Faculty of Engineering and Built Environment

School of Engineering

Mechancial Engineering Project B

Semester 1 - 2019

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FINAL YEAR PROJECT

PROJECT TITLE

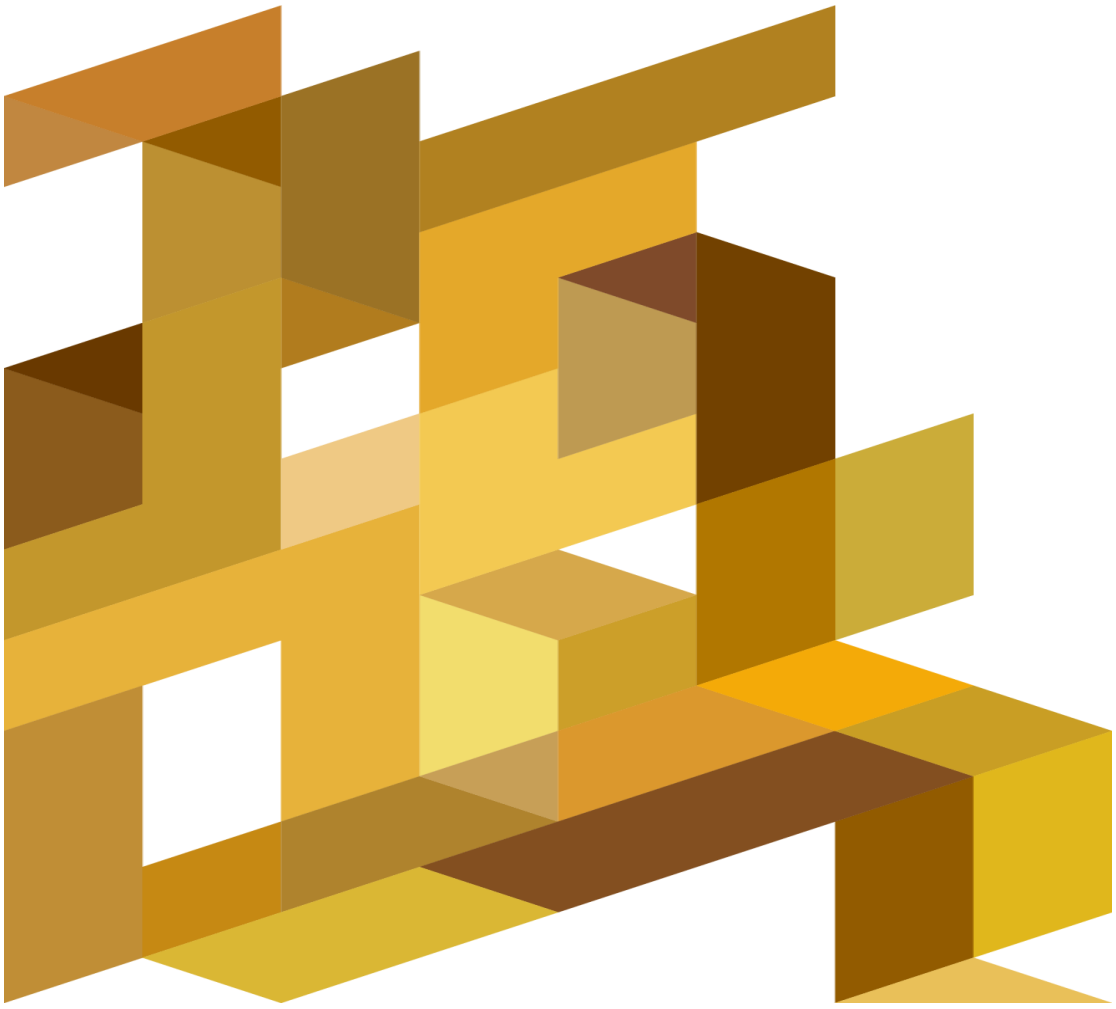
Optical Flow Based Obstacle Avoidance for a Fixed Wing UAV

NAME

Patrick Prell

SUPERVISOR

Dr Christopher Renton



Optical Flow Based Obstacle Avoidance for a Fixed Wing UAV

Final Year Project Report - MECH4841 Part B

June 2019

Patrick Prell [1,??]

[1] *Student of Mechatronics Engineering,*

The University of Newcastle, Callaghan, NSW 2308, AUSTRALIA

Student Number: 3204734

E-mail: Patrick.Prell@uon.edu.au

Abstract

Remember that executive summary may include the following information:

- Defines the intention of the report.
- Places the report in context so the reader knows why it is important to read it.
- Why is it important?
- What problem is addressed?
- Briefly states the results
- Briefly presents the implications and recommendations

Contents

Listings

1 Introduction

Exploring and navagating unknown environments is an essential instinct in biology and avoiding obstacles is an major component. For robotic vehicles to perform high level tasks, they must also have these instincts.

- **Position**: Show there is a problem and that it is important to solve it.
- **Problem**: Describe the specifics of the problem you are trying to address
- **Proposal**: Discuss how you are going to address this problem. Use the literature to back-up your approach to the problem, or to highlight that what you are doing has not been done before

The rest of the report is organised as follows. **??** describes items related to the core content. **??** concludes the report. Appendix shows an example of how to make a Table.

2 Background

To the average human or most mammals for that matter, visual tasks such as identifying objects and interpreting visual cues might seem like a trivial task. However, implementing algorithms that mimic a mammalian visual cortex is infuriatingly difficult and largely remains an unsolved problem (**?**). One fact often overlooked is that for all of its amazing abilities, approximately 1/3 of the human brain is dedicated to analysing information collected from our eyes. This statistic helps to put into perspective how challenging the field of computer vision can be.

The concept of computer vision has been explored since the inception of artificial intelligence in the late 1960s. Computer vision was made possible by leaps in computer power, allowing manipulation of large (for the time) data sets such as images and videos. The following decade saw research attempting to extract information from a sequence of images using visual cues embedded in the motion of an image. This brought about research on visual navigation as a viable method for a mobile robot to navigate an environment. Using Visual Navigation on a robot to move through an unstructured environment with no prior knowledge of the map is often based on optical flow. A motion field is estimated from fusing the motion of the robot with the motion of the pixels of a video feed. Furthermore, visual navigation techniques have been inspired by biology. Observing the way that animals and insects utilise their visual systems to move about and map their environments, the processes for which this is achieved can be replicated and implemented on a robotic system (**?**).

The following section builds the background knowledge required to utilise visual cues as input to an obstacle avoidance system. In particular, a history of visual navigation is outlined; the geometry of a camera and the tools for calibrating it are introduced; and the kinematics of vision optical flow and optical flow on the view sphere. Finally, this section discusses the approach taken to validate the obstacle avoidance algorithm.

2.1 Optical Flow

In computer vision, we are constantly running into the problem of dimensional compression. A lot of information is lost in the projection from 3-dimensions to a 2-dimensional plane, and when that projection is inverted, the lost data needs to be inferred. In the case of optical flow, the lost data comes mainly from different assumptions that are made to make the optical flow algorithm solvable. Each Optical flow algorithm has its own set of assumptions and constraints.

Beginning with a basic description of optical flow. Optical flow is the apparent movement of objects, textures and edges between a sequence of frames. Optical flow can be thought of as the time derivative of two images. It is difficult to imagine taking the derivative of an image, and it is this which makes evaluating optical flow so difficult. One major problem with calculating optical flow is the *aperture problem*. This problem manifests in many optical illusions that trick the human brain as well. An example of this effect can be seen in **??**. To an unassuming camera, it is impossible to tell the direction of the striped pattern.

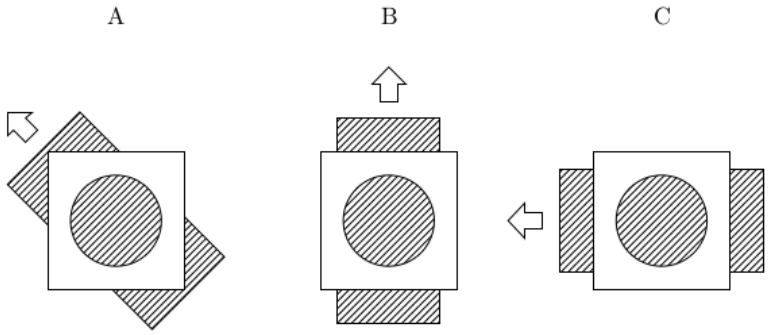


Figure 1: The direction of the stripes through the aperture is unclear without more context

Many optical flow algorithms rely on the derivative of an image. More specifically how does the pixel intensity change from one pixel to the next: This is known as the *image gradient*. Due to the intensity of an image only being defined in discrete locations, an assumption of some underlying continuous intensity function needs to be made. similar to the traditional definition of derivatives, the image gradient is separable. To calculate the partial derivatives if an image, an *image kernal* is designed to capture the change intensity along each particular dimension. Optical flow requires the image gradient in x, y and t. **??** show the image kernels for these directions.

$$\frac{\delta I}{\delta x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \frac{\delta I}{\delta y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \frac{\delta I}{\delta t} = \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}}_{\text{frame } t_0} + \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{frame } t_1} \tag{2.1}$$

There have been many solutions proposed to calculate the motion of pixels in an image. Each method being designed for different tasks and have their own set of advantages and disadvantages in a particular situation they were designed for. Two early methods for calculating optical flow are the *Horn and Schunk* optical flow algorithm and the *Lucas - Kanade* method. Although both of these methods were devised over 20 years ago, they have laid a foundation for many optical flow algorithms following (**?**). The Farneback algorithm, a much more recent algorithm will then be discussed as this is the algorithm used for the project.

2.1.1 Horn & Schunk Optical Flow

Horn and Schunck optical flow is based around an assumption known as the *Brightness constancy*. This says that the intencity of a pixel remains constant over a small step in time.

$$\frac{dI}{dt} = 0 \tag{2.2}$$

Defining a vector field $\mathbf{h} = (u, v) = \left(\frac{du}{dt}, \frac{dv}{dt}\right)$ that hold the brightness constancy true, the chain rule is used to derive the following *optical flow constraint* equation.

$$\nabla I \mathbf{h} + I_t = 0 \tag{2.3}$$

This equation exposes the aperture problem in **??** mathematically. There are more variables to solve for than there are linearly independent equations meaning there is no unique solution for these variables. in **?**, This problem addressed by formulating the estimation of optical flow as a minimisation problem. where the vector field (u, v) is presented as the minimiser for an energy function $J(u, v)$. this energy function is made up of the optical flow constraint based on brightness constancy, and another term

based on the gradient of the flow. The resulting energy function is as follows:

$$J(u,v) = \int_{\Omega} (I_x u + I_y v + I_t)^2 + \alpha^2(|\nabla u|^2 + |\nabla v|^2) \tag{2.4}$$

where α is the weight of the smoothness of the resulting flow field.

