NTNU
Norwegian University of Science and Technology

Autonomous landing of multi-rotor UAV

Øystein Reitstøen Lasson

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Tor Arne Johansen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

Unmanned **A**erial **V**ehicles (UAVs) has seen increased increased use in marine observations systems, [1], [2]. The **O**cean-**A**ir synoptic operations using coordinated autonomous robotic **Sys**tems and micro underwater gliders (OASYS) project seeks to decrease the cost of marine observation system through increased automation. One aspect of this project is automating the operation of multirotor UAVs.

This thesis is not the continuation of a project thesis. As a starting point, the author received a problem description, access to the NTNU UAV-lab and access to the UAV itself. Supervisor Tor Arne Johansen has provided several useful suggestions concerning the theoretical approach to the problem. Co-supervisor Artur Piotr Zolich has provided advice and suggestions concerning the practical experiments. However, the author has ultimately been responsible for the theoretical approach and derivation, choice of software, software implementation, choice and acquisition of hardware and planning and conducting experiments.

I would like to express my gratitude to professor Tor Arne Johansen for his help with the theoretical aspects of this thesis.

I would also like to express my gratitude to Artur Piotr Zolich for his help with the practical aspects of this thesis.

Øystein Lasson
NTNU, June 2018

Abstract

A GPS and camera-based landing system for navigating a multirotor UAV from a set of arbitrary GPS coordinates to a floating platform has been proposed. The camera-based sub-system uses a target (a specific geometrical marking), in order to simplify the process of finding the platform. In terms of robust target detection, an attempt has been made to address the issue of illumination invariance (detecting the target under varying light conditions) and filtering out visual noise (objects mistakenly identified by the system as the target). The 3D (X, Y, Z) position estimation proposed has undergone motionless tests with, and without, attitude compensation. For both cases, the absolute error never exceeded 8% of the ground truth measurements.

Sammendrag

Et GPS og kamerabasert landingssystem for å tillate navigering av en multirotor UAV fra et sett med vilkårlige GPS koordinater tilbake til en flytende plattform har blitt foreslått. Det kamerabaserte delsystemet bruker et mål (en spesifikk geometrisk markør) for å forenkle lokalisering av plattformen. Når det kommer til robust måldeteksjon har det blitt forsøkt å ta høyde for varierende belysning (finne målet under forskjellige lysforhold) og å filtrere ut visuelt støy (objekter som har blitt feilidentifisert som målet av systemet). 3D (X, Y, Z) posisjons estimeringen som er foreslått har undergått testing med og uten rotasjons-kompensasjon. I begge tilfeller overskrider aldri den absolutte feilen aldri 8% av de "sanne" målingene.

List of Figures

List of Tables

Acronyms

BODY **Body**-fixed. 7–9, 11, 43, 67

CAM **Cam**era-fixed. 8, 9, 11, 16, 37, 38, 43, 57, 67

FOV **F**ield **O**f **V**iew. 3, 4, 22, 24, 25, 29, 37, 46

GPS **G**lobal **P**ositioning **S**ystem. 2–4, 21–25, 29, 46, 65, 71, 72

HSV **H**ue **S**aturation **V**alue. 14, 34

IMU **I**nertial **M**easurement **U**nit. 3, 4, 11, 42, 43, 67, 72

MSL **M**ean **S**ea **L**evel. 26

NED **N**orth-**E**east-**D**own. 7, 9, 11, 21, 22, 26, 41, 42, 57

OASYS **O**cean-**A**ir synoptic operations using coordinated autonomous robotic **Sys**tems and micro underwater gliders. i

RGB **R**ed **G**reen **B**lue. 12–14, 32–34

UAV **U**nmanned **A**erial **V**ehicle. i, 1–5, 7, 8, 10, 21–27, 29, 31, 32, 37, 42, 46–48, 56, 63, 65, 67, 71–73

Contents

Chapter 1

Introduction

An **U**nmanned **A**erial **V**ehicle (UAV) is an aircraft that operates without a human crew on board. It can either fly autonomously based on pre-programmed flight plans, or from a remote station. While UAV's are often used in military operations, they have seen increased usage in civilian applications as well.

UAV is an umbrella term covering a wide range of geometric configurations. Broadly speaking, they can be divided into *rotorcraft* and *fixed-wing*. This thesis is only concerned with rotorcrafts. Common for all configurations of rotorcrafts is their reliance on rotors to generate lift, yielding a couple of distinct advantages over their fixed-wing counterparts, namely, vertical landing and takeoff, ability to hover over a specific point and ability to operate at low airspeeds.

This thesis is mainly concerned with prototyping a vision based landing system for a rotorcraft UAV.

1.1. Thesis Objectives

The overall goal of this thesis is to investigate how visual navigation for autonomous landing of a multi-rotor UAV can be achieved. Detailed thesis objectives follows below.

1. **Task:** Present a high-level theoretical overview of how the UAV could navigate from a set of arbitrary GPS coordinates to an anchored landing platform at sea.

2. Investigate how a vision-based system can be made robust with respect to illumination invariance and visual noise. See detailed objectives below.
    (a) The vision system should be able to handle varying light conditions.
        i. **Task:** Investigate how illumination invariance can be achieved, propose a method for addressing the problem of illumination invariance, and conduct experiments to validate said method.
    (b) Here, *visual noise* is defined as objects misidentified by the system as the landing target. Even in the presence of visual noise, the system should be able to correctly identify the target.
        i. **Task:** Investigate how robustness to visual noise can be achieved, propose a method for filtering out visual noise, and conduct experiments to validate said method.
3. **Task:** Present a theoretical proposal for a vision-based 3D-position estimation algorithm, and conduct experiments to validate its efficiency.

1.2. Literature Review

In order to land autonomously, the UAV must be able to navigate its environment. Navigation is dependent on two important design choices, the control scheme used and the set of sensors used [8]. Generally speaking, there needs to be a sensible, weighted assessment of several factors, namely price, total weight of the UAV and performance depending on the exact task the UAV is supposed to undertake.

A common approach to measuring orientation and position is a **I**nertial **M**easurement **U**nit (IMU) for orientation and **G**lobal **P**ositioning **S**ystem (GPS) for positioning. However, GPS might have severe errors in accuracy. In [9], it is noted that the low cost GPS used has horizontal errors of $\pm 2m$ and vertical errors of up to $\pm 5m$, making it extremely unreliable for precision landing. This uncertainty in GPS measurements is a huge motivating factor for exploring vision based solutions for landing.

Vision based landing has multiple advantages over other sensor schemes [10], [11], [12]. In the context of unmanned flights, several camera characteristics are desirable. They are generally light, cheap, and applicable both indoors and outdoors. Additionally, cameras are considered *passive* sensors, meaning that they themselves do not emit external signals that can interfere with other on-board systems.

In order to estimate the position of the UAV relative to a target, a common approach starts with the *projection equations* [13], [14], [15], [16], [17], [9]

$$x_p = f\frac{X}{Z}, \qquad y_p = f\frac{Y}{Z} \tag{1.1}$$

where f is the focal length of the camera, (X, Y, Z) [m] is a point in the world, and (x_p, y_p) [m] represents the projection into the image plane as shown in figure 1.1.

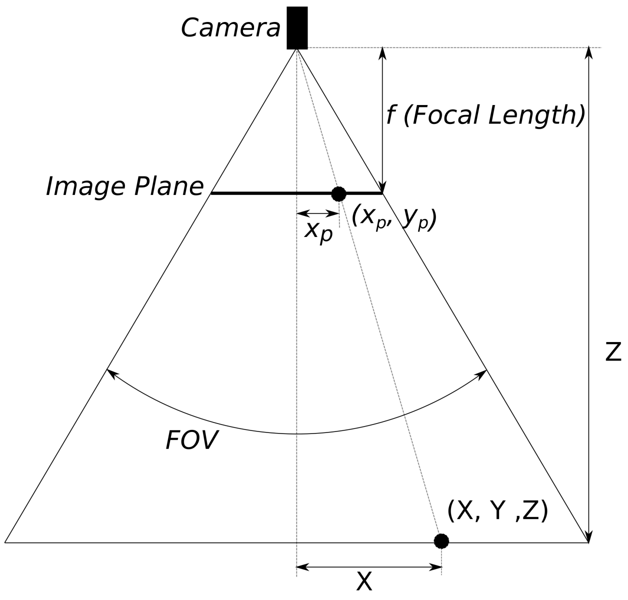


Figure 1.1: Birds-eye view of image projection. A 3D point (X, Y, Z) [m] in the world is projected into the image image plane as a 2D point $x_p = f(X/Z)$ [m], $y_p = f(Y/Z)$ [m] where f [m] is the focal length of the camera (the distance from the optical center to the image plane). FOV $[deg]$ denotes the cameras **F**ield **O**f **V**iew.

One fundamental limitation of applying (1.1), is that (X, Y, Z) must be in the cameras **F**ield **O**f **V**iew (FOV) as shown in figure 1.1.

Withing the scope of this thesis, the goal would be to estimate (X, Y, Z) in (1.1). Assuming that (x_p, y_p) is known, that leaves three unknowns (X, Y, Z) but only two equations. Finding estimates for (X, Y, Z) often begins with first estimating Z which is a process called *range finding*.

In [13], a range finding method based on *optical flow*[1] is presented. The optical flow

[1]Optical flow is the apparent movement of objects between two frames in an image sequence.

equations (the time derivative of (1.1)) given by [13, eq. (2)] is solved for Z. Next, (1.1) can be solved for X and Y to provide an estimate of the horizontal position.

An alternative approach is presented in [18], where multiple cameras, (stereo vision) is used to determine Z. The main idea is that if the distance between two cameras *Left* (L) and *Right* (R) are known, the range can be found by determining the position of a point in (L), relative to its position in (R). Yet another approach is using recursive least squares to estimate Z as shown in [9].

It is also possible to combine the camera with other sensors. For example, [19, Chapter 3] uses a camera and an IMU to achieve a complete state estimation. Put simply, the camera is used to obtain the position, and the IMU is used to obtain the orientation (roll, pitch, yaw) and other angular states. Other sensor combinations exists as-well, [20] uses data from a camera, data from an on-board GPS and data from an on-board IMU as input to a Kalman-filter algorithm. In [21] a camera is combined with a *laser range finder* and an on-board IMU to navigate.

If a camera based sensor scheme is chosen, there is another important consideration. The UAV's velocity must be seen in the context of the on-board system's ability to process images. In [22, 1.1 Challenges of Vision-Based Helicopter Flight] it is claimed that the on-board image processing must happen at a frame rate of *at least* 30Hz in order to achieve efficient vision based object tracking.

Target detection, in this context, essentially refers to finding the platform. More specifically, the target is some kind of geometrical marking placed on the platform, making it easier for the on-board vision system to locate and navigate to the platform. This section seeks to explore different ways to achieve this.

In [15], the geometrical shape of the landing target is four white squares of varying sizes. Additionally, they apply binary thresholding to turn the surrounding are black. They preform feature extraction by detection corners.

In [23] the geometrical marking used is a single, solid blue square. A Gaussian 2D filter is first applied to smooth the image before it is converted from the RGB color-space to the HSV color space. The authors note that the RGB values can vary greatly depending on the light conditions, which is why HSV is used instead. Once a HSV image has been obtained, binary thresholding is applied before calculating the center of the resulting binary blob.

In [9], the target consist of a hexagon-shaped platform with several white colored nested circles on it. The authors chose this geometrical marking due to its uniqueness (hexagons rarely, if ever, occur naturally reducing the risks of false positives), scalability (the differently sized circles allow for target detection at both short and long distances), and simplicity (circles has well-known properties that are easy to exploit in an algorithm). They convert the RGB image to gray-scale before also applying a binary threshold.

1.3. Thesis Outline

The overall structure of the thesis is outlined below. Each chapter is meant to address different aspects relevant to eventually implementing a complete system.

- **Chapter 2: Theory**. This chapter presents the most relevant theory used to design the system in later chapters.
- **Chapter 3: System Overview**. This chapter proposes what a complete system would could eventually look like. More specifically, several modes of operation for the landing system is proposed and described.
- **Chapter 4: Vision System**. This chapter describes how camera images can be used to obtain the relative position of the UAV with respect to the target. It is largely based on the theory presented in chapter 2.
- **Chapter 5: Experiments** This chapter presents the experiments done in an attempt to offer empirical validation of the theoretical concepts from chapter 4.
- **Chapter 6: Conclusion** This chapter presents the tentative conclusions drawn, lessons learned and suggestions for future work.

Chapter 2

Theory

In this chapter, the fundamental theoretical concepts for the vision-based algorithm will be covered.

2.1. Reference Frames

In order to accurately describe the position of the the UAV, in the world, several reference frames must be defined. All reference frames will follow the *right-hand-rule*, as seen in figure 2.1.

Based on the right-hand-rule, the following frames are defined.

- **NED:** The **N**orth-**E**east-**D**own (NED) reference frame $\{n\} = \{X^n, Y^n, Z^n\}$ has its origin $\mathbf{O^n}$ fixed at the center of the landing platform. However, the NED axis does not rotate with the platform. The X^n-axis points towards the *true North*[1], the Y^n-axis points towards *East* and the Z^n axis points *downwards* normal to the Earths surface[24, p.17].
- **BODY:** The **Body**-fixed-fixed (BODY) reference frame $\{b\} = \{X^b, Y^b, Z^b\}$ has its origin $\mathbf{O^b}$ fixed at the center of UAV. The X^b-axis points *forward*, the Y^b axis points to the *right* and the Z^b- axis points from top to bottom [24, p.17].
- **CAM:** The **Cam**era-fixed (CAM) reference frame $\{c\} = \{X^c, Y^c, Z^c\}$ as its

[1]The direction towards the geographic north pole

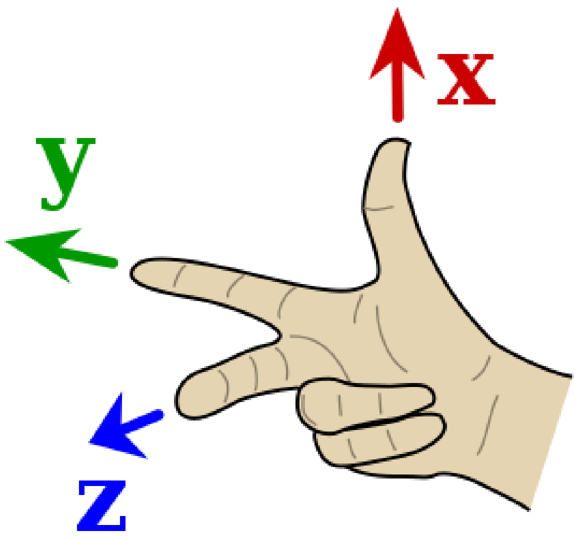


Figure 2.1: The orientation of the Cartesian x, y and z axis when using the right hand rule. Image courtesy of [3].

origin $\mathbf{O^c}$ fixed to the camera aperture[2]. The X^c-axis points *right*, the Y^c axis points *backwards* and the Z^c- axis points towards the ground.

See figure 2.2 for a sketch of how the different frames relate to each other. Due to the construction of the UAV the origins of the CAM and BODY are placed in the same point i.e $\mathbf{O}^b = \mathbf{O}^c$.

2.2. Rotation and Attitude

There are several possibilities for representing the attitude and orientation of the UAV. Two common representations are *Euler angles* and *Unit Quaternions*.

Euler angles are parameterized by [24, p.22]

$$\Theta = [\phi \quad \theta \quad \psi]^T \tag{2.1}$$

[2]The camera aperture is an opening or a hole which light travels through.

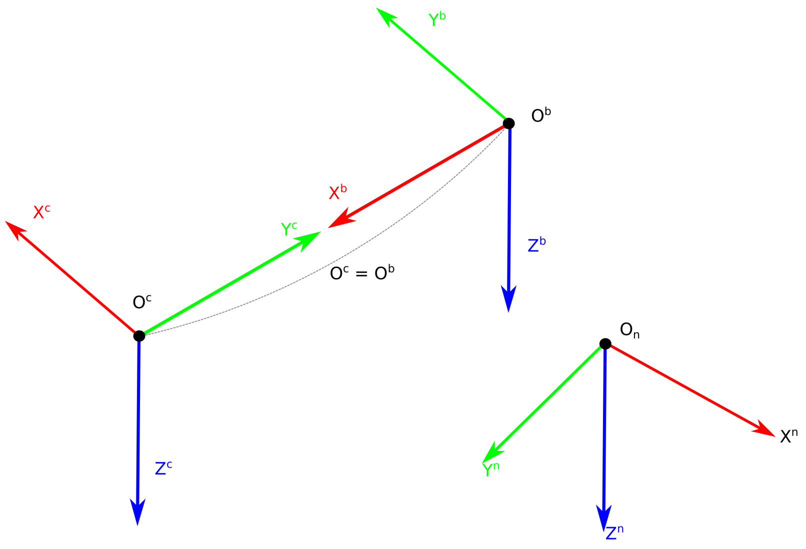


Figure 2.2: Sketch of how the CAM $\{c\}$ frame, BODY frame $\{b\}$ and NED frame $\{n\}$ relates to each other. The BODY and CAM frames are drawn separately for clarity, but their origins are the same i.e $\mathbf{O}^b = \mathbf{O}^c$. Also note that the CAM and BODY frames are meant to have an arbitrary orientation relative to the NED frame.

where ϕ (roll) is rotation about the x-axis, θ (pitch) is rotation about the y-axis and ψ (yaw) is rotation about the z-axis [24, p.22]. Euler angles are fairly easy to understand for a human, and they are good for decomposing rotations into individual degrees of freedom. However, they suffer from a singularity in $\theta = \pm 90°$ [24, p.25] and can suffer from *Gimbal lock* (loss of one degree of freedom) for certain rotations.

An alternative to Euler angles is *unit quaternions* parameterized by [24, p.27]

$$\mathbf{q} = [\eta + i\epsilon_1 + j\epsilon_2 + +k\epsilon_3]$$

where η is the real part and ϵ_1, ϵ_2 and ϵ_3 are the imaginary parts. The main motivation for using quaternions is to avoid the singularity of the Euler angles [24, p.27]. Compared to Euler angles, quaternions are also computationally faster, but they are more difficult to understand on an intuitive level.

Within the scope of this thesis, the UAV is not expected to preform "acrobatic" movements (i.e large values for roll (ϕ) and pitch (θ). Therefore, the singularity in $\theta = \pm 90°$ is deemed highly unlikely to present a problem. While quaternions would be more optimal in terms of computation, the intuitive nature of the Euler angles are weighted higher during the development of the system. Based on this reasoning, Euler angles as shown in (2.1) are chosen for representing the various rotations and the attitude of the UAV.

