Dynamic Image Based Visual Servoing with Unmanned Aerial Vehicles

by

Kenny Leung

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Control Systems

Department of Electrical and Computer Engineering

University of Alberta

Abstract

In recent years rotary-wing unmanned aerial vehicles (UAVs) are seeing more uses in applications such as transmission line inspection, event filming, parcel delivery, and search-and-rescue missions. Recent improvements in embedded systems have lead to more research for improving unmanned aerial vehicle (UAV) autonomy. Traditionally for outdoor use, the inertial measurement unit (IMU) with magnetometer, barometer, or global positioning system (GPS) can be used for pose estimation, where the pose estimate is used for motion control. However, in GPS denied environments, such as indoors, other sensors need to be used for acquiring pose of the UAV in the environment. One method to solve for the vehicle's pose is visual servoing, which uses a camera to collect information about the environment, and estimates the UAV's pose for control. This thesis implements the proposed dynamic image-based visual servoing (DIBVS) controller in Fink et al. [1] using a newer experimental platform to improve the performance. This new platform utilizes the Robot Operating System (ROS) running on an on-board Nvidia Jetson TX1 (TX1) mini computer with a FLIR Chameleon 3 camera for the computer vision (CV) system. ROS is chosen to create a simple framework that would handle necessary components for the UAV, such as image capture and data transfer, while allowing for minimal software changes to test different algorithms. Using this framework, future research projects using the platform can save time troubleshooting issues testing hardware, and instead focus on problems related to the algorithms being tested. Also, future upgrades to the platform would need minimal changes to the software so long as the new system is able to run ROS. For the experiments, the CV system uses color to identify visual features used to compute image moments

which are sent to a PX4 flight management unit (FMU) for control of the UAV over a target. The experimental results are compared to the values found in Fink et al. [1], where the new platform is found to have similar performance to the platform used in the reference paper. This result indicates successful implementation of the DIBVS controller on the new platform with a more modular and flexible CV system.

Acknowledgements

This research was made possible by the support of the people in the Applied Nonlinear Controls Lab (ANCL). Of special mention would be my supervisor Dr. Alan F. Lynch for guiding me through the steps necessary to make this research a success. As well as Dr. Geoffery Fink for putting together the ANCL platform and assisting throughout the development of the experiment. Also a special mention to Luyi Yang a visiting student from China to ANCL for his help getting the computer vision part of the experiment completed. Special thanks to Yunzhi Lin a visiting student from China for his help with performing various tests to ensure that the experiment was working on the platform.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years there has been an increase in research conducted on unmanned aerial vehicle (UAV) to replace manned operations in case of dangerous conditions such as disaster relief. Choices for the type of vehicle can vary from fixed wing aircraft, conventional helicopter, multi-rotor, and hybrid types of aircraft where each type has its own advantages and disadvantages [2]. Of interest is the rotorcraft UAV due to the versatility provided by vertical take-off and landing (VTOL) capabilities, high maneuverability, and ability to perform low speed flights or hover. These benefits allow for a wide field of applications such as transmission line inspection, event filming, parcel delivery, and search-and-rescue missions. A survey of looking at different types of research being done on rotorcraft can be found in [3]. Previous members of the Applied Nonlinear Controls Lab (ANCL) group had worked with a conventional helicopter type UAV, such as [4], where it was found that the vehicle is challenging and complex to control and maintain in an academic research environment. Due to these issues, members of the ANCL had developed a quadrotor platform in [5], which is later referred to as the experimental platform. The choice for a quadrotor platform is inspired by the success of other research projects such as Eidgenössische Technische Hochschule Zürich's (ETHZ) Flying Machine Arena (FMA) [6, 7], University of Pennsylvania's GRASP microunmanned aerial vehicle (MAV) test bed [8], and Massachusetts Institute of Technology (MIT) Aerospace Controls Laborartory's (ACL) RAVEN project [9]. The quadrotor platform proved to be beneficial for use in research over the conventional gas powered helicopter UAV used in [4] as it is simpler to maintain, and more stable in flight. Although during the early conception of the quadrotor, it was harder to implement a working platform for the quadrotor due to the requirement of applying a different input signals for each motor to maneuver. However, this problem was resolved when embedded systems were developed enough to be added on-board the UAV and able to translate simple commands, such as inputs from a remote controller, into the required physi-

cal control signals. A unique aspect of a quadrotor is its ability to choose propeller rotation direction. A typical choice for the rotation direction is for adjacent motors to rotate in opposing directions, while opposite motors rotate in the same direction as shown in Figure 2.1. This choice of propeller rotation is important since the rotation direction cancels out the counter torque of the motors, which simplifies the dynamics of the vehicle.

The quadrotor is an interesting platform to study as it is a complex nonlinear underactuated inherently unstable vehicle, where there are six degrees of freedom with only four inputs. A common research area for these vehicles is autonomous motion control, which is difficult due to the necessity for accurate pose estimation. The issue of pose estimation posed a problem for small UAVs for many decades due to the weight limitations combined with the issue of having limited power supply carried on the vehicle. However, with the recent development of smaller, cheaper, and higher precision embedded systems the possibility of including different sensors on-board have been opening up which can supplement or replace the use of global positioning system (GPS) sensors. A typical choice to solve the pose estimation issue used on-board inertial measurement unit (IMU) in combination with magnetometer, barometer, or GPS signals as a method for solving this control issue. Using these sensors in an outdoor environment would be practical to achieve motion control. However, for places without a GPS signal, such as inside a building, other methods must be used to gather information about the position of the vehicle, such as a camera [10]. An on-board camera is a lightweight, low power, and affordable device that can gather a large amount of information about the environment [11]. Using a camera for motion control is known as visual servoing which takes the information gathered from a camera to control the vehicle.

Research for using a camera to recover information about the real world from an image is called computer vision. In computer vision if given multiple images of the same scene, information related to the position of the camera is embedded in the transformation matrix between the two images called a homography matrix [12]. Using a homography matrix any points in one image can be transformed to their position in the second image. The homography matrix is described by nine parameters, where the ninth parameter is a scale factor due to only being able to find information up to a scale factor in images. Therefore, to solve the homography matrix it requires a minimum of four corresponding points in each image to solve for the parameters. Using projective geometry, points being projected onto the image plane are invariant, which means points are projected as points and lines projected to lines. However, parallel properties of lines in the images may not be preserved when using projective geometry as real world parallel lines may intersect at a finite position in the image. In order to preserve the parallel property of lines, an affine

projection must be used, which moves the intersection of parallel lines in the image back to the line at infinity [13]. To compute the line at infinity, two parallel lines must first be extracted from the image. Computing the intersection of the two lines in projective geometry yields a point at infinity. The line at infinity can then be computed from the intersection of a point at infinity with one of the parallel lines. To transform a projective image into an affine image the intersection of the parallel lines must be moved back to infinity. This is done by computing the line at infinity in the image then replacing the last row in the homography matrix with the computed line at infinity. This new matrix is then used to transform the image into an affine projection. Affine projections are useful since relative distances of objects in the real world would be preserved in the image. Transforming to affine projection will now have all parallel lines intersecting at the line at infinity in the image. By knowing a geometric measurement in the image and having the affine image, the projection can be transformed into a geometric projection. The geometric projection is useful since the affine properties are preserved, and measurements in the image are the same as measurements done in the real world. By using the invariant properties from these different projections, it becomes possible to recover information of the real world from the images.

The controller for motion control commonly used is an inner-outer loop structure, and [14] had proven the stability of this controller design for flight control. The inner-outer loop control structure comes from the quadrotor UAV dynamics which can be split into two subsystems, since the rotational dynamics are independent of the translational dynamics. Therefore, the inner loop control handles orientation of the vehicle while the outer loop deals with translation of the vehicle. Due to the coupling of the translational dynamics to the rotational dynamics, the outer loop controller provides reference angles for the inner loop. Due to the frequency of the two different subsystems, visual servoing is a practical implementation to the common inner-outer loop controller used in UAV control. The frequency difference is used in visual servoing from the assumption that the reference signals are tracked perfectly by the inner loop controller [15]. In practice, visual servoing works since the outer loop control uses the computer vision (CV) system which is mainly limited by the frame rate of the camera that usually runs at rates of around 30 Hz, while the inner loop uses the IMU with systems such as a Vicon Motion Capture System (MCS) running at 200 Hz indoors. The relatively higher frequency of the inner loop ensures that the inner loop reaches the reference orientation which is needed to properly control the position of the UAV. The outer loop controller computes the reference angles and the total force of the motors needed, while the inner loop uses the reference angles to compute the input torques needed for pose control. This thesis intends to implement the controller in [1] on new hardware to improve performance of the

controller. The new implementation includes the use of an open source software called Robot Operating System (ROS), loaded onto the Nvidia Jetson TX1 (TX1) embedded system. The TX1 is chosen due to its capability of using a dedicated Graphics Processing Unit (GPU) to process images, which is typically the most computationally taxing process. The choice for using ROS is to have a flexible CV system where different methods, such as shape or color identification, can be used to determine targets in an image with as high a frame rate as the system can handle. ROS is an improvement from the pixy camera used in [1] since the pixy is limited to seven color options for targets, and a maximum frame rate of 50 Hz. Therefore, using ROS on the embedded system removes limitations on target identification, and control updates are less restricted than the platform used in the reference paper.

1.1 Literature Review

There are many research projects on UAV as discussed in [2, 3], where some works focus on the control of the UAV. As noted before visual servoing can be used for controlling the aircraft. Visual servoing is a method of utilizing a camera to gather information on the environment, and feed it back to the controller for motion control. When using a camera for visual servoing the two common methods implemented are the monocular and binocular setup. The binocular setup has the benefit of giving information about depth of the object used to recover the full 3D coordinates. However, this benefit is gone once distance to the object from the camera greatly exceeds the baseline or distance between the two cameras [16]. Therefore, a more generalized approach would be the monocular setup, since the binocular setup is the same as a monocular one for objects far from the camera. The monocular approach is interesting since the 3D coordinate can be recovered up a scale factor, similar to binocular approach, by using multiple images with the assumption that the environment is static through the use of the homography matrix. To solve the scale factor problem, a solution would be to use an external device to get information regarding the depth. Some examples of external devices used are range finder, ultrasonic devices, and IMU used to gather more information about the environment [2].

Visual servoing is split into two main types of approaches Position-based visual servoing (PBVS) and image-based visual servoing (IBVS) [17]. PBVS control extracts the vehicles estimated pose from the images and uses the error from the desired pose to control the vehicle. However, estimation of the vehicle's pose requires a priori knowledge of the target geometry combined with extracted image features to recover the 3D pose. This constraint means research done would need either computer-aided design models of the target like in [18], or other special markers,

such as april tags which gives relative pose from the marker, to recover the pose. However, this constraint would leave the controller unable to operate in common unstructured environments in the world. Another PBVS control method is based on visual odometry (VO) which uses the changes in images over time to estimate vehicle pose with an assumption that the environment is static. Work in [19] utilizes a vision-based extended Kalman filter (EKF) approach combined with IMU data to estimate pose of the UAV relative to a desired target. Another example is [20], which uses PBVS to control landing the UAV by computing pose of the vehicle relative to the landing pad. Other works include [21] which uses a partial pose based visual servoing (PPBVS) to estimate parts of the vehicle's pose relative to the target for tracking infrastructure.

IBVS controllers differ from PBVS controller since the controller is based directly on the image features found in the image. By directly using image features in the control, computations used to find the pose in PBVS can be avoided which may result in faster performance. An example for a IBVS controller is in [21], where the second proposed controller is an IBVS controller based on a LQR servo controller used to track infrastructure. Dynamic image-based visual servoing (DIBVS) is an IBVS control that integrates the vehicle dynamics into the control. Methods of implementing DIBVS include the spherical projection approach, the homography approach, the virtual spring approach, and the virtual camera approach. In [22], it is shown that the triangular, or passivity-like, properties seen in quadrotor dynamics (2.2), are destroyed by applying the perspective projection which couples the angular velocity with the translation velocity. Therefore, [22] introduces a spherical projection approach that can get back the passivity-like properties of the dynamics, where the translational velocity is not coupled with the angular velocity. Using image features, they derive a controller using back-stepping. The advantage to using the spherical projection is that the depth of objects does not need to be known, however, this also results in the projections being insensitive to altitude changes [23]. Work done in [24] uses a modified spherical projection from [22] while rectifying the issue of depth insensitivity.

As mentioned previously, the homography matrix has information related to the camera pose. Work done in [25] uses a set of images corresponding to current and desired pose to compute homographies for the images, which are then compared to a reference image set. The desired trajectory assumes a collection of images along a path the controller will follow are collected ahead of time. The proposed controller only handles the position of the UAV, while not addressing the yaw. Due to homographies having an unknown scale factor related to the depth, [25] applies an adaptive update law to estimate the depth. Using the proposed controller their simulation was found to be globally asymtopically stable (GAS) for the translational

dynamics. Other works using homography approach include [26] where the angular velocity and homography information is used for the derived locally stable control law. Again the target is assumed planar which limits the uses of the control method.

Work in [27, 28] introduces a virtual spring approach which uses image moments to find the interaction matrix for controlling the vertical translation and yaw. The horizontal translation is controlled through increasing image errors causing the vehicle to rotate and fly towards the target. Due to the possibility of rotations putting the vehicle into an unstable state, a virtual spring is applied to limit rotation of the vehicle. The proposed controller gives the benefit of not needing translational velocity which can save on some sensors, and is proven to be locally asymptotically stable. However, the controller suffers from its assumption that features are parallel to the image plane, which would not hold once there is a need for horizontal translations.

The virtual camera approach transforms points from the actual image plane to a virtual image plane. This approach is proposed in [29, 30] where a virtual image plane is assumed parallel to the planar target, and therefore the points roll and pitch dependency is eliminated. In [31], the virtual camera is applied to the image points to account for underactuation of the vehicle. By defining appropriate image features similar to the ones proposed in [32, 33] to track the vehicle dynamics, [31] derive a control law using back-stepping methods. Using the proposed controller, [31] ran simulations and found that the results had shown improvements over spherical image coordinates. A subset of the work in [31] was done by [5] which examined the translational control, and ran the controller on an experimental platform. The results from the experiment had confirmed that the controller was able to correctly control the translational dynamics, as well as following a moving target. The work was later extended in [1] to include the virtual camera and add in the yaw control. The virtual camera is used to restore a passivity-like property for the image kinematics. Work done in [1] extends [31] by implementing the controller on an experimental platform which few works have done. Furthermore, [1] goes on to prove that the proposed controller is GAS for translational dynamics, and globally exponentially stable (GES) for the rotational dynamics. The results from the experiments performed indicated the UAV was able to center itself on the target features and correct its yaw by minimizing the image feature errors. For all the mentioned virtual approach controllers there is an underlying assumption that the linear velocity of the aircraft is known. There is also the assumption that the image features will not leave the field of view, which may occur for certain poses. Other related works include [34] which apply an adaptive control scheme to account for unknown parameters, as well as compensates for errors observed from the experiments conducted in [5]. Work done in [35], includes input saturation to address the

issue of features leaving the field of view. Another work using virtual camera is [23], which implements a virtual camera but differs by using lines in images, instead of points, for the controller design to tackle problems such as power line inspections.

Some other areas of research look at the application of navigating the UAV within its environment. Examples for this type of controller are simultaneous localization and mapping (SLAM) [36, 37], and parallel tracking and mapping (PTAM) [38] which creates a map of the environment and locates the vehicle in the map. This map is built by looking at prominent features in the image and keeps track of the features as they change in successive images, which can later be used to derive the vehicle location. PTAM differs from SLAM by splitting the tasks for tracking and mapping to improve processing times. Difficulties faced by these types of algorithms is the ability to track large maps of the environment, since key frames are kept in order to estimate the location of the vehicle, and therefore require more storage space as the map expands. Other issues are the ability to identify where loop closure should occur, which is crucial to identify where boundaries of the map. If map boundaries are not identified correctly then the vehicle may get incorrect pose estimates as previously found features will be erroneously identified as new features used in the new pose estimation. A ROS implementation of PTAM is used by [39] to test visual state estimation with observers designed using EKF, where the version of PTAM used is based on an improvement by [11]. The work proposed in [11] improves on the PTAM method proposed by [38] in areas such as limiting the number of key frames, improving the feature identification, and re-initialization of the map.

1.2 Thesis Overview

This thesis is separated into two main topics, Chapter 2 describes the modelling and experimental platform, while Chapter 3 covers the controller and results. Chapter 2 begins by describing the quadrotor UAV dynamics used in the simulation as well as controller design. The dynamics are described in two different frames of reference and the different parameters used are defined. After the UAV model, the pinhole camera model is introduced, which models how the points are projected onto an imaging plane. Section 3.2 uses this camera model to compute reference values to compare the experimental values from the CV system. These models are also used in the simulation to verify how well the controllers respond in Section 3.3. After the modelling, a brief description of the experimental platform used for the experiment is included in Section 2.2. Section 2.2 goes over the hardware and software on the Pixhawk autopilot flight management unit (FMU) or PX4, the companion PC, and the camera.