Horn and Schunck's optical flow algorithm was a huge step forward in computer vision research, although this algorithm is limited in that it typically can only estimate small motions, the method fails in the presence of large motion when the gradient of the image is not smooth enough (**?**).

2.1.2 Lucas - Kanade Optical Flow

Since its introduction in 1981, techniques pioneered by the Lucas-Kanade method have become some of the most widely used tools for in computer vision. The main idea behind the Lucas-Kanade method is image alignment, where a region in an image is translated and deformed to minimise the least squared difference between the region and the following frame (**?**). Furthermore, the Lucas-Kanade method operates under the assumption that the flow is effectively constant in a small cluster of pixels, the method can then solve the optical flow equation for all the pixels in that neighbourhood using least squares regression. The Lucas-Kanade method improved on previous methods by using the spatial intensity gradient to find the disparity vector h, reducing the algorithms calculation time to $O(M^2 log(N))$ for the average case (**?**).

The Lucas - Kanade Optical Flow algoithm is known as a sparce optical flow algorithm, calculating the pixel flow based on feature matching. The goal of Lucas-Kanade optical flow algorithm is to match a subset of pixels $\mathbf{T}(x)$ from an image at time t_0 to a subsequent template image $\mathbf{I}(x)$ at time t_1. The necessary movement of $\mathbf{T}(x)$ becomes the optical flow for that region.

Considering a frame $I(x, y)$, assuming only small motion occurs between the time step, The new image can be represented as $H(x, y) = I(x + \delta x, y + \delta y)$. Finally, by using a Taylor series expansion the

Lucas-Kanade equation can be written as:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{2.5}$$

where δx and δy is the flow in thier respective directions.

Similar to the horn and Schunk method in **??**, assumptions must be made to solve this equation. the two main assumptions made to constrain the equation are, the brightness constancy, mentioned in **??** and a *spacial coherance* constraint. The spacial coherance encapulates the idea that there are regions within the image where all the pixels are moving together, possibly coresponding to an object moving in the frame.

2.1.3 Farneback Optical flow

A more recent algorithm for calculationg optical flow is presented by Gunnar Farneback. This method is based on the quadratic polynomial expansion transform. The idea begind the Farneback algorithm is to model an image as a function by locally fitting a polynomial to the image at each pixel. The polynomial expansion allows a neighbourhood for each pixel to be aproximated **?**. The signal model used in the Farneback algorithm is given by:

$$f(\mathbf{x}) \approx \mathbf{x}^\top \mathbf{A}\mathbf{x} + \tilde{\mathbf{b}}^\top + c \tag{2.6}$$

where $\mathbf{x}$ is the local pixel coordinates, $\mathbf{A}$ is a symetric matrix b is a vector adn c a constant. A least squares fitting for the pixel nebourhood is then performed to find these coefficents. Due to the polynomial being based on a window surrounding each pixel, there is a weighting applied making the function more sensitive to pixels closer to the centre of the window. For this reason the choise of the neibourhood size is an important consideration as it determins the scale of the features that this algorithm is sensitive to. **??** shows the opperation of the Farneback algorithm as a one dimentional hystogram, where the height y at each point represents the pixel intensity as in the 2-dimensional case and.

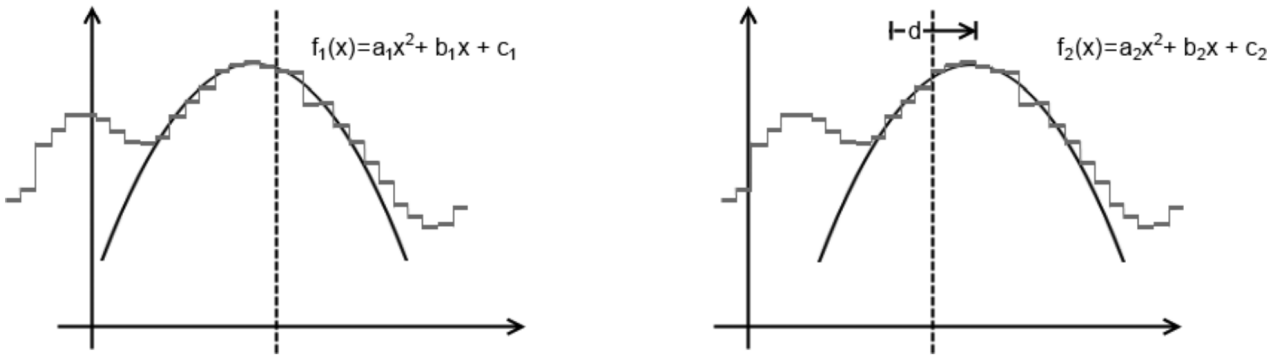


Figure 2: A one dimentional illustration of the Farneback algorithm. I_{t0} (left) shows the image before a displacement d and I_{t1} shows the image after. Image from (**?**)

The resulting quadratic polynomials from one timestep to the next, f_1 and f_2 in **??**, can be related through the equations:

$$\begin{aligned} a_1 &= a_2 \\ b_1 &= b_2 + 2a_1 \\ c_1 &= c_2 + b_1 d - a_1 d^2 \end{aligned} \tag{2.7}$$

The displacement d can then be solved for through:

$$d = -\frac{1}{2a_1}(b_2 - b_1) \tag{2.8}$$

There is a probelm with this aproach on its own, that is this method will only work for movement that can be captured in the neibourhood chosen for the polynomial. However this issue can be resolved if some information about the displacement is already known, for instance from a previous calculation of the algorithm. So rather than comparing two images at the same point, It is possible to compare the points based on an estimated displacement, and refine the results form there. Another method for dealing with large movements is a pyramydal aproach. This involves calculating the flow on a pair of lower resolution images where large movement apears small. This aproximation can then be refined by feeding the results forward to higher order pyramids (**?**).

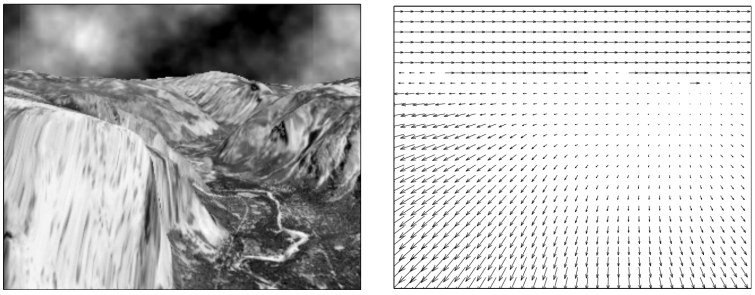


Figure 3: Results from the Farneback Optical flow algorithm run on the Yosemite optical flow evaluation sequence(**?**)

Technique	Average error	Standard deviation	Density
Lucas & Kanade [9]	2.80°	3.82°	35%
Uras *et al.* [10]	3.37°	3.37°	14.7%
Fleet & Jepson [11]	2.97°	5.76°	34.1%
Black & Anandan [12]	4.46°	4.21°	100%
Szeliski & Coughlan [13]	2.45°	3.05°	100%
Black & Jepson [14]	2.29°	2.25°	100%
Ju *et al.* [15]	2.16°	2.0°	100%
Karlholm [16]	2.06°	1.72°	100%
Lai & Vemuri [17]	1.99°	1.41°	100%
Bab-Hadiashar & Suter [18]	1.97°	1.96°	100%
Mémin & Pérez [19]	1.58°	1.21°	100%
Farnebäck, constant motion [1, 6]	1.94°	2.31°	100%
Farnebäck, affine motion [1, 6]	1.40°	2.57°	100%
Farnebäck, segmentation [2, 6]	1.14°	2.14°	100%
Constant motion, 1 iteration	3.94°	4.23°	100%
Constant motion, 3 iterations	2.60°	2.27°	100%
Affine motion, 1 iteration	4.19°	6.76°	100%
Affine motion, 3 iterations	2.08°	2.45°	100%

Figure 4: Farneback Algorithm compared with Lucas-Kanade, Horn & Schunk along with many other methods not discussed (**?**)

The Farneback algorithm was evaluated against a common optical flow evaluation dataset known as Lynn Quams Yosemite sequence. This dataset is a synthetic sequence of images with a known motion field, against which an algorithm can be evaluated. Furthermore, the dataset features discontinuity at oclution boundaries and depth variation. **??** and **??** show the results obtained by **?**.

2.2 Visual Navigation

For the majority of autonomous rovers the navigation system is the key component to successfully carrying out its task. The rover collects data about its environment from sensors on the platform,

attempting to extract meaningful information about its own state, such as its inclination and position relative to some reference. These sensors have typically included inertial measurment units (IMU), global positioning systems (GPS) and light detection and rainging (LiDAR) devices (**?**).

Visual navigation refers to the technique of navigationg an environment, primarily using input from a vision sensor such as a camera (**?**). Navigating an environemt sucessfully relys on the rover having an understanding of the relationship between its actions and the environment surrounding it. And it is this relationship that is so dificult to design into an algorithm (**?**). In recent years, the use of cameras on autonomous vehicle has drastically increased thanks to the increased availability of camera hardware to the average hobbiest. The advent of more accessable hardware means that much more attention has been places on the task of designing an algorithm to interprate the data form a robotic vision system.

2.2.1 Inverse Optimal Control

The traditional way of designing an autonomous system is to create a cost function for the system and use this cost function to produce an optimal policy. This method often involves tweaking tuning prametrs and weight values that are too complicated to understnd intuitively (**?**). This is known as optimal control. as the name suggests, inverse optimal control flips these techniques around. Inverse optimal control can be a far more intuitive method of obtaining the desired behavior from a robot.

Rather than defining a cost function that generates the policy for the robot, inverse optimal control uncovers the cost function that best explains demonstrated optimal behavior (**?**). This behavior is derived from from expert demonstrations of the optimal policy (**?**). In nature, there are many systems found that have all ready implemented a near optimal control stratergy for many visual tasks. The question that inverse optimal control answers is, how can we attempt to mimic the behavior seen nature for the purposes of our goals.

2.2.2 Biomimicry

Biology so often holds a wealth of inspiration for engineers. From the material science of the microstructures in a butterfly wing to the computational fluid dynamics of a Kingfisher diving into water (**?**). visual sensors are found so frequently in nature, making it very productive to look at nature and tap into the eons of knowledge distilled by evolution.

Due to the immense complexity of understanding and implementing a vision based navigation system, reaserchers have looked to honeybees for clues. Honeybees have been noticed to display superb agylity and remarkable compitency in navigating their environment (**?**). Using an inverse optimal control framework, the interpritation that the honeybee has of its environment was investigated. Reaserchers at the university of queensland were able to devise and setup an experiment (**??**) with the goal of deducing how a bee negotialtes its environment.