The notation introduced by [24] will be used for convenience. Let

$$\cos(\theta) = c\theta \quad \text{and} \quad \sin(\theta) = s\theta$$

Additionally, for clarity, the usage of the subscript and superscript will be specified, according to [24, p.20, eq]. Consider a transformation between two vectors $\mathbf{v}^l$ and $\mathbf{v}^k$ in arbitrary reference frames $\{l\}$ and $\{k\}$ consisting of a rotation and translation. The transformation *from* $\{k\}$ *to* $\{l\}$ is given by

$$\mathbf{v}^l = \mathbf{R}_k^l \mathbf{v}^k \quad \text{reads as} \quad \mathbf{v}^{to} = \mathbf{R}_{from}^{to} \mathbf{v}^{from}$$

Rotating an angle about the $\{x, y, z\}$ axis are given by [24, p.22, eq. (2.15)]

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \tag{2.2}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 0 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \tag{2.3}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

Note that the matrices in (2.2), (2.3) and (2.4) are orthogonal which implies the following property [24, p.20, eq. (2.7)]

$$\mathbf{R}^{-1} = \mathbf{R}^T \quad \text{and} \quad \left(\mathbf{R}_k^l\right)^T = \mathbf{R}_l^k \tag{2.5}$$

where $\{l\}$ and $\{k\}$ are arbitrary reference frames. Once the rotation from $\{l\}$ to $\{k\}$ is found, the rotation from $\{k\}$ to $\{l\}$ can immediately be found by (2.5). Also note that the

Next, a sequence of rotations is defined according to [24, p. 22], which is called the *(zyx)-convention*. Let $\Theta_{nb} = \begin{bmatrix}\phi^{nb} & \theta^{nb} & \psi^{nb}\end{bmatrix}^T$ describe the Euler angles between the $\{n\}$ NED-frame and $\{b\}$ BODY-frame. The rotation *to* $\{n\}$ *from* $\{b\}$ is then given by [24, p.22 eq. (2.16)]

$$\begin{aligned}\mathbf{R}_b^n(\Theta_{nb}) &= \mathbf{R}_z(\psi^{nb})\mathbf{R}_y(\theta^{nb})\mathbf{R}_x(\phi^{nb}) \\ &= \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}\end{aligned} \tag{2.6}$$

where the An on-board **I**nertial **M**easurement **U**nit (IMU) will be responsible for providing Θ_{nb}. Next, let $\Theta_{cb} = \begin{bmatrix}\phi^{cb} & \theta^{cb} & \psi^{cb}\end{bmatrix}^T$ be the angles between the CAM-frame and BODY-frame. The camera will be mounted to the same rigid body as the IMU, implying $\Theta_{cb} = \mathbf{const}$. Next, from figure 2.2, it can be seen that

$$\phi^{cb} = 0°, \quad \theta^{cb} = 0°, \quad \psi^{cb} = -90° \tag{2.7}$$

$$\begin{aligned}\mathbf{R}_b^c(\Theta_{cb}) &= \mathbf{R}_z(\psi^{cb})\mathbf{R}_y(\theta^{cb})\mathbf{R}_x(\phi^{cb}) \\ &= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{2.8}$$

Finally, note that (2.5) applied to (2.6) or (2.8) will yield the inverse rotational transformation Which one is used is a matter of convenience.

2.3. Image Processing

2.3.1. Binary Thresholding

As shown in the literary review in section 1.2, thresholding and more specifically, *binary thresholding* was a rather popular approach to simple image segmentation for vision based landing.

The principle is quite straightforward. As an exampled, consider a pixel with spatial coordinates (u, v) in an image $\mathbf{C}(u, v)$. Here, $\mathbf{C}(u, v)$ can be a single numerical value (gray-scale image) or a vector (RGB image). Next, a condition $Condition(\mathbf{C}(u, v), \mathbf{T})$ which is a binary function of a pixel and a threshold value $\mathbf{T}$ (which can also either be a single scalar or a vector) can be placed on each pixel. The binary pixel $I_{bin}(u, v)$ corresponding to $\mathbf{C}(u, v)$ can then be defined as

$$I_{bin}(u, v) = \begin{cases} 1 & \text{if} \quad Condition(\mathbf{C}(u, v), \mathbf{T}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where $Condition$ uses one or more binary operator(s) (depending on the specific purpose of the thresholding) $(\leq, \geq, <, >, \neq, ==)$ etc. to return either 0 or 1. As a concrete example, if $C(u, v)$ is a gray-scale pixel consisting of a single intensity value and T_{gray} is some numerical threshold value, then

$$I_{bin}(u, v) = \begin{cases} 1 & \text{if} \quad C(u, v) > T_{gray} \\ 0 & \text{otherwise} \end{cases}$$

will set all pixels with intensity greater than T_{gray} to one, and all other pixels to zero in the binary image.

2.3.2. Color-spaces

A *color space* outlines an abstract mathematical model for representing colors. A generic example is

$$\text{Color} = \left[\text{value}_1 \ \text{value}_2 \ ...\text{value}_n\right]^T$$

that is, the color is represented as a set of values. There exists a wide range of color spaces, each with their own advantages and disadvantages, depending on what one is trying to achieve.

Red-Green-Blue

The **R**ed **G**reen **B**lue (RGB) color space is arguably the most commonly used color space. It consist of three color channels which can be thought of as "layers" in an image. A visual representation of the RGB color space can be seen in figure (2.3).

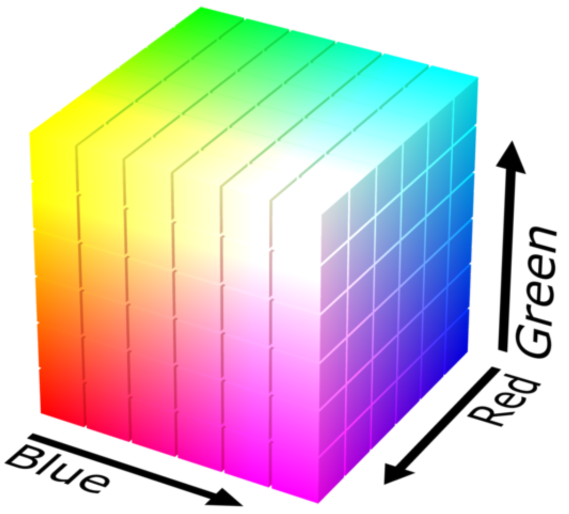


Figure 2.3: Visual representation of the 3D RGB color-space. Each RGB color is a point contained within the cube, and how far one moves along each axis will describe how much red, green or blue a color consist of. Image courtesy of [4].

Consider an RGB image $\mathbf{I}_{RGB}$ with arbitrary spatial dimensions V (height) and U (width). Mathematically, the image can be seen as a matrix with dimensions $(U \times V \times 3)$. Let $0 \leq u \leq U$ and $0 \leq v \leq V$ be arbitrary spatial coordinates of a pixel in the RGB image. Evaluating I_{RGB} at (u, v) then returns a three element color vector on the form

$$\mathbf{C}_{RGB}(u,v) = \begin{bmatrix} R(u,v) & G(u,v) & B(u,v) \end{bmatrix}^T \tag{2.9}$$

where the color components $R = R(u,v))$, $G = G(u,v)$ and $B = B(u,v)$ are often limited to the range $[0, 255]$. The amount of each component (i.e how far along you are on each colors axis in figure 2.3) will determine what color the pixel at (u, v) is.

The "naive" RGB-model described (2.9) is not particularly robust to changes in illumination. If the same scene is captured after being exposed to some light source the change in intensity can be modeled by multiplying (2.9) by some unknown factor β

$$\beta \mathbf{C}_{RGB}(u,v) = \beta \begin{bmatrix} R(u,v) & G(u,v) & B(u,v) \end{bmatrix}^T \tag{2.10}$$

It is fairy easy to derive a representation of the three components in (2.10) that is independent of β [25]

$$\overline{\mathbf{C}}_{RGB}(u,v) = \begin{bmatrix} \frac{R}{R+G+B} & \frac{G}{R+G+B} & \frac{B}{R+G+B} \end{bmatrix}^T \tag{2.11}$$

Now, if (2.11) is applied to all pixels in an RGB image $\mathbf{I}_{RGB}$, it will produce $\overline{\mathbf{I}}_{RGB}$ which is a *normalized* RGB image. The representation (2.11) is then more robust to changes in lighting conditions than (2.9), but it should be noted that this is based on a very simplistic model of the camera response. It is not necessarily the case that all components of (2.9) is scaled by the same factor β, and much more sophisticated methods for illumination-invariance exists [25].

Hue Saturation Value

The **H**ue **S**aturation **V**alue (HSV) color-space is an alternative representation of the RGB color-space. In this model, colors are represented in a cylindrical space, as shown in figure 2.4.

Let $\mathbf{C}$ be a RGB color vector as shown in (2.9). The HSV components seen in figure (2.4) are mathematically related to $\mathbf{C}$ as follows (adapted from [26, eq. (2), (3) and (4)])

$$V = \max(\mathbf{C}_{RGB}), \qquad \mathbf{C}_{RGB} = \begin{bmatrix} R & G & B \end{bmatrix}^T$$

$$S(\mathbf{C}_{RGB}) = \begin{cases} \frac{\max(\mathbf{C}_{RGB}) - \min(\mathbf{C}_{RGB})}{\max(\mathbf{C}_{RGB})} & \text{if} \quad \max(\mathbf{C}_{RGB}) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

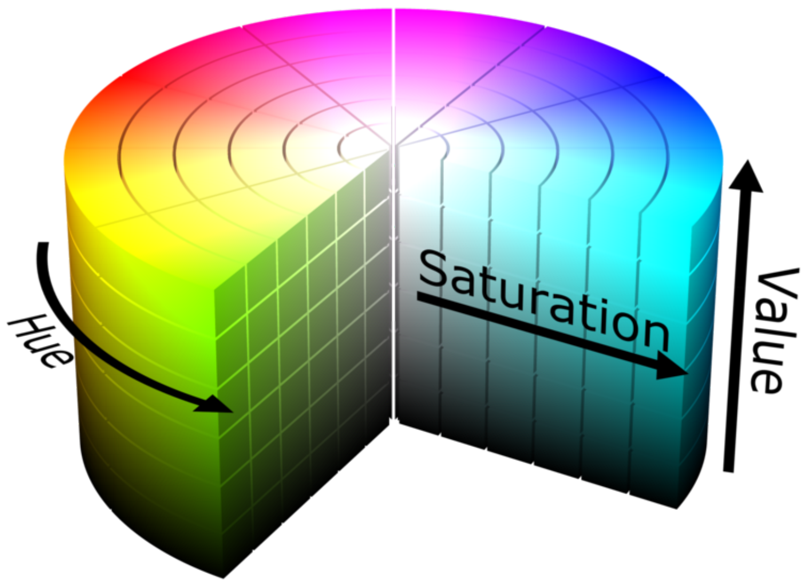


Figure 2.4: A visual representation of the HSV color space. *Hue* is given by an angle and represents the color. *Saturation* describes the amount to which that color is mixed with white. *Value* describes the amount to which that color is mixed with black. Image courtesy of [5].

$$H(\mathbf{C}_{RGB}) = 60^\circ \times \begin{cases} \text{undefined} & \text{if} \quad S(\mathbf{C}_{RGB}) = 0 \\ \frac{G-B}{\max(\mathbf{C}_{RGB}) - \min(\mathbf{C}_{RGB})} & \text{if} \quad R = \max(\mathbf{C}_{RGB}) \\ 2 + \frac{B-R}{\max(\mathbf{C}_{RGB}) - \min(\mathbf{C}_{RGB})} & \text{if} \quad G = \max(\mathbf{C}_{RGB}) \\ 4 + \frac{R-G}{\max(\mathbf{C}_{RGB}) - \min(\mathbf{C}_{RGB})} & \text{if} \quad B = \max(\mathbf{C}_{RGB}) \end{cases}$$

which can be combined into a vector similar to (2.9)

$$\mathbf{C}_{HSV}(u,v) = \begin{bmatrix} H(u,v) & S(u,v) & V(u,v) \end{bmatrix}^T \tag{2.12}$$

The HSV values can be thought of as follows [23]

- **HUE** represents the color.

- **SATURATION** describes the amount to which that color is mixed with *white*. Notice how the "center" of the cylinder in figure 2.4 is white.
- **VALUE** describes the amount to which that color is mixed with *black*. Notice how the bottom of the cylinder in figure 2.4 is black.

2.4. The Camera

In this thesis, a simple camera model is used, called the *pinhole camera model*. This model describes the relationship between a 3D point in the word expressed in metric units [m] and the corresponding 2D point in the image plane expressed in discrete image units [pixel].

Homogeneous Coordinates

First, *homogeneous coordinates* allow for a mapping between 2D euclidean space and 3D euclidean space [27, pp. 154]. Let (y_1, y_2) be a point in $\mathbb{R}^2$ and (x_1, x_2, x_3) be a point in $\mathbb{R}^3$ and g be a constant. The $\mathbb{R}^3 \mapsto \mathbb{R}^2$ map is then described by

$$\begin{bmatrix} y_1 \\ y_2 \\ g \end{bmatrix} = s \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2.13}$$

where s is a scaling factor needed to solve (2.13). By inspection, one obtains $s = g/x_3$. This principle will be used when deriving the camera model.

Extrinsic Parameters

A point observed in the world $\mathbf{P} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ [m] expressed in some arbitrary reference frame is mapped to the corresponding point $\mathbf{P^c} = \begin{bmatrix} X^c & Y^c & Z^c \end{bmatrix}^T$ [m] in the CAM reference frame as (in homogeneous coordinates)

$$\begin{bmatrix} \mathbf{P^c} \\ 1 \end{bmatrix} = \lambda_0 \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P} \\ 1 \end{bmatrix} \tag{2.14}$$

where $\mathbf{R}$ is an arbitrary rotation matrix and $\mathbf{T}$ is a translation vector and $\lambda_0 = 1$.

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} T_{11} \\ T_{21} \\ T_{31} \end{bmatrix}$$

The pair $(\mathbf{R}, \mathbf{T})$ are the extrinsic parameters.

Intrinsic Parameters

Next, the point $\mathbf{P}^c$ from (2.14) is mapped to an image point $\mathbf{p}^i = \begin{bmatrix} x^i & y^i \end{bmatrix}^T$ [m] in the image plane which is located at $Z^c = f$ (see figure 2.5) where f is the focal length of the camera [27, pp. 153] through a projection matrix Π_1 (in homogeneous coordinates)

$$\begin{bmatrix} \mathbf{p}^i \\ 1 \end{bmatrix} = \lambda_1 \Pi_1 \begin{bmatrix} \mathbf{P}^c \\ 1 \end{bmatrix} \tag{2.15}$$

where Π_1 is given by

$$\Pi_1 = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.16}$$

and where λ_1 is a scaling factor needed to solve (2.15) due to the homogeneous coordinates. By inspection, one can see that $\lambda_1 = 1/Z^c$. Multiplying out (2.15) leads to

$$x^i = f\frac{X^c}{Z^c}, \qquad y^i = f\frac{Y^c}{Z^c} \tag{2.17}$$

where (2.17) describes a map from 3D euclidean space $\mathbb{R}^3$ to 2D euclidean space $\mathbb{R}^2$, i.e

$$(X^c, Y^c, Z^c) \mapsto (fX^c/Z^c, fY^c/Z^c)$$

The equation (2.15) is still in metric units [m] so a final transformation is needed to express the image point (2.17) in discrete image coordinates with unit [pixel]. Let the point $\mathbf{p}^p = \begin{bmatrix} u & v \end{bmatrix}^T$

express discrete image coordinates with unit [pixels]. The transformation from the image point $\mathbf{p}^i$ in (2.15) to $\mathbf{p}^p$ is then (in homogeneous coordinates),

$$\begin{bmatrix} \mathbf{p}^p \\ 1 \end{bmatrix} = \lambda_2 \Pi_2 \begin{bmatrix} \mathbf{p}^i \\ 1 \end{bmatrix} \tag{2.18}$$

where $\lambda_2 = 1$ and (adapted from [28, Lesson 20: Intrinsic camera parameters])

$$\Pi_2 = \begin{bmatrix} s_x & s_x \cot(\gamma) & u_0 + v_0 \cot \gamma \\ 0 & s_y / \sin(\gamma) & v_0 \sin(\gamma) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

where s_x and s_y are scaling factors attempting to account for non-square pixels, (u_0, v_0) represent the coordinates of where the optical axis intersects with the image plane, $\cot(\gamma) = 1/\tan(\gamma)$ and γ attempts to account for any possible non-orthogonality between the rows and columns in the image. At this point, the following assumptions are made [28, Lesson 20: Intrinsic camera parameters, Improving Intrinsic Parameters])

- The rows and columns in the image are orthogonal, implying $\gamma = pi/2$.
- The pixels in the image are square, implying $s_c = s_x = s_y$. This assumption are based on the values found in table 5.1.

Applying the above assumptions, (2.19) simplifies to

$$\Pi_2 = \begin{bmatrix} s_c & 0 & u_0 \\ 0 & s_c & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

Next, define the *intrinsic parameter matrix* $\mathbf{K}_{3\times4}$ and $\mathbf{K}_{3\times3}$ based on (2.16) and (2.20) as follows

$$\mathbf{K}_{3\times 4} = [\mathbf{K}_{3\times 3} \quad \mathbf{0}] = \Pi_2\Pi_1 = \begin{bmatrix} s_c f & 0 & u_0 & 0 \\ 0 & s_c f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.21}$$

and

$$\mathbf{K}_{3\times 3} = \begin{bmatrix} s_c f & 0 & u_0 \\ 0 & s_c f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.22}$$

A summary of the intrinsic parameters can be seen in figure (2.5).

World to Image Plane

A complete mapping from an arbitrary point $\mathbf{P}$ in the world to discrete image coordinates $\mathbf{p}^p$ can now be expressed as (in homogeneous coordinates)

$$\begin{aligned} \begin{bmatrix} \mathbf{P}^p \\ 1 \end{bmatrix} &= \lambda \cdot \quad \mathbf{K}_{3\times 4} \quad \cdot \quad \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad \cdot \begin{bmatrix} \mathbf{P} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \lambda \cdot \begin{bmatrix} s_c f & 0 & u_0 & 0 \\ 0 & s_c f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_{11} \\ R_{21} & R_{22} & R_{23} & T_{21} \\ R_{31} & R_{32} & R_{33} & T_{31} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \end{aligned} \tag{2.23}$$

where $\lambda = \lambda_0\lambda_1\lambda_2$ is a scale factor needed to solve the camera model (2.23). The rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{T}$ will be left as arbitrary for now, but will be made concrete later.

