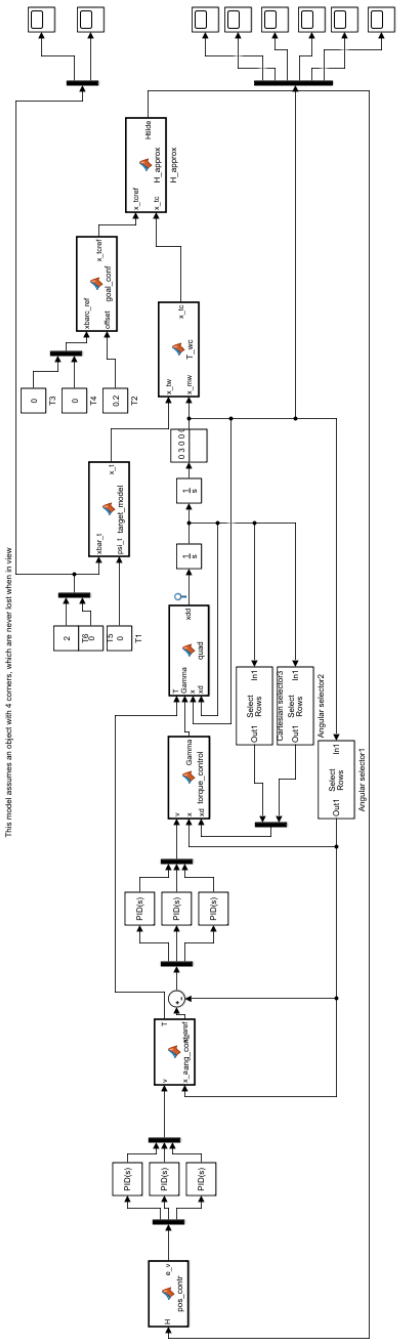


Figure 2: Homography based controller in Simulink

Bibliography

[1] Insightness AG. Evk portfolio brochure, 2017.

[2] Angelo Arleo, Fabrizio Smeraldi, and Wulfram Gerstner. Cognitive navigation based on nonuniform gabor space sampling, unsupervised growing networks, and reinforcement learning, 2004. BR,U,D,L.

[3] Quentin Bateux, Eric Marchand, Jurgen Leitner, Francois Chaumette, and Peter Corke. Visual servoing from deep neural networks, 2017. BR,S,S,L.

[4] A.M. Baumberg and D.C. Hogg. An efficient method for contour tracking using active shape models, 1994.

[5] Selim Benhimane and Ezio Malis. Homography-based 2d visual servoing, 2006. CI,R,S,H.

[6] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozzi. Event-based visual flow, 2013.

[7] Xavier Berthelon, Guillaume Chenegros, Nicolas Libert, Jose-Alain Sahel, Kate Grieve, and Ryad Benosman. Full-field oct technique for high speed event-based optical flow and particle tracking, 2017.

[8] Pascual Campoy, Ivan F. Mondragon, Miguel A. Olivares-Mendez, and Carol Martinez. Visual servoing for uavs, 2010. CI,U,D,H.

[9] Tiago Gomes Carreira. Quadcopter automatic landing on a docking station, 2013.

[10] Andrea Censi, Jonas Strubel, Christian Brandli, Tobi Delbruck, and Davide Scaramuzza. Low-latency localization by active led markers tracking using a dynamic vision sensor, 2013.

[11] Francois Chaumette. Image moments: a general and useful set of features for visual servoing, 2004.

[12] Francois Chaumette and Seth Hutchinson. Visual servoing and visual tracking, 2014. C(IPH),N,S,(NR).

[13] Yunqiang Chen, Yong Rui, and Thomas S. Huang. Jpdaf based hmm for real-time contour tracking, 2001.

[14] Jorg Conradt, Raphael Berner, Matthew Cook, and Tobi Delbruck. An embedded aer dynamic vision sensor for low-latency pole balancing, 2009.

[15] G.C.H.E De Croon, H.W. Ho, C. de Wagter, E. van Kampen, B. Remes, and Q.P. Chu. Optic-flow based slope estimation for autonomous landing, 2013.

[16] Guido C. H. E. De Croon. Bioinspiration and biomimetics, 2016. BS,U,S,O.

[17] Henry De Plinval, Pascal Morin, Philippe Mouyon, and Tarek Hamel. Visual servoing for underactuated vtol uavs: a linear, homography-based approach, 2011. CI,S,S,H.

[18] T. Delbruck and P. Lichtsteiner. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system, 2007.