In **??** the honeybee startes at one end of a corridor lined with a grating pattern. The pattern along the corridor can be horisontally translated, simulating a slower or faster image speed. (**?**) found that when the grating patterns were staitonary, the honeybees fly through the middle of the corridor **??**D. Interestingly, this was irrispective of the spatial period of the pattern. Furthermore, it was found that when one side of the grating pattern was translated, the honeybee would hug the wall that coresponded to the lowest apparent angular image velocity, again irrispective of the spatial period of the pattern. The conclution that (**?**) found was that the honeybees are attempting to equalise the apparent angular motion of the images either side of it. Further suggestiong that bees can accurately compute the speed of the image in each eye, mostly independent from the texture of the image.

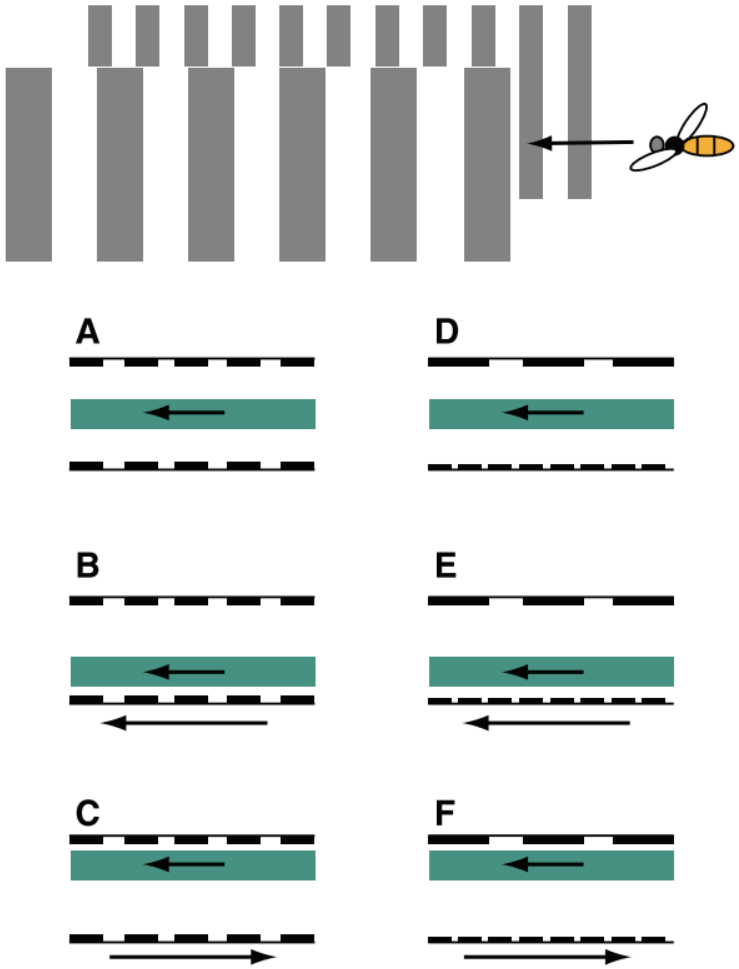


Figure 5: Inverse optimal control experiment used to identify the mechanism that bees use to maintain their distance to a corridor wall

2.2.3 Optical Flow for Visual Navigation

Determining motion and structure from visual cues is a major aspect of visual navigation. There have been many attempts to aquire such information from various artificial visual systems (**?**). Sterio vision is one such method usetd to compute depth information about the environment. This method uses a concept known as multiview geometry to reconstruct a depth map of the environemt surrounding the camera (**?**). Another way of visually determining the depth of an environment is by interpreting the blur of an image as depth iformation, this technique is known as depth from focus (**?**).

Like optical flow, these methos are commonly observed in nature. however, they are often seen in anamals with a highly developed visual cortex. This suggest that the comutational power required

to mimic these techniques is not ideal for real time depth map extraction on mobile and lightweight computer hardware. In **??** the use of optical flow by insects was discussed. These insects have been found not to have the visual cortex that mamals have (**?**), suggesting the the optical flow based visual navigation techniques insects have evolved to use must be comutationally inexpencive.

Determining motion from a plainar perspective projection of optic flow requires a dificult inversion of a non-linear transformation. A solution for this inversion ha been obtained by various authors, however these studies have relied on the fact that the motion field being used is accurate to withinn 1%. **?** presents the technique of projecting the percieved optic flow onto a view sphere. This method is able to produce the motion of the flow field unambiguously.

The method presented by **?** and further implemented by **?** uses a *derotation algorithm* to resolve the rotation on the view sphere. This algorithm uses a search based method to rotate the flow vectors looking for a solution that produces a distinct focus of expansion (FOE) and focus of contraction (FOC). The axis passing from the FOC to the FOE is the unambiguous direction of travel.

Another method outlined by **?** implements optical flow on the view sphere such that the translational and rotational velovities can be estimated. Extending on work of **?** and **?**, the method that **?** outlines utelises all of the flow vectors on the view sphere to imply the state of the camera.

2.2.4 Optical Flow for Obstical Avoidance

Object detection and avoidance is a major aspect of navigationg an environment, and utelising optical flow information to achieve this has proven to be an exceptionally dificult problem. There have been various attempts to devise a robust and accurate interpritation of the flow field that can detect and avoid obstacles.

One such obstacle avoidace method is presented by **?**, where a *ballance strategy* is implemented. The strategy uses the idea of balancing the amount of left and right side flow maintaining an equal relative distance from objects. This is dome by subdeviding the image into three vertical regions for which the flow can be balanced. **?** implements their method on a ground based rover, demostrating the rover is able to traverse its environment without collisions. The commanded control action for this strategy

is calculated as follows:

$$\Delta(F_L - F_R) = \frac{\sum \left\|\vec{W}_L\right\| - \sum \left\|\vec{W}_R\right\|}{\sum \left\|\vec{W}_L\right\| + \sum \left\|\vec{W}_R\right\|} \tag{2.9}$$

where $\Delta(F_L - F_R)$ is the commanded torques on the left and right side of the rover, and $\sum \left\|\vec{W}\right\|$ is the sum of the magnitudes of optic flow in the visual hemifields.

? presents an improved balance strategy implemented on a quad rotor drone. Unlike the ground based balance method, the improved method devides the image into 9 reigons coresopnding to the 6 DOF of the quadrotor. Furthermore, information of *time to contact* (TTC) is extracted from the optical flow. The actuation command for the improved balace strategy calculated as a combination of TTC and flow Ballance **??**.

$$Cmd = \alpha Cmd_{IBS} + \beta Cmd_{TTC} \tag{2.10}$$

where Cmd_{IBS} is the command calculated by the balance strategy, Cmd_{TTC} is the command calculated from TTC and α and β are tuning parameters intended to add a weight to the different control allocation.

According to the motion paralax theory, motion of an object far away will corespond to a smaller optical flow than that of a nearby object with the same abolute motion (**?**). This observation is extremely useful for object avoidance, as it suggests that true depth is unnesessary to simply avoid obstacles. the apparent motion of an image can essentially, reveals all of the objects that are of important to avoid. for instatnce, if there is a portion of a flow field that has high flow, this must correspond to an object that is either far away but converging fast on the rover, or it is an object that is close enough to collide with.

? proposes a method for a divergence-based control schemes to planar surfaces of arbitrary orientation that enables the design of a unified control law for landing and docking. The aproach utelises the concept of maximum divergance (max-div) on the view sphere to regulate the aproach velocity in a docking or landing manuver. The following theorem was proven in **?**.

Theorem 4.1 *Given an approach angle $\theta_{nt} \in [0, \frac{\pi}{2})$ toward a planar surface, the maximum divergence induced by the relative motion between an infinite planar surface and view sphere will always occur halfway along the shortest arc of the great circle connecting* **?**

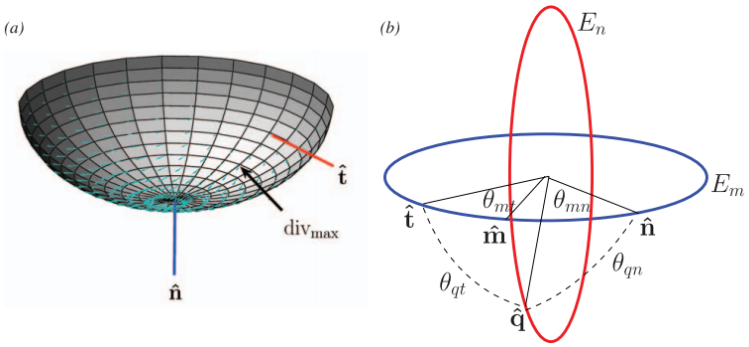


Figure 6: The maximum divergence property.
(a) shows the location of max-div on the view sphere, (b) is the framework illustrating **Theorem 4.1** image credits **?**

The experimental results in **?** confirm that divergance of the optical flow field is a valid method for landing and docking. This concept of divergence of optical flow can similarly be used for obstacle avoidance, by finding the reigon of least flow and following that direction.

2.3 Reference Frames and Coordinate Systems

In general, the motion of an object needs to be described relative to some coordinate system and the reference frame. Defined by **?**, a reference frame is a perspective from which the motion is described by an observer. A reference frame can be defined by a set of at least 3 non-colinear points that are rigidly connected. The reference frames used in this project are $\mathcal{C}$, $\mathcal{B}$ and $\mathcal{N}$ for the camera, the vehicle (body) and the world frames respectively.

Furthermore, a coordinate system is a way to describe the position and motion of objects relative to its reference frame. A coordinate system in three-dimensional Euclidean geometry is made up of at least three orthogonal basis vectors of unit length. This report uses the same notation as used in **?** to describe a vector from point O to point P as $\vec{r}_{P/O}$. **??** shows an example reference frame $\mathcal{C}$ attached to a camera with an associated coordinate system $\{c\}$.

When designing computational models, it is often convenient to represent vectors as column matrices

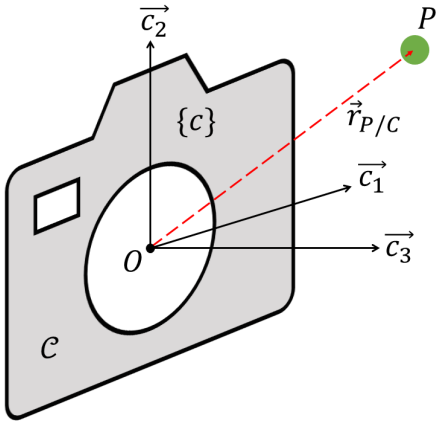


Figure 7: A reference frame $\mathcal{C}$ with its associated coordinate system $\{c\}$

in $\mathbb{R}^3$. The basis of the original vector is denoted as a superscript of the matrix. That is, a vector $\vec{v}$ in basis $\{a\}$ is written as $\mathbf{v}^a$, or more generally:

$$\mathbf{v}^a = \begin{bmatrix} v_1^a \\ v_2^a \\ v_3^a \end{bmatrix} \tag{2.11}$$

In this form, the dot product of two vectors $\vec{u} \cdot \vec{v}$ (inner product) in the same basis becomes a matrix operation:

$$(\mathbf{u}^a)^T \mathbf{v}^a = \begin{bmatrix} u_1^a & u_2^a & u_3^a \end{bmatrix} \begin{bmatrix} v_1^a \\ v_2^a \\ v_3^a \end{bmatrix} \tag{2.12}$$

Additionally, the cross product of two vectors $\vec{w} = \vec{u} \times \vec{v}$ can be represented as the *skew-symmetric* matrix of $\mathbf{u}^a$ multiplied by $\mathbf{v}^a$. That is:

$$\mathbf{w}^a = \mathbf{S}(\mathbf{u}^a)\mathbf{v}^a \tag{2.13}$$

The skew-symmetric operator $\mathbf{S}(\mathbf{u})$ arranges the elements in u such that the resulting matrix is skew-symmetric.

$$\mathbf{S}(\mathbf{u}) = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \tag{2.14}$$

2.3.1 Coordinate Transforms and Rotation Matrices

This project uses multiple coordinate systems to deal with the complex geometry of the camera and vehicle. Therefore, in order to have context for measurements and actions of different sensors and actuators, their orientation needs to be transformed into the relevant coordinate system. The following report will use rotation matrices, as derived from a coordinate transform.