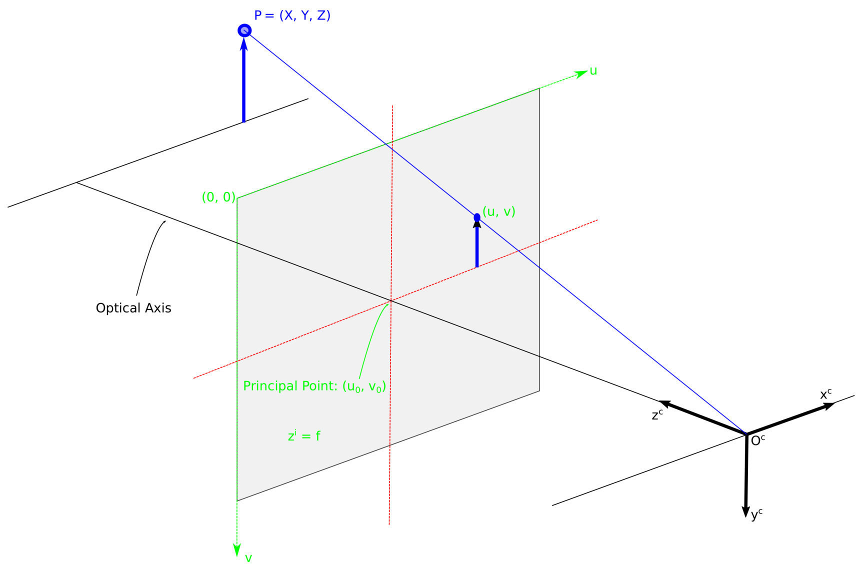


Figure 2.5: The pinhole camera model and its intrinsic parameters. Some point in the world is observed as $\mathbf{P}^c$ in the camera frame. The point (u, v) is the 2D projection of $\mathbf{P}^c$ into the image plane. The offsets (u_0, v_0) are constants to shift the image origin (upper left corner), so that it aligns with the *Optical Axis*. The point where the Optical Axis intersects the image plane is called the *Principal Point* and has coordinates (u_0, v_0). The image plane is located a fixed length $Z^c = f = const$ from the camera frame's origin $\mathbf{O}^c$ where f is the focal length of the camera.

Chapter 3

System Overview

3.1. Assumptions

In practice, the system cannot operate under any given conditions. As a trivial example, if there is a full blown hurricane, the UAV will not be able to land or navigate due to high speed winds. Additionally, the 3D motion of the platform is directly linked to the waves, which in turn is linked to the wind speed. Therefore, the *wind speed* can be a useful metric for placing restrictions on the operational range of the system in terms of weather. A scale that can be used to describe different sea conditions in terms of wind speed, is the *Beaufort Wind Force Scale* [29].

Certain assumptions must also be made about the landing platform itself. These assumptions are mainly centered around limiting the possible area where the platform can be found, relative to a point with known **G**lobal **P**ositioning **S**ystem (GPS) coordinates. Additionally, more specific constraints are placed on its motion in all three directions.

The following assumptions are made about the system

- The UAV will not fly unless the observed sea conditions are in the range [0 - 3] as described by the Beaufort Wind Force scale as presented by [29].
- Let $\Theta^{pn} = \begin{bmatrix} \phi^{pn} & \theta^{pn} & \psi^{pn} \end{bmatrix}^T$ denote the Euler angles between the platform fixed body frame and the inertial NED frame. Within the scope of this thesis, it is then assumed that
$$\Theta^{pn} \approx \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$

meaning that the platforms body frame is approximately aligned with the NED-frame at all times. Note that this also implies

$$\dot{\Theta}^{pn} \approx [0 \quad 0 \quad 0]^T$$

meaning that the angular motion of the platform is assumed to be zero for this thesis.

- The *surge* and sway are negligible, since the platform is anchored and since the UAV will only fly if the observed sea conditions are calm, as described by the Beaufort Wind Force Scale.
- The landing platform is anchored so that it cannot drift away with currents. The anchor itself has known fixed-GPS coordinates

$$\mathbf{P}_{anchor}^{GPS} = [X_{anchor}^{GPS} \quad Y_{anchor}^{GPS}]^T$$

meaning that the platform will be found within a circular area with radius r_{plat}, and center $(X_{anchor}^{GPS}, Y_{anchor}^{GPS})$.

These assumptions may have to be relaxed for the final system, but they are made to simplify the development phase and tighten the scope of this thesis.

3.2. Landing Procedure

Suppose the UAV has preformed some task at sea. The last part of its mission will consist of safely returning to a landing platform from a set of arbitrary GPS coordinates for recharging, maintenance etc. The overall process of returning to the platform can be outlined by the following steps

1. **Search** for the platform at some sensible altitude and get the platform into the cameras FOV.
2. **Track** the platform and use the camera based vision system to position UAV over the landing platform and descend to a lower altitude.
3. **Observe** the platform if necessary and attempt to compensate for *heave*.
4. Ensure a controlled **Touchdown**[1].

Based on the steps outlined above, four distinct modes of operation are defined.

[1]Touchdown is defined as the moment where the UAV makes contact with the landing platform

3.2.1. Search Mode (GPS)

The assumed starting point for this mode is

- The UAV has preformed its designated task and is located at some point expressed in the GPS-frame

$$\mathbf{P}_{UAV}^{GPS} = [X_{uav}^{GPS} \quad Y_{uav}^{GPS} \quad Z_{uav}^{GPS}]^T$$

Then the relative position between the UAV and the anchor is

$$\mathbf{P}_{rel}^{GPS} = \Bigg(\mathbf{P}_{anchor}^{GPS} - \mathbf{P}_{UAV}^{GPS}\Bigg) + \mathbf{E}_{error}^{GPS} \tag{3.1}$$

where

$$\mathbf{E}_{error}^{GPS} = [X_{error}^{GPS} \quad Y_{error}^{GPS} \quad Z_{error}^{GPS}]^T \tag{3.2}$$

is an error term quantifying the uncertainty of the GPS measurements.

Next, an expression for a *search altitude* must be derived. Specifically, an expression for a search altitude must be found such that the UAV will find the platform, even in the worst case scenario, described in figure 3.1.

The proposed general form of this search altitude is

$$Z_{search} = ceil\Big(Z_{ideal} + Z_{error}^{H} + V_{error}\Big) \tag{3.3}$$

where $ceil(x)$ is the ceiling function (ensuring that Z_{search} is always rounded up), Z_{ideal} denotes the search altitude if the GPS measurements were perfect, ,Z_{error}^{H} is a correction factor that attempts to account for how the maximum horizontal GPS error from (3.2) might affect the search height and V_{error} is simply the magnitude of the vertical GPS error.

Consider the possible worst case scenario in figure 3.1, where the platform is in its maximum extended position to the left, and the UAV approaches from the right. The **green** UAV$_1$ has a (hypothetical) perfect GPS, meaning that $\mathbf{P}_{rel}^{GPS} = \mathbf{0}$ in its position shown in figure 3.1. Even if its search altitude is only set to Z_{ideal}, it will find the platform in this scenario.

The **red** UAV has an imperfect GPS where the error is described by (3.2). Then, the error terms are defined as

$$\begin{aligned} H_{error} &= \max_{X,Y}\left(\left|\mathbf{E}^{GPS}_{error}\right|\right) \\ V_{error} &= \left|Z^{GPS}_{error}\right| \end{aligned}$$

In its position shown in figure 3.1, the **red** UAV$_2$ will not find the anchor if its searching at Z_{ideal}, due to the error terms H_{error} and V_{error}.

The **yellow** UAV$_3$ in figure 3.1 has the same imperfect GPS as UAV$_2$, but compensates for this by increasing its search altitude to Z_{search} as shown in (3.3), where Z_{error} is a correction factor proportional to the error term H_{error}.

By applying basic trigonometry to figure 3.1, expressions for Z_{ideal} and Z_{error} in (3.3) can be derived. Let FOV_h and FOV_v denote the cameras FOV in the horizontal and vertical direction respectively. Let r_{anchor} denote biggest possible radius the platform can be found within. The following is then obtained

$$\begin{aligned} Z_{ideal} &= \frac{r_{anchor}}{\tan\left(\min\left(FOV_h, FOV_v\right)/2\right)} \\ Z^{H}_{error} &= \frac{H_{error}}{\tan\left(\min\left(FOV_h, FOV_v\right)/2\right)} \end{aligned}$$

which can be plugged into (3.3) to obtain the proposed expression for the search height

$$\begin{aligned} Z_{search} &= ceil\left(\frac{r_{anchor}}{\tan\left(\min\left(FOV_h, FOV_v\right)/2\right)} + \frac{H_{error}}{\tan\left(\min\left(FOV_h, FOV_v\right)/2\right)} + V_{error}\right) \\ &= ceil\left(\frac{r_{anchor} + H_{error}}{\tan\left(\min\left(FOV_h, FOV_v\right)/2\right)} + V_{error}\right) \end{aligned} \tag{3.4}$$

To get a feel for the magnitude of the search altitude in (3.4), suppose the UAV is fitted with the low cost GPS found in the literary review in [9] which has the following error margins

$$\begin{aligned} X^{GPS}_{error} = Y^{GPS}_{error} = \pm 2m \quad &\Rightarrow \quad H_{error} = |\pm 2m| = 2m \\ Z^{GPS}_{error} = \pm 5m \quad &\Rightarrow \quad V_{error} = |\pm 5m| = 5m \end{aligned}$$

Additionally, assume the following

$$r_{anchor} = 15\text{m}, \qquad \min(FOV_h, FOV_v) = 50°$$

Plugging these values into (3.4), the search altitude is calculated to be

$$Z_{search} = ceil(33.5294\text{m}) = 34\text{m}$$

meaning that the UAV must send $Z_{cmd} = Z_{search} = 34m$ into the existing control hierarchy to be guaranteed to get the landing platform into its FOV, given the GPS error margins.

As a slight caveat, even if the platform is in the camera's FOV, it does no good if the vision-system is unable to lock on to the target on the platform due to it being to far away. Therefore, it is critical that the target is sufficiently large to be detected at Z_{search}, given the image resolution used in the system.

3.2.2. Tracking Mode

When the platform is in the UAV's FOV and the target is detected, the tracking mode will attempt to align the UAV's center with the targets center, and descend to an altitude given by $Z_{track} = \alpha_{track} Z_{search}$, where $0 < \alpha_{track} < 1$ is a scaling factor so that Z_{track} is some percentage of Z_{search}.

Let

$$\mathbf{P}^n_{target} = \begin{bmatrix} X^n_{target} & Y^n_{target} & Z^n_{target} \end{bmatrix}^T$$

denote the position of the center of the target expressed in the NED frame and

$$\mathbf{P}^n_{uav,rel} = \begin{bmatrix} X^n_{uav,rel} & Y^n_{uav,rel} & Z^n_{uav,rel} \end{bmatrix}^T$$

be the UAV's position *relative* to $\mathbf{P}^n_{target}$ expressed in the NED frame. Ideally, the relative position in this mode would be

$$\mathbf{P}^n_{uav,rel} = \begin{bmatrix} 0 & 0 & Z_{track} \end{bmatrix}^T$$

but demanding $X^n_{uav,rel} = Y^n_{uav,rel} = 0$ is unrealistic. One can relax this constraint by putting a threshold on the allowed magnitude of the horizontal relative position as

$$\left(X^n_{rel,x}\right)^2 + \left(Y^n_{rel,x}\right)^2 \leq r^2_{track} \tag{3.5}$$

where r_{track} [m] is a design parameter. The constraint (3.5) demands that the horizontal relative position of the UAV is within a circle with radius r_{track} and center $\left(X^n_{target}, Y^n_{target}\right)$.

In this mode, the platforms *heave* is ignored completely. The altitude in this mode is still computed relative to the MSL, not the platform.

Another important function of this mode is to determine if the platform has been lost. The details for how this criteria is defined will be discussed in chapter 4. In the event that the target is in fact lost, the *target detected* flag should be set to false, and the UAV should return to search mode.

3.2.3. Observation Mode

While the issue of estimating the motion of a floating platform is not addressed in this thesis, this mode is included for completeness. This mode is inspired by the *approach* stage in [30].

As the name suggest, in this mode, the UAV would observe the platform over some pre-defined time interval $t_0 ... t_N$ and attempt to build a model of the platform's vertical position on the form (adapted from [31, eq. (2)])

$$Z^n_{PLAT}(t) = \sum_{i=1}^{N} A_i \sin\left(\frac{2\pi}{T}t + \varphi_i\right) \tag{3.6}$$

where where the amplitudes A_i, frequency $(2\pi/T)$ and phases φ_i are unknown constants, assumed to range over some fixed interval. The details of how this is done is outside the scope of this thesis.

3.2.4. Landing Mode

The main function of the landing mode is to ensure a controlled, vertical descent, and to ensure that the UAV hits the platform. This mode is inspired by the *touchdown* stage from [30].

First, it should impose a desired vertical descent velocity constraint

$$\dot{Z}_{cmd} \leq 0.5\text{m/s}$$

Second, it should attempt to keep the UAV horizontally aligned with the platform , meaning

$$\phi_{cmd} = 0 \qquad \theta_{cmd} = 0$$

i.e the control system should drive pitch and roll towards zero.

Third, it should enforce a constraint similar to (3.5), but it might be necessary to tighten it further to

$$\left(X^n_{rel,x}\right)^2 + \left(Y^n_{rel,x}\right)^2 \leq r^2_{land} \leq r^2_{track} \tag{3.7}$$

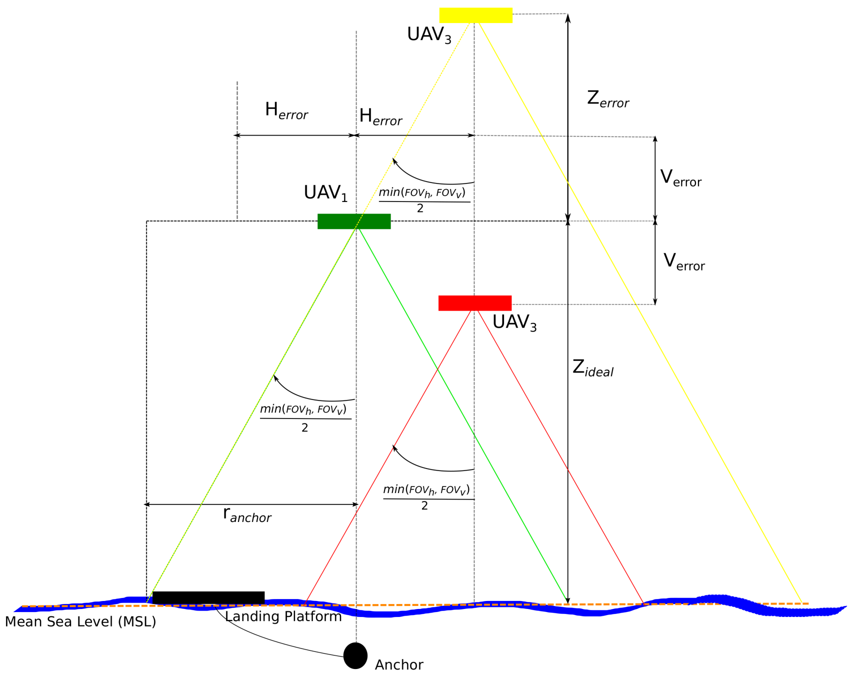


Figure 3.1: Sketch of several scenarios when the UAV is in search mode. Let the anchor be in its maximum extended position, and have the UAV approach from the right. The **green** UAV$_1$ has a perfect GPS, meaning that it is perfectly aligned with the anchor, and will have the platform in its FOV, even if it is searching at Z_{ideal}. The **red** UAV$_2$ has an imperfect gps with H_{error} denoting the maximum error of the GPS in the horizontal plane and V_{error} denoting the maximum error of the GPS in the vertical direction. If the **red** UAV attempts to search at Z_{ideal}, it runs the risk of not finding the platform, even when it is close to the anchor, due to its *actual* altitude being distorted by H_{error} and V_{error}. Finally, the **yellow** UAV$_3$ has the same imperfect GPS as UAV$_3$, but its search altitude is chosen so that it compensates for H_{error} and V_{error} , meaning that it will find the platform, even in the worst case scenario.

Chapter 4

Vision System

This chapter will present a theoretical overview of the vision system. The vision system will be based on the on-board downward-pointing camera. Its main purpose is to calculate the UAV's position relative to the center of a landing target with certain known characteristics/features.

4.1. Image Processing

Based on the characteristics mentioned in section 4.1.1, it is possible to formulate a general idea of what the target will look like. The target will likely be composed of one or more colored (in this case red) blobs[1]. Therefore, the image processing algorithm will be designed to detect these blobs.

4.1.1. Landing Platform Characteristics

The landing platform characteristics can will influence the choice of image processing algorithms and techniques used. Therefore, it is important to establish exactly what those characteristics are.

For the purposes of this thesis, the following three characteristics have been deemed most important:

[1]In the context of computer vision, *blob* refers to a region in a digital image that differ in their properties, for example brightness or color

- **Geometric Marking:** The landing platform will be marked with a simple geometric shape. Shapes with well known properties such as circles or squares are preferable.
- **Physical Size:** The physical size of the geometrical marking must meet two fundamental requirements. Fist, it must be small enough to appear in its entirety in the image when the UAV is close to the platform. Second, it must be large enough to yield robust measurements from higher altitudes.
- **Color:** Since the landing platform will be ocean based, the color of the marking can be exploited.

The chosen target can be seen in figure 4.1. The reasoning for choosing this particular target was threefold. First, red is not a color that is likely to occur naturally at sea. It is also fairly easy to separate it from the approximately uniformly blue background of the ocean in both the RGB and HSV color spaces. Second, circles have well-known geometric properties that can easily be exploited with respect to robust target detection. Third, the nested circle scheme allows the target to easily to be scaled up (simply add more circles).

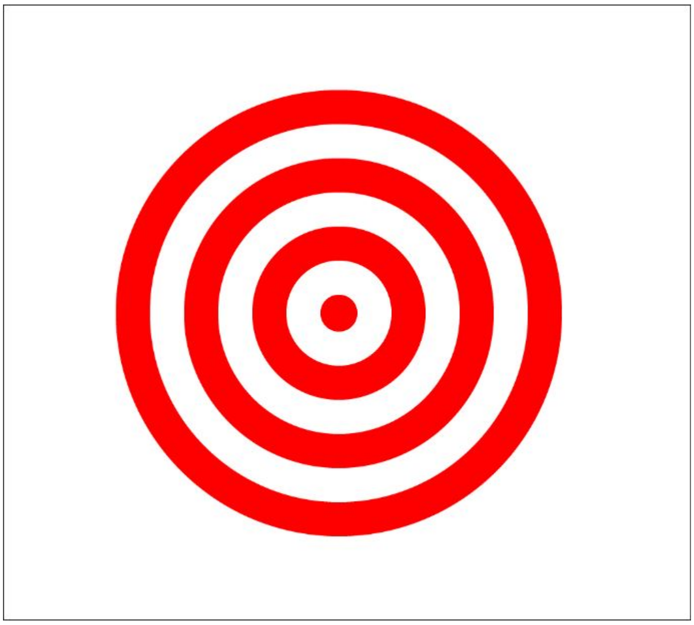


Figure 4.1: The target used in this thesis. Image courtesy of [6].