[19] Lukas Everding and Jorg Conradt. Low-latency line tracking using event-based dynamic vision sensors, 2018.

[20] Geoff Fink, Hui Xie, Alan F. Lynch, and Martin Jagersand. Experimental validation of dynamic visual servoing for a quadrotor using a virtual camera, 2015. CH,U,S,N.

[21] G.J. Garcia, J. Pomares, F. Torres, and P. Gil. Event-based visual servoing with features prediction, 2014. CI,R,S,N.

[22] L. Gerstmayr. Image-based visual servoing, 2009. Lecture slides.

[23] P. Gil, G.J. Garcia, C.M. Mateo, and F. Torres. Active visual features based on events to guide robots, 2014. CI,R,D,N.

[24] Tarek Hamel and Najib Metni. A uav for bridge inspection: Visual servoing control law with orientation limits, 2007.

[25] Hann Woei Ho, Guido De Croon, Erik-Jan Van Kampen, and Q.P. Chu. Adaptive control strategy for constant optical flow divergence landing, 2016. BS,U,S,O.

[26] Hamed Jabbari, Giuseppe Oriolo, and Hossein Bolandi. An adaptive scheme for image-based visual servoing of an underactuated uav, 2013. CI,S,S,N.

[27] Xavier Lagorce, Sio-Hoi Meyer, Cedric Ieng, David Filliat, and Ryad Benosman. Asynchronous event-based multi-kernel algorithm for high speed visual features tracking, 2014.

[28] Alex X. Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration, 2017. BR,S,S,L.

[29] Daewon Lee, Tyler Ryan, and H. Jin. Kim. Autonomous landing of a vtol uav on a moving platform using image-based visual servoing, 2012. CP,U,S,N.

[30] Seong Hun Lee and Guido C. H. E. De Croon. Stability-based scale estimation of monocular slam for autonomous quadrotor navigation, 2017. BS,U,S,O.

[31] A. Linares-Barranco, F. Gomez-Rodriguez, V. Villanueva, L. Longinotti, and T. Delbruck. A usb3.0 fpga event-based filtering and tracking framework for dynamic vision sensors, 2015.

[32] M. Litzenberger, C. Posch, D. Bauer, A.N. Belbachir, P. Schon, B. Kohn, and H. Garn. Embedded vision system for real-time object tracking using an asynchronous transient vision sensor, 2006.

[33] M. Litzenberger, C. Posch, D. Bauer, A.N. Belbachir, P. Schon, B. Kohn, and H. Garn. Embedded vision system for real-time object tracking using an asynchronous transient vision sensor, 2006.

[34] E. Malis, F. Chaumette, and S. Boudet. 2 1/2 d visual servoing, 2005. CH,R,S,H.

[35] Rafik Mebarki, Vincenzo Lippiello, and Bruno Siciliano. Nonlinear visual control of unmanned aerial vehicles in gps-denied environments, 2015.

[36] Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning, 2005. BR,S,S,L.

[37] Ralf Moller, Dimitrios Lambrinos, Rolf Pfeifer, and Rudiger Wehner. Insect strategies of visual homing in mobile robots, 1998. BS,R,S,N.

[38] P. Morin, Philippe Mouyon, and T. Hamel. Visual servoing for underactuated vtol uavs: a linear, homography-based framework, 2014. CH,U,S,H.

[39] E. Mueggler, C. Bartolozzi, and Davide Scaramuzza. Fast event-based corner detection, 2017.

[40] Titus R. Neumann and Heinrich H. Bulthoff. Insect inspired visual control of translatory flight, 2001. CI,U,D,N.

[41] Zhenjiang Ni, Aude Bolopion, Jol Agnus, Ryad Benosman, and Stephane Regnier. Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics, 2012.

[42] Giorgio Panin, Alexander Ladikos, and Alois Knoll. An efficient and robust real-time contour tracking system, 2006.

[43] Bas J. Pijnacker Hordijk, Kirk Y.W. Scheper, and Guido C.H.E. de Croon. Vertical landing for micro air vehicles using event-based optical flow, 2017.

[44] Marinela Georgieva Popova. Visual servoing for a quadrotor uav in target tracking applications, 2015. CI,S,D,(RLH).

[45] Lorenzo Rosa. Visual servoing for unmanned aerial vehicles, unkown. CH,U,S,(RLH).

[46] Nitin Sanket, Chahat Deep Singh, Kanishka Ganguly, Cornelia Fermuller, and Yiannis Aloimonos. Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight, 2018.