Chapter 3 goes over the controller design, verification of the modelling for the image moment features, and finally the simulation, and experiment of the UAV. Chapter 3 focuses on the outer loop controller of the inner-outer loop control structure, while a PID controller is used for the inner loop. Section 3.1 describes how points are transformed to a virtual camera, and defines the image moment features. An analysis of the proposed image moment features in [32, 33] shows that using the proposed image moments, the passivity-like property from the dynamics can be restored. The image moment features are used with the UAV dynamics to derive a controller from the reference paper [1], and the stability is shown to be stable. Section 3.2 describes the steps taken to verify that the modelling of the camera, described in Chapter 2, matches with the results found using the on-board CV system. The test are carried out on a desktop computer running the ROS with a camera mounted on the desk. Nine tests were carried out to ensure that the image moment features would be computed correctly for use in the controller described before, where two test are carried out to ensure the passivity-like property is restored for the image moment features. For Section 3.3, a simulation is run on simulink to verify that the controller with chosen gains gives a stable flight for a given initial condition. The results from simulation are compared to the results in [1], which showed a stable response. Finally, the controller is run on the experimental platform to test the performance on a real world platform. The resulting flight data is analyzed and compared to the reference paper [1].

1.3 Contribution

The contributions of this thesis are focused on setting up a working easily accessible platform for research to be conducted on UAV. Some main tasks can be described as follows

- Developed a framework for the CV system that is easy to access and implement for future research projects. Using this framework built on ROS the experimental platform controllers can be tested safely on desktops to ensure proper functionality before deployment on the experimental platform.
- Implementing the DIBVS controller proposed in [1] with new hardware on the platform to improve performance of the proposed algorithm. Initial test results indicate similar performance using the new platform with potential for more improvements using the new platform.

Chapter 2

Modelling and Platform

This chapter focuses on the modelling used for the simulations and theory then goes into brief detail on the platform used for the final experiment. In Section 2.1 the models for the unmanned aerial vehicle (UAV) and camera are described. In Section 2.2 the experimental platform hardware and parameters are described.

2.1 Modelling

In this section the quadrotor dynamics are described first followed by the camera model. For this section a cross configuration quadrotor is considered with the body frame oriented such that b_1 indicates the front of the vehicle, b_2 to the right of the vehicle, and b_3 facing down. However, it should be noted that a plus configuration gives the same rotational and translational kinematics with only the torque definitions being different. For simplicity the camera frame $\mathcal{C}$ is considered to coincide with the body frame. In practice there could be an simple transformation required, such as a translation down for a downward facing camera mounted below the chassis. For the camera a pinhole model with perspective projection is used and any distortions in the camera are not considered in the model.

2.1.1 Quadrotor Model Dynamics

The model for the quadrotor is typically described using two frames: the body frame $\mathcal{B}$ and the navigation frame $\mathcal{N}$. The body frame is situated at the center of mass of the quadrotor while the navigation frame is centered around a point on earth, such as the point of takeoff. The frame basis vectors are chosen to follow the right-hand rule as shown in Figure 2.1, where the third axis of each frame is assumed to be positive pointing down. For defining the positive direction of the b_1 axis there are two possible definitions: a cross configuration where the b_1 axis is between motor 1 and 3, and the plus configuration where b_1 points toward motor 1. For defining

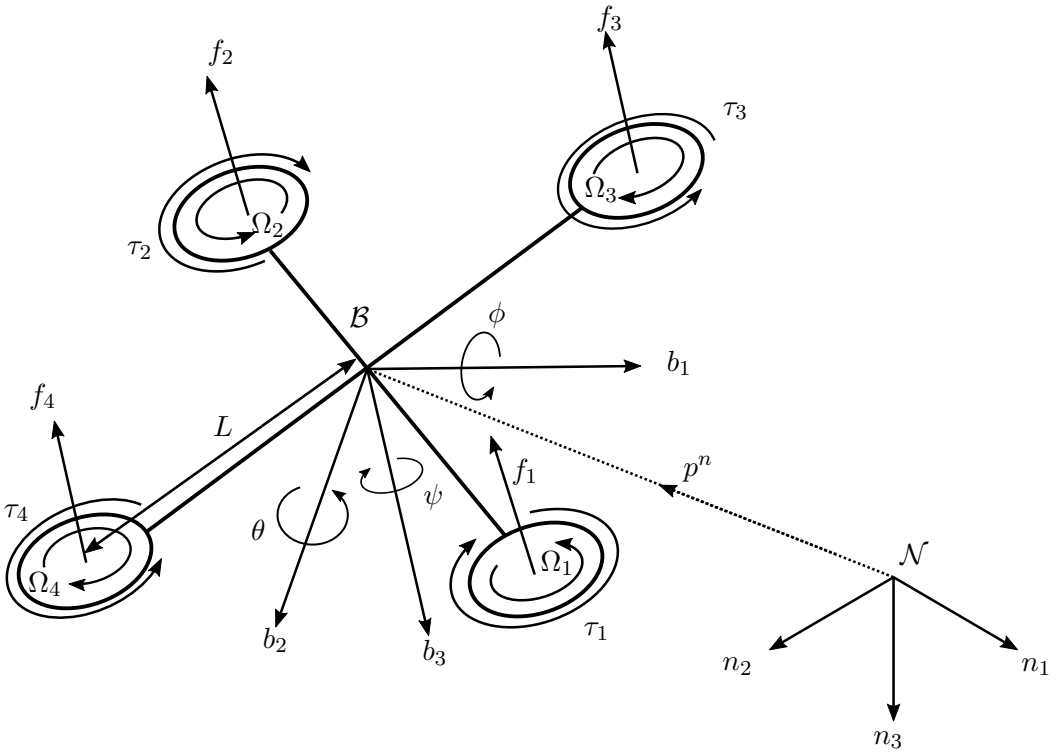


Figure 2.1: Model of quadrotor based on [39]

the directions of roll, pitch, and yaw the right-hand rule is used. Therefore, when looking at the quad from above the roll will be defined positive rolling towards motor 1, the pitch positive with the nose is pointing up, and the yaw rotating in a clockwise fashion. To transform vectors between frames (2.1) introduces a rotation matrix from the body frame to the navigation frame, which is parameterized in ZYX Euler angles.

$$R_b^n(\eta) = \begin{bmatrix} c_\psi s_\theta & c_\psi s_\theta s_\phi - s_\phi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi - c_\phi c_\phi & s_\psi s_\theta c_\phi + c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{2.1}$$

where $R \in SO(3)$, ϕ is the roll, θ is the pitch, ψ is the yaw described by Figure 2.1, $c_x = \cos x$, and $s_x = \sin x$. For transforming points from the navigation frame to the body frame the inverse of the rotation matrix will be used. The vehicle dynamics

that describe the translational and rotational dynamics are as follows:

$$\dot{p}^n = v^n \tag{2.2a}$$

$$m\dot{v}^n = -R_b^n T_M n_3 + n_3 g m \tag{2.2b}$$

$$\dot{R}_b^n = R_b^n \,\mathrm{sk}(\omega^b) \tag{2.2c}$$

$$J\dot{\omega}^b = -\omega^b \times J\omega^b + \tau^b \tag{2.2d}$$

where p^n is the position, v^n is the velocity, g is gravity, m is the mass of the UAV, n_3 is the third basis vector in $\mathcal{N}$, ω^b is the angular velocity, J is the inertia matrix, R is (2.1), $sk(.)$ is the skew symmetric matrix defined by

$$\mathrm{sk}(x) = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$$

and T_M is the total thrust from the four motors. From (2.2) a typical choice for states are $x = [p^T, v^T, \eta^T, \omega^T]^T$, where η is the ZYX Euler angles as defined in Figure 2.1. Using these definitions the translational dynamics are described by (2.2a) and (2.2b), while the rotational dynamics by (2.2c) and (2.2d). It can be seen in (2.2c) and (2.2d) that the rotational dynamics are independent of the translational dynamics. However, the translational dynamics are related to the rotational dynamics in (2.2b) through the rotation matrix. As mentioned in Chapter 1, it is due to this relation between the translational and rotational dynamics that the inner-outer loop control structure is chosen.

For the rotational dynamics, the Euler rates can be found in terms of the angular velocity ω by expanding and simplifying (2.2c)

$$\dot{\eta} = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \omega^b \tag{2.3}$$

where $t_x = \tan x$, and $\eta = [\phi, \theta, \psi]^T$. It can be seen in (2.3) that there is a singularity at $\theta = \pm 90^\circ$, therefore, this state will need to be specially handled based on this parameterization. However, this singularity is not a special physical state and can be safely dealt with by assuming values, such as $\frac{c_\phi}{c_\theta} = 0$, if the singularity is encountered. For the purpose of the controller, the UAV is assumed to be at or near hover, therefore, the singularity can be safely ignored. To simplify the expression for (2.2d), the inertia matrix is assumed to be a diagonal matrix, $J = \mathrm{diag}(J_1, J_2, J_3)$, for a

symmetric UAV which gives:

$$\dot{\omega}^b = \begin{bmatrix} \frac{\omega_2^b \omega_3^b (J_2 - J_3)}{J_1} \\ \frac{\omega_3^b \omega_1^b (J_3 - J_1)}{J_2} \\ \frac{\omega_1^b \omega_2^b (J_1 - J_2)}{J_3} \end{bmatrix} + \begin{bmatrix} \frac{\tau_1^b}{J_1} \\ \frac{\tau_2^b}{J_2} \\ \frac{\tau_3^b}{J_3} \end{bmatrix} \tag{2.4}$$

where ω is the angular velocity, J_i is inertia in the ith direction, and τ is the torque. The torques are defined in (2.5) using the definition for the angle described in Figure 2.2.

$$\tau_1^b = L s_\Theta(-f_1 + f_2 + f_3 - f_4) \tag{2.5a}$$

$$\tau_2^b = L c_\Theta(f_1 + f_3 - f_2 - f_4) \tag{2.5b}$$

$$\tau_3^b = -K_\tau \tilde{\Omega}_1 - K_\tau \tilde{\Omega}_2 + K_\tau \tilde{\Omega}_3 + K_\tau \tilde{\Omega}_4 \tag{2.5c}$$

where L is the length of the arm from the center of mass to the motor, f_i is the thrust from the i^{th} motor, K_τ is a constant, Θ is the angle as defined in Figure 2.2, and $\tilde{\Omega}$ the physical signal to the motor. In practice the physical signal will be handled in a mixer which takes torques as input and correctly sends the required signal to the motors. It should be noted that (2.5) assumes angle Θ is the same between the front facing arms and rear facing arms relative to the b_2 axis.

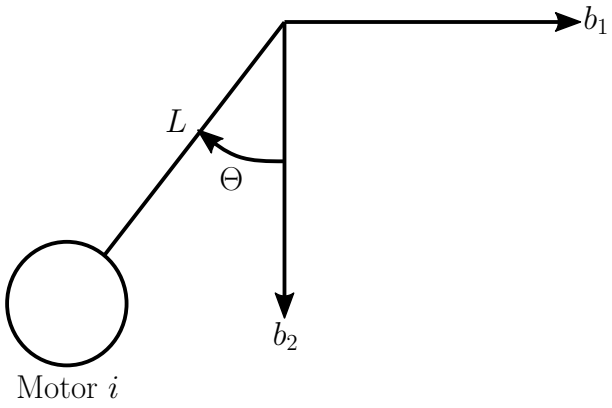


Figure 2.2: Motor Angle Used for Torque

An alternative representation to ZYX Euler angles used to represent the orientation is quaternions. Both Euler angles and quaternions are valid representations, which can be transformed from one to the other. It should be noted that there are two quaternion representations for each Euler representation. Quaternions are a useful representation since there is no singularity when the pitch angle is perfectly perpendicular to the horizon, or $\theta = \pm 90°$. Therefore, quaternions are favored for use in controllers to avoid unexpected behaviour occurring for any possible orien-

tation. To convert Euler angles to the quaternion representation the relation (2.6) can be used.

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} c_{\phi/2}c_{\theta/2}c_{\psi/2} + s_{\phi/2}s_{\theta/2}s_{\psi/2} \\ s_{\phi/2}c_{\theta/2}c_{\psi/2} - c_{\phi/2}s_{\theta/2}s_{\psi/2} \\ c_{\phi/2}s_{\theta/2}c_{\psi/2} + s_{\phi/2}c_{\theta/2}s_{\psi/2} \\ c_{\phi/2}c_{\theta/2}s_{\psi/2} - s_{\phi/2}s_{\theta/2}c_{\psi/2} \end{bmatrix} \tag{2.6}$$

where $q = [q_0, q_1, q_2, q_3]$ is the quaternion. A drawback for using quaternion representation comes from the difficulty of visualizing the physical orientation of the vehicle with just the vectors. Therefore, if quaternions are given (2.7) can be used to convert into ZYX Euler angles.

$$\phi = \arctan\left(\frac{2(q_0q_1 + q_2q_3)}{1 - 2(q_1^2 + q_2^2)}\right) \tag{2.7a}$$

$$\theta = \arcsin\big(2(q_0q_2 - q_1q_3)\big) \tag{2.7b}$$

$$\psi = \arctan\left(\frac{2(q_0q_3 + q_1q_2)}{1 - 2(q_2^2 + q_3^2)}\right) \tag{2.7c}$$

where q is the quaternion.

2.1.2 Camera Model

The camera model used for this thesis is the pinhole model, illustrated in Figure 2.3, which assumes that all rays enter a pinhole before reaching an image plane on the other side. Due to the image being inverted the image plane can be moved in front of the pinhole as shown in Figure 2.3 to simulate the inversion done after the image is captured. For this thesis the objects of interest are described in $\mathcal{N}$ which are then projected onto the image plane π_p^+. This model is used due to the properties such as the mapping of real world points to image points, and the preservation of lines to lines in the image as discussed in 1. By having these properties it is possible to use projective geometry to derive real world information from the image, such as finding parallel lines in real world objects if they are parallel in the image. To extend upon this concept, by knowing something about an object in the real world it also becomes possible to get a real world measurement from the image as explained in Chapter 1. In order to map the 3D world coordinate to the 2D image coordinates a perspective projection is used. The perspective projection means points are scaled by the depth, or distance in the c_3 axis, from the camera. This is done since the image has less information than the real world, therefore, any information that is found in the image can only be found up to a scale factor [12]. After the scaling, the intrinsic parameters of the camera are used to project the points onto the image plane. This projection relation is described by (2.8) where any rotations or translation are

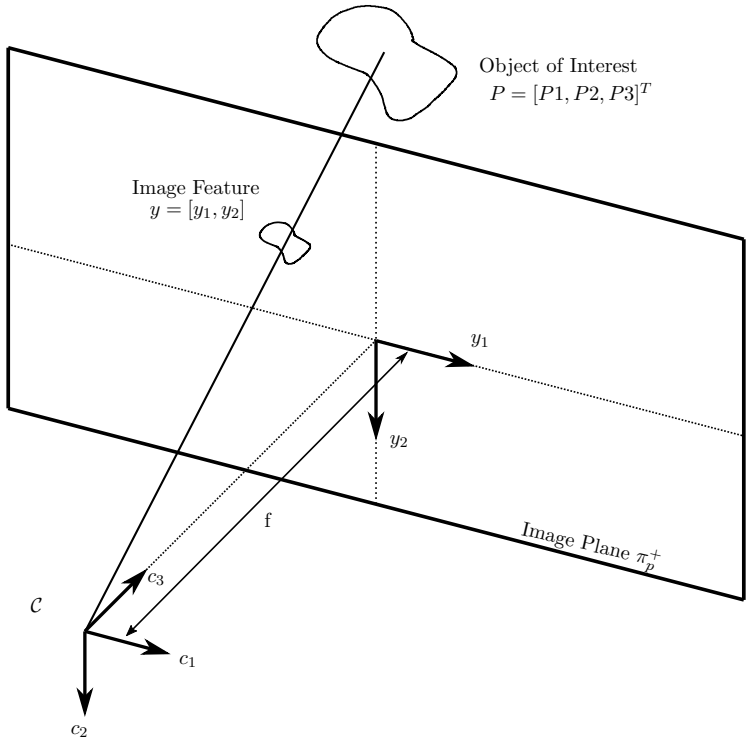


Figure 2.3: Pinhole camera model based on [39], where image feature, y, is a point on object, P, projected onto image plane π_p^+

assumed already be applied to the points.

$$\begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} = H \begin{bmatrix} P_1/P_3 \\ P_2/P_3 \\ 1 \end{bmatrix} \tag{2.8}$$

where (y_1, y_2) is the image feature coordinate, P is the real world coordinate, and H is the projection matrix. The projection matrix describes the intrinsic parameters of the camera and are arranged into a matrix as follows

$$H = \begin{bmatrix} \lambda_1 & s & y_{10} \\ 0 & \lambda_2 & y_{20} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

where λ_i is the focal length in pixels, s is the skew in the image, and y_{i0} is the principal point. The principal point is considered to be the image plane center, and typically assumed to be half the resolution of the image. The skew is a factor which relates to the difference in pixel size along the horizontal and vertical direction. For this thesis, the skew is assumed to be zero since with the advancement of technology most modern cameras are made with square pixels. The focal length in meters is

related to the focal length in pixel through the following relation

$$\lambda_i = \frac{f}{\rho_i} \tag{2.10}$$

where f is the focal length in meters, and ρ_i is the pixel size in meters. The relation (2.10) is useful to know as camera calibration programs, such as the ones used in Section 3.2, are able to directly find the value of λ. Defining $s = [y_1, y_2]^T$ as the image feature, the image kinematics can be found by combining (2.8) with (2.2) which gives

$$\dot{s} = \begin{bmatrix} \dot{y_1} \\ \dot{y_2} \end{bmatrix} = \begin{bmatrix} -\frac{\lambda}{P_3} & 0 & \frac{y_1}{P_3} & \frac{y_1 y_2}{P_3} & -\frac{y_1^2+\lambda^2}{\lambda} & y_2 \\ 0 & -\frac{\lambda}{P_3} & \frac{y_2}{P_3} & \frac{y_2^2+\lambda^2}{\lambda} & -\frac{y_1 y_2}{P_3} & -y_1 \end{bmatrix} \begin{bmatrix} v^c \\ \omega^c \end{bmatrix} \tag{2.11}$$

where (y_1, y_2) is the horizontal and vertical position respectively. The interaction matrix in (2.11) relates the image features to the UAV dynamics in the form

$$\dot{s} = L \begin{bmatrix} v_c^c \\ \omega_c^c \end{bmatrix} \tag{2.12}$$

where L is the interaction matrix.