4.1.2. Search for Red Pixels

Recall from section 2.3.2 that each pixel in an RGB image is essentially a small vector consisting of three values; amount of red, amount of green, amount of blue, returning a color vector $\mathbf{I}_{RGB}(u,v) = \begin{bmatrix} R(u,v) & G(u,v) & B(u,v) \end{bmatrix}^T$. Each of the RGB values is assumed to be in the range $[0, 255]$. Since the target is chosen to be red the pixels in the blob would ideally be perfectly red: $\begin{bmatrix} 255 & 0 & 0 \end{bmatrix}^T$. However, this is obviously not the case in real life, meaning that the detection method must have some tolerance for variations of red.

Additionally, the color space chosen must be able to find red pixels under different light conditions. Both color spaces presented in 2.3.2 will be tested for illumination invariance in chapter 5.

1. Search for red pixels in the captured **R**ed **G**reen **B**lue (RGB) image.
2. Convert the RGB image to a binary image.
3. Extract and filter blob properties.

Each of the three steps mentioned are described in more detail in the next sections.

Normalized RGB

A condition for testing the "redness" of the pixel can be imposed on every color vector in the RGB image. In order detect red pixels, two methods will be proposed. Consider a normalized RGB vector given by (2.11) which will now be expressed as

$$\overline{\mathbf{C}}_{RGB}(u,v) = \begin{bmatrix} \overline{R}(u,v) & \overline{G}(u,v) & \overline{B}(u,v) \end{bmatrix}^T$$

where (u,v) [pixel] are discrete image coordinates in the image plane. Let $\overline{\mathbf{I}}_{RGB}$ be an $(M \times N \times 3)$ normalized RGB image and T_1 and T_2 be numerical threshold values. For the purposes of this thesis, the following conditions expressed as a function of a normalized RGB color vector will be tested:

$$C_1\left(\overline{\mathbf{C}}_{RGB}, T_1, T_2\right) = \overline{R} > T_1 \cdot \overline{G}) \quad \textbf{and} \quad (\overline{R} > T_1 \cdot \overline{B}\Big] \tag{4.1}$$

$$C_2\left(\overline{\mathbf{C}}_{RGB}, T_1, T_2\right) = \left[\overline{R} > T_2 \cdot (\overline{G} + \overline{B})\right] \tag{4.2}$$

The $(M \times N \times 1)$ binary image $\mathbf{I}_{bin}$ is constructed by applying 4.1 and 4.2 to every color vector $\overline{\mathbf{C}}_{RGB}(u,v) \in \bar{\mathbf{I}}_{RGB}$, where, assuming appropriate values for T_1 and T_1, $\mathbf{I}_{bin}$ will have the red pixels marked as 1's.

HSV

The approach to detecting red pixels in the HSV image is slightly different than for the normalized RGB image. While the RGB threshold considers all three RGB values in a single condition, the HSV threshold will work on each layer in the HSV image separately, before combining the results. First, recall the expression for an HSV color vector given by (2.12). Instead of expressing a color vector, let $\mathbf{I}_{HSV}$ denote a $M \times N \times 3$ HSV image where $\mathbf{H}, \mathbf{S}$ and $\mathbf{V}$ are the three $(M \times N \times 1)$ layers in $\mathbf{I}_{HSV}$. Next, let $\mathbf{T}_H, \mathbf{T}_S$ and $\mathbf{T}_V$ be (1×2) threshold vectors containing numerical threshold values as follows

$$\mathbf{T}_H = [T_H^{lower} \quad T_H^{upper}], \qquad \mathbf{T}_S = [T_S^{lower} \quad T_S^{upper}] \qquad \mathbf{T}_V = [T_V^{lower} \quad T_V^{upper}]$$

Next, three $(M \times N \times 1)$ masks are created by evaluating

$$\mathbf{H}_{mask} = \left[\Big((H(u,v) \le T_H^{lower}) \quad \textbf{or} \quad (H(u,v) \ge T_H^{upper}) \Big) \quad \forall \quad H(u,v) \in \mathbf{H} \right]$$

$$\mathbf{S}_{mask} = \left[\Big((S(u,v) \ge T_S^{lower}) \quad \textbf{and} \quad (S(u,v) \le T_S^{upper}) \Big) \quad \forall \quad S(u,v) \in \mathbf{S} \right]$$

$$\mathbf{V}_{mask} = \left[\Big((V(u,v) \ge T_V^{lower}) \quad \textbf{and} \quad (V(u,v) \le T_V^{upper}) \Big) \quad \forall \quad V(u,v) \in \mathbf{V} \right]$$

Then the three masks are combined as follows to create a $(M \times N \times 1)$ binary image

$$\mathbf{I}_{HSV}^{bin} = \left[\mathbf{H}_{mask} \quad \textbf{and} \quad \mathbf{S}_{mask} \quad \textbf{and} \quad \mathbf{V}_{mask} \right]$$

where, assuming appropriate values for $\mathbf{T}_H, \mathbf{T}_S$ and $\mathbf{T}_V$, $\mathbf{I}_{bin}$ will have the red pixels marked as 1's.

4.1.3. Target Detection

Once the red pixels have been found in an image, it is necessary to determine which blobs qualifies as the target. It is unrealistic to expect the threshold to perfectly only extract the red nested circles, so the underlying assumption here is that there will always be some visual noise in the image and a robust method for filtering out this noise must be found.

Contour Filter

The first step after the algorithm preforms after the thresholding, is to trace the contours $\left(\mathbf{Contours}_{blobs}\right)$ of the binary blobs in the image. In practice, $\mathbf{Contours}_{blobs}$ is a set of vectors where each vector contains all the pixel coordinates (u, v) of the contour of the blob they correspond to.

Let $\mathbf{C}_{blob}$ be the vector containing all the pixel coordinates of the contour of a single blob i.e $\mathbf{C}_{blob} \in \mathbf{Contours}_{blobs}$. The circumference of the blob c_{blob} can then be approximated by counting the number of pixels in $\mathbf{C}_{blob}$ as

$$c_{blob} = length\left(\mathbf{C}_{blob}\right) \tag{4.3}$$

The first constraint placed on the contours is that the circumference of any given blob must pass

$$c_{blob} > T_{c_{blob}} \tag{4.4}$$

where $T_{c_{blob}}$ is a numerical value. Next, the diameter can be approximated by finding the horizontal max and min values of the blob

$$\begin{aligned} u_{min} &= \min_u \left(\mathbf{C}_{blob}\right), \qquad & u_{max} &= \max_u \left(\mathbf{C}_{blob}\right) \\ v_{min} &= \min_v \left(\mathbf{C}_{blob}\right), \qquad & v_{max} &= \max_v \left(\mathbf{C}_{blob}\right) \end{aligned}$$

The diameter of the blob can then be approximated by

$$\begin{aligned} d^u_{blob} &= \frac{(u_{max} - u_{min})}{2} \\ d^v_{blob} &= \frac{(v_{max} - v_{min})}{2} \\ d_{blob} &= \frac{d^u_{blob} + d^v_{blob}}{2} \end{aligned} \tag{4.5}$$

The reason for averaging over both the u and v direction in (4.5) is that a circle, even as a binary blob, should have approximately the same diameter in both directions. The area of the blob can then be calculated as

$$A_{blob} = \pi \left(\frac{d_{blob}}{2} \right)^2 \tag{4.6}$$

Since the algorithm looks for circles, there should be a approximately constant ratio between (4.3), (4.5) and (4.6) if the blob is circular. In the ideal case (perfect circle) the following would be true

$$\frac{c_{ideal}}{d_{ideal}} = \pi, \qquad \frac{A_{ideal}}{c^2_{ideal}} = \frac{1}{4\pi}$$

which can be used to set sensible threshold values for the non-ideal blob case

$$k_1 + \pi \leq \frac{c_{blob}}{d_{blob}} \leq k_2 + \pi \tag{4.7}$$

$$k_3 + \frac{1}{4\pi} \leq \frac{A_{blob}}{c^2_{blob}} \leq k_4 + \frac{1}{4\pi} \tag{4.8}$$

where k_1, k_2, k_3 and k_4 are small correction factors used to relax the constraints in (4.7) and (4.8).

Additionally, a constraint is placed directly on the area calculated in (4.6). Let A_{image} be the total area of the entire image. Then, the area of any given blob is bounded by

$$\alpha_{min} A_{image} \leq A_{blob} \leq \alpha_{max} A_{image} \tag{4.9}$$

where $0 < \alpha_{min} < \alpha_{max} < 1$. The conditions in (4.4), (4.7), (4.8) and (4.9) will be tested in chapter 5.

4.2. Position Estimation

Suppose the camera has the target in its FOV. The goal is to estimate the translation between the UAV and the center of the target. At this point, it is assumed that the contour filter has been applied, and that all contours detected is part of the target.

4.2.1. Image Plane to CAM Frame

The first step in the translation estimation is to take the feature points found in the image with unit [pixel], and map them to the position in the CAM frame. By plugging (2.18) into (2.15) one obtains

$$\begin{bmatrix} \mathbf{p}^p \\ 1 \end{bmatrix} = \lambda_2 \lambda_1 \Pi_1 \Pi_2 \begin{bmatrix} \mathbf{P}^c \\ 1 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \mathbf{p}^p \\ 1 \end{bmatrix} = \lambda_2 \lambda_1 \mathbf{K}_{3\times 4} \begin{bmatrix} \mathbf{P}^c \\ 1 \end{bmatrix}$$

in its expanded form

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda_2 \lambda_1 \begin{bmatrix} s_c f & 0 & u_0 & 0 \\ 0 & s_c f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X^c \\ Y^c \\ Z^c \\ 1 \end{bmatrix} \tag{4.10}$$

By inspection, $\lambda_2 \lambda_1 = 1/Z^c$. Multiplying out one obtains

$$u = s_c f \frac{X^c}{Z^c} + u_0 \tag{4.11}$$

$$v = s_c f \frac{Y^c}{Z^c} + v_0 \tag{4.12}$$

A key assumption for the next part of the derivation is that **all** points on the target is approximately the same distance from the camera frame origin $\mathbf{O}^c$, i.e Z^c is the same for all points. Now, let (u_1, v) and (u_2, v) be two discrete points in the image plane on the same horizontal line. Let (X_1^c, Y^c) and (X_2^c, Y^c) be the corresponding points in the CAM frame. Consider the distance between (u_1, v) and (u_2, v) in the u direction given by

$$\begin{aligned} u_2 - u_1 &= \left(s_c f \frac{X_2^c}{Z^c} + u_0 \right) - \left(s_c f \frac{X_1^c}{Z^c} + u_0 \right) \\ u_2 - u_1 &= \frac{s_c f}{Z^c} (X_2^c - X_1^c) \\ Z^c &= s_c f \frac{\left(X_2^c - X_1^c \right)}{(u_2 - u_1)} \end{aligned} \tag{4.13}$$

By observing that $Z^c > 0$ (recall the CAM frame in figure 2.2) and $s_c f > 0$, the absolute value can be applied to both sides of (4.13) to obtain

$$Z^c = s_c f \frac{\left| X_2^c - X_1^c \right|}{\left| u_2 - u_1 \right|} \tag{4.14}$$

where $\left| X_2^c - X_1^c \right|$ [m] is a known real-world distance and $\left| u_2 - u_1 \right|$ [pixel] is the corresponding discrete distance in the image plane. The derivation used to arrive at (4.14) holds for a set of points on a vertical line in the v direction (u, v_1), (u, v_2), (X^c, Y_1^c) and (X^c, Y_2^c) where one obtains

$$Z^c = s_c f \frac{\left| Y_2^c - Y_1^c \right|}{\left| v_2 - v_1 \right|} \tag{4.15}$$

Instead of considering only two points, consider generalized versions of (4.14) and (4.15). Suppose the contour filter in section 4.1.3 has been applied and the contours of the nested circles is figure 4.1 has been found. Let

$$u_i^{min} = \min_u \left(\mathbf{C}_i^{blob} \right), \qquad u_i^{max} = \max_u \left(\mathbf{C}_i^{blob} \right) \tag{4.16}$$

$$v_i^{min} = \min_v \left(\mathbf{C}_i^{blob} \right), \qquad v_i^{max} = \max_v \left(\mathbf{C}_i^{blob} \right) \tag{4.17}$$

where $\left(u_i^{min}, v\right)$, $\left(u_i^{max}, v\right)$, is the minimum and maximum pixel coordinates of the i'th contour in the u direction, and $\left(u, v_i^{min}\right)$, $\left(u, v_i^{max}\right)$ is the is the minimum and maximum pixel coordinates of the i'th contour in the v direction as shown in figure 4.2. Next, the geometric simplicity of the target can be exploited. There is a corresponding real world distance d_{real} to

$$d_1^{u,min} = |u_1^{min} - u_2^{min}|$$

which can be obtained by simply measuring the corresponding real world distance on the target between the points u_1^{min} and u_2^{min}. In fact, due to the symmetry of the target, this real world distance is the same for all discrete distances on the form

$$d_i^{u,min} = |u_i^{min} - u_{i+1}^{min}|, \qquad d_i^{u,max} = |u_i^{max} - u_{i+1}^{max}| \tag{4.18}$$

$$d_i^{v,min} = |v_i^{min} - v_{i+1}^{min}|, \qquad d_i^{v,max} = |v_i^{max} - v_{i+1}^{max}| \tag{4.19}$$

which can be seen from figure 4.2.

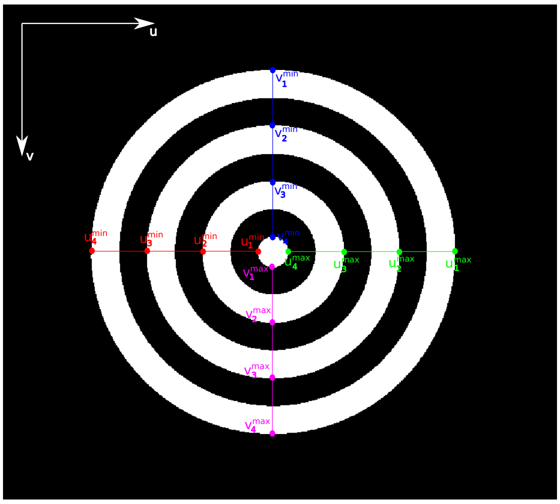


Figure 4.2: Illustration of what feature points the algorithm looks for. For each distance on the form (4.18) or (4.19), there is a corresponding real world distance d_{real}, which is obtained by simply measuring it on the physical target.

Now, assuming a target similar to figure 4.1 is used, a proposed generalization for (4.14) and (4.15) is

$$Z_u^c = \frac{s_c f d_{real}}{2(N-1)} \sum_{i=1}^{N-1} \left(\frac{1}{\left|u_i^{min} - u_{i+1}^{min}\right|} + \frac{1}{\left|u_i^{max} - u_{i+1}^{max}\right|} \right) \tag{4.20}$$

$$Z_v^c = \frac{s_c f d_{real}}{2(N-1)} \sum_{i=1}^{N-1} \left(\frac{1}{\left|v_i^{min} - v_{i+1}^{min}\right|} + \frac{1}{\left|v_i^{max} - v_{i+1}^{max}\right|} \right) \tag{4.21}$$

where $2N$ is the number of points found in each direction (N *max* points plus N *min* points), and d_{real} has simply been factored out.

Next, Z^c can be expressed as the average of (4.20) and (4.21)

$$Z_c = \frac{Z_u^c + Z_v^c}{2} \tag{4.22}$$

The reasoning behind using (4.22) instead of just (4.14) or (4.15) is that the circles detected might be slightly deformed, potentially causing inaccuracies if a single distance pair $\left|X_2^c - X_1^c\right|, \left|u_2 - u_1\right|$ is used.

Next, the *target center* and the *optical center* should be calculated, as it is required to find X^c and Y^c. The target center (u_{target}, v_{target}) is calculated as

$$u_{target} = \frac{1}{2N} \sum_{i=1}^{N} \left(u_i^{max} + u_i^{min} \right) \tag{4.23}$$

$$v_{target} = \frac{1}{2N} \sum_{i=1}^{N} \left(v_i^{max} + v_i^{min} \right) \tag{4.24}$$

and the optical center (u_0, v_0) is calculated as

$$u_0 = \frac{U}{2}, \qquad v_0 = \frac{V}{2} \tag{4.25}$$

where U [pixel] is the image width and V [pixel] is the image height.

Once Z^c, (u_{target}, v_{target}) and (u_0, v_0) is known, X^c and Y^c can be found by re-arranging (4.11) and (4.12) into

$$X^c = Z^c \frac{(u_{target} - u_0)}{s_c f} \tag{4.26}$$

$$Y^c = Z^c \frac{(v_{target} - v_0)}{s_c f} \tag{4.27}$$

which means that $\mathbf{P}^c$ can be found by calculating (4.22), (4.26) and (4.27), using information extracted from the image plane and a set of known distances in the real world.