[47] O. Tahri and Francois Chaumette. Point-based and region-based image moments for visual servoing of planar objects, 2005.

[48] David Tedaldi, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. Feature detection and tracking with the dynamic and active-pixel vision sensor (davis), 2016.

[49] Andrew Vardy and Ralf Moller. Biologically plausible visual homing methods based on optical flow techniques, 2005. Is,R,S,N.

[50] V. Vasco, A. Glover, E. Mueggler, D. Scaramuzza, L. Natale, and C. Bartolozzi. Independent motion detection with event-driven cameras, 2017.

[51] Valentina Vasco, Arren Glover, and Chiara Bartolozzi. Fast event-based harris corner detection exploiting the advantages of event-driven cameras, 2016.

[52] J. Zeil, M. Hofmann, and J. Chahl. Catchment areas of panoramic snapshots in outdoor scenes, 2003.

[53] Dongliang Zheng, Hesheng Wang, Jingchuan Wang, Siheng Chen, Weidong Chen, and Xinwu Liang. Image-based visual servoing of a quadrotor using virtual camera approach, 2017. CH,U,D,N.

[54] Alex Zhu, Nikolay Atanasov, and Kostas Daniilidis. Event-based feature tracking with probabilistic data association, 2017.

III

Conclusions and Recommendations

In this research, successful landings were performed using IBVS logic applied to features in the virtual camera frame, combined with an adaptation of optical flow divergence based landing rationale. The algorithm was applied to a range of starting heights and divergence settings in Gazebo simulations in Paparazzi.

These landings were performed solely on textured objects, because the algorithm requires the tracking of at least 2 corners and preferably more, because more corners lead to a more stable approximation of the size increase and translation of the object in the camera view. If not enough texture can be found in the region specified in the camera view, the MAV directly goes into the final landing phase, which means it will descend with a constant thrust setting, slightly lower than what would be required for hover.

From the experiments it is clear that D_b can be used to replace D_{LK} which is the result of global optical flow estimations. Thanks to the small errors on the linear approximation and the overall availability, in contrast to the outcome of the Lukas-Kanade scheme, a well functioning system, robustly servoing and landing on a specified location in the screen, can be created based on D_b.

The results of the D_b experiments would probably be even better when $\frac{v_{z,i}+v_{z,i-1}}{2}$ is used as ground truth D, since that is the value $D_b = \frac{dw}{wdt} = \frac{w_i - w_{i-1}}{w_i dt}$ actually approximates; the approximation is shifted by $\frac{1}{2}T_{frame}$. Another way to improve the approximation power of D_b as $\hat{D}$ would be applying a filter to them based on data from the accelerometer. The effect of assuming the camera to be a perfect convex lens, can further be countered by initializing the window more gradually, by for example only reinitializing features from an outer sub-window in an inner sub-window. When a feature for example moves to a location at 70 % of the image width it can be reinitialized between 0 and 50 %, avoiding a difference in accuracy for different window initializations.

From the experiments it is evident that the controller becomes less stable for lower starting altitudes. This is probably due to the fact that the controller gains are already too high at the start of the tuning process, which seems to agree with the fact that landing phases 1 & 2 are quite short at these low altitudes, meaning oscillations are found after only a slight increase of the controller gains. Reducing the starting gain and the step-wise increase, are therefore likely to increase the performance.

Improving the horizontal performance could also be achieved by putting effort into finding a better fit for the ratio of the horizontal and vertical gain, as the horizontal control seems to be more aggressive than the vertical control. For now the horizontal gain is tuned based on the vertical gain, probably the controller performance would however be better when a similar type of gain tuning is applied to the horizontal gain as is done for the vertical gain, in a separate second oscillation detection phase. This phase could for example have as a goal to keep the object centered in the virtual camera frame. Similar to the vertical gain tuning, the control gain for this task would then be increased until the MAV is clearly oscillating over the object.

A real world experiment was attempted to validate the results acquired from Gazebo simulation. It was however unsuccessful, due to the occasional complete loss of tracked corners between two frames, twice in a row, triggering flight phase 4 e.g. a constant thrust descend.

An increase in performance can also be achieved by a change in hardware: using an event-based camera instead of a regular frame based camera. The Dynamic Vision Sensor (DVS) developed by IniLabs is such an event-based camera, sending the pixel location and timestamp of a pixel changing by more than a threshold in greyscale intensity. A tracking algorithm, similar to the one used throughout this research, was created for event data from such a camera and could be used to replace the current object tracking module. Upsides of using the DVS as a vision sensor are its exceptionally high update rate and its natural edge indication, since edges cause brightness changes in a dynamic environment. This natural edge indication enables corner detection at low processing cost, while the high update rate enables fast corrections for instantaneous changes.