2.2 Experimental Platform

The experimental platform was built in [39] with inspiration from the success of other research groups platforms as mentioned in Chapter 1. The quadrotor is a custom-built four arm, cross configuration quadrotor frame with the front of the UAV between two blue arms of the aircraft as shown in Figure 2.4. This platform is used since commercial quadrotors are difficult to program or have limited programming to allow for custom controllers. This platform is also built on the principle of using open-source projects [39]. The angle between the two front arms or back arms, along the b_1 direction, is 120°, while the angle between the front arm and back arm, along the b_2 direction, is 60°. For this thesis, the platform consist mainly of three components: the Pixhawk autopilot flight management unit (FMU) or PX4, the companion PC, and the camera on the UAV. The data flow between the components is shown in Figure 2.5, which uses a protocol called mavlink to translate data sent between the systems. The computer vision (CV) system gets images from the camera and orientation from the PX4 which are used to compute image moment features. The computed image moment features are then sent to a Robot Operating System (ROS) package called mavros which handles communication with the PX4. The PX4 combines rotational data from the inertial measurement unit (IMU),

translation velocity from the Motion Capture System (MCS), and the image moment features to compute the control inputs for motion control. The environment also includes a Vicon MCS that uses eight cameras with a desktop PC to compute information about the vehicle's pose. A 12.2 V LiPO battery is used to power all the motors, and systems on-board the aircraft, which gives the aircraft approximately 15 minutes of operational time with motors running before needing replacement. A custom-built power distribution board made in [39] is used to power the PX4, the companion PC, and the camera.

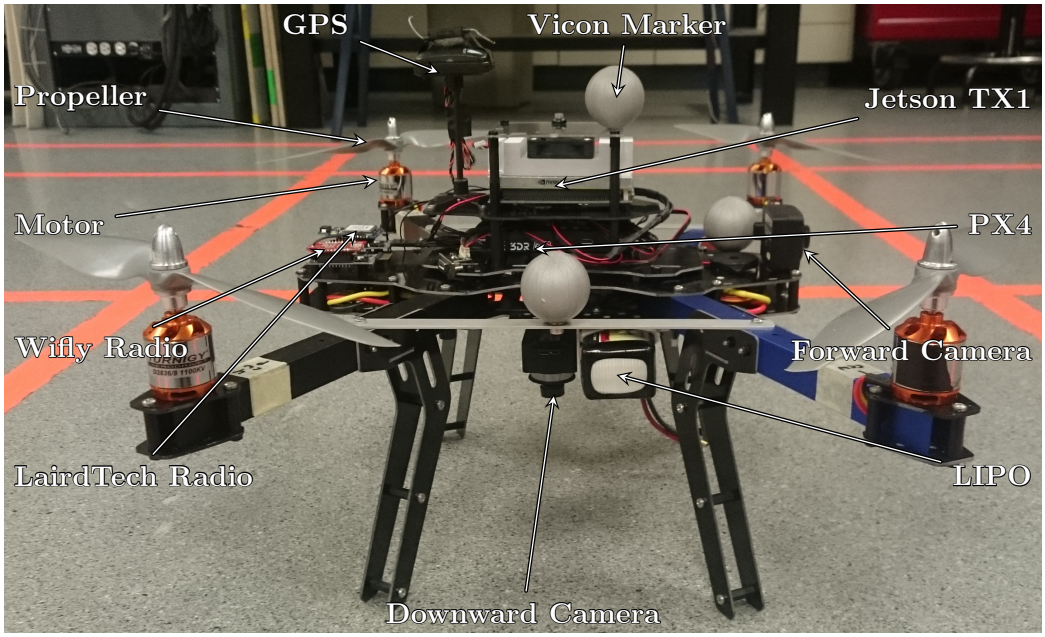


Figure 2.4: ANCL experimental platform taken from [39], showing the various components of the UAV

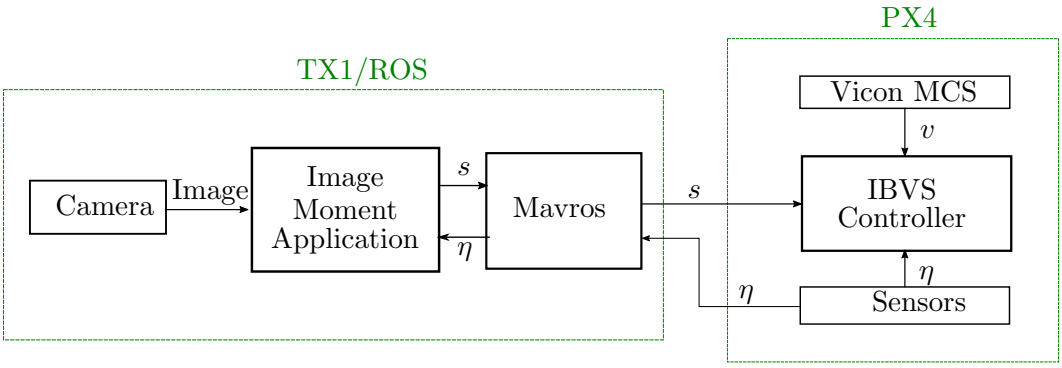


Figure 2.5: Data flow showing the communication between ROS and PX4 to control the UAV

Parameter	Value
L	0.28m
m	1.6kg
Θ	30°
J_1	0.03kgm^2
J_2	0.03kgm^2
J_3	0.05kgm^2
λ	414pixel
y_0	(256,320)pixel

Table 2.1: ANCL experimental platform physical parameters

2.2.1 Flight Management Unit

The navigation system is a Pixhawk autopilot FMU, or PX4, originally created by [40], which controls four motors mounted on the end of each arm of the aircraft. The PX4 takes inputs from the Vicon MCS, IMU, and CV system to compute control signals for the experiment. The PX4 uses data streams called topics to publish data, which any application can subscribe to read the data. Topics are useful when running multiple applications that need to read the same set of data, which is used in Section 3.4 for the comparison of outputs between a known working position controller, and the proposed image-based visual servoing (IBVS) controller for tuning purposes. A logger is set up on the PX4 to begin logging all the data that is published on the topics when the vehicle is armed. This log continues until the vehicle enters an unarmed state. As described in Chapter 1, the PX4 has an inner loop controller that controls the vehicle's attitude, and an outer loop controller that does position control using the inputs from the MCS. A Vicon MCS running on a ground station computer sends the pose of the aircraft to the PX4 through a laird tech radio for pose control. This communication runs with a frequency of 100 Hz, which is shown to be the optimal frequency in [39] for sending data. This frequency is necessary for the controller design described in Chapter 3, where it is assumed the inner loop can perfectly track the reference. The Vicon MCS computes pose through the use of special vicon markers, placed on the aircraft, based on how each camera in the room sees these markers. The data sent from the MCS includes position from the room center in meters, velocity in meters per second, and orientation in quaternions.

2.2.2 Companion PC

The aircraft has a Nvidia Jetson TX1 (TX1) mini computer which handles the CV system, and sends data to the PX4 over a serial connection. The TX1 is an embedded system that runs with a custom version of Ubuntu 16.04 ARM operating

system called Linux4Tegra, that comes from Nvidia's Jetpack software development kit (SDK). In terms of hardware the system has a quad-core ARM Cortex-A57 Central Processing Unit (CPU), and an Nvidia Maxwell 256 CUDA core Graphics Processing Unit (GPU). The TX1 is placed on an Auvidia J100 carrier board[1], which also provides two universal serial bus (USB) 3.0 ports, and is mounted on top of the UAV. One of the USB ports is connected to the downward facing FLIR (previously known as Point Grey) Chameleon 3 camera[2], while the other USB port has a FTDI cable connected to the PX4 for communication purposes. A wireless adapter is also included on the TX1 to communicate with a ground station. On the software side, the TX1 is loaded with ROS that handles running applications, known as packages in ROS, as well as the communication between applications or other computers running ROS. ROS is used in many robotics application and CV systems which allows for ease of development and deployment of code. The development and deployment of ROS is used in Section 3.2 to test the CV system before the experiment done in Section 3.4.

One task that ROS handles is the images taken by the camera, the FLIR Chameleon 3, using a third party ROS package to grab each frame. By using ROS, the tests done in Section 3.2 and Section 3.4 use the same image processing code in a custom ROS package to compute the image moments, and send data to the PX4. The image processing ROS package used in this thesis utilizes the open source OpenCV library for processing of the images, and identifies objects based on hue, saturation, and value (HSV) thresholds when looking at pixels. A binary image is created by setting pixels with HSV within a desired threshold range to one, otherwise the pixel is set to zero. This binary image will show bright portions for the objects and dark for any other pixel. Next morphology operations, or shape processing, is carried out in an attempt to remove possible specular highlights, also known as bright spots, or other erroneous holes detected in the object. The operations carried out scans an area around the pixel in question and assigns the maximum value for erode operations or minimum value for dilate operations. For this thesis, the operation uses a combination of alternating erode and dilate operations to remove holes in objects, while trying to undo any unwanted effects done to the edges of markers. This operation is carried out to make the detected object more robust to noise. The modified binary image is then processed using OpenCV libraries which follow the algorithm in [41] to find the contours of the objects. This is done by a raster scan, row by row from top to bottom and left to right scanning the image and identifying borders from changes in adjacent pixels values, i.e. changes from one to zero or zero to one. Border pixels found are marked with a number to identify

[1]https://auvidea.com/j100/
[2]Part no.:CM3-U3-13Y3C-CS

which contour the pixel belongs to. An assumption for the ROS package is that no objects in the image will leave the field of view, and the largest N objects within the desired HSV range are the desired targets. The image moments are also only able to be computed if there are $N > 1$ objects in the image. Using the contours found, the area and centroid for each object is computed. The computed centroids are used to compute the image moment features described in Section 3.1, and sent to the PX4 for motion control.

Communication with the PX4 is handled by the mavros ROS package which translates the data from ROS format, and converts it into the appropriate mavlink data packet form that the PX4 can read. The vehicle runs the mavlink protocol to transmit over the radio, where the end computer has the same mavlink protocol which can translate and read the data. Another benefit to having ROS running is the ability to log the ROS topics during the flight. Similar to the PX4, ROS is configured to begin logging data on all topics once the vehicle has entered the armed state until the vehicle is unarmed. As mentioned before, the inner loop is assumed to perfectly track the reference [15]. This is done by having the inner loop running at a higher frequency to achieve a faster stable state for orientation which then allows the aircraft to stabilize the position states. This is beneficial for the real world experiment as well since the camera runs at rates around 30Hz compared to the MCS communication which is part of the inner loop control.

2.2.3 Camera

For this thesis, two different types of camera are utilized: FLIR Chameleon 3 and FLIR Chameleon 2 camera[3]. The chameleon 3 is a USB 3 camera used on the experimental platform, while the chameleon 2 is a USB 2 camera used during development of the ROS packages. The chameleon 3 is chosen to be used for the experimental platform due to having higher specifications such as higher frame rate, and higher compatibility on the TX1. The camera can take images with a resolution of 1280x1024 and frame rates of up to 149 fps, frames per second, in color. For the experiment, the camera is run in a binning mode that averages every two pixels cutting the resolution in half. Using camera calibration applications, the calibrated focal length is found to be $\lambda = 414$pixel. This camera is a downward facing camera mounted on the bottom of the vehicle, and as close to the center of mass as possible. Although the camera is able to run at rates higher than 149 fps with smaller resolutions, the maximum rate found with the minimum programs running was approximately 90 fps before any image processing is done. The chameleon 2 camera is used in Section 3.2 due the software being similar to the chameleon 3 camera, as well as some technical difficulties encountered with the chameleon 3 on certain

[3]Part no:11510997

desktop PC. The chameleon 2 has a max resolution of 1296x964 and max frame rate of 18 fps resulting in it being undesirable for use in the controller due to low frame rate, however, by using the ROS packages any experimental code can be tested on the desktop first before deploying on the experimental platform. In Section 3.2, this advantage is used during development of the algorithms for validating the camera modelling and computation of image moment features, since both cameras use the same ROS packages. This setup saves time and effort from having to constantly change batteries when powering the CV system for testing, as well as not needing to unmount the camera when testing. Also, as frame rate does not affect the computation accuracy of features only the rate of outputs, the choice for developing on the chameleon 2 is used to check accuracy of computed values before final deployment on the chameleon 3.

Chapter 3

Controller

This chapter describes how the vehicle kinematics in Section 2.1 are used with the image features to derive the control law. The type of controller chosen for this thesis is an inner-outer loop control shown in Figure 3.1, where the controllers are run on the PX4. An assumption for this thesis is that the environment contains the target markers with hue, saturation, and value (HSV) within a certain threshold, where the N largest visual features found are the target. Section 3.1 describes the control law derived from the kinematics of the unmanned aerial vehicle (UAV) and image features. In Section 3.2, the image feature extraction application is tested to verify how well the features extracted match the modelling. Once verification is done, a simulation is run to test how the UAV responds with the control law. Lastly, the controller is run on the experimental platform where the results are analyzed and compared to results from [1] which uses the same controller.