4.2.2. CAM Frame to NED Frame

Recall the full map from a arbitrary world point $\mathbf{P}$ to a discrete image point $\mathbf{p}^p$ in (2.23). Consider its re-written version

$$\begin{bmatrix} \mathbf{P}^p \\ 1 \end{bmatrix} = \lambda \mathbf{K}_{3\times3}\Big(\mathbf{RP} + \mathbf{T}\Big) \quad \Rightarrow \quad \frac{1}{\lambda}\mathbf{K}_{3\times3}^{-1}\begin{bmatrix} \mathbf{P}^p \\ 1 \end{bmatrix} = \Big(\mathbf{RP} + \mathbf{T}\Big) \tag{4.28}$$

Since $\mathbf{P}$ is in an arbitrary reference frame $\mathbf{P} = \mathbf{P}^n$ be a point in the NED frame. Next, observe that

$$\mathbf{P}^c = \frac{1}{\lambda}\mathbf{K}_{3\times3}^{-1}\begin{bmatrix} \mathbf{p}^p \\ 1 \end{bmatrix}$$

Also, note that the cameras position in the world is given by [32, p.11]

$$\mathbf{C} = -\mathbf{R}^T\mathbf{T} \tag{4.29}$$

where $\mathbf{R}, \mathbf{T}$ are the extrinsic parameters. Now, (4.28) becomes

$$
\begin{aligned}
&\Rightarrow \quad \mathbf{P}^c = \mathbf{R}_n^c \mathbf{P}^n + \mathbf{T}^c \\
&\Rightarrow \quad \mathbf{T}^c = \mathbf{P}^c - \mathbf{R}_n^c \mathbf{P}^n \\
&\Rightarrow \quad \left(\mathbf{R}_n^c\right)^T \mathbf{T}^c = \left(\mathbf{R}_n^c\right)^T \mathbf{P}^c - \left(\mathbf{R}_n^c\right)^T \mathbf{R}_n^c \mathbf{P}^n \\
&\Rightarrow \quad -\left(\mathbf{R}_n^c\right)^T \mathbf{T}^c = -\left(\mathbf{R}_n^c\right)^T \mathbf{P}^c + \left(\mathbf{R}_n^c\right)^T \mathbf{R}_n^c \mathbf{P}^n \\
\text{applying (4.29)} &\Rightarrow \quad \mathbf{P}_{cam}^n = -\left(\mathbf{R}_n^c\right)^T \mathbf{T}^c = \mathbf{P}_{target}^n - \mathbf{R}_c^n \mathbf{P}^c \\
&\Rightarrow \quad \mathbf{P}_{cam}^n - \mathbf{P}^n = -\mathbf{R}_c^n \mathbf{P}^c \qquad (4.30)
\end{aligned}
$$

Now, recall that in section 2.1, it was established that $\mathbf{O}^c = \mathbf{O}^b$, meaning that $\mathbf{P}_{cam}^n$ is effectively the UAV's position in the NED frame. Therefore, for clarity, let $\mathbf{P}_{uav}^n = \mathbf{P}_{cam}^n$. Then, (4.30) becomes

$$\mathbf{P}_{uav}^n - \mathbf{P}^n = -\mathbf{R}_c^n \mathbf{P}^c$$

The goal is to obtain the UAV's position relative to the target. Let $\mathbf{P}^n = \mathbf{P}_{taget}^n$, where $\mathbf{P}_{taget}^n$ is the center of the target in the NED frame . Additionally, the rotation $\mathbf{R}_c^n$ is decomposed into $\mathbf{R}_c^n = \mathbf{R}_b^n \mathbf{R}_c^b$. Then, the UAV's *relative* position to the target is

$$\mathbf{P}_{uav,rel}^n = \mathbf{P}_{uav}^n - \mathbf{P}_{target}^n = -\mathbf{R}_b^n \left(\mathbf{R}_c^b \mathbf{P}^c\right) \qquad (4.31)$$

where $\mathbf{P}_{target}^n = \mathbf{O}^n$ based on the NED definition in section 2.1, $\mathbf{R}_b^n$ can be computed using (2.6) and the on-board IMU data, $\mathbf{P}^c$ can be obtained based on the equations derived in section 4.2.1, and $\mathbf{R}_c^b$ is the transpose of (2.8).

4.3. Algorithm Overview

This section is an attempt to essentially summarize the entire chapter. The steps of the envisioned algorithm is given below

1. Capture RGB image.

2. Search for red pixels using one of the methods in 4.1.2.
3. Extract contours from the binary image.
4. Apply the contour filter from section 4.1.3.
5. **If** the number of contours found **after** the contour filter is greater than 2 (the algorithm needs at least 2 nested circles from the target), go to the next step. **Otherwise**, go to *search mode* (section 3.2.1.)
6. Extract feature points. See figure 4.2 for a visual representation of the feature points.
7. Using the extracted feature points, compute $\mathbf{P}^c$ according to (4.22), (4.26) and (4.27).
8. Using IMU-data, and the known rotation from the CAM frame to the BODY frame, compute the *relative position* according to (4.31).

Chapter 5

Experiments

5.1. Hardware

For the experiments in this chapter, the Raspberry Pi Camera Module v2 (see table 5.1) and the Raspberry pi Model 3B was used, both of which can be seen in figure 5.15a. The reason for choosing this particular combination of hardware was mainly

- **Ease of use:** The author was already familiar with the raspberry pi, meaning that it did not require substantial effort to become familiarized with the hardware.
- **Instant availability:** NTNU has a student-driven electronics shop called *Omega Verksted* which sells multiple raspberry pi products. This means that this particular combination of hardware was instantly available.
- **Online documentation and support**: Raspberry pi offers detailed online documentation and a large online community with for additional support.

The author fully acknowledges that there exists other (potentially better) combinations of hardware for the purpose of an autonomous vision-based landing system, but the raspberry pi was deemed sufficient for the scope of this thesis.

5.2. Software

	Camera Module v2
Price	$25
Weight	3g
Still resolution	8 Megapixels
Sensor image area	3.68mm $\times$ 2.76mm
Pixel size	1.12 μm $\times$ 1.12 μm
Horizontal FOV	62.2 °
Vertical FOV	48.2 °

Table 5.1: Key specifications for the raspberry pi camera used in this thesis. All values are taken from [7], where the full table can be found.

Matlab

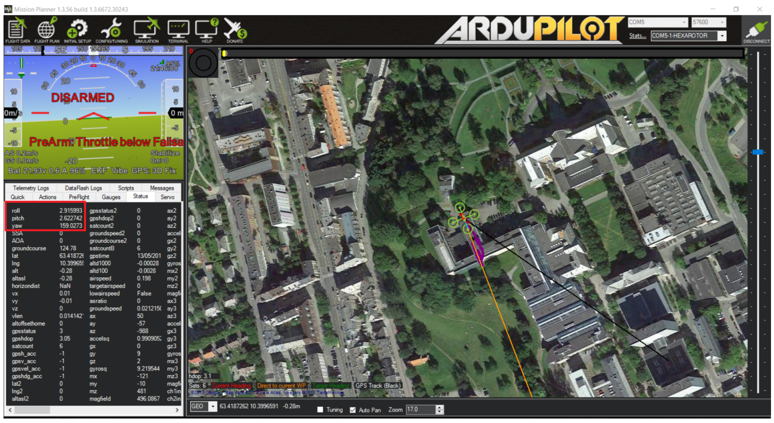
Matlab was used for the prototype implementation of the vision system described in chapter 4. It was chosen primarily for its visualization capability, ease of use (the author is very familiar with Matlab), a wide section of built in and relevant functions, and a large online community offering additional support.

Mission Planner

In order to read relevant data from the on-board UAV systems, the Ardu Pilot *Mission Planner* (see figure 5.1) was used. It communicates with the UAV using the Micro Air Vehicle Link [1] (**MAVLink**) protocol.

Despite having access to the on-board GPS when using the Mission Planner in conjunction with the UAV, it was deemed unwise to attempt to use it for position ground truth measurements, as the signal quality suffered indoors.

[1]A communications protocol specifically designed for communication with small aerial vehicles.



```
roll           2.915993   gpsstatus2       0          ax2
pitch          2.622742   gpshdop2         0          ay2
yaw            159.0273   satcount2        0          az2
SSA            0          groundspeed2     0          accel
AOA            0          groundcourse2    0          gx2
groundcourse   124.78     satcountB        6          gy2
lat            63.41872   gpstime          13/05/201  gz2
lng            10.39965   altd1000         -0.00028   gyros
alt            -0.28      altd100          -0.0028    mx2
altasl         -0.28      airspeed         0.198      my2
horizondist    NaN        targetairspeed   0          mz2
vx             0.01       lowairspeed      False      magfi
vy             -0.01      asratio          0          ax3
vz             0          groundspeed      0.021215   ay3
vlen           0.014142   ax               50         az3
altoffsethome  0          ay               -57        accel
gpsstatus      3          az               -988       gx3
gpshdop        3.05       accelsq          0.990905   gy3
satcount       6          gx               0          gz3
gpsh_acc       -1         gy               9          gyros
gpsv_acc       -1         gz               2          mx3
gpsvel_acc     -1         gyrosq           9.219544   my3
gpshdg_acc     -1         mx               -121       mz3
lat2           0          my               -10        magfi
lng2           0          mz               481        ch1in
altasl2        0          magfield         496.0867   ch2in
```

Figure 5.1: The Mission Planner interface. The "Status" page shown on the right allows quick and easy access to some of the most mission critical data. The UAV's orientation, roll, pitch and yaw, are highlighted by the red bounding box.

5.3. Error Metrics

Two common error metrics are presented here. Let p_{GT} [m] denote the *ground truth* and p_{EST} [m] be the corresponding estimate of p_{GT}. Then

$$e_{abs}^{m} = \left| |p_{GT}| - |p_{EST}| \right| \tag{5.1}$$

$$e_{abs}^{\%} = 100\% \cdot \frac{e_{abs}^{m}}{|p_{GT}|} \tag{5.2}$$

The reason for the nested absolute value signs in (5.1) is that some ground truth values and estimates can be negative. This is also the reason for applying the absolute value sign to p_{GT} in (5.2).

5.4. Robustness

The robustness of the visual system in this thesis is mainly concerned with the following:

1. It must be able to correctly identify the target under varying light-conditions.

2. It must be able to correctly identify the target with visual noise present in the binary image.

Put simply, the UAV should be able to land if its sunny, cloudy, etc., and the vision system should be able to handle a red piece of plastic that happens to drift into view.

This section presents some preliminary tests that attempts to achieve these goals.

5.4.1. Illumination Invariance

Goal

The goal of this section is to test the robustness of the two color spaces presented in section 2.3.2 with respect to illumination invariance.

More specifically, the goal is to find threshold values for the thresholding methods proposed in section 4.1.2, meaning, T_1, T_2 for **Normalized RGB** and $\mathbf{T}_H, \mathbf{T}_S, \mathbf{T}_V$ for **HSV**. This is important, as adverse light conditions could potentially cause the binary target to become deformed, as shown in figure 5.2.

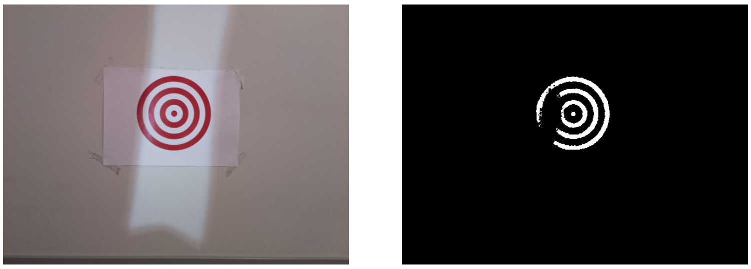
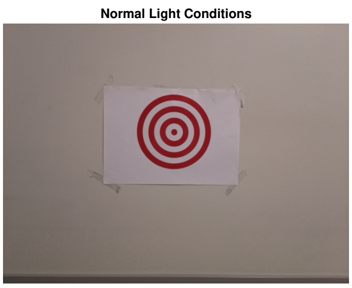

(a) Original RGB image. (b) Binary output.

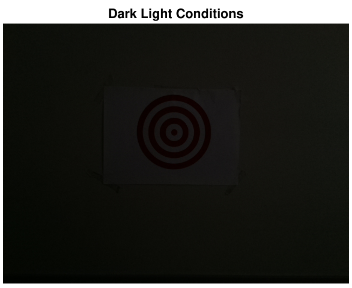
Figure 5.2: Example of how light conditions can deform the binary output of the thresholding. Even though the target is the only red object and the target is not physically obstructed as shown in (a), the light conditions can still cause the thresholding to produce a deformed output shown in (b).

Setup

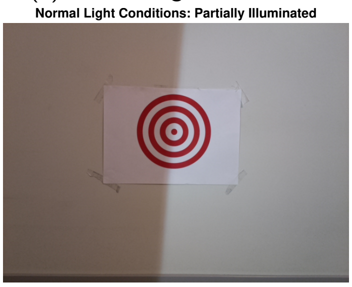
Images was taken of target the target shown in figure 4.1 at a fixed distance, under varying light conditions. A sample of these light conditions can be seen in figure 5.3. There were 12 images in total in this test set.



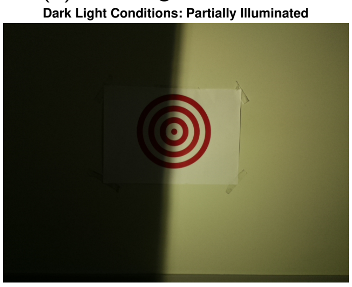
(a) Normal Light conditions.



(b) Dark Light conditions.



(c) Normal Light conditions: Target partially illuminated.



(d) Dark Light conditions: Target partially illuminated.

Figure 5.3: A sample of the data-set used to test the color-spaces in section 2.3.2 for illumination invariance.

Error Metric

The error metric used here is conceptually similar to (5.2), but is stated explicitly here for clarity. A practical way of comparing different threshold value is needed, as inspecting 12 binary outputs for each set of threshold values can quickly become ineffective. Therefore, a simple error metric was used. First, as a reference point, the light conditions seen in figure 5.3a was defined as *normal light conditions*, which will be used as a reference when testing the thresholding values under varying light conditions. With that in mind, the error metric was defined as follows:

1. Given one of the two methods described in section 4.1.2 and a set of corresponding thresholding values, threshold image (5.3a) and visually inspect the output to ensure that the reference is not deformed.

2. Place a bounding box around the target as shown in figure 5.4.

3. Count the number of 1's inside the bounding box and call this number *Number of red pixels reference*, N_{ref}^{pixel}.

Next, a method can then be tested on a set of images as follows

1. Given one of the two methods described in section 4.1.2 and a set of corresponding thresholding values, threshold the current test image.
2. Place a bounding box around the target as shown in figure 5.4.
3. Count the number of 1's inside the bounding box and call this number *Number of red pixels found* N_{found}^{pixel}.
4. Compute the *Absolute Error* metric as shown in (5.3).
5. Repeat steps 1-4 until all images has been tested.

$$e_{abs}^{\%} = 100\% \cdot \left| 1 - \left(N_{found}^{pixel} \middle/ N_{ref}^{pixel} \right) \right| \tag{5.3}$$

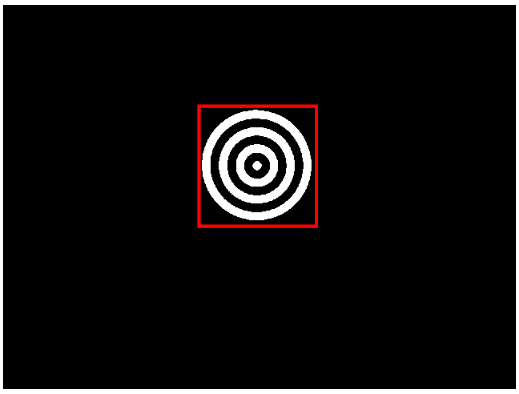


Figure 5.4: Bounding box used to define *Number of pixels reference* in (5.3). The binary output shown is based on image (5.3a).

The main idea underpinning the metric in (5.3), is that the number of pixels inside the bounding box in figure 5.4 should stay somewhat constant, regardless of light conditions if the thresholding values are good with respect to illumination invariance. For example, the deformed output in image (5.2b) would have an absolute error of roughly 35% due to a large number of "missing" pixels.

Results: HSV

Seven different sets of thresholding values for the HSV-based method proposed in section 4.1.3 are presented in table 5.2. It proved quite difficult to find a set of HSV thresholding values which yielded a good response with respect to the metric in (5.3), as shown in figure 5.5. The general approach was to let $\mathbf{T}_S$ and $\mathbf{T}_V$ have a fairly large range to allow for tolerance of different shades of red (i.e red mixed with different amounts of white and black).

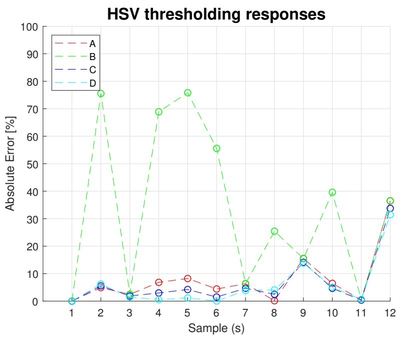
Threshold Value Sets (HSV)	$\mathbf{T}_H$	$\mathbf{T}_S$	$\mathbf{T}_V$
A	$[0.05\,0.97]$	$[0.3\,1]$	$[0.01\,1]$
B	$[0.01\,0.97]$	$[0.3\,1]$	$[0.01\,1]$
C	$[0.05\,0.97]$	$[0.4\,1]$	$[0.02\,1]$
D	$[0.05\,0.97]$	$[0.5\,1]$	$[0.03\,1]$
E	$[0.01\,0.50]$	$[0.6\,1]$	$[0.04\,1]$
F	$[0.00\,0.97]$	$[0.3\,1]$	$[0.01\,1]$
G	$[0.03\,0.97]$	$[0.3\,1]$	$[0.01\,1]$

Table 5.2: Threshold value sets used to generate the output in figure 5.5.

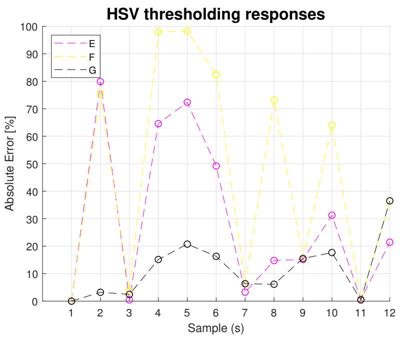
The responses from sets **B** (seen in figure 5.5a), **E** and **F** (seen in figure 5.5b) were visually inspected due to the extremely high error. All of them produced severely deformed targets. Looking at their corresponding values in table 5.2, it was difficult to establish a clear pattern answer as to *why* they preformed so poorly, but they were scrapped based on their poor responses.

The responses of sets **A**, **C**, **D** (seen in figure 5.5a) and **G** (seen in figure 5.5b) , preform reasonable well compared with **B**, **E** and **F** for samples $s \in [1, 11]$, but all have a significant error spike for sample $s = 12$. This could imply that **A**, **C**, **D** and **G** might produce severely deformed targets when exposed to certain light conditions.

Based on these tests, all the tested HSV threshold values are considered non-robust, at least within the scope of these tests.



(a) HSV thresholding responses for thresholding value sets A-D



(b) HSV thresholding responses for thresholding value sets E-G

Figure 5.5: Absolute error responses for different threshold value sets for the HSV method. The results are partitioned into 2 plots for readability.

Results: Normalized RGB

Seven different sets of thresholding values for the normalized RGB method proposed in section 4.1.3 are presented in table 5.3. Compared to the HSV method, it proved much easier to find a set of RGB thresholding values which yielded a decent response with respect to the metric in (5.3), as shown in figure 5.6. A possible explanation might be that the RGB method only requires tuning of 2 parameters, whereas the HSV method requires tuning of 6. Recall the conditions (4.1) and (4.2). The first parameter to be considered is T_1 in (4.1). *Increasing* it will lessen the tolerance for the amount of *green* **and** *blue* present, and conversely, decreasing it will *increase* the tolerance for the the amount of *green* **and** *blue* present. Finding the balance between the two is key. The second parameter to be considered is T_2 in (4.2). If T_2 is *increased* it will lessen the tolerance for the amount of *green* **or** *blue* present, and conversely, if it is *decreased* it will increase the tolerance for the amount of *green* **or** *blue* present. The other parameter T_1 works slightly differently, increasing it will lessen the tolerance for the amount of *green* **and** *blue* present, and conversely, decreasing it will *increase* the tolerance for the the amount of *green* **and** *blue* present. Finding the balance between the two is key.