Next steps for application of this new control logic would be to apply it to landing on a moving platform, or a platform with a vertical offset from the background. Because the information taken from the images is restricted to the tracked object, the algorithm should also work for those situations in which the visual queues from the object differ from those belonging to the background.

The DVS seems to simply be a better fit for a computer vision task such as the one described in this paper. If a step was made to apply the strategy to landing on a moving platform, the benefits become even more

evident, especially since the DVS does not suffer from motion blur and is less sensitive to changes in lighting conditions.

IV

Appendices

.1. Flight test height over time

This chapter shows the height over time for an MAV in successful landings in Gazebo simulation using the D_b approach starting from heights $\boldsymbol{h_0} = [2,3,4,5,6]$ with divergence settings $\boldsymbol{D_b} = [0.02, 0.04, 0.08, 0.16, 0.24]$. The runs are combined in one figure per height setting.

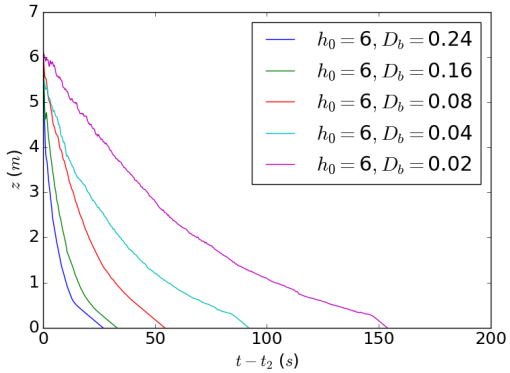


Figure 1: Height of MAV for runs starting from a height of $6m$

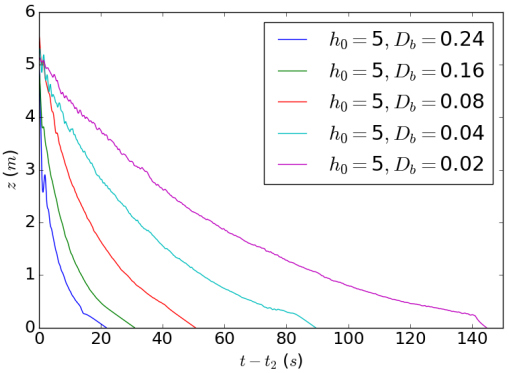


Figure 2: Height of MAV for runs starting from a height of $5m$

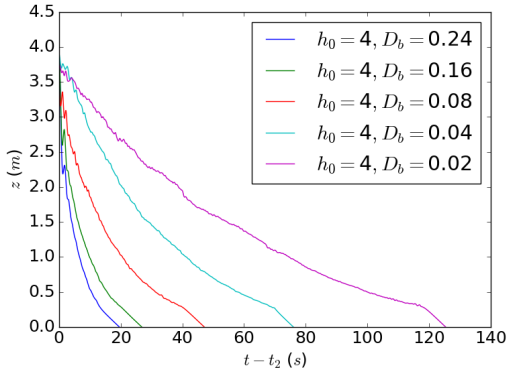


Figure 3: Height of MAV for runs starting from a height of $4m$

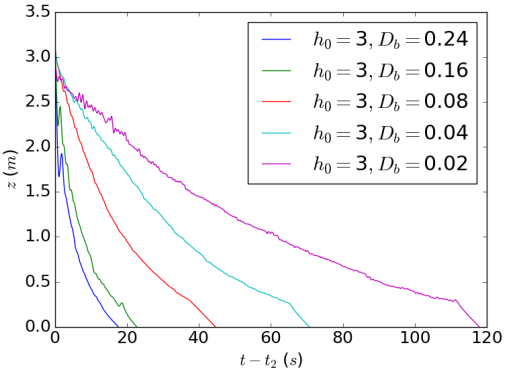


Figure 4: Height of MAV for runs starting from a height of $3m$

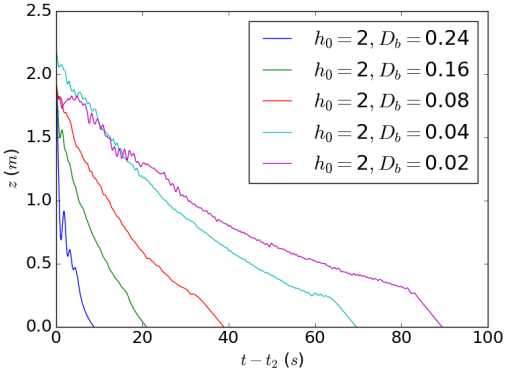


Figure 5: Height of MAV for runs starting from a height of $2m$

.2. Flight test horizontal error over time

This chapter shows the horizontal offset to the landing spot over time for an MAV in successful landings in Gazebo simulation using the D_b approach starting from heights $\boldsymbol{h_0} = [2,3,4,5,6]$ with divergence settings $\boldsymbol{D_b} = [0.02, 0.04, 0.08, 0.16, 0.24]$. The runs are combined in one figure per height setting.