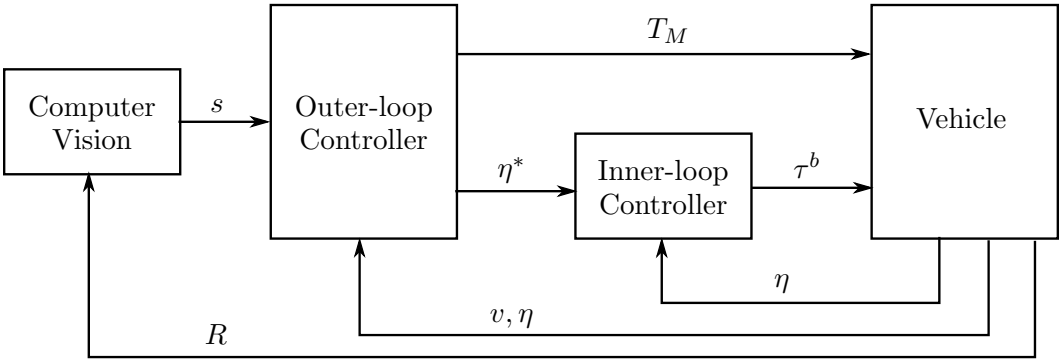


Figure 3.1: DIBVS inner-outer loop controller structure where the outer loop controls the translational dynamics and the inner loop controls the rotational dynamics

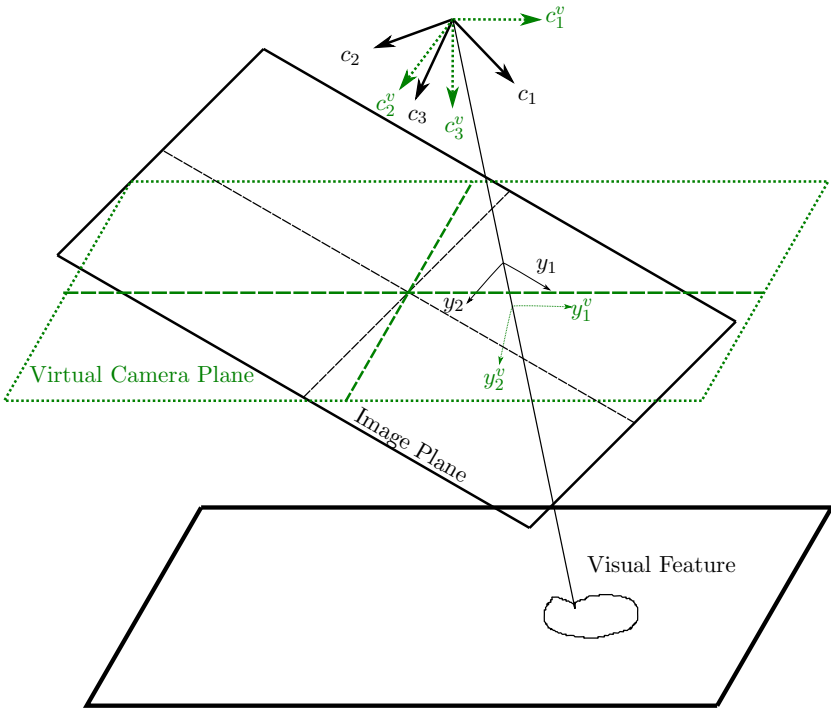


Figure 3.2: Visualization of virtual camera with image plane in black to virtual camera plane in green

3.1 IBVS

Using the knowledge that points in an image are related to real world coordinates it is possible to create a controller for the UAV as shown in [1]. The control structure selected is an inner-outer loop controller where the outer loop controls the rotational dynamics, and the inner loop controls the translational dynamics. The focus for this section will be on the outer loop controller. For the outer loop control, it can be seen that (2.3) does not depend on the value of ψ, and therefore, it would only be necessary to control ϕ and θ with ψ controlled separately.

3.1.1 Virtual Camera

For the controller, a virtual camera is utilized which transforms points on the actual image plane to a new virtual image plane that is parallel with the real world plane as shown in Figure 3.2. These new virtual points are denoted by the v superscript to differentiate the two different coordinates, and have the roll and pitch dependency eliminated from the points. The virtual camera simplifies the controller thus allows for a simpler control of the yaw, and only adds a single transform which should not affect the performance of the controller much. For the equations below, it is assumed that the focal length in the horizontal and vertical direction are the same, otherwise, a different derivation would be required [39]. Another assumption used is

that objects would not leave the field of view. To begin the points being projected onto the virtual plane are described by

$$p^v = R_{\phi\theta} P \tag{3.1}$$

where $R_{\phi\theta}$ is rotation matrix with only the roll and pitch

$$R_{\phi\theta} = \begin{bmatrix} c_\theta & s_\phi s_\theta & c_\phi s_\theta \\ 0 & c_\phi & -s_\phi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

and P is the point in the camera frame. The kinematics for the transformed points is

$$\dot{p^v} = -\dot{\psi}\,\mathrm{sk}(n_3)p^v - v^v \tag{3.2}$$

where v^v is velocity in virtual frame, and $\dot{\psi}$ is the angular velocity for yaw. Similar to Section 2.1.2, a perspective projection is used with the principal point assumed at $(0, 0)$, then the projected points can be described as

$$y^v = \lambda \begin{bmatrix} \frac{p_1^v}{p_3^v}, \frac{p_2^v}{p_3^v} \end{bmatrix}^T \tag{3.3}$$

where λ is the focal length in pixel. Combining (3.3) with (3.1) the transformation of real coordinates to the virtual camera coordinates can be described as

$$y^v = \frac{1}{\beta} \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \end{bmatrix} R_{\phi\theta} \begin{bmatrix} \lambda y_1 \\ \lambda y_2 \\ \lambda^2 \end{bmatrix} \tag{3.4}$$

where β is

$$\beta = n_3^T R_{\theta\phi} \begin{bmatrix} \lambda y_1 \\ \lambda y_2 \\ \lambda^2 \end{bmatrix} \tag{3.5}$$

n_3 is basis vector in navigation frame, and p_i is the normalized real world coordinates. Combining (3.4) with (3.2) yields

$$\dot{y}^v = \frac{1}{p_3^v} \begin{bmatrix} -\lambda & 0 & y_1^v \\ 0 & -\lambda & y_2^v \end{bmatrix} v^v + \begin{bmatrix} y_2^v \\ -y_1^v \end{bmatrix} \dot{\psi} \tag{3.6}$$

describing the kinematics in the virtual camera coordinates. From (3.6), it can now be seen that the points are decoupled from the roll and pitch.

3.1.2 Image Moment Features

Given an image, there are many ways to identify and track an object in the image, where two methods either use distinct features on an object or color of the object. A common example of using distinct features is using two eyes to identify a face, or two eyes and a mouth. This method of tracking is useful as it prevents some erroneous identification of objects, such as if a single ellipse shaped object found in an image frame will not be recognized as a face. Conversely, this method suffers from a problem if the object of interest is being occluded in the image which would fail to identify the object. The other method of identifying objects is based on the pixel color in the image, such as identifying green balls. Using color to identify objects assumes the environment only contains the desired color for the target of interest, which typically limits the usage to preset environments or targets. Another problem that using color may encounter are specular highlights, or bright spots, caused by light being reflected directly back at the camera, which causes certain pixels to have higher color intensity. For this thesis, the color method is used to identify the objects of interest using the HSV of each pixel, and morphology operations are applied to remove specular highlights. To allow for flexibility in identification of the target it is assumed that the largest K objects in the image are the desired targets, and any other objects are just noise.

At this point it is assumed that objects are able to be correctly identified in the image, and that the object is a planar target. Since the cameras used are mostly digital cameras the following will look mainly at discrete images. With the assumptions mentioned above the moment for an object with N pixels are defined as

$$m_{ij} = \sum_{k=1}^{N} (y_{1k})^i (y_{2k})^j \tag{3.7}$$

where y_{ik} indicates the k^{th} point in the y_i direction. The centered image moment moments are defined as

$$\mu_{ij} = \sum_{k=1}^{N} (y_{1k} - y_{1g})^i (y_{2k} - y_{2g})^j \tag{3.8}$$

where y_g is the centroid of the object described by $y_{1g} = m_{10}/m_{00}$ and $y_{2g} = m_{01}/m_{00}$. For the discrete case it should be noted that $m_{00} = N$ for N points used to track an object. The kinematics can now be found by taking the derivative of (3.7) which yields

$$\dot{m}_{ij} = \sum_{k=1}^{N} i y_{1k}^{i-1} y_{2k}^{j} \dot{y}_{1k} + j y_{1k}^{i} y_{2k}^{j-1} \dot{y}_{2k} \tag{3.9}$$

The same procedure applied to the centered image moments yields

$$\dot{\mu}_{ij} = \sum_{k=1}^{N} \Big(i(y_{1k} - y_{1g})^{i-1}(y_{2k} - y_{2g})^{j}(\dot{y}_{1k} - \dot{y}_{1g}) + j(y_{1k} - y_{1g})^{i}(y_{2k} - y_{2g})^{j-1}(\dot{y}_{2k} - \dot{y}_{2g}) \Big) \tag{3.10}$$

Now the depth of any 3D objects, assuming they are planar, can be described as follows [31, 32]

$$\frac{1}{P_3} = Ay_1 + By_2 + C \tag{3.11}$$

Similar to (2.12), an interaction matrix for the image moments can be defined as

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = [L_{ij}] \begin{bmatrix} v^c \\ \omega^c \end{bmatrix} \tag{3.12}$$

where L is the interaction matrix necessary to implement visual servoing. Using the depth defined in (3.11), and substituting with the single point image kinematic (2.11) gives

$$\dot{y}_1 = -(Ay_1 + By_2 + C)v_1^c + (Ay_1 + By_2 + C)y_1 v_3^c + y_1 y_2 \omega_1 - (y_1^2 + 1)\omega_2 + y_2 \omega_3 \tag{3.13a}$$

$$\dot{y}_2 = -(Ay_1 + By_2 + C)v_2^c + (Ay_1 + By_2 + C)y_2 v_3^c + (y_2^2 + 1)\omega_1 - y_1 y_2 \omega_2 - y_1 \omega_3 \tag{3.13b}$$

Now using (3.13) substituted into (3.9) the interaction matrix can be found as [39]

$$L_{ij} = \begin{bmatrix} -i(Am_{i,j} + Bm_{i-1,j+1} + Cm_{i-1,j}) \\ -j(Am_{i+1,j-1} + Bm_{i,j} + Cm_{i,j-1}) \\ (i+j)(Am_{i+1,j} + Bm_{i,j+1} + Cm_{i,j}) \\ (i+j)m_{i,j+1} + jm_{i,j-1} \\ -(i+j)m_{i+1,j} - im_{i-1,j} \\ im_{i-1,j+1} - jm_{i+1,j-1} \end{bmatrix}^T \tag{3.14}$$

likewise using the depth in (3.10) with (3.9) the interaction matrix for the centered

image moments as noted in [33, 39] is

$$L_{ij} = \begin{bmatrix} -iA\mu_{ij} - iB\mu_{i-1,j+1} \\ -jA\mu_{i+1,j-1} - jB\mu_{ij} \\ -A\mu_{\omega 2} + B\mu_{\omega 1} + (i+j)Cm_{ij} \\ (i+j)\mu_{i,j+1} + iy_{1g}\mu_{i-1,j+1} + (i+2j)y_{2g}\mu_{ij} - i\frac{\mu_{11}}{m_{00}}\mu_{i-1,j} - j\frac{\mu_{02}}{m_{00}}\mu_{i,j-1} \\ -(i+j)\mu_{i+1,j} - (2i+j)y_{1g}\mu_{i,j} - jy_{2g}\mu_{i+1,j-1} + i\frac{\mu_{20}}{m_{00}}\mu_{i-1,j} + j\frac{\mu_{11}}{m_{00}}\mu_{i,j-1} \\ i\mu_{i-1,j+1} - j\mu_{i+1,j-1} \end{bmatrix}^T \tag{3.15}$$

To simplify the interaction matrix, apply the virtual camera transformation to the points. This transforms the points such that they are parallel to the image plane resulting in $A = 0, B = 0$ in 3.11. Now for visual servoing select the image features for control of the translational kinematics. From [33], the chosen image features are $s_1 = y_{1g}, s_2 = y_{2g}$, and (3.19) for the translational dynamics. These are intuitive choices, since the centroids relate to the y_1, y_2 direction, and the area to p_3. Using (3.14), the interaction matrix for the chosen features, if using infinite points to track an object to be followed, gives (3.16), where (3.16c) uses the continuous interaction matrix from [39].

$$L_{y1g} = \begin{bmatrix} -C & 0 & Cy_{1g} & \left(y_{1g}y_{2g} + \frac{4\mu_{11}}{m_{00}}\right) & -\left(1 + y_{1g}^2 + \frac{4\mu_{20}}{m_{00}}\right) & y_{2g} \end{bmatrix} \tag{3.16a}$$

$$L_{y2g} = \begin{bmatrix} 0 & -C & Cy_{2g} & -\left(1 + y_{2g}^2 + \frac{4\mu_{02}}{m_{00}}\right) & \left(y_{1g}y_{2g} + \frac{4\mu_{11}}{m_{00}}\right) & -y_{1g} \end{bmatrix} \tag{3.16b}$$

$$L_a = \begin{bmatrix} 0 & 0 & 2Cm_{00} & 3m_{01} & -3m_{01} & 0 \end{bmatrix} \tag{3.16c}$$

In (3.16), there is a coupling between the chosen features with v_3, and the third feature also has different scaling relation to the corresponding velocity. To remove the coupling and restore passivity-like properties a proper normalization can be applied to the features which gives the redefined features as

$$s_3 = P_3^* \sqrt{\frac{a^*}{a}} \tag{3.17a}$$

$$s_1 = s_3 y_{1g} \tag{3.17b}$$

$$s_2 = s_3 y_{2g} \tag{3.17c}$$

where P_3^* is the desired depth. Using the new features in (3.17) and noting that

$P_3^*\sqrt{a^*} = P_3\sqrt{a}$ the interaction matrix for these features are

$$\begin{bmatrix} L_{s1} \\ L_{s2} \\ L_{s3} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & s_3\left(\frac{-y_{1g}y_{2g}}{2} + \frac{4\mu_{11}}{m_{00}}\right) & -s_3\left(1 - \frac{y_{1g}^2}{2} + \frac{4\mu_{20}}{m_{00}}\right) & s_2 \\ 0 & -1 & 0 & -s_3\left(1 - \frac{y_{2g}^2}{2} + \frac{4\mu_{02}}{m_{00}}\right) & \left(y_{1g}y_{2g} + \frac{4\mu_{11}}{m_{00}}\right) & -s_1 \\ 0 & 0 & -1 & \frac{-3s_3y_{2g}}{2} & \frac{-3s_3y_{1g}}{2} & 0 \end{bmatrix} \tag{3.18}$$

where (3.18) shows that through the normalization in (3.17a) it is possible to decouple the interaction matrix for the chosen features. For a discretized image, the area is not chosen due to the fact that areas may result in no changes depending on how the objects are tracked as shown in [39], where using only N points to track an object results in $\dot{a} = 0 \forall t$ as a is constant. Therefore, as proposed in [33] the new feature is selected to be the second order image moments

$$s_3 = a = \mu_{02} + \mu_{20} \tag{3.19}$$

as these features are invariant to translation as shown in (3.15) for the parallel assumption stated earlier setting $A = 0, B = 0$. Using the new feature (3.19) a similar interaction matrix to (3.18) can be found for the translational kinematics [33]. For the rotational kinematics, the feature chosen is based on the second order image moments defining an object's orientation, ψ defined as

$$s_4 = \psi = 0.5 \arctan\left(\frac{\mu_{11}}{\mu_{20} - \mu_{02}}\right) \tag{3.20}$$

where ψ is the angle. This orientation approximates an ellipse inside the set of points and uses the principle axis theorem to define the angle from the b_1 axis. As shown in [39], the interaction matrix for the chosen orientation is given by

$$L_{s4} = \begin{bmatrix} 0 & 0 & 0 & \alpha & \beta & -1 \end{bmatrix} \tag{3.21}$$

where

$$\alpha = \frac{-y_{1g}(2\mu_{11}^2 + \mu_{02}(\mu_{02} - \mu_{20})) + \mu_{11}s_2 + 5(\mu_{12}(\mu_{20} - \mu_{02}) + \mu_{11}(\mu_{03} - \mu_{21}))}{(\mu_{20} - \mu_{02}^2) + 4\mu_{11}^2}$$

$$\beta = \frac{\mu_{11}s_1 - y_{2g}(2\mu_{11}^2 + \mu_{02}(\mu_{02} - \mu_{20})) + 5(\mu_{21}(\mu_{20} - \mu_{02}) + \mu_{11}(\mu_{03} - \mu_{21})))}{(\mu_{20} - \mu_{02}^2) + 4\mu_{11}^2}$$

Combining these four image moment features from (3.17) and (3.20) with the virtual camera in Section 3.1.1 the triangular or passivity-like property can be recovered for the translational dynamics.