By inspecting figure 5.6a, it is clear that the values for set **A** can immediately be discarded., due to a massive $\approx 45\%$ error spike for $s = 2$. This is likely due to the relatively low value of T_2, which is why T_2 is higher in table 5.3 for all other thresholding value sets. The responses from sets **B** (seen in figure 5.6a) and **E** (seen in figure 5.6a) have, compared to sets **C**, **D**, **F** and **G** a higher error in $s = 2$ and

Threshold Value Sets (RGB)	T_1	T_2
A	1.165	0.51
B	1.165	0.61
C	1.165	0.71
D	1.165	0.81
E	1.2	0.61
F	1.25	0.61
G	1.3	0.71

Table 5.3: Threshold value sets used to generate the output in figure 5.6.

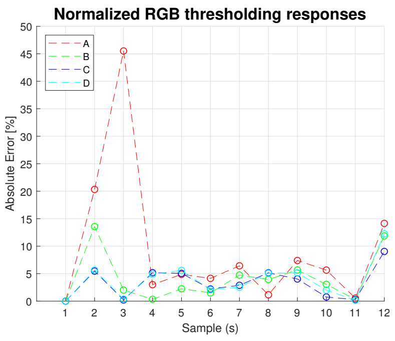
$s = 12$. Based on this, the general trend was that increasing T_1 and T_2 produced error responses closer to 0%, which is why T_1 and T_2 increases for sets **C**, **D**, **F** and **G** in table 5.3.

The sets **C**, **D** and G were the final candidates. Both **C** and **D** had a higher average error than **G**, as did all of the values tested for the HSV method. Therefore, the final choice, based on figure 5.6 and 5.5, was normalized RGB with $T_1 = 1.3$ and $T_2 = 0.71$. Its performance on the full illumination invariance data-set is seen in appendix A.

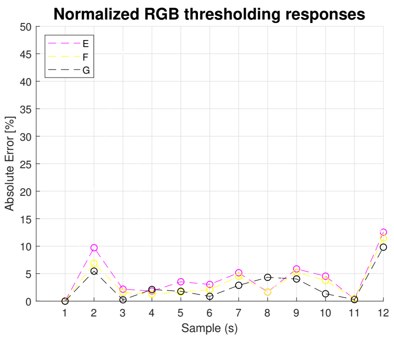
Remarks

There are a several important remarks with respect to the illumination invariance tests.

1. Only two color-spaces were tested, based on the findings from the literary review. There could be other color representations even better suited for illumination invariance which the literary review did not reveal.
2. Only one thresholding scheme per color space were tested. Conceivably, there could exist other, better, thresholding schemes for both HSV and normalized



(a) Normalized RGB thresholding responses for thresholding value sets A-D



(b) Normalized RGB thresholding responses for thresholding value sets E-G

Figure 5.6: Absolute error responses for different threshold value sets for the normalized RGB method. The results are partitioned into 2 plots for readability.

RGB.

3. A limited set of thresholding values were tested. This was mainly due to the limited time-scope of this thesis, where it was necessary to eventually make a decision. However, the author fully acknowledges that better thresholding values could exists.

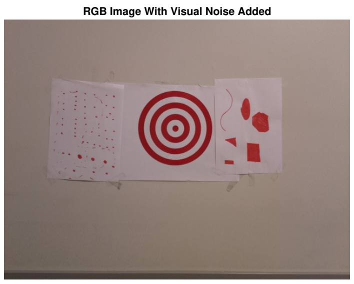
4. It is difficult to cover every lighting condition scenario and the data-set in appendix A is likely lacking in this respect.

As a summary, the results in this section should **not** be interpreted as anything other than a tentative conclusion, where the best performance found was for the proposed normalized RGB thresholding scheme.

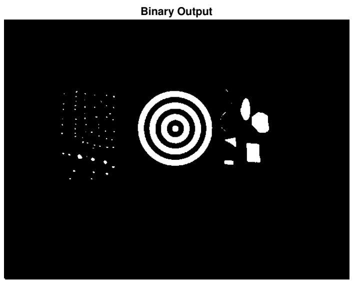
5.4.2. Contour Filter

Since the illumination invariance tests presented in section 5.4.1 are mainly concerned with not producing deformed targets, it is important that the algorithm can lock on to the target, even with visual noise is present.

Visual noise, in this context, refers to binary blobs that has passed the red-pixel search which is not the actual target. Consider figure 5.7 as an example, where additional red blobs has been added.



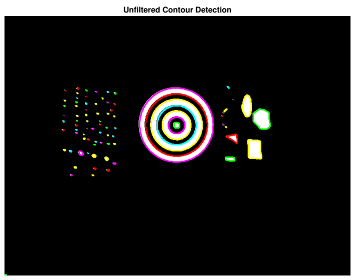
(a) RGB image.



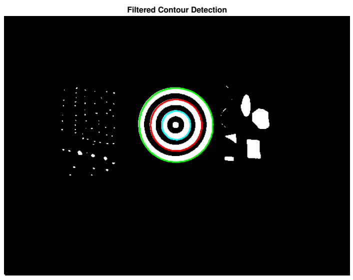
(b) Binary output with visual noise.

Figure 5.7: Example of visual noise. The red pixel search cannot reasonably be expected to filter out the additional blobs, which implies the need for a robust way to separate the target from the added blobs.

Now, consider what would happen if an unfiltered contour search was applied to image (5.7b). The result can be seen in image (5.8a) in figure 5.8. Another example can be seen in figure 5.9.



(a) Unfiltered contour search.



(b) Filtered contour search.

Figure 5.8: Example of a unfiltered and filtered contour search.

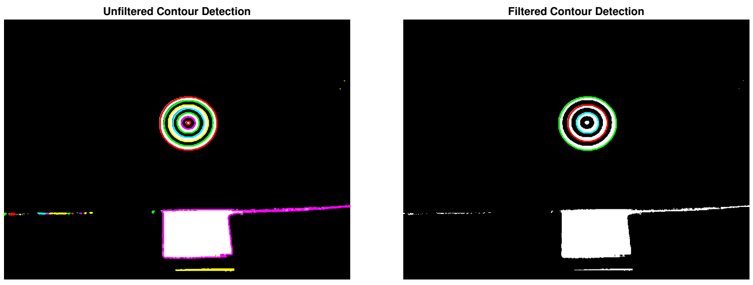
Due to the noise in images (5.8a) and (5.9a), the equations (4.20) and (4.21) would be fed a huge number of discrete distances in the image plane for which there are no corresponding known real world distances, likely resulting in complete failure when attempting to estimate Z^c in (4.22) and by extension X^c and Y^c in (4.26) and (4.27) respectively.

The contour filter this section is referring to, consists of taking all detected contours in image (5.8a) and applying the conditions (4.4), (4.7), (4.8) and (4.9).

Image (5.8b) is the output of applying the contour filter. Its parameters can be seen in table 5.4.

Parameter	k_1	k_2	k_3	k_4	$T_{c_{blob}}$	α_{min}	α_{max}
Value	-0.5416	0.0656	0.0079	$5k_3$	100	$(0.1/100)$	$(20/100)$

Table 5.4: Numerical values for the parameters of the contour filter.



(a) Unfiltered contour search. (b) Filtered contour search.

Figure 5.9: Another example of a unfiltered and filtered contour search. This image was taken from inside the UAV with the camera mounted as shown in figure 5.15 with sunlight streaming into the room. There were no other red objects in the image, but there was still a number of false positives. Even with a large amount of visual noise, the contour filter still manages to correctly identify the target.

Remarks

Despite the approximately uniform blue background the ocean will provide, it is still possible that an image will be contaminated by noise due to varying light conditions. Recall that the illumination invariance scheme proposed in section 5.4.1 is mainly concerned with preserving the integrity of the target, meaning that it is possible that the red pixel threshold can produce false positives. Therefore, a contour filter is added as an additional layer of robustness.

5.5. Position Estimation

Before any actual flight test can be conducted, the proposed position estimation scheme must be tested under controlled conditions. The main goal of these tests is to verify that the algorithm gives reasonable results.

The methodology for generating test-data on some range $[a, b]$ [m] can be outlined as follows:

1. Set the camera to some position $p_{GT} = a$ relative to the target.
2. Capture an image and note p_{GT}.
3. Increment p_{GT} to $p_{GT} \leftarrow p_{GT} + \Delta$
4. Repeat steps 2-3 up to and including $p_{GT} = b$.

Test data refers to images taken at 640×480 resolution. Once the data was gathered, it was fed to a `Matlab` implementation consisting of the following steps

1. Apply the red pixel search scheme chosen in section 5.4.1.
2. Detect contours and apply the contour filter from section 5.4.2.
3. Extract discrete image plane distances from the filtered contours.
4. Compute $\mathbf{P}^c$ according to (4.22), (4.26) and (4.27) using the extracted discrete image plane distances and their corresponding known real world values.

Finally, for these tests, the platform is assumed to be completely stationary meaning no surge, sway or heave and no angular motion in any direction.

5.5.1. Conditions

These tests were conducted under the condition that the CAM frame was aligned directly with the NED frame, meaning that the body frame was ignored for these tests (these tests were not done with the actual hexacopter). This means

$$\mathbf{R}_b^n = \mathbf{R}_c^b = \mathbf{I}_{3\times3} \tag{5.4}$$

where $\mathbf{I}_{3\times3}$ is the 3×3 identity matrix which reduces (4.31) to

$$\mathbf{P}_{uav,rel}^n = -\mathbf{P}^c$$

5.5.2. Vertical Position Estimation

A simple test rig was constructed for testing the vertical position estimation. It consisted of the actual raspberry pi camera mounted on a simple rig, facing the target attached to a wall as seen in figure 5.10. The rig was built so that $X^c \approx X^n_{target} = 0$ and $Y^c \approx Y^n_{target} = 0$. Next, a measurement interval Δ [m] was defined. For the vertical translation tests, $\Delta = 0.1m$. The camera was then moved on the range $[a, b] = [0.2, 2m]$ in Δ increments, generating a total of 19 samples ($s \in [1, 19]$), consisting of 20 images.

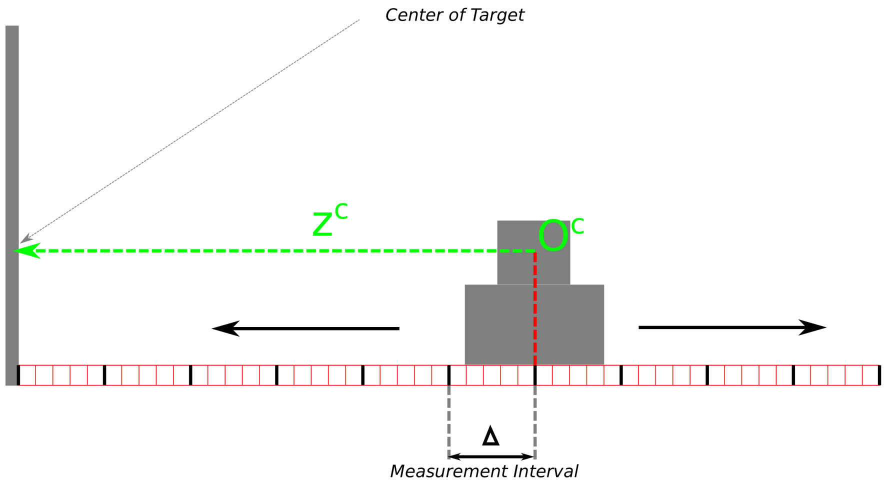
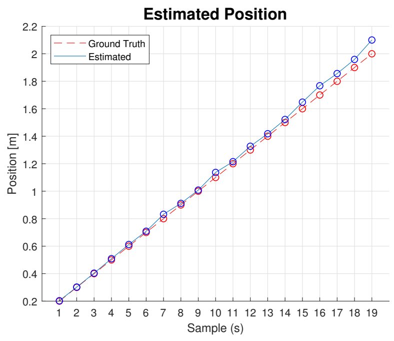


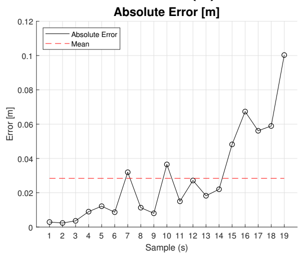
Figure 5.10: Simple sketch of the test rig for vertical position estimation. The camera can be moved along the Z^c axis in Δ delta increments.

The result of the vertical position estimation and can be seen in figure 5.11. Note that technically, it is $\left|\mathbf{P}^n_{uav,rel}\right|$ that is shown, since the sign of $\mathbf{P}^n_{uav,rel}$ is irrelevant for this specific test.

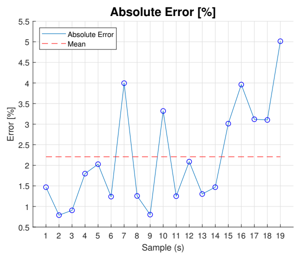
Based on figure 5.11b and 5.11b, it would seem that the overall trend seems to be that the error grows as the distance grows. While the error is $< 6\%$ on the range $[0.1m, 2m]$, it is worth noting that this test is, of course, very idealized, in the sense that there is no platform motion or rotation.



(a) Vertical position estimation.



(b) Absolute error [m].



(c) Absolute error [%].

Figure 5.11: Result of the vertical position estimation in the z-direction.

5.5.3. Horizontal Position Estimation

A simple test rig was constructed for testing the horizontal position estimation as well. Since these tests encompassed both the X^c and Y^c direction, the rig was constructed so that the camera could be rotated to allow for displacement in both horizontal directions. It consisted of the actual raspberry pi camera mounted on a simple rig, facing downwards towards the target on the ground as seen in figure 5.12. The rig was built so that $|Z^c|$ was fixed, i.e $|Z^c| \approx 0.8m = const$. Next, a measurement interval Δ [m] was defined. For the horizontal translation tests, $\Delta = 0.05m$. The camera was then moved on the range $[a, b] = [-0.3m, 0.3m]$ in Δ increments, generating a total of 12 samples each for the X^c and Y^c direction ($s \in [1, 12]$).

The results for the position estimates in the x-direction can be seen in figure 5.13a and the results of the position estimates in the y-direction can be seen in figure 5.14a . The order of magnitude of the absolute error in figure 5.13b and 5.14b is

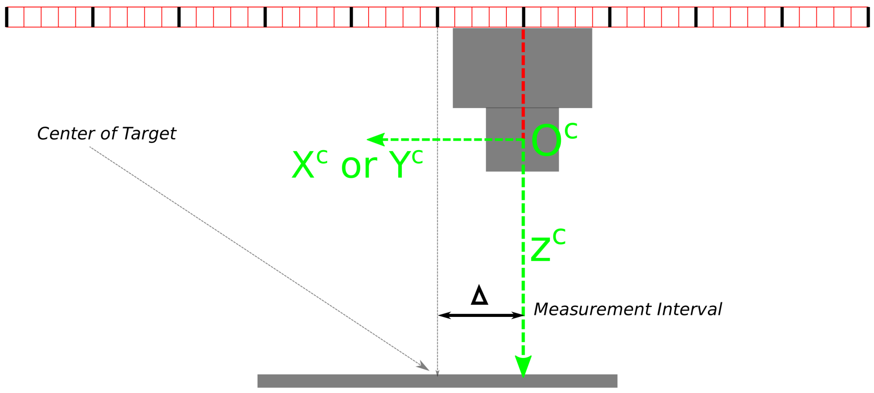


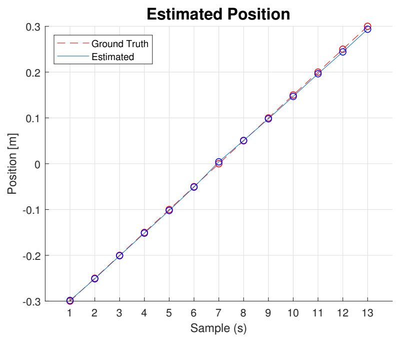
Figure 5.12: Simple sketch of the test rig for horizontal position estimation. The camera can be moved along the X^c or Y^c axis (depending on the camera's orientation) in Δ delta increments.

roughly 10^{-3} (with a couple of exceptions for the y-direction), which is quite precise. However, generally speaking, the percentage error from (5.2) for the y-direction is much more spread out than in the x-direction, showing $\approx 6.2\%$ for sample $s = 9$ in figure 5.14c. Possible sources for this error will be discussed in section 5.5.4.

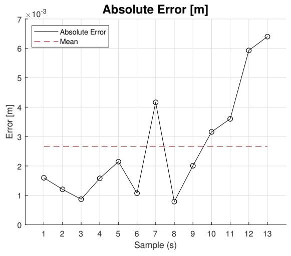
The results in figure 5.13 and 5.14, while good, should be seen in the context of the test setup which was very idealized. Additionally, the range was only $[-0.3m, 0.3m]$. As seen with the vertical position estimation in figure 5.11, the trend is that the error grows as the distance grows, meaning that it is conceivable that larger error margins would be present for larger ranges.

5.5.4. Sources of Error

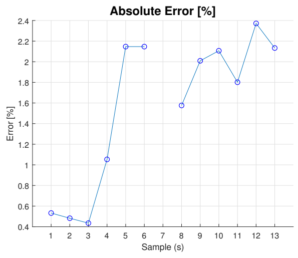
There are several possible sources of error for the results in figure 5.11 5.13 and 5.14. Even though the test rigs shown in figure 5.10 and 5.12 are *theoretically ideal*, there might still be construction errors leading to the assumption in (5.4) not



(a) Horizontal (x) position estimation.



(b) Absolute error [m].

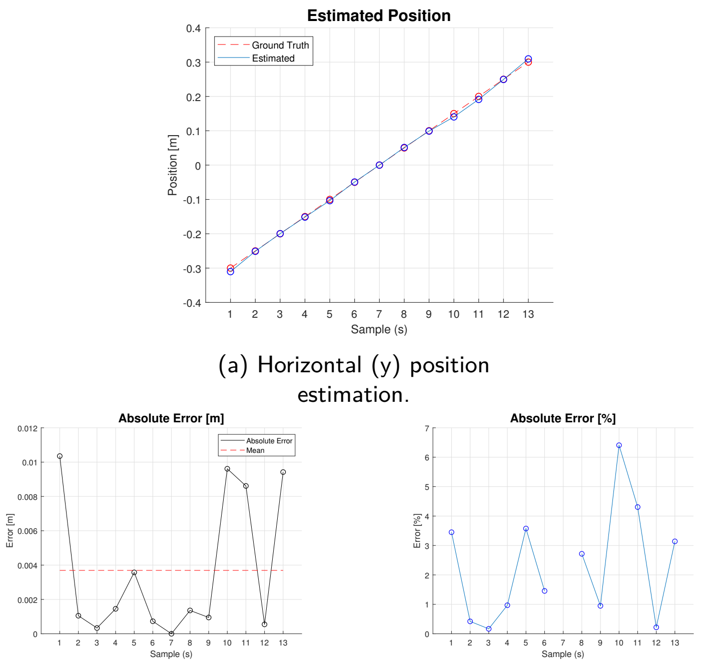


(c) Absolute error [%].