A vector can be described from multiple bases, for instance, $\vec{r}$ can be represented in both basis $\{n\}$ and basis $\{c\}$ as:

$$\begin{aligned} \vec{r} &= r_1^n \vec{n}_1 + r_2^n \vec{n}_2 + r_3^n \vec{n}_3 \\ \vec{r} &= r_1^c \vec{c}_1 + r_2^c \vec{c}_2 + r_3^c \vec{c}_3 \end{aligned} \tag{2.15}$$

Expressing $\vec{r}$ as an inner product of the basis $\{c\}$ and r^n:

$$\vec{r} = \begin{bmatrix} \vec{c}_1 & \vec{c}_2 & \vec{c}_3 \end{bmatrix} \begin{bmatrix} r_1^c \\ r_2^c \\ r_3^c \end{bmatrix} \tag{2.16}$$

Finally, express both sides in $\{n\}$:

$$\mathbf{r}^n = \begin{bmatrix} \mathbf{b}_1^c & \mathbf{b}_2^c & \mathbf{b}_3^c \end{bmatrix} \mathbf{u}^c \tag{2.17}$$

This can be denoted more generally in matrix form:

$$\mathbf{r}^c = \mathbf{R}_n^c \mathbf{r}^n. \tag{2.18}$$

In robotics, it is often desirable to rotate a vector to a particular angle. The method used in this project is known as *euler rotation*. Euler angles discretise a complex three-dimensional rotation into

three separate planar rotations about each basis vector. The rotation matrices about each basis vector are given by:

$$\begin{aligned}
\mathbf{R}_x(\phi) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \\
\mathbf{R}_y(\theta) &= \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \\
\mathbf{R}_z(\psi) &= \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{2.19}$$

The order of these rotations effects the final position, for this reason in this project the common robotics convention of *Roll, Pitch, Yaw* (RPY) will be used. The notation for a rotation matrix from $\{n\}$ to $\{c\}$ is given by $\mathbf{\Theta}_c^n \triangleq \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^\top$. using the consecutive planar rotations from **??** we get $r_a^b = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)r_b^a = \mathbf{R}(\mathbf{\Theta}_c^n)r_b^a$. From this, the structure of a full rotation matrix:

$$\mathbf{R}(\mathbf{\Theta}_c^n) = \begin{bmatrix} c_\psi c_\theta & s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi c_\phi s_\theta \\ s_\psi c_\theta & c_\psi c_\phi + s_\phi s_\theta s_\psi & c_\psi s_\phi + s_\psi c_\phi s_\theta \\ s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{2.20}$$

where $s \triangleq sin(\cdot)$ and $c \triangleq cos(\cdot)$.

Due to the orthogonality of rotation matrices, they have the property that the inverse of the rotation matrix is equal to its transpose:

$$\mathbf{R}^{-1} = \mathbf{R} \tag{2.21}$$

This is useful, as rotating a vector back to its original orientation is as simple as applying the transposed rotation matrix. That is $\mathbf{R}_a^b = \mathbf{R}_b^{a\top}$.

2.3.2 Time Derivative of Vectors and Transport Theorem

The subtle difference between the time-derivative of a scalar magnitude and a vector magnitude is that the vector derivative depends on the reference frame from which they are being observed **?**. **??** shows a vector $\vec{r}$ observed from two separate reference frames, $\mathcal{A}$ and $\mathcal{B}$.

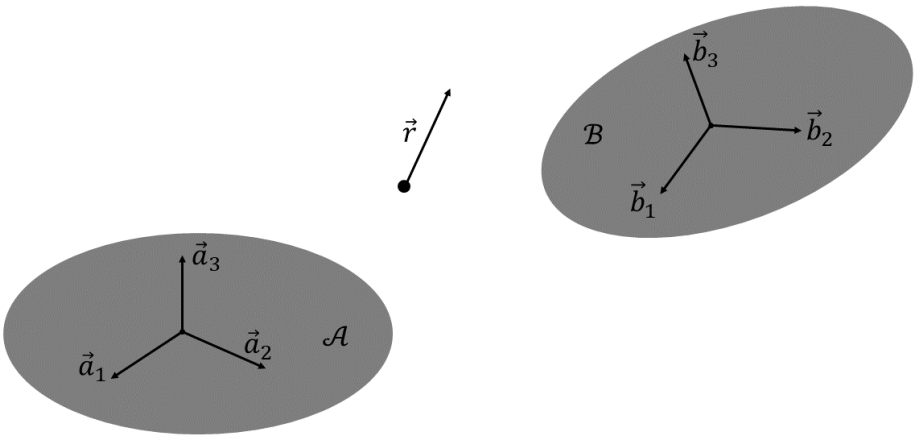


Figure 8: A vector $\vec{r}$ seen from two reference frames.

Taking the time derivative of $\vec{r}$ in each vector frame:

$$\begin{aligned} \frac{^{\mathcal{A}}d\vec{r}}{dt} &= \frac{dr_1^a}{dt}\vec{a}_1 + \frac{dr_2^a}{dt}\vec{a}_2 + \frac{dr_3^a}{dt}\vec{a}_3 \\ \frac{^{\mathcal{B}}d\vec{r}}{dt} &= \frac{dr_1^b}{dt}\vec{b}_1 + \frac{dr_2^b}{dt}\vec{b}_2 + \frac{dr_3^b}{dt}\vec{b}_3 \end{aligned} \tag{2.22}$$

Consider the case of a frisbee flying through the air carrying a spider S as a passenger. Reference frame $\mathcal{F}$ is attached rigidly to the frisbee and an observer O is watching it from the field in reference frame $\mathcal{O}$. Clearly, the movement of the spider is different from $\mathcal{A}$ as it is from $\mathcal{F}$ or more generally:

$$\frac{^{\mathcal{O}}d\vec{r}}{dt} \neq \frac{^{\mathcal{F}}d\vec{r}}{dt} \tag{2.23}$$

To find the time derivative of the spider relative to $\mathcal{O}$, two methods could be used: the time derivative

of the vector $\vec{r}_{S/O}$ can be taken, which will prove to be a tedious exercise. Or the *transport theorem* can be used.

In accordance with **?**, the transport theorem states that there exists a unique vector $\vec{\omega}_{\mathcal{B}/\mathcal{A}}$ called the **angular velocity** of $\mathcal{B}$ with respect to $\mathcal{A}$ such that:

$$\frac{^{\mathcal{A}}d\vec{r}}{dt} = \frac{^{\mathcal{B}}d\vec{r}}{dt} + \vec{\omega}_{\mathcal{B}/\mathcal{A}} \times \vec{r} \tag{2.24}$$

Using the transport theorem, the velocity of the spider relative to $\mathcal{O}$ is simply the velocity of the frisbee's centre P plus the rotational velocity of the frisbee, resulting in:

$$\frac{^{\mathcal{O}}d}{dt}\vec{r}_{S/O} = \frac{^{\mathcal{F}}d}{dt}\vec{r}_{P/O} + \vec{\omega}_{\mathcal{F}/\mathcal{O}} \times \vec{r}_{P/O} \tag{2.25}$$

2.4 Kinematics of Vision

While discussing the motion of objects in a space, it is helpful to properly define a geometric definition for these points and the relative motion between them. Kinematics defines these such relationships. More specifically, kinematics defines the reference frame from which the motion of a point can be described and the transformation that associates this motion in a separate reference frame. These concepts will be used to describe the relative motion between a camera mounted on a vehicle in an environment.

2.4.1 Optical Flow on the View Sphere

In **??**, the concept of projecting optical flow onto a view sphere in order to better obtain the translational and rotational velocities of a camera in motion was introduced. This is the concept that was used to calculate the artificial optical flow in simulation. The following derivation is addapted from **?** and **?**

Assuming the viewsphere has a unit radius, the point P can be projected onto the view sphere to point S by:

$$\vec{r}_{S/C} = \frac{\vec{r}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} \tag{2.26}$$

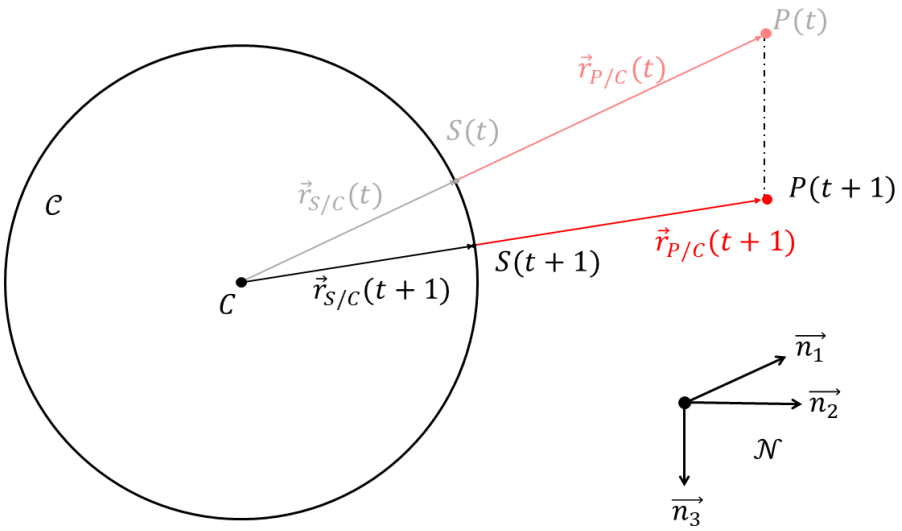


Figure 9: Projection of a point P from a camera C onto a unit sphere S and its movement through time. adapted from **?**