3.1.3 IBVS Controller

Recall from Section 2.1 that the rotational dynamics can be linearized and separately controlled from the translational dynamics which is implemented through the separation of outer loop control from the inner loop control. Once the image points have been transformed to the virtual camera coordinates, the next step would be to use the points to compute the image moment features. Recall that the image moment features are described by

$$s_1 = s_3 y_{1g} \tag{3.22a}$$

$$s_2 = s_3 y_{2g} \tag{3.22b}$$

$$s_3 = \sqrt{\frac{\mu_{20}^* + \mu_{02}^*}{\mu_{20} + \mu_{02}}} \tag{3.22c}$$

$$s_4 = \frac{1}{2} \arctan\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right) \tag{3.22d}$$

where μ^* is the reference image moment for the desired pose, μ is the centered image moments (3.8), y_{ig} is the centroid, and N is the number of points. It should be noted that the reference image moment can be assumed $\mu^* = \mu_{20}^* + \mu_{02}^*$ without loss of generality, therefore, only requiring a single value is used as reference. Each of the image moments feature values in (3.22) describe a physical property of the group of points where (s_1, s_2) describe the (y_1, y_2) position of the center of the points, s_3 describes how close the points are to the desired distance, and s_4 describes the orientation for the set of points based on an ellipse drawn between the points [32]. It should also be noted that if $\mu_{20} = \mu_{02}$ then the angle or s_4 is a singularity, but can be considered to be $s_4 = 90°$. The simple choice for regulating the four image moments is $[s_1, s_2, s_3, s_4]^T = [0, 0, 1, 0]^T$ which indicates that the points are centered in the image, at the desired distance from the camera, and the points form an ellipse oriented along the y_1 axis. The choice for reference image moments features are arbitrary for s_3, s_4, however, for (s_1, s_2) other reference values may lead to the points leaving the field of view causing the controller to fail. For T_M, a small angle approximation can be used to describe the force in the virtual frame since the controller regulates the vehicle at hover which gives $T_M^v \approx T_M [\theta, \phi, 1]^T$. Using (3.22) the image moment kinematics can be described by

$$\begin{bmatrix} \dot{s_1} \\ \dot{s_2} \\ \dot{s_3} \end{bmatrix} = \frac{-1}{p_3^*} \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{v}^v + \begin{bmatrix} s_2 \\ -s_1 \\ 0 \end{bmatrix} \dot{\psi} \tag{3.23a}$$

$$\dot{s_4} = -\dot{\psi} \tag{3.23b}$$

where p_3^* is the reference depth, and

$$\tilde{v}^v = R_{\tilde{\psi}}^T (v^n - \bar{v}_t^n)$$

$$\tilde{\psi} = \psi - \psi_t$$

with R_ψ defined by (3.36). Using the description of the above image moments kinematics in (3.23) and combining with the vehicle dynamics (2.2) a control law can be derived as

$$\dot{s_1} = -k_2 v_1^v + s_2 \dot{\psi} \tag{3.24a}$$
$$\dot{s_2} = -k_2 v_2^v - s_1 \dot{\psi} \tag{3.24b}$$
$$\dot{s_3} = -k_1 v_3^v \tag{3.24c}$$
$$\dot{v_1^v} = v_2^v \dot{\psi} + u_\theta \tag{3.24d}$$
$$\dot{v_2^v} = -v_1^v \dot{\psi} + u_\phi \tag{3.24e}$$
$$\dot{v_3^v} = u_F \tag{3.24f}$$
$$\dot{s_4} = -\dot{\psi} \tag{3.24g}$$
$$\dot{\psi} = u_\psi \tag{3.24h}$$

where $k_1 = 1/P_3^*$, $k_2 = \lambda k_1$, s is the image moments, v is the velocity, ψ is the yaw, and u is the inputs described by [1]

$$u_F = -\frac{T_M}{m} + g \tag{3.25a}$$
$$u_\phi = \frac{T_M \theta}{m} \tag{3.25b}$$
$$u_\theta = -\frac{T_M \phi}{m} \tag{3.25c}$$
$$u_\psi = \int_0^t \frac{\tau_3}{J_3} \mathrm{d}t \tag{3.25d}$$

where T_M is the total force from the motors, g is the force of gravity, (ϕ, θ) is the roll and pitch respectively, τ is the torque, and J_3 is the momentum. From [39] by

choosing the inputs as

$$u_F = -k_{h2}\left(\frac{v_3^v}{k_{h1}} - s_3 + 1\right) \tag{3.26a}$$

$$u_\phi = -k_{l2}\left(\frac{v_2^v}{k_{l1}} - s_2\right) \tag{3.26b}$$

$$u_\theta = -k_{l2}\left(\frac{v_1^v}{k_{l1}} - s_1\right) \tag{3.26c}$$

$$u_\psi = \frac{k_\psi s_4}{J_3} \tag{3.26d}$$

where k are positive gains the system can be shown to be stable. For the gain values in (3.26), they are chosen such that $P_3^* k_{l2} > k_{l1}^2$ and $P_3^* k_{h2} > k_{h1}^2$ to ensure that the system is asymptotically stable. The gain values used in this thesis is listed in Table 3.15 which are tuned values from [1] due to the difference in platform specifications.

As shown in [39], choosing the correct Lyapunov function candidate the translational subsystem can be shown to be globally asymtopically stable (GAS) and the yaw subsystem is globally exponentially stable (GES). To start, consider the translational subsystem described by (3.24a) to (3.24f), and define the following errors

$$\delta_1 = \frac{1}{\lambda} s_1 \tag{3.27a}$$

$$\delta_2 = \frac{1}{\lambda} s_2 \tag{3.27b}$$

$$\delta_3 = s_3 - 1 \tag{3.27c}$$

$$\delta_4 = \frac{v_1^v}{k_{l1}} - \delta_1 \tag{3.27d}$$

$$\delta_5 = \frac{v_2^v}{k_{l1}} - \delta_2 \tag{3.27e}$$

$$\delta_6 = \frac{v_3^v}{k_{h1}} - \delta_3 \tag{3.27f}$$

where $(k_{h1}, k_{h2}, k_{l1}, k_l 2)$ are gains. Using these error functions, first consider the height subsystem with chosen Lyapunov function candidate as

$$V_1 = \frac{1}{2}\left(\delta_6^2 + \delta_3^2\right) \tag{3.28}$$

a positive definite function. Taking the derivative of (3.28) gives the following

$$\dot{V}_1 = -k_{h1} k_1 (2\delta_3^2 + \delta_3 \delta_6) + \frac{\delta_6 u_F}{k_{h1}} + \frac{k_1}{k_{h1}} \delta_6 (\delta_3 + \delta_6) \tag{3.29}$$

now let $u_F = k_{h2}\delta_6$

$$\dot{V}_1 = -k_{h1}k_1\delta_3^2 - \frac{1}{k_{h1}}(k_{h2} - k_{h1}^2 k_1)\delta_6^2 \tag{3.30}$$

where choosing $k_{h2} > k_{h1}^2 k_1$ the function would be negative definite. Now consider translational subsystem and second Lyapunov function candidate

$$V_2 = \frac{1}{2}\left(\delta_1^2 + \delta_2^2 + \delta_4^2 + \delta_5^2\right) \tag{3.31}$$

taking the derivative yields

$$\dot{V}_2 = \frac{(u_\theta\delta_4 + u_\phi\delta_5)}{k_{l1}} + k_{l2}k_1(-\delta_1^2 - \delta_2^2 + \delta_4^2 + \delta_5^2) \tag{3.32}$$

rewriting (3.26b) and (3.26c) in terms of the new error signals yields $u_\phi = k_{l2}\delta_5$ and $u_\theta = k_{l2}\delta_4$ which are substituted into (3.32) yields

$$\dot{V}_2 = -k_{l1}k_1(\delta_1^2 + \delta_2^2) - \frac{(k_{l2} - k_1 k_{l1}^2)(\delta_4^2 + \delta_5^2)}{k_{l1}} \tag{3.33}$$

a negative definite function if choosing $k_{l2} > k_{l1}^2 k_1$. The factor $1/\lambda$ for s_1, s_2 in (3.26b) and (3.26c) is assumed to be part of the gain k_{l2} without loss of generality. Finally, using the Lyapunov function candidate

$$V = V_1 + V_2$$

a positive definite function with

$$\dot{V} = \dot{V}_1 + \dot{V}_2$$

a negative definite derivative for gains chosen as $k_{h2} > k_{h1}^2 k_1$ and $k_{l2} > k_{l1}^2 k_1$, for all defined signals then it can be concluded that the translational subsystem is GAS. For the yaw subsystem, controller input (3.26d) is substituted into the dynamics (3.24g) and (3.24h) gives

$$\dot{s_4} = -\frac{k_\psi}{J_3}s_4 \tag{3.34}$$

which is negative definite for all values of s_4, thus the yaw subsystem is GES with the chosen controller. For the inner loop control a simple PID controller described

by

$$\tau_1{}^b = k_{p\psi} e_\phi + k_{i\psi} \int e_\phi \, de_\phi + k_{d\psi} \dot{e_\phi} \tag{3.35a}$$

$$\tau_2{}^b = k_{p\theta} e_\theta + k_{i\theta} \int e_\theta \, de_\theta + k_{d\theta} \dot{e_\theta} \tag{3.35b}$$

$$\tau_3{}^b = k_{p\psi} e_\psi + k_{i\psi} \int e_\psi \, de_\psi + k_{d\psi} \dot{e_\psi} \tag{3.35c}$$

where τ is the torque, k is the gain, and e_η is error in the angle. Gain values chosen for the experiment are listed in Table 3.15, which are based on [1] and tuned for the experimental platform.

3.2 Image Moment Validation

Recall from Section 3.1.2 that objects are identified by finding where the HSV of pixels fall within a threshold range. In this section, centers of the object are computed giving a pair (y_1, y_2) coordinate, where y_1 is the horizontal position, and y_2 is the vertical position used to compute the image moments from (3.22a) to (3.22d). For this section, the camera frame is assumed to coincide with the navigation frame. Recall from Section 3.1 that the ideal values for controlling the image moment features are $[s_1, s_2, s_3, s_4]^T = [0, 0, 1, 0]^T$, for a given reference $\mu^* = \mu^*_{02} + \mu^*_{20}$, which is related to the distance between the target and camera.

3.2.1 Testing Equipment & Procedure

For the experiment run in Section 3.2.4 the following equipment is used.

- FLIR (formerly Point Grey) Chameleon 2 Camera. Serial number: 11510997
- FLIR (Point Grey) ACC-01-4002 lens with focal length of 8 mm
- Target board with known geometry, shown in Figure 3.3
- Ruler to measure distance
- Calibration board

The camera chosen for this section is the FLIR Chameleon 2 camera due to problems encountered with the platform as mentioned in Section 2.2, however, all Robot Operating System (ROS) packages used for the Chameleon 2 camera are the same packages for a Chameleon 3 camera. For the target board, four green markers are placed on a board in the four cardinal directions from the center. The approximate dimensions of the markers are listed in Table 3.1 with an error of 0.1 cm for each measurement taken with respect to the center. The center points of the two middle

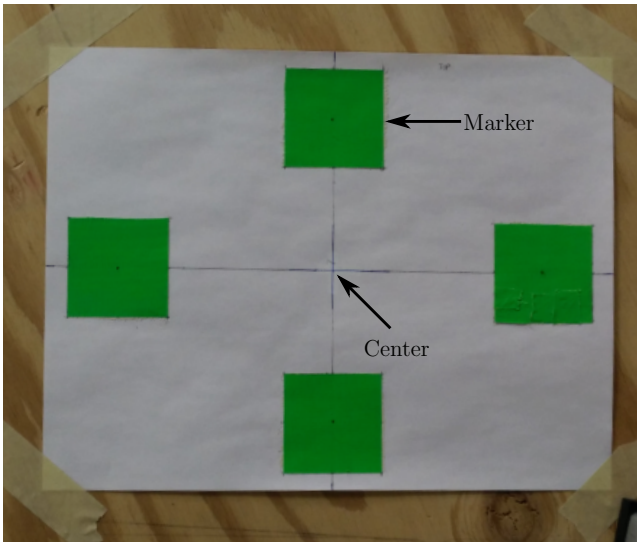


Figure 3.3: Target board with 4 markers

markers are spaced 14.8 cm vertically apart, while the left and right squares are spaced 21 cm horizontally. This spacing is chosen so the orientation for the group of points is pointing from the center to the right marker.

Marker Name	**Width**(cm)	**Height**(cm)	**Center** y_1(cm)	**Center** y_2(cm)
Top Square	4.8	4.8	0	7.4
Bottom Square	4.8	4.8	0	-7.4
Left Square	4.8	4.8	-10.5	0
Right Square	4.8	4.8	10.5	0

Table 3.1: Dimensions of the target markers with errors of 0.1cm

To ensure that theoretical values computed match values computed in the ROS package, the target center is aligned with the optical center of the camera first then the target is moved to simulate any offset. For all tests in this section, the target is set up and assumed to be parallel with the image plane. It is also assumed that any rotations are parallel to the image plane, and therefore, can be described by

$$R_\psi = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.36}$$

where R_ψ is the rotation matrix in the c_3 axis. Figure 3.4 illustrates the experiment setup where the camera is fixed to the desk on the right, and the relocatable target on the left.

Figure 3.4: Image moment reference setup with relocatable target in front of the fixed camera

3.2.2 Camera Calibration

Simulating the camera requires its intrinsic parameters which can be acquired through calibrating the camera. A few different applications used for this calibration are listed in Appendix A. For this thesis, the Camera Calibration Application from the Computer Vision Toolbox on Matlab is used to get the intrinsic parameters for the projection matrix using the steps listed in Appendix A.0.2. This toolbox is chosen due to its ease of use, as well as being able to get the error when calculating the intrinsic parameters of the camera. To use the application, a set of images of a known checkerboard patterned calibration board is used as input to the application. This application finds the corners of the pattern for each image, and uses the known dimensions of each square in the pattern to compute the intrinsic parameters. Images used should have sharp lines along edges of the squares to reduce the error when computing the corners. Values acquired for this thesis are listed in Table 3.2, and the projection matrix can be found by substituting the values into (2.9). Note that the focal length in both directions are within the computed error range of each other, and therefore are assumed to be the same. Another way to compute the focal length uses (2.10) which yields very similar values.

Parameter	Value (pixel)	Error (pixel)
λ_{y1}	2194	4.67
λ_{y2}	2192	4.54
C_{y1}	658	2.26
C_{y2}	440	2.57

Table 3.2: Projection matrix values and errors

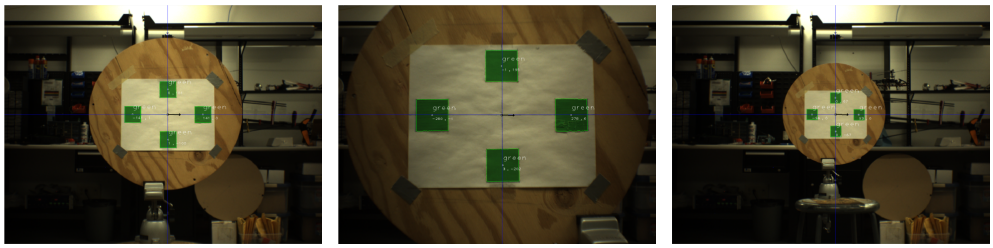
3.2.3 Modelling Values

Using the known dimensions of the target board, and projection matrix from Section 3.2.1, a Matlab script was created to compute the expected values for s_1 to s_4 from (3.22). Table 3.3 lists the testing conditions for each experiment, where eight of the experiments are designed to have the target at the extreme for a single image moment features. For example, the "Target_Closer" and "Target_Farther" experiment are used to test decreasing and increasing s_3 respectively. Figure 3.5 shows a snapshot for each of the nine experiments carried out, where the Figure 3.5a is the reference setup used to set $s = [0, 0, 1, 0]^T$.

Experiment	Depth (m)	Offset x (m)	Offset y (m)	Angle (°)
Target_Reference	1.6	0	0	0
Target_Closer	0.8	0	0	0
Target_Farther	2.4	0	0	0
Target_Up	1.6	0	0.20	0
Target_Down	1.6	0	-0.20	0
Target_Right	1.6	0.25	0	0
Target_Left	1.6	-0.25	0	0
Target_Tilt_Left	1.6	0	0	50.1
Target_Tilt_Right	1.6	0	0	-47.2

Table 3.3: List of Experiments Test Conditions

For experiments with offsets and rotations, the initial points are first rotated then translated before they are projected onto the image plane when computing the reference values. Recall from Section 2.1.2 that the pinhole model is used, therefore, the measured points are projected onto the image plane using (2.8), and the values for computed projected center points, (y_1, y_2), of the target board are described in Table 3.4 with units of pixels.



(a) Reference target setup

(b) Target moved closer to camera

(c) Target moved away from camera

(d) Target moved up on image plane

(e) Target moved down on image plane

(f) Target moved left on image plane

(g) Target moved right on image plane

(h) Target tilted left on image plane

(i) Target tilted right on image plane

Figure 3.5: A snapshot for each of the nine experimental setups used to test the computation of the image moment features

Experiment	Top Marker	Bottom Marker	Left Marker	Right Marker
Target_Reference	(0,101)	(0,-101)	(-143,0)	(143,0)
Target_Closer	(0,203)	(0,-203)	(-287,0)	(287,0)
Target_Farther	(0,67.6)	(0,-67.6)	(-95.7,0)	(95.7,0)
Target_Up	(0,375)	(0,173)	(-143,274)	(143,274)
Target_Down	(0,-173)	(0,-375)	(-143,-274)	(143,-274)
Target_Right	(343,101)	(343,-101)	(199,0)	(486,0)
Target_Left	(-343,101)	(-343,-101)	(-486,0)	(-199,0)
Target_Tilt_Left	(-77.8,65.0)	(77.8,-65.0)	(-92.0,-110)	(92.0,110)
Target_Tilt_Right	(74.4,68.9)	(-74.4,-68.9)	(-97.5,105)	(97.5,-105)

Table 3.4: Reference Values for Projected Points (y_1, y_2)

Table 3.4 values are centered around the principal point known as the center of the image. Using (3.8) the required centered image moment are described as follows

$$\mu_{11} = \sum_{j=1}^{K} (y_{1j} - y_{1g})(y_{2j} - y_{2g}) \tag{3.37a}$$

$$\mu_{20} = \sum_{j=1}^{K} (y_{1j} - y_{1g})^2 \tag{3.37b}$$

$$\mu_{02} = \sum_{j=1}^{K} (y_{2j} - y_{2g})^2 \tag{3.37c}$$

where (y_{1j}, y_{2j}) is the center of the j^{th} point, and (y_{1g}, y_{2g}) is the average value of (y_1, y_2). The computed centered image moments are listed for each experiment in Table 3.5.