Figure 5.13: Result of the horizontal position estimation in the x-direction. The discontinuity in $s = 7$ in (5.13c) is due to the *ground truth* being 0 for sample 7, meaning it cannot be calculated by the metric in (5.2)

being completely accurate. Additionally, the *ground truth* measurements was done manually by the author, leading to some uncertainty. Since the measuring tape had a resolution of $1mm$, the order of magnitude of this uncertainty is likely $10^{-3}m$, but human error could conceivably lead to much larger errors.

Additionally, for the horizontal measurements, the rig was physically rotated $90°$ after conducting the tests in the x-direction to allow for testing in the y-direction. It is possible that this movement caused some minor physical displacement of the camera, causing the larger error margins seen in figure 5.14c compared to figure 5.13c.



(a) Horizontal (y) position estimation.

(b) Absolute error [m].

(c) Absolute error [%].

Figure 5.14: Result of the horizontal position estimation in the y-direction. The discontinuity in $s = 7$ in (5.14c) is due to the *ground truth* being 0 for sample 7, meaning it cannot be calculated by the metric in (5.2)

5.6. Attitude Compensation

After the position estimation system were shown to preform reasonably well with no rotation in section 5.5, the next step was to include rotation to see how the system preformed.

In order to achieve this, three simple trials were devised, involving pitch and roll (yaw was excluded due to reasons stated in section 5.6.1). Each trial consisted of letting the roll and pitch vary over some pre-determined interval $[a_\phi, b_\phi]$ [deg] for the roll, and $[a_\theta, b_\theta]$ [deg] for the pitch. Each trial also had a set of measurement increments δ_ϕ [deg] for the roll, and δ_θ [deg] for the pitch. Each trial can be summarized as

Trial 1: Test the systems ability to compensate for roll. Let $[a_\phi, b_\phi] \approx [-25°, 25°]$ and $\delta_\phi \approx 5°$. Keep the pitch angles small, $[a_\theta, b_\theta] \approx [-3°, 3°]$ and let $\delta_\theta = 0$.

Trial 2: Test the systems ability to compensate for pitch. Let $[a_\theta, b_\theta] \approx [-25°, 25°]$ and $\delta_\theta \approx 5°$. Keep the roll angles small, $[a_\phi, b_\phi] \approx [-3°, 3°]$ and let $\delta_\phi = 0$.

Trial 3: Test the systems ability to compensate for roll and pitch. Let $[a_\phi, b_\phi] \approx [-25°, 25°]$, $\delta_\phi \approx 5°$ and $[a_\theta, b_\theta] \approx [-25°, 25°]$, $\delta_\theta \approx 5°$.

This methodology was used to produce the data shown in 5.5 (**Trial 1**), 5.6 (**Trial 2**) and 5.7 (**Trial 3**). As a final note, due to the nature of the setup, it proved difficult to match the proposed intervals and measurement increments exactly. Therefore, the intervals and increments proposed in the three trials listed above are only approximations, and does not match the exact values seen in table 5.5, 5.6 and 5.7.

5.6.1. Conditions

In this case the rotation matrix in (4.31) is used as is, with one exception. The testing was done indoors at the NTNU UAV-lab, meaning that the on-board magnetometer produced false yaw measurements. This was remedied by setting $\psi^{nb} \approx 0$ in (2.6).

5.6.2. Setup

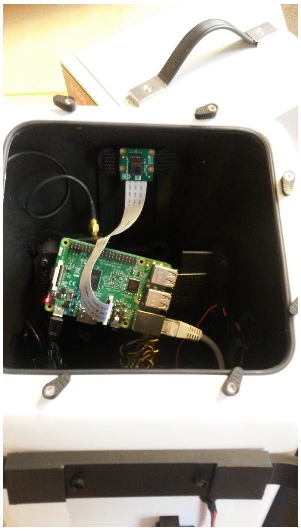
The camera and raspberry pi was mounted inside the UAV as shown in figure 5.15. A power supply was used to power the UAV's on-board systems, while the raspberry pi had its own separate power supply for these tests.

Since the UAV was not flying, a simple method for inducing roll and pitch was used, namely, displacing the uav as shown in figure 5.16. The target was then placed on the floor beneath the UAV. For these tests, a laser range finder was available, making the process of finding ground truth measurements much easier than if a ruler was used.

5.6.3. Results

The results for (**Trial 1**) is seen in table 5.5 with corresponding error metrics seen in figure 5.17, the results for (**Trial 2**) is seen in table 5.6 with corresponding error metrics seen in figure 5.6 and the results for (**Trial 3**) is seen in table 5.7 with corresponding error metrics seen in figure 5.7.

The Z^n error in terms of meters for both **Trial 1** in figure 5.17a and **Trial 2** in figure 5.18a are quite large compared to the other two directions. Recall the results


(a) Raspberry pi camera mounted.

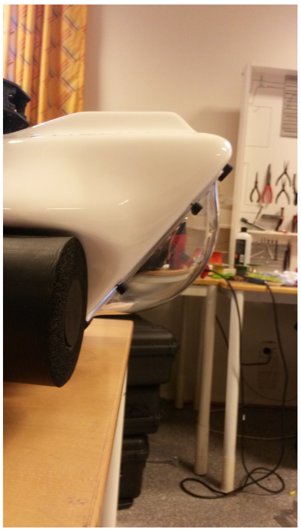

(b) Sideways view.

Figure 5.15: The raspberry pi camera was mounted as shown in (a), and was then covered by the plastic dome shown in (b).

presented in figure 5.11 in section 5.5. There, the [%] error for the Z^n direction never exceeds $\approx 5\%$ while it exceeds $\approx 7\%$ for both **Trial 1** and **Trial 2** as shown in figure 5.17b and 5.18b respectively. This confirms (perhaps not surprisingly) that introducing roll and pitch has an adverse effect on the distance estimation in the Z^n direction. This is important to note, as the error in Z^c will propagate to the estimation of X^c and Z^c (recall (4.26) and (4.27)), and, by extension X^n and Y^n. The [%] based absolute error seen in figure 5.17b for **Trial 1** and 5.18b for **Trial 2** captures this effect better than the meter based absolute error, as both X^n and Y^n are much closed to Z^n in terms of [%] error for both trials.

For **Trial 1**, the X^n absolute error in terms of meters seen in figure 5.17a is, relative to the Y^n and Z^n error, fairly small. This is likely due to a couple of reasons. First, the magnitude X^n is quite small ($\approx 0.1m$), and in section 5.5 the general trend found was that the error grows as the distance from the target grows. Second, by inspecting the rotation matrix in (2.6), one can see that the roll does not have a huge impact on the X^n direction, i.e the roll angle does not "distort" the distances seen by the camera in the X^n direction. This reasoning also seems to apply to **Trial 2**, where is Y^n error in 5.18a is quite small, relative to the X^n and Z^n error. Here, the pitch does not distort the distances seen in the Y^n direction.

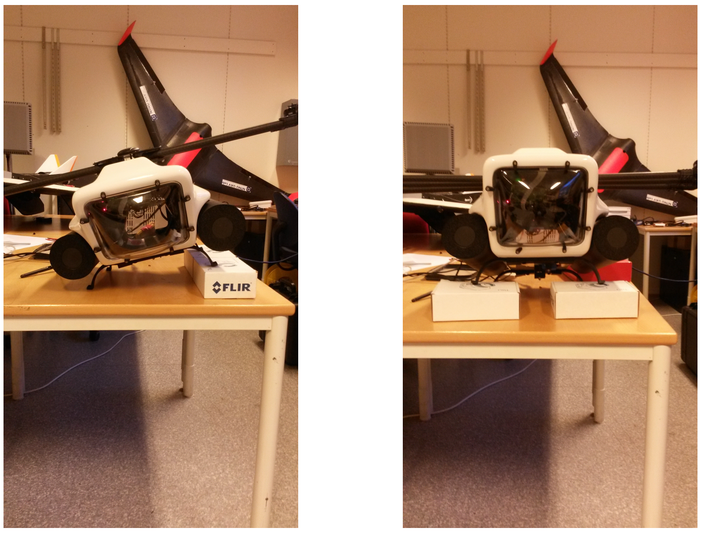

(a) Inducing roll. (b) Inducing pitch.

Figure 5.16: The (very simple) method for inducing roll (a) and pitch (b).

In **Trial 3** (table 5.7 and figure 5.19), both roll and pitch is increased. Compared to **Trial 1** and **Trial 2**, the mean error has increased for both error metrics in all three directions. This is not completely unexpected, as the system now has to account for large pitch and roll values simultaneously. As seen in figure 5.19, the overall trend is that the error metrics decrease towards the middle. Coupling this observation with the data from table 5.7, the data confirms that the vision system's performance is adversely affected by larger Euler angles and larger distances to the target.

These are important considerations when envisioning the complete system in chapter 3. The tracking mode presented in section 3.2.2 will use a GPS-based height estimate Z_{track} to align the UAV with the platform, i.e $X^n_{n,rel}$ and $Y^n_{n,rel}$ will depend on Z_{track} (recall (4.26) and (4.27)). As briefly discussed in the literature review, the GPS-based height estimate can be subject to quite large errors in the vertical direction (recall that the one found in [9], with vertical error $\pm 5m$). Now, if the UAV couples an uncertain estimate of the altitude (Z_{search}) with large Euler angles, aligning the UAV according to (3.5) could prove difficult as $X^n_{n,rel}$ and $Y^n_{n,rel}$ would (as suggested by the results in figure 5.17b, 5.18b and 5.19b) presumably have a [%] based error relatively close to Z_{search}. Therefore, it could be wise to constrain the roll and pitch in the tracking mode, so that the UAV error introduced by the Euler angles is kept

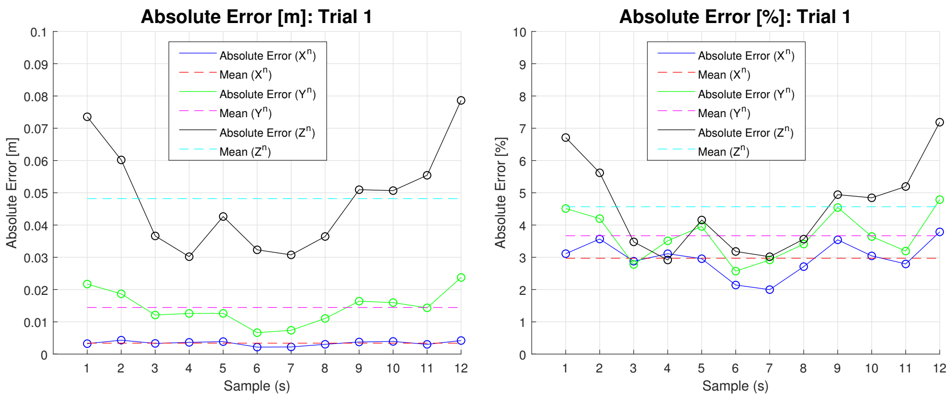
Sample (s)	**Angles [deg]**		**EST [m]**			**GT [m]**		
s	Roll (ϕ)	Pitch (θ)	X^n	Y^n	Z^n	X^n	Y^n	Z^n
1	-25.043	2.571	0.101	0.459	-1.022	0.104	0.481	-1.096
2	-20.141	2.603	0.125	0.464	-1.131	0.121	0.445	-1.071
3	-14.616	1.78	0.118	0.424	-1.016	0.115	0.436	-1.053
4	-10.403	2.637	0.113	0.372	-1.065	0.117	0.359	-1.035
5	-5.119	1.453	0.135	0.332	-1.069	0.131	0.319	-1.026
6	0.821	2.151	0.103	0.265	-0.984	0.101	0.258	-1.016
7	0.911	3.076	0.113	-0.246	-1.050	0.111	-0.253	-1.019
8	4.874	2.889	0.109	-0.335	-1.059	0.112	-0.324	-1.023
9	10.131	1.344	0.102	-0.377	-0.980	0.106	-0.361	-1.031
10	14.762	1.391	0.133	-0.454	-1.097	0.129	-0.438	-1.046
11	20.095	2.36	0.105	-0.463	-1.122	0.108	-0.449	-1.067
12	24.533	2.324	0.107	-0.472	-1.015	0.111	-0.496	-1.094

Table 5.5: Results for trial 1. Here, the roll is gradually increased while keeping the pitch angles small. The corresponding error metrics can be seen in figure 5.17.

beneath a certain percentage. A possible general form for such a bound could simply be

$$|\phi_{cmd}| \leq \phi_{bound} = const, \qquad |\theta_{cmd}| \leq \theta_{bound} = const$$

where ϕ_{cmd} and θ_{cmd} are the commands to the control system and ϕ_{bound} and θ_{bound} are some sensible values in degrees. While it is tempting to suggest concrete values for ϕ_{bound} and θ_{bound} based on the error margins in figure 5.17, 5.18 and 5.19, it might be inappropriate, since the ground truth distances considered in table 5.5, 5.6 and 5.7 are relatively small compared to what might be encountered in a "real world"



(a) Absolute Error [m].

(b) Absolute Error [%].

Figure 5.17: Error metrics for the results seen in table 5.5.

scenario.

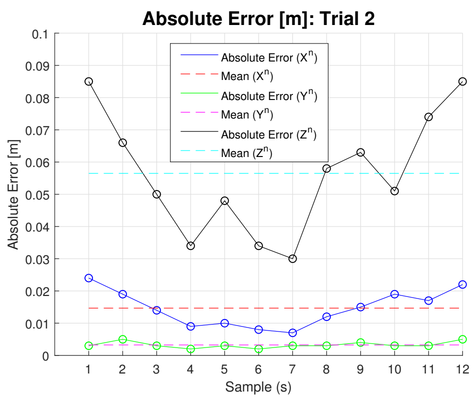
5.6.4. Sources of Error

Potential sources for the error margins seen in figure 5.17, 5.18 and 5.19, are listed below

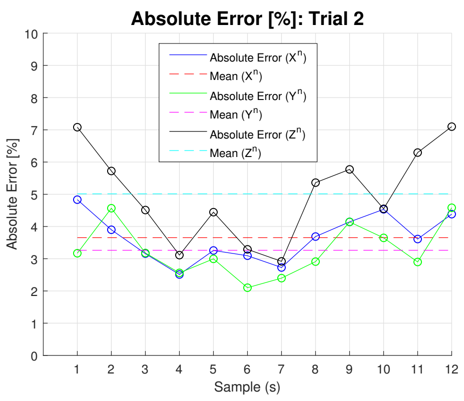
- Consider the mounting of the raspberry pi camera as seen in figure 5.15a. Due to time constraints, a proper fastening mechanism was not devised. While saving time, it could cause a discrepancy in the alignment between the BODY and CAM frame. Specifically, it could quite possibly perturb the angles seen in (2.7), causing the rotation in (4.31) to be slightly off.
- The IMU values were manually read to only three decimal points from the Mission planner interface seen in figure 5.1. Even when the UAV stood still, the values fluctuated. The angle values presented in table 5.5, 5.6 and 5.7 are therefore approximations of the actual IMU values.
- Human error could also have influenced the results. The laser range finder used was hand-held, so it is conceivable that the measurements are slightly off.

Sample (s)	**Angles [deg]**		**EST [m]**			**GT [m]**		
s	Roll (ϕ)	Pitch (θ)	X^n	Y^n	Z^n	X^n	Y^n	Z^n
1	0.981	-24.957	0.529	0.101	-1.292	0.505	0.098	-1.207
2	2.712.	-19.887	0.459	0.105	-1.095	0.478	0.100	-1.161
3	2.432	-14.928	0.455	0.105	-1.162	0.441	0.108	-1.112
4	1.816	-10.223	0.355	0.098	-1.132	0.364	0.096	-1.098
5	2.439	-5.261	0.328	0.096	-1.028	0.318	0.099	-1.076
6	3.787	0.53	0.273	0.101	-1.052	0.265	0.103	-1.018
7	2.987	0.81	-0.250	0.106	-1.045	-0.257	0.109	-1.015
8	1.939	4.941	-0.328	0.104	-1.139	-0.316	0.107	-1.081
9	1.029	9.872	-0.352	0.108	-1.033	-0.367	0.104	-1.096
10	2.345	15.113	-0.441	0.098	-1.068	-0.422	0.095	-1.119
11	3.011	20.106	-0.476	0.106	-1.098	-0.459	0.109	-1.172
12	2.631	25.098	-0.524	0.110	-1.116	-0.502	0.105	-1.201

Table 5.6: Results for trial 2. Here, the pitch is gradually increased while keeping the roll angles small. The corresponding error metrics can be seen in figure 5.18.

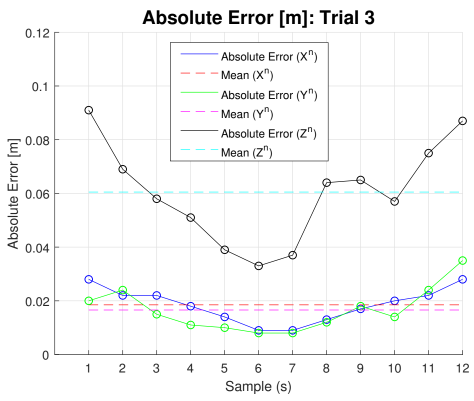


(a) Absolute Error [m].

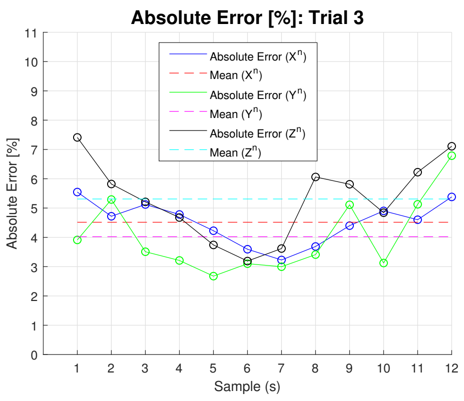


(b) Absolute Error [%].

Figure 5.18: Error metrics for the results seen in table 5.6.



(a) Absolute Error [m].



(b) Absolute Error [%].

Figure 5.19: Error metrics for the results seen in table 5.7.

Sample (s)	**Angles [deg]**		**EST [m]**			**GT [m]**		
s	Roll (ϕ)	Pitch (θ)	X^n	Y^n	Z^n	X^n	Y^n	Z^n
1	-25.142	-25.341	0.535	0.521	-1.314	0.507	0.501	-1.223
2	-20.091	-19.193	0.441	0.428	-1.109	0.463	0.452	-1.178
3	-14.908	-15.012	0.451	0.424	-1.177	0.429	0.439	-1.119
4	-10.115	-9.891	0.350	0.337	-1.037	0.368	0.348	-1.088
5	-4.776	-5.075	0.335	0.347	-1.086	0.321	0.357	-1.047
6	0.982	1.124	0.242	0.256	-0.987	0.251	0.248	-1.020
7	1.182	1.028	-0.262	-0.250	-0.978	-0.271	-0.258	-1.015
8	5.107	5.055	-0.346	-0.353	-1.123	-0.359	-0.341	-1.059
9	10.111	9.937	-0.359	-0.371	-1.046	-0.376	-0.353	-1.111
10	15.075	15.066	-0.431	-0.463	-1.112	-0.411	-0.449	-1.169
11	19.871	20.776	-0.465	-0.494	-1.276	-0.487	-0.470	-1.201
12	25.167	25.123	-0.542	-0.478	-1.316	-0.514	-0.513	-1.229

Table 5.7: Results for trial 3. Here, both the pitch and roll is gradually increased. The corresponding error metrics can be seen in figure 5.19.