Applying the concepts from **??**, the time derivitive of $\vec{r}_{S/C}$ with respect to reference frame $\mathcal{N}$ can be taken:

$$\frac{^{\mathcal{N}}d\vec{r}_{S/C}}{dt} = \frac{^{\mathcal{N}}d}{dt}\left(\frac{\vec{r}_{P/C}}{\left\|\vec{r}_{P/C}\right\|}\right) \tag{2.27}$$

Noticing $\left\|\vec{r}_{P/C}\right\|$ can be written as $(\vec{r}_{P/C}\cdot\vec{r}_{P/C})^{-\frac{1}{2}}$ and substituting this in:

$$\frac{^{\mathcal{N}}d\vec{r}_{S/C}}{dt} = \frac{^{\mathcal{N}}d}{dt}\vec{r}_{P/C}(\vec{r}_{P/C}\cdot\vec{r}_{P/C})^{-\frac{1}{2}} \tag{2.28}$$

Now that the equation is in the form of $f(x)g(y)$ the product rule can be aplied and vector derivitives written as velocities to simplify, giving:

$$\begin{aligned}\frac{^{\mathcal{N}}d\vec{r}_{S/C}}{dt} &= \frac{^{\mathcal{N}}d}{dt}\left(\vec{r}_{P/C}\right)\left(\vec{r}_{P/C}\cdot\vec{r}_{P/C}\right)^{-\frac{1}{2}} + \vec{r}_{P/C}\frac{^{\mathcal{N}}d}{dt}(\vec{r}_{P/C}\cdot\vec{r}_{P/C})^{-\frac{1}{2}} \\ \vec{v}_{S/C} &= \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} + \vec{r}_{P/C}\frac{^{\mathcal{N}}d}{dt}(\vec{r}_{P/C}\cdot\vec{r}_{P/C})^{-\frac{1}{2}}\end{aligned} \tag{2.29}$$

Following this, the chain rule can be applied to the second term of the equation:

$$\begin{aligned}\vec{v}_{S/C} &= \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} - \frac{1}{2}\vec{r}_{P/C}(\vec{r}_{P/C}\cdot\vec{r}_{P/C})^{-\frac{3}{2}}\frac{^{\mathcal{N}}d}{dt}(\vec{r}_{P/C}\cdot\vec{r}_{P/C}) \\ &= \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} - \frac{1}{2}\vec{r}_{P/C}(\vec{r}_{P/C}\cdot\vec{r}_{P/C})^{-\frac{3}{2}}\left(2\vec{r}_{P/C}\frac{^{\mathcal{N}}\vec{r}_{P/C}}{dt}\right)\end{aligned} \tag{2.30}$$

Again, the vector norm can be substituted back into the equation and terms collected

$$\vec{v}_{S/C} = \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} - \frac{\vec{r}_{P/C}(\vec{r}_{P/C}\vec{v}_{P/C})}{\left\|\vec{r}_{P/C}\right\|^3} \tag{2.31}$$

Rearranging **??** to get $\vec{r}_{P/C} = \vec{r}_{S/C}\left\|\vec{r}_{P/C}\right\|$ this equation can be substituted in, and like terms canceled:

$$\begin{aligned}&= \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} - \frac{\vec{r}_{S/C}\left\|\vec{r}_{P/C}\right\|(\vec{r}_{S/C}\left\|\vec{r}_{P/C}\right\|\vec{v}_{P/C})}{\left\|\vec{r}_{P/C}\right\|^3} \\ \vec{r}_{S/C} &= \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} - \frac{\vec{r}_{S/C}(\vec{r}_{S/C}\vec{v}_{P/C})}{\left\|\vec{r}_{P/C}\right\|}\end{aligned} \tag{2.32}$$

As previously mentioned these calculations were with respect to the world reference frame $\mathcal{N}$. However, spherical flow is measured in the camera reference frame $\mathcal{C}$. Therefore we need to apply the transport theorem outlined in **??** obtaining the velocity of a point S as seen from $\mathcal{C}$

$$\begin{aligned}\frac{^{\mathcal{C}}d\vec{r}_{S/C}}{dt} &= \frac{^{\mathcal{N}}d\vec{r}_{S/C}}{dt} + \vec{\omega}_{\mathcal{C}/\mathcal{N}}\times\vec{r}_{S/C} \\ \frac{^{\mathcal{C}}d\vec{r}_{S/C}}{dt} &= \frac{^{\mathcal{N}}d\vec{r}_{S/C}}{dt} - \vec{\omega}_{\mathcal{N}/\mathcal{C}}\times\vec{r}_{S/C} \\ \frac{^{\mathcal{C}}d\vec{r}_{S/C}}{dt} &= \frac{\vec{v}_{P/C}}{\left\|\vec{r}_{P/C}\right\|} - \frac{\vec{r}_{S/C}(\vec{r}_{S/C}\vec{v}_{P/C})}{\left\|\vec{r}_{P/C}\right\|} + \vec{r}_{S/C}\times\vec{\omega}_{\mathcal{C}/\mathcal{N}}\end{aligned} \tag{2.33}$$

Once the transport theorem has been applied to $\frac{^{\mathcal{C}}d\vec{r}_{S/C}}{dt}$ It can no longer be thoiught of as a velocity. This is because it is now a rate of change of angel i.e. its unity are *rads/s*. the equation can be written more clearly as:

$$^{\mathcal{C}}\dot{\vec{r}}_{S/C} = \frac{\vec{v}_{P/C} - \vec{r}_{S/C}(\vec{r}_{S/C}\vec{v}_{P/C})}{\left\|\vec{r}_{P/C}\right\|} + \vec{r}_{S/C}\times\vec{\omega}_{\mathcal{C}/\mathcal{N}} \tag{2.34}$$

The numerator of **??** can be rearranged into a form such that the triple product can be aplied:

$$\begin{aligned}\vec{v}_{P/C} - \vec{r}_{S/C}(\vec{r}_{S/C}\vec{v}_{P/C}) =& (\vec{r}_{S/C} \cdot \vec{v}_{P/C})\vec{v}_{P/C} - (\vec{r}_{S/C} \cdot \vec{v}_{P/C})\vec{r}_{S/C} \\ &= \vec{r}_{S/C} \times \vec{v}_{P/C} \times \vec{r}_{S/C}\end{aligned} \tag{2.35}$$

?? can be further rearranged by realising that $\vec{v}_{P/C} = -\vec{v}_{C/N}$ to get:

$$^{\mathcal{C}}\dot{\vec{r}}_{S/C} = \frac{\vec{r}_{S/C} \times \vec{r}_{S/C} \times \vec{v}_{C/N}}{\left\|\vec{r}_{P/C}\right\|} + \vec{r}_{S/C} \times \vec{\omega}_{\mathcal{C}/\mathcal{N}} \tag{2.36}$$

In an open environment, the distance, or depth to an object, captured by $\left\|\vec{r}_{P/C}\right\|$ can often be ∞ (think, looking at the sky). This causes the problem of deviding by infinity. For this reason it is convenient to use inverse depth.

$$\rho_{P/C} \triangleq \frac{1}{\left\|\vec{r}_{P/C}\right\|} \tag{2.37}$$

Substituting **??** into **??**.

$$^{\mathcal{C}}\dot{\vec{r}}_{S/C} = \rho_{P/C}\vec{r}_{S/C} \times \vec{r}_{S/C} \times \vec{v}_{C/N} + \vec{r}_{S/C} \times \vec{\omega}_{\mathcal{C}/\mathcal{N}} \tag{2.38}$$

As mentioned in **??**, it is helpfull when creating a comutational model to use the skew symetric matrix to compute the cross product. To do this, **??** is expressed in camera coordinates $\{c\}$ and the skew symetric matrix is applied to all cross products.

$$^{\mathcal{C}}\dot{\mathbf{r}}^c_{S/C} = \rho_{P/C}\mathbf{S}(\mathbf{r}^c_{S/C})\mathbf{S}(\mathbf{r}^c_{S/C})\mathbf{v}^c_{C/N} + \mathbf{S}(\mathbf{r}^c_{S/C})\omega^c_{\mathcal{C}/\mathcal{N}} \tag{2.39}$$

?? can finally be rearranged into matrix form to obtain a mathmatical model for the optical flow on the view shpere.

$$^{\mathcal{C}}\dot{\mathbf{r}}^{c}_{S/C} = \mathbf{S}(\mathbf{r}^{c}_{S}/C)\begin{bmatrix}\rho_{P/C}\mathbf{S}(\mathbf{r}^{c}_{S/C}) & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{v}^{c}_{C/N}\\ \omega^{c}_{C/N}\end{bmatrix} \tag{2.40}$$

?? shows the resulting spherical flow of a camera moving in each degree of freedom.

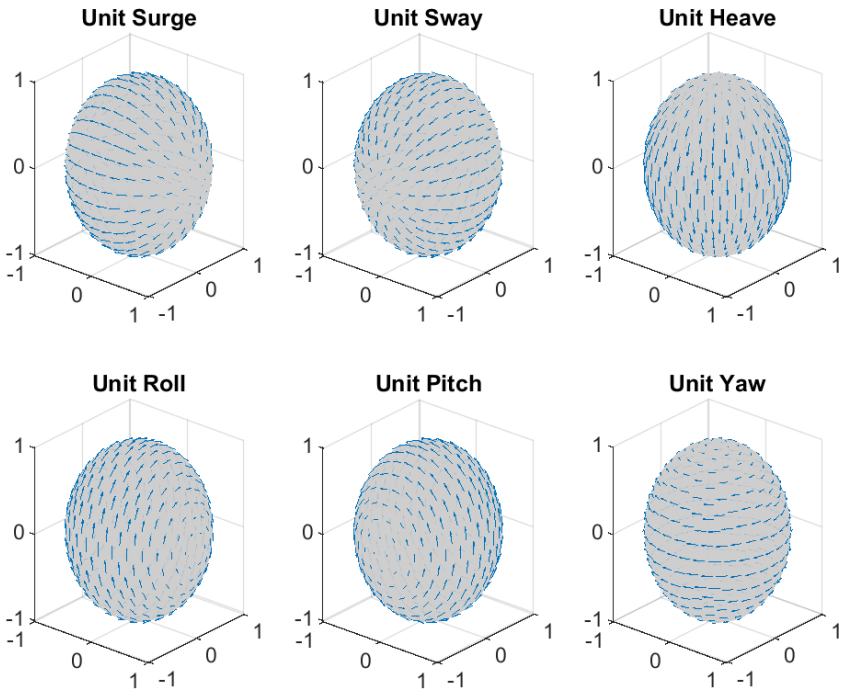


Figure 10: Optical flow calulated from; Top row: A unit translation in x, y and z. Bottom row: a unit rotation about x, y and z

2.5 Camera Calibration

Computer vision begins with the camera detecting light from a scene. This light has travelled from its source, reflecting from an object, through the camera lens, and is recorded on the image sensor. For these rays of light that make up an image to be useful for representing the real-world, the geometry that transforms these rays from the three-dimensional real world to a two-dimensional image plane must be understood; i.e. the transformation from world coordinates to image coordinates.