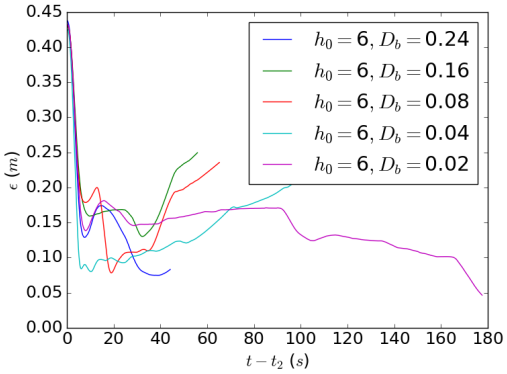


Figure 6: Horizontal offset of MAV to the landing spot for runs starting from a height of $6m$

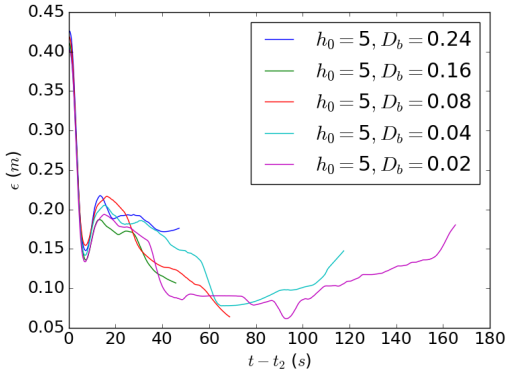


Figure 7: Horizontal offset of MAV to the landing spot for runs starting from a height of $5m$

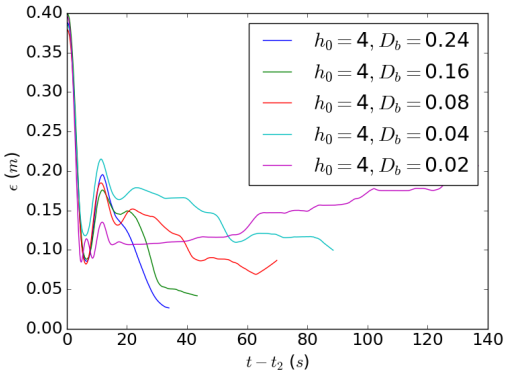


Figure 8: Horizontal offset of MAV to the landing spot for runs starting from a height of $4m$

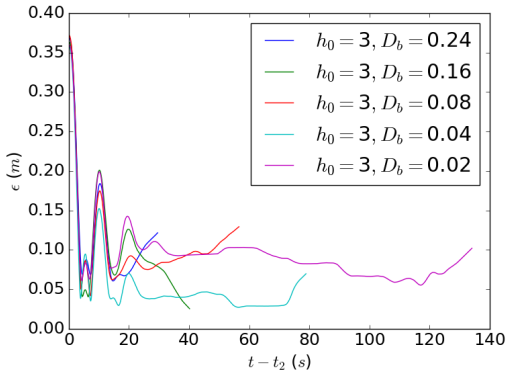


Figure 9: Horizontal offset of MAV to the landing spot for runs starting from a height of $3m$

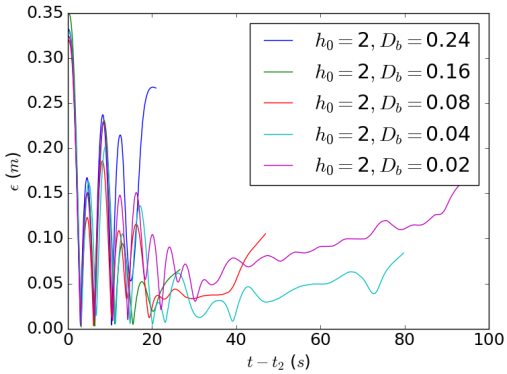


Figure 10: Horizontal offset of MAV to the landing spot for runs starting from a height of $2m$