Experiment	μ_{11}	μ_{20}	μ_{02}
Target_Reference	0	41170	20559
Target_Closer	0	164681	82237
Target_Farther	0	18298	9137
Target_Up	0	41170	20559
Target_Down	0	41170	20559
Target_Right	0	41170	20559
Target_Left	0	41170	20559
Target_Tilt_Left	10123	29055	32658
Target_Tilt_Right	-10255	30088	31627

Table 3.5: Reference Values for μ_{11},μ_{20}, and μ_{02}

For all experiments in this section, the reference $\mu^* = \mu_{20} + \mu_{02}$ is taken from

the "Target_Reference" experiment listed in Table 3.5. Using the centered image moments the image feature moments described by (3.22) are computed for each experiment condition and are listed in Table 3.6.

Experiment	s_1	s_2	s_3	s_4
Target_Reference	0	0	1	0
Target_Closer	0	0	0.5	0
Target_Farther	0	0	1.5	0
Target_Up	0	274	1	0
Target_Down	0	-274	1	0
Target_Right	343	0	1	0
Target_Left	-343	0	1	0
Target_Tilt_Left	0	0	1	-50.1
Target_Tilt_Right	0	0	1	47.2

Table 3.6: Reference Values for s_1-s_4

As illustrated in Table 3.6, each set of two experiments after the "Target_Reference" experiment test a single image moment feature in the following order: $s_3 \rightarrow s_2 \rightarrow s_1 \rightarrow s_4$. The results from the experiment shows that using the image moment features selected the passivity-like property is successfully restored resulting in only a single image moment feature changing.

3.2.4 ROS Experimental Results

Following the experiment procedure steps listed in Appendix B.1, the ROS package is used to compute the image moment features s_1 to s_4 with the target board in the different positions listed in Table 3.3. Algorithm 1 is implemented on the ROS package running on a desktop PC which handles all the computer vision (CV) processes and image moments calculations.

Algorithm 1 Image Moment Calculation

```
convert RGB image to HSV
get threshold value for desired color
if maxThreshold > pixel > minThreshold then
    return 255
else
    return 0
end if
Apply Morphology to image
for object > minSize do
    find center of object
    Shift pixel values to middle of the image
    Apply normalization to the image points
    Apply virtual camera transform to the points
end for
Sort detected objects from largest to smallest area
Compute the image moments with centers for N largest objects
Send image moments to outer loop controller
```

Experiment	**Top Marker**	**Bottom Marker**	**Left Marker**	**Right Marker**
Target_Reference	(0.337,100)	(1.14,-101)	(-141,0.698)	(141,-0.234)
Target_Closer	(-1.52,196)	(3.29,-202)	(-281,-1.18)	(278,-0.647)
Target_Farther	(0.007,67.3)	(0.649,-67.0)	(-94.1,0.741)	(93.9,-0.741)
Target_Up	(-0.980,370)	(-0.976,171)	(-142,272)	(139,269)
Target_Down	(0.629,-173)	(0.908,-382)	(-144,-276)	(144,-279)
Target_Right	(342,102)	(347,-102)	(202,-0.4)	(487,0.986)
Target_Left	(-341,100)	(-343,-102)	(-485,-0.209)	(-201,-1.86)
Target_Tilt_Left	(-76.2,68.0)	(78.8,-64.4)	(-92.5,-106)	(93.7,107)
Target_Tilt_Right	(71.4,69.3)	(-76.4,-70.1)	(-99.1,104)	(93.7,-104)

Table 3.7: Experiment Values for Projected Points y_1, y_2

For each setup, the average value over 500 computed image frames are used when comparing to the same values listed in Section 3.2.3. The average center locations for each marker are listed in Table 3.7. Once the centers are found, the points are sorted from largest to smallest based on the area associated to each center point found. The largest four areas are then used to compute the centered image moment values listed in Table 3.8.

Experiment	μ_{11}	μ_{20}	μ_{02}
Target_Reference	-212	39766	20231
Target_Closer	-813	156313	79198
Target_Farther	-181	17677	9028
Target_Up	-470	39520	19848
Target_Down	-519	41598	21686
Target_Right	-282	40784	20780
Target_Left	-47.1	40219	20522
Target_Tilt_Left	9624	29339	31577
Target_Tilt_Right	-9676	29518	31204

Table 3.8: Experimental Values for μ_{11},μ_{20}, and μ_{02}

With the centered image moments computed, the values are used with (3.22) to compute the image moment features where the average value is listed in Table 3.9.

Experiment	s_1	s_2	s_3	s_4
Target_Reference	0.340	0.0803	1.01	0.624
Target_Closer	-0.080	-1.01	0.512	0.606
Target_Farther	0.192	0.101	1.52	1.20
Target_Up	-1.50	276	1.02	1.37
Target_Down	0.489	-274	0.988	1.50
Target_Right	345	0.177	1.00	0.810
Target_Left	-346	-1.02	1.01	0.138
Target_Tilt_Left	0.967	1.19	1.01	-48.4
Target_Tilt_Right	-2.60	-0.170	1.01	47.6

Table 3.9: Experimental Values for s_1-s_4

where s is the image moment feature computed for the 4 markers. All results in Table 3.9 used the reference value from Table 3.5 of the reference setup $\mu^* = 61731$.

3.2.5 Error Analysis

The average error for each value over the nine tests are listed in Tables 3.10 to 3.12. Using the pinhole projection model for a camera and the projection matrix (2.8), the errors in the points can be found after some derivation to be described by (3.38).

$$\Delta y_1 = \sqrt{\left(\frac{\lambda_{y1}}{P_3}\Delta P_1\right)^2 + \left(\frac{P_1 * \lambda_{y1}}{P_3^2}\Delta P_3\right)^2 + \left(\frac{P_1}{P_3}\Delta\lambda_{y1}\right)^2 + \left(\Delta C_{y1}\right)^2} \tag{3.38a}$$

$$\Delta y_2 = \sqrt{\left(\frac{\lambda_{y2}}{P_3}\Delta P_2\right)^2 + \left(\frac{P_2 * \lambda_{y2}}{P_3^2}\Delta P_3\right)^2 + \left(\frac{P_2}{P_3}\Delta\lambda_{y2}\right)^2 + \left(\Delta C_{y2}\right)^2} \tag{3.38b}$$

where $(\lambda_{y1}, \lambda_{y2})$ is the focal length in pixel, (P_1, P_2, P_3) is the real world coordinates relative to the camera position in metres, and (C_{y1}, C_{y2}) is the principal in pixels. The results of using (3.38) for the target board are described in Table 3.10 for the nine experiments. An example for each step to compute the errors in y_1 is in Appendix B.2.

Experiment	**Top Marker**	**Bottom Marker**	**Left Marker**	**Right Marker**
Target_Reference	(2.64,3.18)	(2.64,3.18)	(3.21,2.91)	(3.21,2.91)
Target_Closer	(3.55,6.32)	(3.55,6.32)	(8.03,3.76)	(8.03,3.76)
Target_Farther	(2.44,2.79)	(2.44,2.79)	(2.57,2.73)	(2.57,2.73)
Target_Up	(2.64,5.58)	(2.64,3.64)	(3.21,4.53)	(3.2,4.53)
Target_Down	(2.64,3.64)	(2.64,5.58)	(3.21,4.53)	(3.21,4.53)
Target_Right	(5.09,3.18)	(5.09,3.18)	(3.66,2.91)	(6.71,2.91)
Target_Left	(5.09,3.18)	(5.09,3.18)	(6.71,2.91)	(3.66,2.91)
Target_Tilt_Left	(2.64,3.18)	(2.64,3.18)	(3.21,2.91)	(3.21,2.91)
Target_Tilt_Right	(2.64,3.18)	(2.64,3.18)	(3.21,2.91)	(3.21,2.91)

Table 3.10: Error in Values of Computed u and v

These errors amount to approximately 1% error for the size of the image. To compute the error in μ, it is necessary to first find the error in the average value of (y_1, y_2) described by

$$\Delta y_{1g} = \sqrt{\sum_{j=1}^{K}\left[\frac{\Delta y_{1j}}{K}\right]^2} \tag{3.39a}$$

$$\Delta y_{2g} = \sqrt{\sum_{j=1}^{K}\left[\frac{\Delta y_{2j}}{K}\right]^2} \tag{3.39b}$$

where K is the number of points detected in the image, (y_{1g}, y_{avg}) is the average value of (y_1, y_2) of all points in the image. For a more general error calculation, it will be assumed that the points are independent of each other but have the same error in the average value, since any object in the image is converted to a single point. Using the error values found from previous equations, the error for each of

the centered image moments values can be computed using (3.40) to (3.42).

$$\Delta\mu_{11} = \left(\sum_{j=1}^{K} \left[(dv_j)\Delta u_j\right]^2 + \left[-(dv_j)\Delta y_{1g}\right]^2 + \left[(du_j)\Delta v_j\right]^2 + \left[-(du_j)\Delta y_{2g}\right]^2 \right)^{1/2} \tag{3.40}$$

$$\Delta\mu_{20} = \sqrt{\sum_{j=1}^{K} \left[2(du_j)\Delta u_j\right]^2 + \left[-2(du_j)\Delta y_{1g}\right]^2} \tag{3.41}$$

$$\Delta\mu_{02} = \sqrt{\sum_{j=1}^{K} \left[2(dv_j)\Delta v_j\right]^2 + \left[-2(dv_j)\Delta y_{2g}\right]^2} \tag{3.42}$$

where $dy_{1j} = y_{1j} - y_{1g}$ and $dy_{2j} = y_{2j} - y_{2g}$ for each point. The computed values are listed in Table 3.11.

Experiment	$\Delta\mu_{11}$	$\Delta\mu_{20}$	$\Delta\mu_{02}$
Target_Reference	702	1302	912
Target_Closer	1834	6517	3627
Target_Farther	436	696	533
Target_Up	995	1303	1351
Target_Down	995	1303	1351
Target_Right	939	2193	913
Target_Left	939	2193	913
Target_Tilt_Left	758	1018	1079
Target_Tilt_Right	754	1046	1066

Table 3.11: Error Values for μ_{11},μ_{20}, and μ_{02}

Finally, combining all the results together, error in the computed image moment features are described by (3.43) to (3.46).

$$\Delta s_1 = \sqrt{\left(y_{1g}\Delta s_3\right)^2 + \left(s_3\Delta y_{1g}\right)^2} \tag{3.43}$$

$$\Delta s_2 = \sqrt{\left(y_{2g}\Delta s_3\right)^2 + \left(s_3\Delta y_{2g}\right)^2} \tag{3.44}$$

$$\Delta s_3 = \frac{\sqrt{\mu^*}}{2\sqrt[3]{\mu_{20} - \mu_{02}}} \sqrt{\Delta\mu_{20}{}^2 + \Delta\mu_{20}{}^2} \tag{3.45}$$

$$\Delta s_4 = \frac{1}{(\frac{\lambda_{y2}}{\lambda_{y1}}\mu_{20} - \frac{\lambda_{y1}}{\lambda_{y2}}\mu_{02})^2 + 4\mu_{11}^2} \sqrt{dW + \left[\left(\frac{\lambda_{y2}}{\lambda_{y1}}\mu_{20} - \frac{\lambda_{y1}}{\lambda_{y2}}\mu_{02}\right)\Delta\mu_{11}\right]^2} \tag{3.46}$$

where dW is given by

$$dW = \frac{\mu_{11}^2}{\lambda_{y1}^2}\left[\left(\mu_{20}\Delta\lambda_{y2}\right)^2 + \left(\lambda_{y2}\Delta\mu_{20}\right)^2 + \left(-\frac{\mu_{20}\Delta\lambda_{y1}}{\lambda_{y1}}\right)^2\right] + \frac{\mu_{11}^2}{\lambda_{y2}^2}\left[\left(\mu_{02}\Delta\lambda_{y1}\right)^2 + \left(\lambda_{y1}\Delta\mu_{02}\right)^2 + \left(\frac{\mu_{02}\Delta\lambda_{y2}}{\lambda_{y2}}\right)^2\right]$$

and μ^* is from the reference setup. The expected error values are computed and summarized in Table 3.12

Experiment	Δs_1	Δs_2	Δs_3	Δs_4
Target_Reference	2.93	3.05	0.013	1.95
Target_Closer	2.90	2.52	0.008	1.27
Target_Farther	3.76	4.14	0.024	2.72
Target_Up	2.93	6.18	0.015	2.76
Target_Down	2.93	6.18	0.015	2.76
Target_Right	8.36	3.05	0.019	2.61
Target_Left	8.36	3.05	0.019	2.61
Target_Tilt_Left	2.93	3.05	0.012	2.07
Target_Tilt_Right	2.93	3.05	0.012	2.08

Table 3.12: Error Values for s_1-s_4

where $(\Delta s_1, \Delta s_2)$ are in units of pixels, Δs_3 is unitless, and Δs_4 is in degrees. Again the values listed in Table 3.12 are the average values for each image set.

3.2.6 Comparison of Results

The difference between the expected values listed in Section 3.2.3 and the average experimental value listed in Section 3.2.4 for the projected points are summarized in Table 3.13 where all values are in units of pixels. Comparing these values to the values computed in Section 3.2.5 it can be seen that the majority of the y_1 values are computed within this error range with only two experiments where one of the marker's y_1 is outside the range. Analyzing y_2 values a similar result can be seen where only one of the markers y_2 value failed to fall within the expected range for four experiments. These results are acceptable since the error in the y_1 or y_2 values never exceed more than half of the marker's location for more than half the total number of experiments, and are only marginally more than the computed expected error. Therefore, it was assumed that the resulting larger error could be the result of unaccounted camera distortion or misalignment of the target.

Table 3.14 list the average value of the computed difference between Section 3.2.3

Experiment	Top Marker	Bottom Marker	Left Marker	Right Marker
Target_Reference	(0.337,-0.889)	(1.14,0.739)	(2.33,0.699)	(-2.61,-0.232)
Target_Closer	(-1.52,6.82)	(3.29,0.758)	(6.19,-1.18)	(8.58,-0.647)
Target_Farther	(-0.007,-0.266)	(0.649,0.555)	(1.57,0.717)	(-1.71,-0.741)
Target_Up	(-0.980,-5.23)	(-0.976,-1.77)	(0.926,-1.56)	(4.86,4.90)
Target_Down	(0.629,-0.754)	(0.908,6.31)	(-0.535,-1.62)	(0.977,5.02)
Target_Right	(-0.131,0.623)	(4.58,-0.502)	(2.30,-0.400)	(1.38,0.986)
Target_Left	(1.19,-1.06)	(-0.655,-0.919)	(0.524,-0.209)	(-2.35,-1.86)
Target_Tilt_Left	(1.65,2.90)	(1.01,0.628)	(-0.436,3.80)	(1.63,-2.57)
Target_Tilt_Right	(2.99,0.403)	(-1.96,-1.25)	(-1.61,-1.48)	(3.77,1.65)

Table 3.13: Difference Between Expected and Experimental Values for (y_1, y_2)

compared to Section 3.2.4 for the computed centered image moments and image moment features. Comparing the μ values it can be seen that all the μ_{11} values are within the range of its expected value with error. Looking at μ_{20} it is within the error except for three experiments, while μ_{02} is within the expected range for all but a single experiment. Again these results are acceptable due to there being less than half of the experiments outside the expected range. Analyzing the image moment feature values it can be seen that s_1, s_2, s_4 are all within range of the expected value with error, while s_3 only fails three experiments.