Chapter 6

Conclusion

A possible high-level solution for allowing an UAV to autonomously return to a landing platform from a set of arbitrary GPS coordinates has been presented. As a part of this system, a detailed proposition of a vision-based landing algorithm has also been presented. Within said algorithm, an attempt has been made to account for varying light conditions, robust target detection through filtering of visual noise and ensuring a reliable 3D position estimation with attitude compensation.

The illumination invariance method chosen was tested on a set of 12 images which displayed varying light conditions. From figure 5.6b it can be see that the method performed with $< 6.5\%$ error for $s \in [1, 11]$, but shoved an error spike of $\approx 10\%$ for the last sample (see figure A.1x in appendix A for the binary output of $s = 12$). Any definitive conclusions regarding satisfactory performance would have to wait until the system has been tested outdoors, but based on the results in appendix A, the proposed scheme at least shows promise with respect to achieving illumination invariance. The robust target detection method, consisting of the contour filter in section 5.4.2, has been shown to be able to filter out visual noise (as it is defined in section 1.1). Its parameters shown in 5.4 was found experimentally through trial and error, but should not be considered immutable.

The position estimation scheme proposed was put through two test stages. First, it was tested without rotation, where the goal was to validate the basic estimation of $\mathbf{P}^c$ as given by (4.26), (4.27) and (4.22). The [%] based absolute error seen in figure 5.11, 5.13c and 5.14c never exceeded $\approx 7\%$ for these tests, which was deemed sufficient for further testing. Next, the system was tested with rotation at the NTNU UAV-lab. Even when introducing both pitch and roll, the [%] based absolute error is bounded by $< 8\%$ for all three directions as seen in figure 5.19b. However, the average error metric also shown in figure 5.19b shows an increase in error for all three direction when compared to 5.11, 5.13c and 5.14c. While it is difficult to precisely define what would constitute satisfactory performance, it is argued that based on the meter based absolute error seen in figure 5.19a, the system's performance at least deems it worthy of further real-world tests (an error of $\approx 3cm$ in the X^n and Y^n direction is unlikely to cause the UAV to miss the platform entirely).

6.1. Future Work

There are several possibilities for future work regarding the system proposed in this thesis. All suggestions here are intended to advance the development of the system towards an actual flight test on the ocean.

- A sensible first step would be to implement the vision-system in a real-time suited programming language. The author would propose `C/C++`, not only for their excellent real-time performance, but also because they have open source libraries such as `OpenCV` available to them, which provides a lot of useful functionality for implementing the vision system proposed in this thesis. As a concrete example, in order to extract the contours with `Matlab`, a function called `bwboundaries` was used. `OpenCV` has an analogous function called `findCountours`, simplifying the implementation process.

- The illumination invariance method proposed here should be tested outdoors.

- Conduct a manual flight with the algorithm running on-board. The algorithm would have no actual control over the UAV here, but it would be possible to inspect the results of the position estimation and attitude compensation in a real world setting without risking that the algorithm crashes the UAV. The main challenge for conducting such a test will be finding a reliable method to measure the ground truth. One possibility is using the GPS, however, as discussed earlier, the GPS may have non-trivial error margins which should be taken into considerations.

- Care should be taken to synchronize IMU data and camera data, when actual flight tests are conducted. If they are not synchronized, the algorithm will be fed an image with a set of incorrect Euler angles, likely causing erroneous position estimation. One possible method of synchronization is using matched time-stamps for both the IMU data and the camera data.

6.2. Lessons Learned

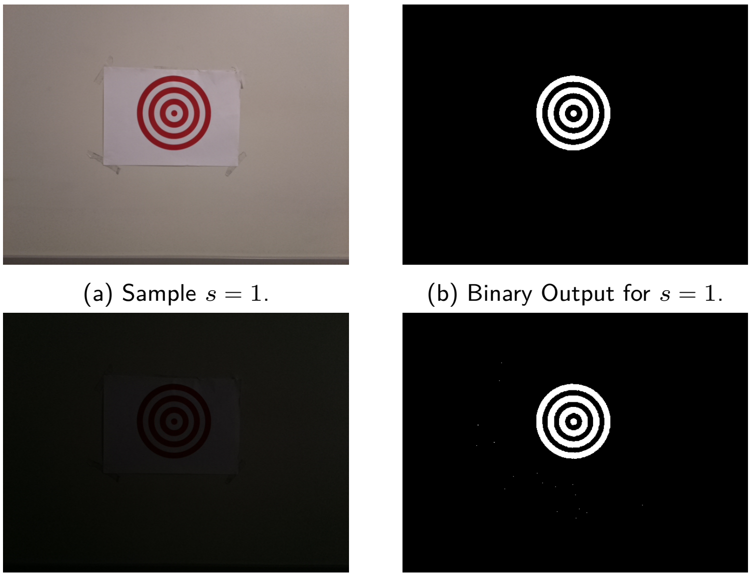
There were several lessons learning during the course of this thesis. Admittedly, some can be considered obvious, but the author would like to list some of them in the hopes that they can help another student who might happen to read this paper in the future.

- Don't underestimate the usefulness of the literature review. It can be incredibly powerful for finding a sensible "angle of attack" for the problem at hand. For example, a very useful result produced by the literature review in this thesis was that a common approach to vision-based pose estimation started with the projection equations as discussed in section 1.2. This formed the basis for the vision system proposed in chapter 4.
- Experiments will likely be much more time-consuming than first anticipated. For example, while it may not be obvious, the data presented in table 5.5, 5.6 and 5.7 were the result of many, *many* hours of meticulous work at the UAV-lab, due to the time-consuming process of gathering accurate ground truth measurements.
- Testing parts of the system should be done in a careful and controlled manner. Start by considering a simple case. For example, this is the reason for the tests presented in section 5.5, where the system is tested with no attitude compensation. The author initially tried to test the system with rotation, but struggled to find the errors in the equations when the results produced were nonsensical. When the simpler case in section 5.5 was considered, the culprit was quickly identified as an erroneous expression for computing Z^c in the earlier drafts of the vision system.

Appendices

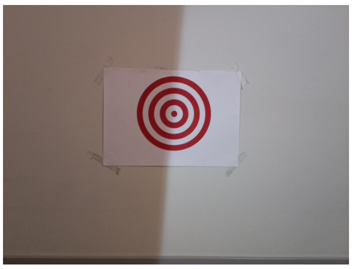
Appendix A

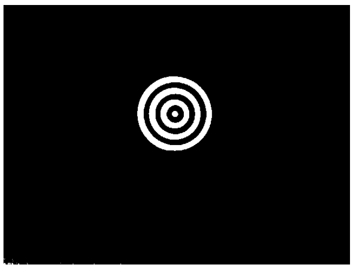
Robustness Results

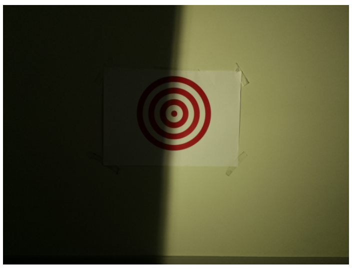

(a) Sample $s = 1$. (b) Binary Output for $s = 1$.

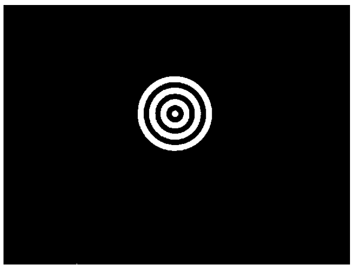
(c) Sample $s = 2$. (d) Binary Output for $s = 2$.

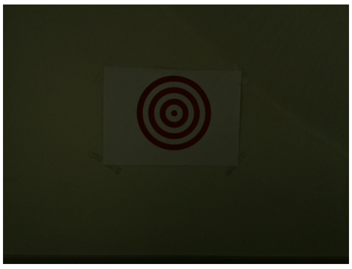
Figure A.1: Results of illumination invariance testing.

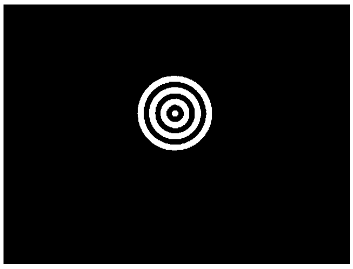

(e) Sample $s = 3$.

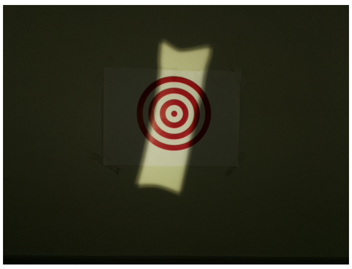

(f) Binary Output for $s = 3$.


(g) Sample $s = 4$.


(h) Binary Output for $s = 4$.


(i) Sample $s = 5$.


(j) Binary Output for $s = 5$.


(k) Sample $s = 6$.

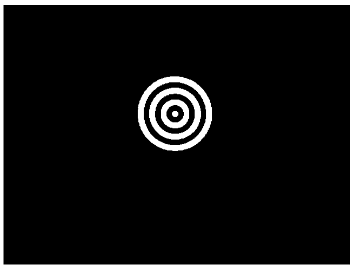
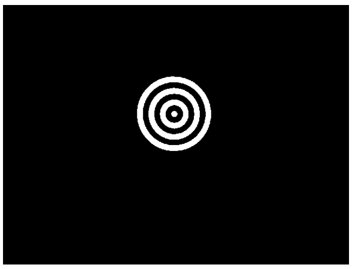
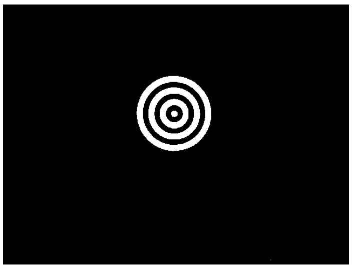

(l) Binary Output for $s = 6$.

Figure A.1: (Continued) Results of illumination invariance testing.

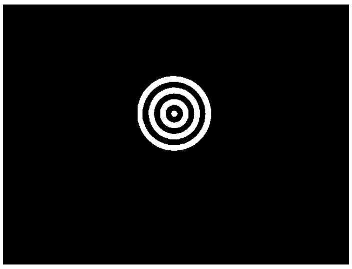
(m) Sample $s = 7$.

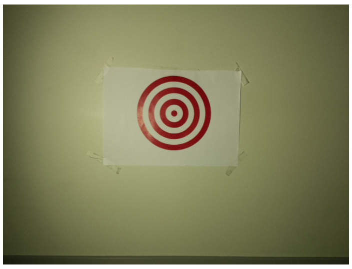

(n) Binary Output for $s = 7$.

(o) Sample $s = 8$.


(p) Binary Output for $s = 8$.

(q) Sample $s = 9$.


(r) Binary Output for $s = 9$.


(s) Sample $s = 10$.

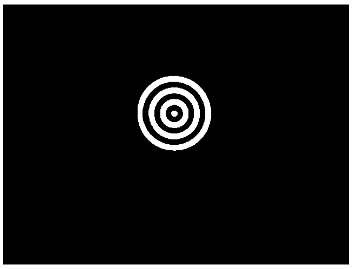
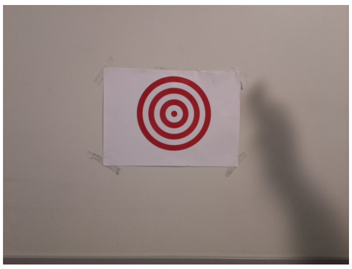
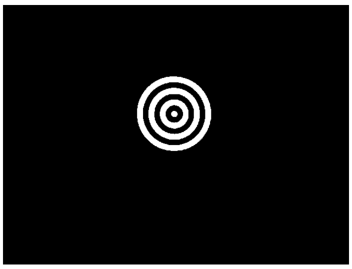
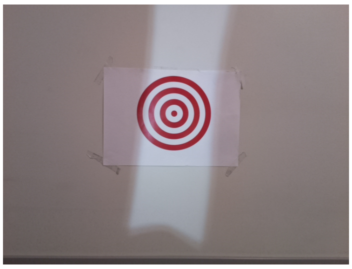

(t) Binary Output for $s = 10$.

Figure A.1: (Continued) Results of illumination invariance testing.


(u) Sample $s = 11$.


(v) Binary Output for $s = 11$.


(w) Sample $s = 12$.

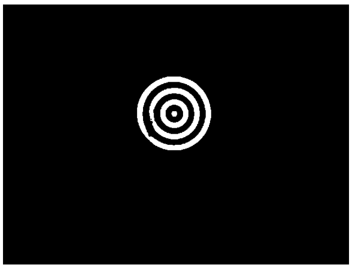

(x) Binary Output for $s = 12$.

Figure A.1: (Continued) Results of illumination invariance testing.

Bibliography

[1] A. Hodgson, N. Kelly, and D. Peel, *Unmanned aerial vehicles (uavs) for surveying marine fauna: a dugong case study,* PloS one **8**, e79556 (2013).

[2] I. I. Kaminer, O. A. Yakimenko, V. N. Dobrokhodov, M. I. Lizarraga, and A. M. Pascoal, *Cooperative control of small uavs for naval applications,* in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, Vol. 1 (IEEE, 2004) pp. 626–631.

[3] *Image: Right-hand rule,* `https://en.wikipedia.org/wiki/File:Right_hand_rule_Cartesian_axes.svg` (2008), [Online; accessed 25-March-2018].

[4] *Image: RGB color-space,* `https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png` (2008), [Online; accessed 1-April-2018].

[5] *Image: HSV color-space,* `https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png` (2008), [Online; accessed 1-April-2018].

[6] *Image: Red bullseye target,* `http://www.targets.ws/bullseye-targets.htm` (2018), [Online; accessed 14-March-2018].

[7] *Raspberry Pi Camera Module V2,* `https://www.raspberrypi.org/documentation/hardware/camera/README.md` (2016), [Online; accessed 12-March-2018].

[8] P. Castillo, R. Lozano, and A. Dzul, *Stabilization of a mini rotorcraft with four rotors,* Control Systems, IEEE **25**, 45 (2005).

[9] F. Kendoul, K. Nonami, I. Fantoni, and R. Lozano, *An adaptive vision-based autopilot for mini flying machines guidance, navigation and control,* Autonomous Robots **27**, 165 (2009).

[10] S. Saripalli, J. Montgomery, and G. Sukhatme, *Visually guided landing of an unmanned aerial vehicle,* Robotics and Automation, IEEE Transactions on **19**, 371 (2003).

[11] K. Nordberg, P. Doherty, G. Farnebäck, P.-E. Forssén, G. Granlund, A. Moe, and J. Wiklund, *Vision for a uav helicopter,* in *International Conference on Intelligent Robots and Systems (IROS), workshop on aerial robotics. Lausanne, Switzerland* (2002) pp. 29–34.

[12] A. D. Wu, E. N. Johnson, and A. A. Proctor, *Vision-aided inertial navigation for flight control,* Journal of Aerospace Computing, Information, and Communication **2**, 348 (2005).

[13] P. Smith, B. Sridhar, and B. Hussien, *Vision-based range estimation using helicopter flight data,* (IEEE Publishing, 1992) pp. 202–208.

[14] J. Hintze, *Autonomous landing of a rotary unmanned aerial vehicle in a non-cooperative environment using machine vision,* (2004).

[15] C. Xu, L. Qiu, M. Liu, B. Kong, and Y. Ge, *Stereo vision based relative pose and motion estimation for unmanned helicopter landing,* Information Acquisition, 2006 IEEE International Conference on , 31 (2006).

[16] V. Dobrokhodov, I. Kaminer, K. Jones, and R. Ghabcheloo, *Vision-based tracking and motion estimation for moving targets using small uavs,* (IEEE, USA, 2006).

[17] T. Daquan and Z. Hongyue, *Vision based navigation algorithm for autonomic landing of uav without heading attitude sensors,* Signal-Image Technologies and Internet-Based System, 2007. SITIS '07. Third International IEEE Conference on , 972 (2007).

[18] M. Dunbabin, P. Corke, and G. Buskey, *Low-cost vision-based auv guidance system for reef navigation,* (IEEE, USA, 2004) pp. 7–12.

[19] R. Lozano, *Unmanned aerial vehicles: Embedded control* (John Wiley & Sons, 2010).

[20] G. Chatterji, P. Menon, and B. Sridhar, *Gps/machine vision navigation system for aircraft,* Aerospace and Electronic Systems, IEEE Transactions on **33**, 1012 (1997).

[21] D. Hubbard, B. Morse, C. Theodore, M. Tischler, and T. Mclain, *Performance evaluation of vision-based navigation and landing on a rotorcraft unmanned aerial vehicle,* (IEEE, 2007) pp. 5–5.

[22] O. Amidi, *An autonomous vision-guided helicopter,* PHD Thesis (1996).

[23] I. N. Thiang, Dr.Lumaw, and H. M. Tun, *Vision-based object tracking algorithm with ar. drone,* International Journal of Scientific & Technology Research **4**, 135 (2015).

[24] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control,* (2011).

[25] G. D. Finlayson, B. Schiele, and J. L. Crowley, *Comprehensive colour image normalization,* in *Computer Vision — ECCV'98*, edited by H. Burkhardt and B. Neumann (Springer Berlin Heidelberg, Berlin, Heidelberg, 1998) pp. 475–490.

[26] A. Hanbury and J. Serra, *A 3d-polar coordinate colour representation suitable for image analysis,* submitted to Computer Vision and Image Understanding (2002).

[27] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision second edition,* Cambridge University Press (2000).

[28] *Introduction to Computer vision by Georgia Tech (Free course),* `https://www.udacity.com/course/introduction-to-computer-vision--ud810` (2011), [Online; accessed 25-March-2018].

[29] U. M. Office, *Beaufort wind force scale,* `https://www.metoffice.gov.uk/guide/weather/marine/beaufort-scale` (2016), [Online; accessed 7-March-2018].

[30] S. M. Esmailifar and F. Saghafi, *Autonomous unmanned helicopter landing system design for safe touchdown on 6dof moving platform,* in *Autonomic and Autonomous Systems, 2009. ICAS'09. Fifth International Conference on* (IEEE, 2009) pp. 245–250.

[31] L. Marconi, A. Isidori, and A. Serrani, *Autonomous vertical landing on an oscillating platform: an internal-model based approach,* Automatica **38**, 21 (2002).

[32] S. Shah, *Real-time image processing on low cost embedded computers,* Techincal report No. UCB/EECS-2014–117 (2014).