The camera calibration process describes the camera with two sets of parameters, namely the intrinsic parameters, and the extrinsic parameters. The intrinsic parameters describe the geometric relationship between the camera model and its projection. And the extrinsic parameters describe the location and orientation or pose, of the camera with respect to the world frame.

This project uses a GoPro Hero 7 to test and verify the calibration routine. This section will cover camera models and outline why a planar model is used for the GoPro. It covers the mathematical tools used to capture the different effects that make our camera imperfect for representing the real world, and how these deviations can be correrected.

2.5.1 Camera Models

The most elementary way of representing a camera is with the pinhole model. Light enters the aperture (pinhole) after reflecting from objects in a scene. For an ideal pinhole camera, only one ray enters the aperture from any point in the scene. The light is then projected onto a surface behind the aperture resulting in an image of the scene that is scaled proportionally to the focal length f.

?? shows an ideal pinhole camera model and its parameters, where f is the focal length, Z is the distance along the optical axis to an object, x is the size of the projected object and X is the size of the object. For this model, (**??**) is the relationship between the projected image and the object

$$\frac{-x}{f} = \frac{X}{Z} \tag{2.41}$$

Due to how light enters the camera, the image projection from a pinhole model is projected upside down. This issue is fixed in a planar camera model. The planar model is similar to a pinhole model,

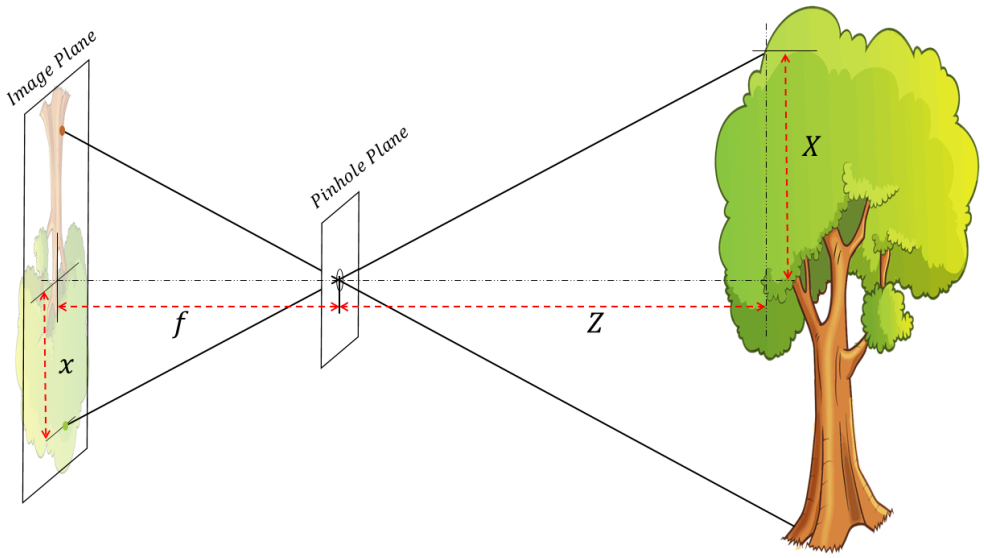


Figure 11: Elementary pinhole model of a camera: In practice, the appeture must be big enough to let light in, allowing a narrow ray of light from each point in the scene blurring the image. Therefore there is a tradeoff between brightness and sharpness.

however rather than passing through an aperture, light travels from each point in the scene, intersects the image plane and converges to a single point known as the *centre of projection* **??**. The Plainar Model better represents the geometry of a camera with a lens, albeit a camera with an impossibly perfect lens.

C	:	Centre of projection
$\{c\}$	:	Camera co-ordinates
P	:	Point in world space
P'	:	Point on image plane
$\vec{u}, \vec{v}$	:	Image co-ordinates
f	:	Focal length
I	:	Image plane

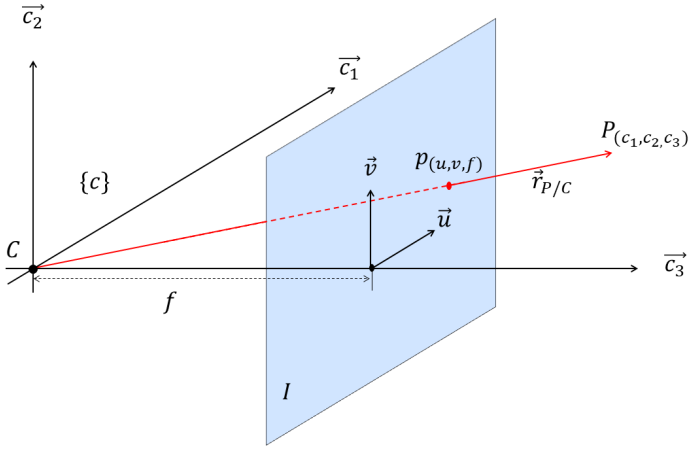


Figure 12: Ideal camera model. A point P in camera coordinates (c_1, c_2, c_3) is projected and displayed as a point p in image co-ordinates (u, v)

The idea behind modelling and calibrating a camera is to identify and cancel out imperfections caused by slight defects in the manufacturing process. Firstly, the image sensor is often mounted out of line

from the optical centre of the lens. Therefore, the parameters u_0 and v_0 are introduced to model this displacement from the optical centre. Furthermore, each pixel is not necessarily square, causing a small discrepancy between the focal length in the u and v directions. These discrepancy are captured in the f_u and f_v parameters. The following **??** project a point P_{c_1,c_2,c_3} to image sensor co-ordinates (u, v):

$$\begin{aligned} u =& f_u \times \frac{c_1}{c_3} + u_0 \\ v =& f_v \times \frac{c_2}{c_3} + v_0 \end{aligned} \tag{2.42}$$

2.5.2 Projective Geometry

Projective geometry is a tool describing the relationship that transforms a point from world coordinates (n_1, n_2, n_3) into image coordinates (u, v). When applying such transformations, it is convenient to add an extra dimension to each vector in the transform. for instance, a point that lies in a n dimensional space is represented as a $n + 1$ dimensional vector. this coordinate system is known as *homogeneous coordinates*. for example, a point lying on a plane at (x, y) is represented by the homogeneous coordinates (x, y, z). It is important to note that in homogeneous coordinates, any two vectors that share a common factor represents the same point in Euclidean geometry. This has the effect that a point in Euclidean geometry can be represented by an infinite number of homogeneous vectors. for instance $\alpha * (q_1, q_2, q_3) \equiv (q_1, q_2, q_3)$. The original primitive point is recovered by dividing through by q_3.

the utility of homogenious coordinates becomes clear when common transformations such as translation, rotation and scailing are attempted in . Homogenious coordinates allow these transformations to be implemented with matrix opperations. This is how projective geometry allows the camera intrinsic parameters f_u, f_v, u_0, v_0 to be arranges into a single 3×3 *camera intrinsic matrix* matrix. Bringing all of thees concepts together, the **??** can be implemented as a single martix opperation of the form:

$$p = \mathbf{M}\mathbf{r}^C_{P/C} \tag{2.43}$$

where:

$$P = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, M = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{r}^C_{P/C} = \begin{bmatrix} c^c_1 \\ c^c_2 \\ c^c_3 \end{bmatrix} \tag{2.44}$$

Some older camera manufacturing processes produce nonrectangular image sensors, i.e. they produced parallelogram shaped sensors. To account for this, a final *skew* parameter was included. Most modern cameras are produced with close to no skew therefor the error that this effect causes is below the noise floor and assumed to be 0.

2.5.3 Lens Distortion

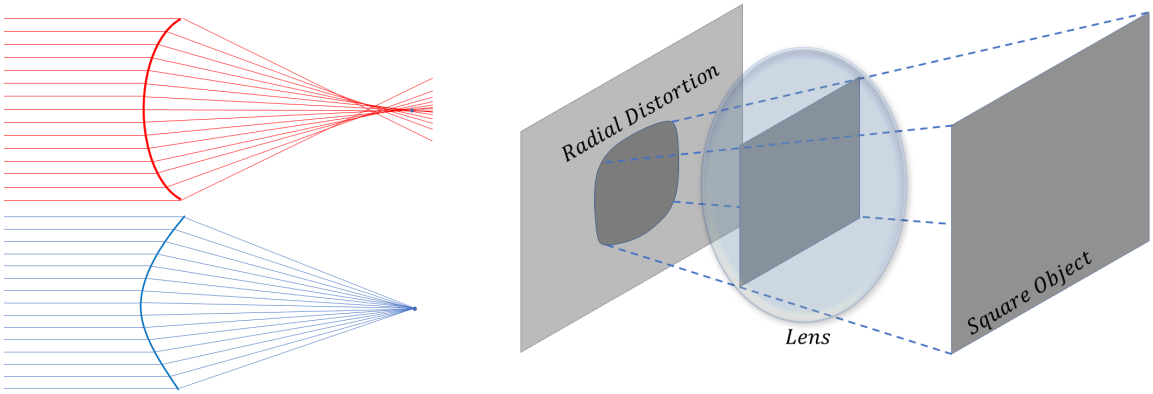
The optics inside a camera are yet another source of error in an image. Similar to correcting for the cameras geometry (captured in **??**), lens distortion must be accounted for. The leading method for correcting Lens distortion uses a *Brown-Conrady model* from **?**. This model characterises the two most prominent sources of distortion, naimley *radianl distortion* and *tangental distortion* and undistorts the image pixel locations.

In an ideal camera, the lense would take the shape of a perfect paraboloid. However it is much easier and cheeper to produce a sperical lens. The spherical shape of the lens tends to "over bend" rays entering farther from the optical center producing the effect known as *Radial Distortion* **??**. Typically a taylor series is used to characterise the distortion caused by the lens **?**. This polynomial takes the form:

$$f(r) = a_0 + a_1 r + a_2 r + a_3 r \dots \tag{2.45}$$

where $r = \sqrt{u^2 + v^2}$ is the radial distance from the optical centre.