Experiment	μ_{11}	μ_{20}	μ_{02}	Δs_1	Δs_2	Δs_3	Δs_4
Target_Reference	-212	1406	-328	0.3043	0.080	0.014	0.624
Target_Closer	-813	8350	-3041	-0.080	-1.01	0.012	0.606
Target_Farther	-181	-621	-110	0.192	0.100	0.020	1.20
Target_Up	-470	1642	-690	-1.5	1.9	0.020	1.37
Target_Down	-519	436	1148	0.489	0.066	-0.013	1.50
Target_Right	-282	-251	230	209	0.177	0.0002	0.810
Target_Left	-47.1	-816	-27.8	-2.70	-1.02	0.007	0.138
Target_Tilt_Left	-499	283	-1082	0.968	0.968	0.006	1.70
Target_Tilt_Right	580	-571	-423	-2.60	-0.170	0.008	0.372

Table 3.14: Difference Between Expected and Experimental Values for μ and s_1-s_4

From these nine experiments, the results indicate that the ROS package computes image moment features with satisfactory accuracy for use in the flight test experiment. These experiments also verified that the image moment features from Section 3.1.2 are decoupled values that describe a single physical property of the image features.

3.3 Simulation

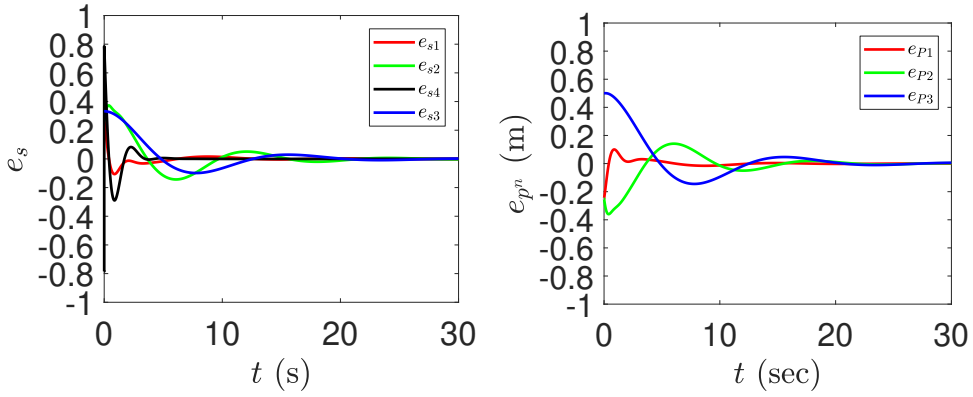
For the simulation, a simulink model running in Matlab is used to simulate the UAV dynamics described in Section 2.1 with special simulink blocks that act as the controller described in Section 3.1. The simulink model includes a block for the vehicle dynamics, the camera, image moments computation, outer loop controller and inner loop controller. The parameters for the UAV used in the model are described in Table 2.1, while the gain values used for the controller are listed in Table 3.15. The inner loop equations are described by (3.35), and the outer loop by (3.26).

The target markers used in the simulation has the first marker located at $X_1 = [0, -0.114, 0]^T$m, and the second marker at $X_2 = [0, 0.114, 0]^T$m. These points are chosen to simulate the target board used in Section 3.4 as shown in Figure 3.8. The desired pose of the vehicle is set to $X^* = [0, 0, -1.5]^T$m with $\psi^* = 0$. The vehicle is given an initial condition of $X_0 = [0.25, 0.25, -1]^T$m with a starting yaw of 45°, which simulates switching from a position controller at a non-ideal pose. Results from the simulation are plotted in Figure 3.6, which shows a stable response for the given initial conditions.

The error in position is described by $e_p = P^{n*} - P^n$ giving the relative distance from the reference, which can be seen in Figure 3.6b to be always converging, and close to zero at the point $t = 20$ s. For the error in image moment features, the error values are computed as $e_s = s^* - s$ where $s^* = [0, 0, 1, 0]^T$ and plotted in Figure 3.6a. The plot shows that the values exhibit similar behaviour as the position, and is converging to zero around $t = 20$ s. From Figure 3.6d, it can be seen that the vehicle corrects for the initial yaw to zero well before $t = 20$s when the other parameters reach their desired values. Therefore, from Figure 3.6 the controller chosen is stable as shown in Section 3.1. Comparing the results with values from [1] it can be seen that the new controller is slower at reaching the desired result taking twice the amount of time. Also, comparing the image feature trajectory plots, the new controller is overshooting the desired pose before correcting itself compared to [1] which directly goes to the desired position. These differences are likely due to the chosen gain values not being tuned perfectly for the platform, however, since the results were still stable, these gain values give an acceptable result for use in the real world experiment.

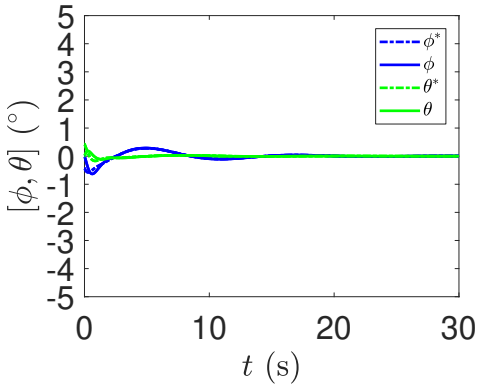
3.4 Experiment

For the experiment, multiple steps were taken to verify the results during implementation of the controller. The target is chosen as two semi-circular green markers

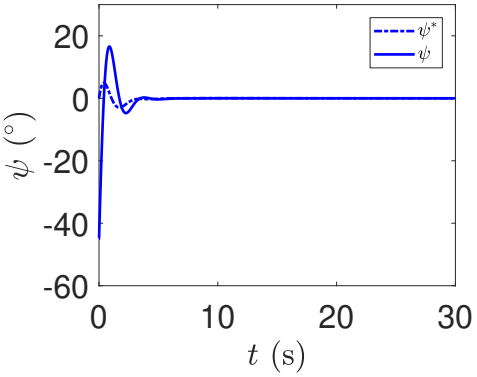


(a) Error in Image Moments s

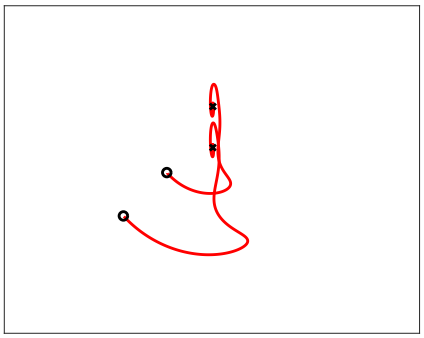
(b) Error in Position P^n



(c) Roll ϕ and Pitch θ



(d) Error in Yaw ψ



(e) Image Feature Trajectory

Figure 3.6: Simulated results using the proposed controller in [1] to control pose of UAV.

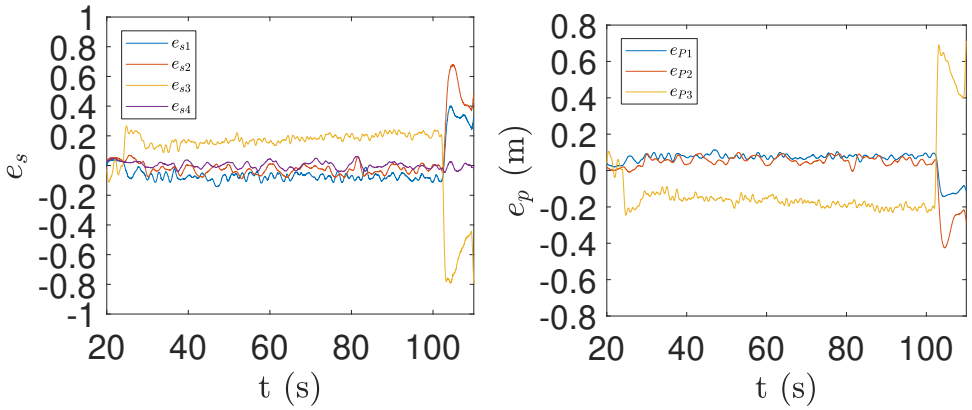
placed in the center of the Motion Capture System (MCS) space. The target marker shape is chosen to avoid having specular highlights, or bright spots, in the image which may cause the centers to be computed incorrectly. The semi-circular shape will be less likely to reflect the overhead lights directly back into the camera. For the experiment, values (s_1, s_2, s_3) used are normalized values to minimize impacts of noise on the image moment features used. Also, the thrust is normalized such that approximately half the total thrust will counteract the force of gravity. The experiment only replaces the outer loop control of a known working inner-outer loop position controller from [39], therefore, only outputs of the two outer loop controllers are compared. Reusing the inner loop controller is beneficial as both controllers use the same structure for the inner loop which reduces the need to remake and test the inner loop. It also eliminated the inner loop variable when tuning the gains, as the inner loop is already tuned for the vehicle. The first step taken to verify the controller is a hand held experiment, where the UAV will be held up in the air above the target, and the computed control outputs are checked to ensure that the control signals values are close to a known working controller. The hand held experiment runs the same setup as the final experiment, but with the motors unplugged to prevent accidental start up of the motors. This step is taken to ensure that the control signals are in a safe operating range to prevent damaging the vehicle and environment since large control signals may cause the vehicle to crash. The hand held experiments were done by having the Vicon MCS track the UAV, and providing the necessary pose information to the controllers used on the UAV. During the experiment, the working position controller is also turned on to record the output for comparison with the image-based visual servoing (IBVS) controller outputs. The UAV is moved in five second intervals to a new pose such that each pose would only affect one of the outputs, e.g. moving the UAV up to check if thrust increases, until all four outputs have been checked. If the output for both controllers are within a small range of each other, and with the same shape then it was concluded that the IBVS controller is giving a satisfactory output and the experiment may proceed to the next step. If the results were not showing good outputs, the controller gains are tuned and the experiment done again. The next step of the experiment entailed flying the UAV autonomously using the MCS with the known working position controller to test the IBVS controller under flight conditions. The outputs of the IBVS controller are compared to the position controller output to verify whether the outputs from the hand held test results carry over to the vehicle under real flight conditions since the states change rapidly.

Once the output shows promising results, more flights are done while replacing one of the four outputs from the position controller with a output from the IBVS controller. If the UAV shows signs of instability from hover, the flights are stopped

Gain	Value
k_{h1}	0.9
k_{h2}	0.28
k_{l1}	0.89
k_{l2}	0.28
k_{ψ}	0.6
$[k_{p\phi}, k_{i\phi}, k_{d\phi}]$	[10,1,3]
$[k_{p\theta}, k_{i\theta}, k_{d\theta}]$	[10,1,3]
$[k_{p\psi}, k_{i\psi}, k_{d\psi}]$	[6,1,3]

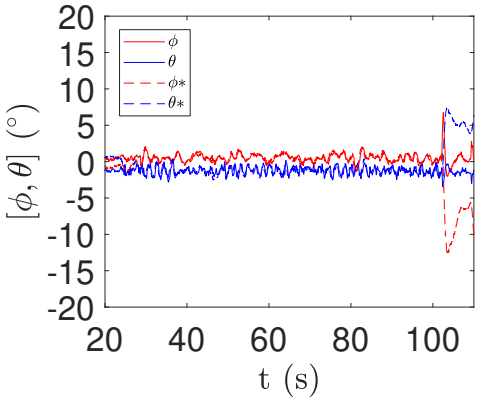
Table 3.15: Gains Used in IBVS Controller

then the logs from the PX4 are collected and analyzed to make appropriate changes to the controller gains. This test is done to ensure that each part of the IBVS controller is working while keeping the vehicle safe by limiting only one part of the control signals to the untested controller during flight, thus decreasing the risk of crashing the vehicle due to incorrect control signals. Also, if the vehicle was found to oscillate the flight was aborted to ensure the vehicle did not put itself into an unstable pose. The final values after tuning are listed in table 3.15 with the assumption that factors, such as m/T_M, is absorbed into the gains k due to being unable to directly measure the force T_M. Also, a constant is added to the thrust to account for gravity. After confirming that each part of the controller is working within the expected range the UAV is run for a longer duration, e.g. 60 s, to confirm whether the controller is stable or only marginally stable for the final experiment. The Vicon MCS is considered the ground truth and used to compute all the error values. To perform the experiment, the UAV is manually flown near the expected set point of $X = [0, 0, -1]^T$, at which point the MCS position controller is used to move and stabilize the vehicle to the set point. The position controller is used to ensure that the target will be in the field of view of the camera before switching to the IBVS controller. The position controller is only run for a limited amount of time to avoid draining the battery significantly. Once stabilized near the set point, the IBVS controller is engaged to control the vehicle. It should also be noted that the average position of the quadrotor may be offset a little from the direct center $(0, 0)$ of the MCS reference due to the target not being perfectly aligned with the rooms center. This offset is accounted for when setting up the reference value used to compute the image moment feature. After the trial time elapses, the vehicle is placed back into the position controller to again stabilize the vehicle before being landed manually.

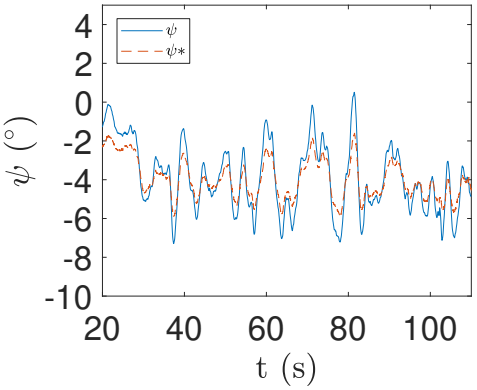


(a) Error in Image Moments s

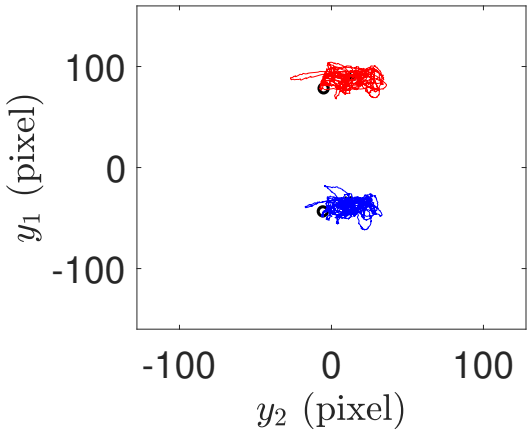
(b) Error in Position P^n



(c) Roll ϕ and Pitch θ



(d) Error in Yaw ψ



(e) Image Feature Trajectory

Figure 3.7: Experimental results using the DIBVS controller for pose control of UAV.

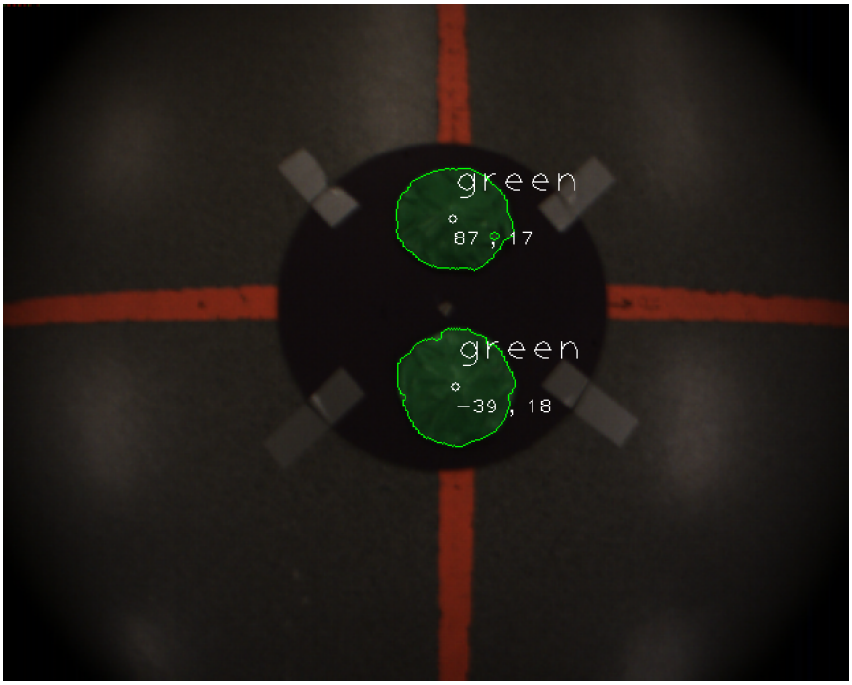


Figure 3.8: Snapshot of processed image during flight outlining green target markers

Error Name	Mean Value	Standard Deviation
e_{s1}	-0.077	0.021
e_{s2}	-0.030	0.027
e_{s3}	0.178	0.032
e_{s4}(rad)	0.001	0.021
e_{P1}(m)	0.078	0.011
e_{P2}(m)	0.058	0.020
e_{P3}(m)	-0.17	0.029
$e_{\psi}(^{\circ})$	1.62	1.61

Table 3.16: Average Errors in Values

After running the final experiment, it was found that the image moments were being computed at an average rate of 34.6 frames per second. The results are plotted in Figure 3.7 where the IBVS controller is running between the time $t = 28.5\,\text{s}$ to $t = 102\,\text{s}$ for a total flight time of $t = 72.5\,\text{s}$. During the transitions between controllers there is a chance that the vehicle may move away from the expected pose due to the non-smooth transition between different controller set points. This behaviour is seen in Figure 3.7b at $t = 103\,\text{s}$ when the UAV moved to the right as the position controller switched in. The errors for each output are computed and listed in Table 3.16. The position error is taken as $e_P = P^* - P$ where the reference value the measured center of the target given by the Vicon MCS which is acquired prior

to the flight tests. Errors in the image moment features are taken to be $e_s = s^* - s$ with the reference $s^* = [0, 0, 1, 0]^T$, and the reference μ^* used is measured from the computed values of the ROS package with the UAV held at the desired pose. The error in yaw is computed as $e_\psi = \psi^* - \psi$ where the reference yaw is orientation of the target measured using the MCS. A snapshot from the camera during the flight is shown in Figure 3.8.