However, for radial distortions, at the optical centre of the lense, deviation cause to the light rays is minimal i.e. $f(r) = 0$ at $r = 0$. This further implies that $a_0 = 0$. Additionally, becaues the distortion must be symetrical in r, the coefficents of the odd powers of r equal 0. This means that only the parameters needed for radial distortion correction are the coefficents of even powers of r **?**. Typically, no more than three radial distortion parameters (k_1, k_2, k_3) are needed. However, for high *field of view* cameras, upto K_6 may be necessary. Once these parameters are found, each pixel will be rescaled



(a) Red - "spherical" lens
Blue - paraboloidal lens

(b) Radial distortion is most clear on square objects

Figure 13: Radial distortion also known as *Barrel Distortion*. The farther from the centre a ray enters, the larger the barrel distortion bends it.

according to the following equations:

$$\begin{aligned} u' &= u \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ v' &= v \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.46}$$

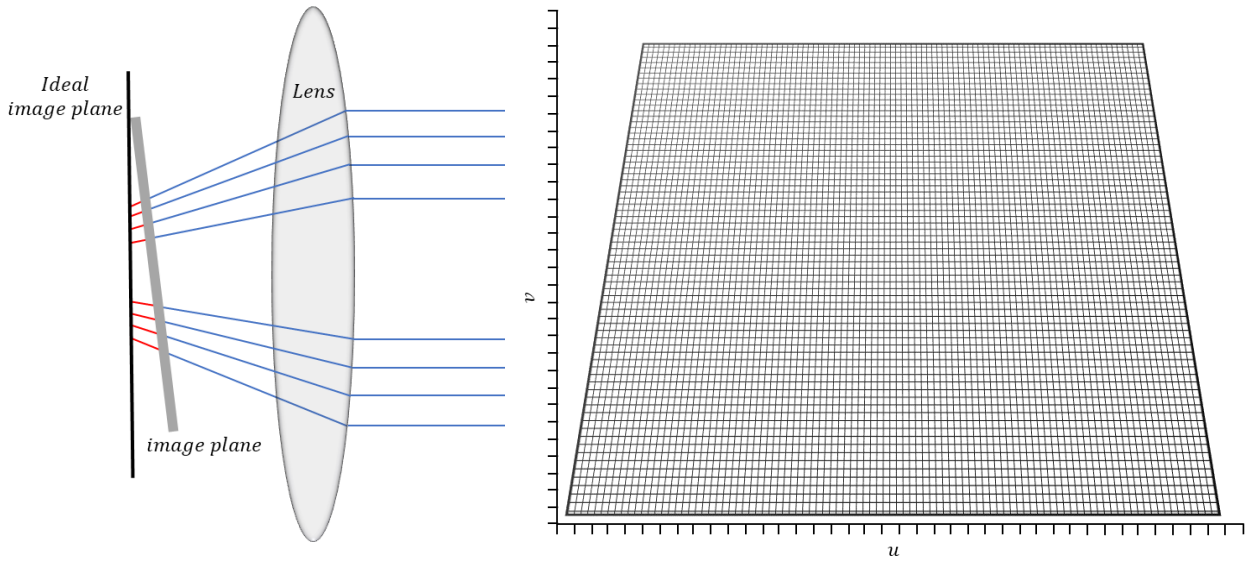
where u' and v' are the undistorted pixel coordinates.

Not only is it expensive and dificult to produce a lens that perfectly trains rays of light onto the *centre of projection* of a camera, it is also dificult to mechanically mount the image sensor to the camera body such that it is parallel to the lense.

Two additional parameters p_1 and p_2 are introduced to minimally characterise this tangental distortion such that:

$$\begin{aligned} u' &= u + [2p_1 uv + p_2(r^2 + 2u^2)] \\ v' &= v + [p1(r^2 + 2v^2) + 2p_2 uv] \end{aligned} \tag{2.47}$$

There are many other aspects within an average camera that cause image distortion, howerver these effects contribute a minor amount to the accuracy of measurements from an image so for this project



(a) Ray diagram of tangental distortion

(b) Tangental distortion of a flat grid

Figure 14: Extream example of tangental distortion.

will be assumed 0. Both Matlab and Open CV based camera calibration rotines are implemented and evaluated in this project. The camera calibration routine for Matlab *estimateCameraParameters();* returns an object containing the camera matrix **??**, radial distortion parameters **??** and tangental distortion parameters **??**, along with information about the calibration photos. When using *Open CV*, these parametetrs are packaged into a *distortion vector*. For this project, six radial distortion parameters were used to capture the distortion for a Go Pro Hero 7. Therefor the distortion vector from Open CV is organised as $(k_1, k_2, p_2, p_1, k_3, k_4, k_5, k_6)$. From **??** and **??**, the *Brown-Conrady model* is as follows:

$$\begin{aligned} u' &= u\left(\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{k_4 r^2 + k_5 r^4 + k_6 r^6}\right) + 2p_1 uv + p_2(r^2 + 2u^2) \\ v' &= v\left(\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{k_4 r^2 + k_5 r^4 + k_6 r^6}\right) + p_1(r^2 + 2v^2) + 2p_2 uv \end{aligned} \tag{2.48}$$

Now that the camera can be described mathematically through the intrinsic parameters and distortion parameters, pixel location in the image can be *undistorted*. **??** shows the movement of a subset of pixels from a GoPro Hero 7 image. Notice the movement of pixels in the centre of the image move far less than those further away radially.

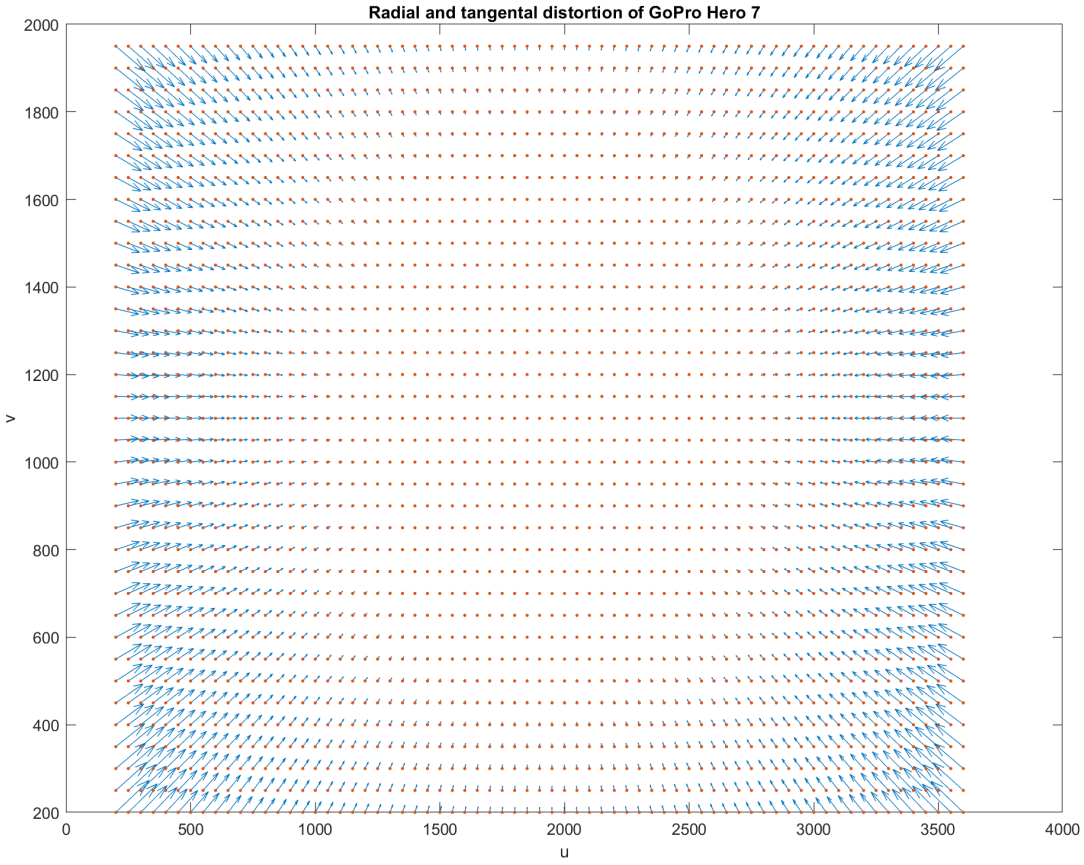


Figure 15: Shows the movement of a sample of pixels based on the parameters of a GoPro Hero 7

3 Obstacle Avoidance Cost Function

The obstacle avoidance algorithm presented in this report is largely based on the divergence of the optical flow field in an image. where the rover attempts to move in the direction of least flow. However, for the rover to be able to complete simple navigational tasks a target directon must be defined, and conciderend in the estimated optimal direction.

A cost funtion that encases both a target vector $\vec{r}_t$ and avoidance vector $\vec{r}_a$ is proposed as follows:

$$\begin{aligned} J(x) &= \alpha D_{flow}(x) + \beta T(x) \\ x* &= arg\min_{(x)} J(x) \end{aligned} \tag{3.1}$$

where $x*$ is a unit vector from camera centre to the optimal direction, D_{flow} is the divergence of the optical flow vector field, T is the cost of deviating from a target vector ($\vec{r}_t$) and α and β are weighting parameters.

The cost T for deviating from the target vector $\vec{r}_t$ was simply calculated by taking the vector dot product from each of the spherical flow direction vectors $\vec{r}_{S/P}$. Furthermore, the the resulting cost function is show in **??**.

Figure 16: Caption

4 Simulation in Unity

Unity 3D game engine was used to create the simulation for collecting the data required to develop and test an obstacle avoidance algorithm. This game engine was a convenient way to create an environment that could obtain a ground truth dataset and real world data could be evaluated against.

In order to utelise the Unity engine as a simulation platform, the behavior of the rover needed to be defined. This was done by implementing basic aerodynamics into the movement of the rover. **??** shows the equations that dictated the rovers movement about the environment.

Listing 1: A simple aerodynamic lift caculation that aproximates the movement of the rover

```
private void calculateLift ()
{
        Vector3 vel = rb.velocity;
        Vector3 localVel = transform.InverseTransformDirection(vel);
        Vector3 dragDirection   = -rb.velocity.normalized;
        Vector3 thrustDirection = transform.forward;
        Vector3 liftDirection   = Vector3.Cross(-dragDirection, transform.right);

        float angleOfAttack   = Mathf.Atan2(-localVel.y, localVel.z);
        float liftCoeff = lc0 + 2*Mathf.PI*angleOfAttack;
        float drag = (0.5f) * Mathf.Abs(roh) * Mathf.Abs(dragCoeff) * vel.sqrMagnitude;
        float lift = (0.5f) * Mathf.Abs(roh) * liftCoeff * vel.sqrMagnitude;

        rb.AddForce(thrustDirection * thrust);
        rb.addforce(dragdirection * drag);
        rb.AddForce(liftDirection * lift);
}
```

Furthermore, control actions were also calculated allowing the rover to be interacted with. For testing purposes these control actions were tied to the **W**, **A**, **S**, **D** keys, allowing the tester to apply control torques to the rover during simulation. The function that applies these torques can be seen in **??**

By utelising the in built physics engine of unity, velocity of the rover was able to be accurately and efficiently extracted from the scene. Both the lift Calculationd and the data was recorded within the **FixedUpdate()** function. This function is built in to unity, and allows processes to be run at exatly 30 frames per second. This is the function that accomodates all of the force calculations, in order to maintain a fixed timestep between adding forces to the sumulated rover.