From the results in Figure 3.7, it can be seen that with the IBVS controller running, the position and the image moments all reach a bounded value near the desired pose. It should be noted that the error in s_3 is expected due to the voltage of the battery dropping from idling as the subsystems boot and lack of integrator term in the controller resulting in decreasing thrust over time. However, as the reference signal is very stable and slowly increasing as the voltage is dropping, the thrust controller is showing the correct response by increasing the signal. Unfortunately, this increase in control signal is not enough to correct the position error. The results from [1] are given as follows: $[e_{s1}, e_{s2}, e_{s3}, e_{s4}]^T = [0.1185, 0.2162, 0.0395, 0.0612]^T$, $[e_{P1}, e_{P2}, e_{P3}]^T = [3.78, -0.16, -5.42]^T cm$, and $e_\psi = 2.86°$. Comparing the values to [1], the results from this experiment shows that the controller is in-line with the previous experiment which uses this controller. The experiment shows slight improvement to the image moment feature tracking and yaw values during the use of the IBVS, while the position error is slightly greater than the results from [1].

Chapter 4

Conclusion

4.1 Summary of Results

The goal of this thesis is to implement the controller from [1] using a different platform to improve performance of the algorithm. Testing was carried out using the Robot Operating System (ROS) framework to investigate its uses for creating a flexible framework for future computer vision (CV) systems. The experimental results confirmed the use of ROS to handle the image taken from a camera, and processing of the image frames to compute image moment features proposed in [32, 33]. The resulting image moment features are successfully used for motion control of the unmanned aerial vehicle (UAV). This result achieves the goal of implementing a flexible CV system in two aspects, the first is an implementation without strict hardware restrictions. Without the hardware restriction, the platform can be upgraded as required so long as ROS can be used and still run with minimal software changes. The second aspect is the ability to implement different CV algorithms with minimal changes to the platform reducing the amount of work required to implement and test different setups. Using the code validated from Section 3.2.4 experimental test were carried out on the platform where the results are analyzed in Section 3.4. From the results in Table 3.16, it can be concluded that the controller is working as tested in simulation, giving a stable flight while hovering above the target. Moreover, when comparing the experimental results to [1] similar error results are achieved with some improvements on the image moment features and yaw. These results indicate the thesis had succeeded in re-implementation of the controller on a different platform, and improved the algorithm performance.

4.2 Future Work

There are still lots of improvement work to be done on this project, one of which is implementation on the Graphics Processing Unit (GPU) for the processing of images. During implementation of the CV system on the Nvidia Jetson TX1 (TX1), the frame rate was initially found to be insufficient resulting in an unstable system during flight. To rectify the issue, the morphology operations were lowered to alleviate the computation times for each image resulting in a less robust result. The cause of the slow processing time was found to be the image processing algorithms, specifically the smoothing functions, taking too long to process images resulting in lower frame rates. Due to time constraints, the algorithm was implemented on the Central Processing Unit (CPU) which resulted in a frame rate of 34.6fps which is lower than some benchmarks[1] indicating frame rates of up 100fps. Therefore, using the GPU might be able to increase the rate of image moment feature updates to the outer loop controller leading to a faster response, and better error correction. Another area of work is the use of other CV based algorithms, such as the 8 point algorithm in [12], to replace the Vicon Motion Capture System (MCS) which could greatly improve the flexibility of the system. Currently, due to the requirement of the MCS providing velocity values to the vehicle it is restricted to be flown indoors within the set airspace of the MCS to use the controller without adding a new sensor. By replacing the MCS, future UAVs could be flown in more diverse environments without having to rely on the MCS providing a reference velocity for the controller. Moreover, the algorithm may solve some drifting problems that certain controllers experience. Also, this research can be extended to include the adaptive dynamic image-based visual servoing (DIBVS) controller proposed in [39]. Due to time constraints, the controller was only tested for the DIBVS controller, however, similar results found in the testing of the two DIBVS controller indicate that the adaptive DIBVS controller should be possible.

[1]Example benchmark result https://devblogs.nvidia.com/parallelforall/jetson-tx2-delivers-twice-intelligence-edge/

Bibliography

[1] G. Fink, H. Xie, A. F. Lynch, and M. Jagersand, "Experimental validation of dynamic visual servoing for a quadrotor using a virtual camera," in *Proceedings of the 2015 International Conference on Unmanned Aircraft Systems*, Denver, CO, Jun. 2015, pp. 1231–1240.

[2] C. Kanellakis and G. Nikolakopoulos, "Survey on computer vision for uavs: Current developments and trends," *Journal of Intelligent & Robotic Systems*, vol. 87, no. 1, pp. 141–168, Jul 2017.

[3] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, Mar./Apr. 2012.

[4] B. Godbolt, "Experimental nonlinear control of a helicopter unmanned aerial vehicle (UAV)," Ph.D. dissertation, Dept. of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, 2013.

[5] G. Fink, H. Xie, A. F. Lynch, and M. Jagersand, "Nonlinear dynamic visual servoing of a quadrotor," *Journal of Unmanned Vehicle Systems*, vol. 3, no. 1, pp. 1–21, 2015.

[6] S. Lupashin, A. Schollig, M. Hehn, and R. D'Andrea, "The flying machine arena as of 2010," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011, pp. 2970–2971.

[7] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

[8] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-UAV testbed," *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 56–65, Sep. 2010.

[9] J. How, B. Bethihke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51–64, Apr. 2008.

[10] F. Fraundorfer and D. Scaramuzza, "Visual odometry : Part II: Matching, robustness, optimization, and applications," *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 78–90, Jun. 2012.

[11] S. M. Weiss, "Vision based navigation for micro helicopters," Ph.D. dissertation, ETH Zürich, Zürich, Switzerland, 2012.

[12] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry, *An Invitation to 3D Vision: From Images to Geometric Models*. New York, NY: Springer-Verlag, 2003.

[13] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., ser. Cambridge Books Online. Cambridge, England: Cambridge University Press, 2003, vol. 1.

[14] F. Kendoul, I. Fantoni, R. Lozano *et al.*, "Asymptotic stability of hierarchical inner-outer loop-based flight controllers," 2008, pp. 1741–1746.

[15] H. Xie, "Dynamic visual servoing of rotary wing unmanned aerial vehicles," Ph.D. dissertation, University of Alberta, Feb. 2016.

[16] D. Scaramuzza and F. Fraundorfer, "Visual odometry: Part I: The first 30 years and fundamentals," *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, pp. 80–92, Dec. 2011.

[17] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[18] O. Kermorgant and F. Chaumette, "Combining ibvs and pbvs to ensure the visibility constraint," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 2849–2854.

[19] A. D. Wu, E. N. Johnson, and A. A. Proctor, "Vision-aided inertial navigation for flight control," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 9, pp. 348–360, 2005.

[20] O. Shakernia, Y. Ma, T. J. Koo, and S. Sastry, "Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control," *Asian Journal Control*, vol. 1, no. 3, pp. 128–145, Sep. 1999.

[21] O. Araar and N. Aouf, "Visual servoing of a quadrotor uav for the tracking of linear structured infrastructures," in *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3310–3315.

[22] T. Hamel and R. Mahony, "Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 2, pp. 187–198, 2002.

[23] H. Xie, A. F. Lynch, and M. Jagersand, "Dynamic IBVS of a rotary wing UAV using line features," *Robotica*, vol. 34, no. 9, pp. 2009–2026, 2014.

[24] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck, "Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 743–749, Jun. 2009.

[25] N. Metni, T. Hamel, and F. Derkx, "Visual tracking control of aerial robotic systems with adaptive depth estimation," in *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, Seville, Spain, Dec. 2005, pp. 6078–6084.

[26] H. de Plinval, P. Morin, P. Mouyon, and T. Hamel, "Visual servoing for underactuated VTOL UAVs: a linear, homography-based framework," *Int. J. Robust. Nonlinear Control*, vol. 24, no. 16, pp. 2285–2308, Apr. 2013.

[27] R. Ozawa and F. Chaumette, "Dynamic visual servoing with image moments for a quadrotor using a virtual spring approach," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011, pp. 5670–5676.

[28] ——, "Dynamic visual servoing with image moments for an unmanned aerial vehicle using a virtual spring approach," vol. 27, no. 9, pp. 683–696, 2013.

[29] D. Lee, T. Ryan, and H. Kim, "Autonomous landing of a VTOL UAV on a moving platform using image-based visual servoing," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, Saint Paul, MN, 2012, pp. 971–976.

[30] D. Lee, H. Lim, H. Kim, Y. Kim, and K. Seong, "Adaptive image-based visual servoing for an underactuated quadrotor system," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 4, pp. 1335–1353, 2012.

[31] H. Jabbari Asl, G. Oriolo, and H. Bolandi, "An adaptive scheme for image-based visual servoing of an underactuated UAV," *International Journal of Robotics and Automation*, vol. 29, no. 1, pp. 92–104, 2014.

[32] F. Chaumette, "Image moments: a general and useful set of features for visual servoing," *Robotics, IEEE Transactions on*, vol. 20, no. 4, pp. 713–723, Aug. 2004.

[33] O. Tahri and F. Chaumette, "Point-based and region-based image moments for visual servoing of planar objects," *Robotics, IEEE Transactions on*, vol. 21, no. 6, pp. 1116–1127, Dec. 2005.

[34] H. Xie, G. Fink, A. F. Lynch, and M. Jagersand, "Adaptive dynamic visual servoing of a UAV," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 5, pp. 2529–2538, 2016.

[35] H. Xie and A. F. Lynch, "Input saturated visual servoing for unmanned aerial vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 2, pp. 952–960, Apr. 2017.

[36] M. Bryson and S. Sukkarieh, "Architectures for cooperative airborne simultaneous localisation and mapping," *Journal of Intelligent Robotic Systems*, vol. 55, no. 4-5, pp. 267–297, Aug. 2009.

[37] ——, *Inertial Sensor-Based Simultaneous Localization and Mapping for UAVs*. Dordrecht: Springer Netherlands, 2015, pp. 401–431.

[38] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, Nara, Japan, Nov. 2007, pp. 225–234.

[39] G. Fink, "Computer vision for unmanned aerial vehicles (uavs)," Ph.D. dissertation, University of Alberta, Jan. 2018.

[40] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation*, Seattle, WA, May 2015, pp. 6235–6240.

[41] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

****

Appendix A

Camera Calibration

For this thesis two methods were tested to acquire the camera intrinsic parameters which are listed in Appendices A.0.1 to A.0.2. The matrix used for the thesis was the matrix result that gave the smaller error, which was found to be the Matlab variation.

A.0.1 ROS Alternative

The camera parameters were acquired by using the camera calibration[1] ROS package, and compiled in a local ROS environment. To acquire the calibration matrix the following steps were taken:

1. Clone the project into a catkin workspace, and build the project using the catkin_make command

2. Create or use the checkerboard calibration pattern that is provided with the camera_calibration[2] ROS package.

3. Run your ROS package for the camera, and confirm the camera is publishing images from the camera to a ROS topic. For this document a Pointgrey, a.k.a. FLIR, chameleon 2 camera is used.

4. Run the calibration tool using the following command, replacing the options, such as size and square, to match your set up.

   ```
   rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.025 image:=/camera/image_raw camera:=/camera
   ```

5. Once the calibration window is open move the calibration board from left to right, and top to bottom of the camera image. During the movement also tilt the board to add skew to the data points. Keep moving the board until the calibration button lights up to indicate enough points are collected.

[1] http://wiki.ros.org/camera_calibration
[2] http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

6. Click the calibration button, and wait for a while. Once done the parameters will appear on the terminal or use the save button to save parameters.

The projection matrix was found for an image with the size of 1280x960. Therefore, the image center as indicated in the projection matrix (660,449) for (y_1, y_2) respectively, which is close to half of the resolution or the assumed center of the image. To ensure that the values computed in theory match the values being computed in experiment the image centers are subtracted such that any value computed would be centered around the center point.

A.0.2 Matlab Alternative

Another method for getting the projection matrix would be to use a Matlab camera_calibration toolbox.[3] Using the Matlab calibration toolbox is recommended since the error in the projection matrix is given for all values being computed. To use the toolbox the same tools as the ROS package is used, but a set of images has to be acquired beforehand with a known naming scheme, ie. image_1.jpg,image_2.jpg... for all images. Once these conditions are satisfied the following steps can be use to find the projection matrix.

1. Start the toolbox by adding it to path and running the command cablib_gui and selecting either mode
2. Choose the extract corners options and identify the corners of the calibration board. Note that the size of the squares has to be entered in millimetres.
3. After all images have their corners extracted the calibration button can be used to compute the projection matrix

optional Using the computed estimates the corners can be extracted again and the calibration sequence run again. Note that the results may be better or worse than the initial values

optional Blurry images can also be removed to give a better result

For the writing of this document the calibration board used is an 8x6 checkerboard pattern that has squares of the size 2.5cm.

[3] http://www.vision.caltech.edu/bouguetj/calib_doc/

Appendix B

Image Moment Validation

B.1 Experiment Procedure

The steps to perform the experiments is listed below, and the experimental conditions for each of the 9 experiments are listed in Table 3.3.

1. Set camera and target on level surfaces
2. Measure distance from the front of the target to the middle of the camera. This measurement is going to be the depth for the target points for calculating the theoretical values.
3. Run the Matlab script supplying it with the specified depth measured from the step before
4. Run the image_moments_calibration ROS node while supplying the μ^* from the previous step unless testing the value of s_3.
5. Move the target, without changing its depth, such that the two marked centers are aligned
6. For any offset move the target the specified distance. For any angles move either the target or camera the specified angle
7. Once target and camera are in position, read the values of s_1-s_4

B.2 Computations

Below is the intermediate steps for the computation of the error for the bottom marker, (u_2, v_2), and the left marker, (u_3, v_3), in the reference test condition. Please note that the other two markers gives the same error as its opposing marker, due to the symmetry of the markers on the target board.

$$\Delta u_2 = \sqrt{\left(\frac{2194}{1.6} * 0.001\right)^2 + \left(\frac{0 * 2194}{1.6^2} 0.02\right)^2 + \left(\frac{0}{1.6} 4.67\right)^2 + (2.26)^2} \tag{B.1}$$

$$\Delta u_2 = \sqrt{\left(1.37^2\right) + (0)^2 + (0)^2 + (2.26)^2} = 2.64 \tag{B.2}$$

$$\Delta v_2 = \sqrt{\left(\frac{2192}{1.6} * 0.001\right)^2 + \left(\frac{-0.0740 * 2192}{1.6^2} 0.02\right)^2 + \left(\frac{-0.0740}{1.6} 4.55\right)^2 + (2.57)^2} \tag{B.3}$$

$$\Delta v_2 = \sqrt{\left(1.37^2\right) + (-1.27)^2 + (-0.210)^2 + (2.57)^2} = 3.18 \tag{B.4}$$

$$\Delta u_3 = \sqrt{\left(\frac{2194}{1.6} * 0.001\right)^2 + \left(\frac{-0.105 * 2194}{1.6^2} 0.02\right)^2 + \left(\frac{-0.105}{1.6} 4.67\right)^2 + (2.26)^2} \tag{B.5}$$

$$\Delta u_3 = \sqrt{\left(1.37^2\right) + (-1.79)^2 + (-0.305)^2 + (2.26)^2} = 3.21 \tag{B.6}$$

$$\Delta v_3 = \sqrt{\left(\frac{2192}{1.6} * 0.001\right)^2 + \left(\frac{0 * 2192}{1.6^2} 0.02\right)^2 + \left(\frac{0}{1.6} 4.55\right)^2 + (2.57)^2} \tag{B.7}$$

$$\Delta v_3 = \sqrt{\left(1.37^2\right) + (0)^2 + (0)^2 + (2.57)^2} = 2.91 \tag{B.8}$$