Depth information from the simulated environment is extracted using raycasts. Typically used for hit

Listing 2: Adds the demanded control torques to the simulated aircraft

```
void calculateControlTorque() {
        Vector3 vel = rb.velocity;
        Vector3 localVel = transform.InverseTransformDirection(vel);

        float elevatorTorque = (elevatorD - elevatorU);
        float aileronTorque = (AileronL - AileronR);
        float rudderTorque = (rudderR - rudderL);

        rb.AddTorque(transform.right * elevatorTorque);
        rb.AddTorque(transform.forward * aileronTorque);
        rb.AddTorque(transform.up * rudderTorque);
}
```

detection in shooter games, the **Physics.Raycast()** function projects a ray into the scene in a defined direction and intersects an object in the scene. Depth can then be recovered from the origen of the ray to the object it has intersected with. The raycast functionality of the Unity engine was utelised by first defining a set of direction vectors along which the depth of the environment could be calculated. These direction vectors were created by itterating through a double for loop that calculated discretised euler angels of a sphere. The calculatins for the raycast direction vectors are seen in **??**

Listing 3: Direction Vectors that define the euler angles for each raycast

```
for (int ii = 1; ii < rX; ii++)
{
        for (int jj = 1; jj < rY; jj++)
        {
                float theta = jj * ((fovY) / (rY)) - fovY/2;
                float phi   = ii * ((fovX) / (rX)) + (Mathf.PI / 2 - fovX / 2);

                dir[ii, jj].x = Mathf.Cos(theta) * Mathf.Cos(phi);
                dir[ii, jj].z = Mathf.Cos(theta) * Mathf.Sin(phi);
                dir[ii, jj].y = -Mathf.Sin(theta);
        }
}
```

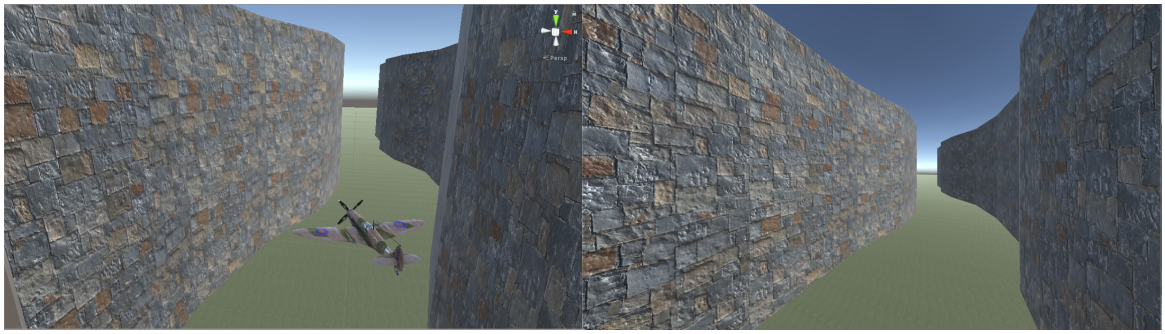
Where **fovX** and **fovY** define the x, y field of view and where **rX** and **rY** define x, y resoulution for the simulated camera.

A data stream to a text file is run during the simulation. The data that was collected from each simulation included;

- Vechical Pose
- Vehical Velocity
- **Inverse** depth of objects (Map)

4.1 Unty Scenes

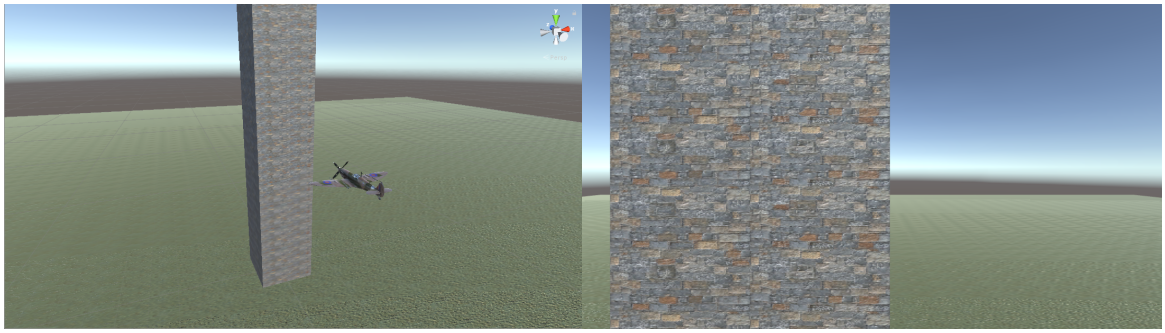
The Unity game engine has the ability to create multiple scenes In order to evaluate the divergence base obstacle avoidance algorithm, multiple senarios were created within the Unity environment.



(a) Red - "spherical" lens
Blue - paraboloidal lens

(b) Radial distortion is most clear on square objects

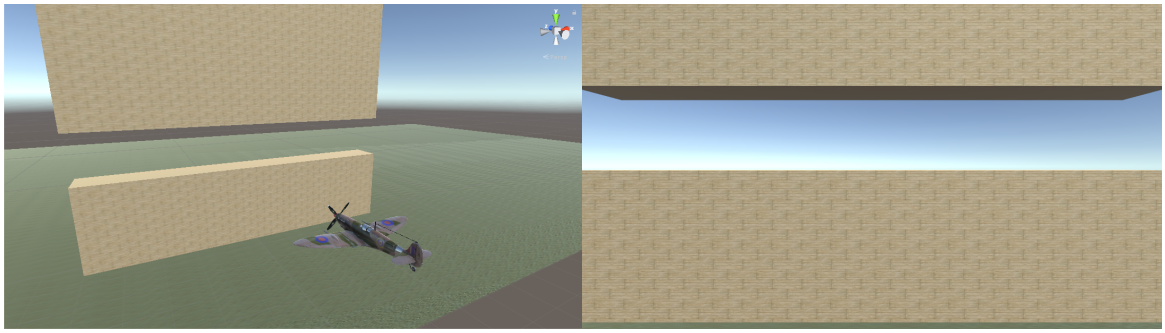
Figure 17: Radial distortion also known as *Barrel Distortion*. The farther from the centre a ray enters, the larger the barrel distortion bends it.



(a) Red - "spherical" lens
Blue - paraboloidal lens

(b) Radial distortion is most clear on square objects

Figure 18: Radial distortion also known as *Barrel Distortion*. The farther from the centre a ray enters, the larger the barrel distortion bends it.



(a) Red - "spherical" lens
Blue - paraboloidal lens

(b) Radial distortion is most clear on square objects

Figure 19: Radial distortion also known as *Barrel Distortion*. The farther from the centre a ray enters, the larger the barrel distortion bends it.

5 Spherical Flow - Simulation

5.1 Measurement Model

5.2 Farneback Algorithm - OpenCV

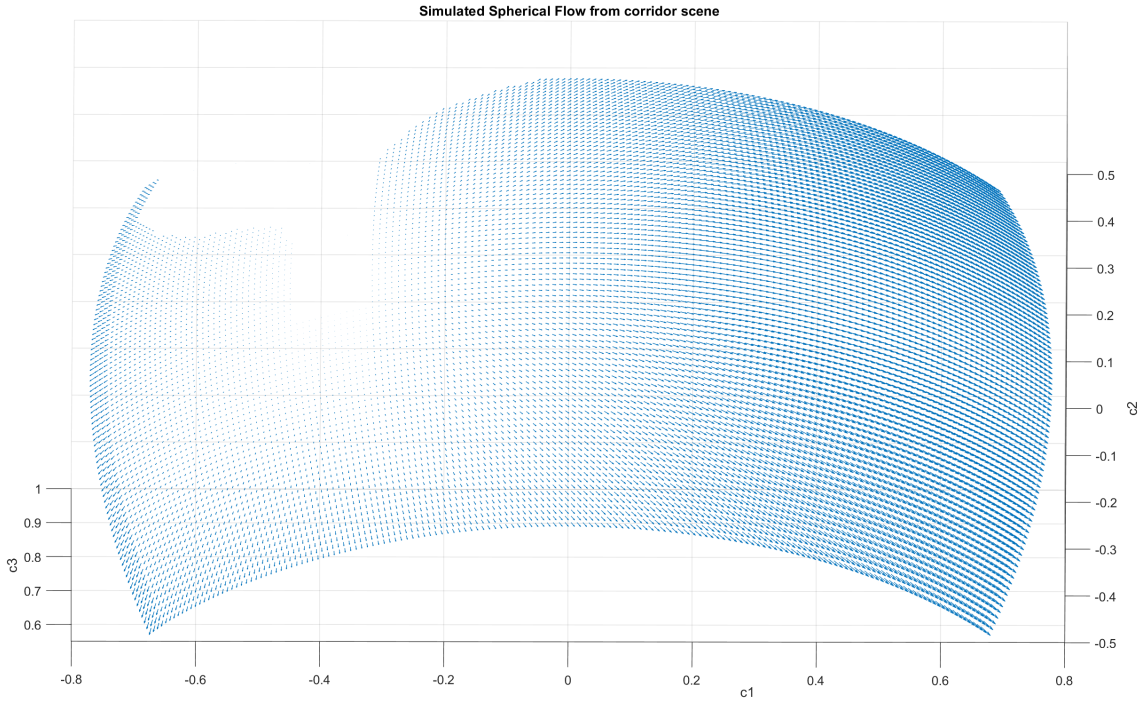


Figure 20: Write something here

6 Camera Calibration

As discussed in **??** it is imperitive to understand and account for the construction and imperfections in a camera, if the measurments taken need to be relevent outside image coordinates. This section outlines the procedure used in this project to correct a GoPro hero 7 using a planar camera model and a 6th order equation to correct for distortion parameters.

6.1 Calibration Boards

Theoretically, to calibrate a camera any object could be used as a calibration object, so log as it is appropriately characterised. However it is much more practical to use a well defined and easily measured object. For this reason, a flat rigid checker pattern was chosen. **??** shows a representation of the layout for the chosen calibration pattern, however the true pattern consisted of a $10x7$ checker grid. These are the measurement input into the calibration algorithm, allowing it to perform a *projective transform* and recover the cameras intrinsic parametres and distortion parametres. For the

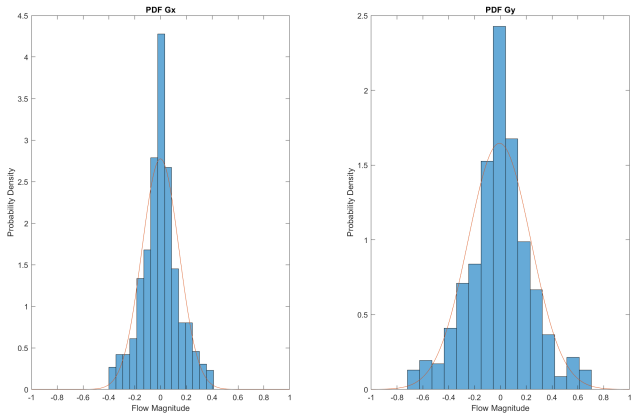


Figure 21: Write something here

actual calibration board used see **??**

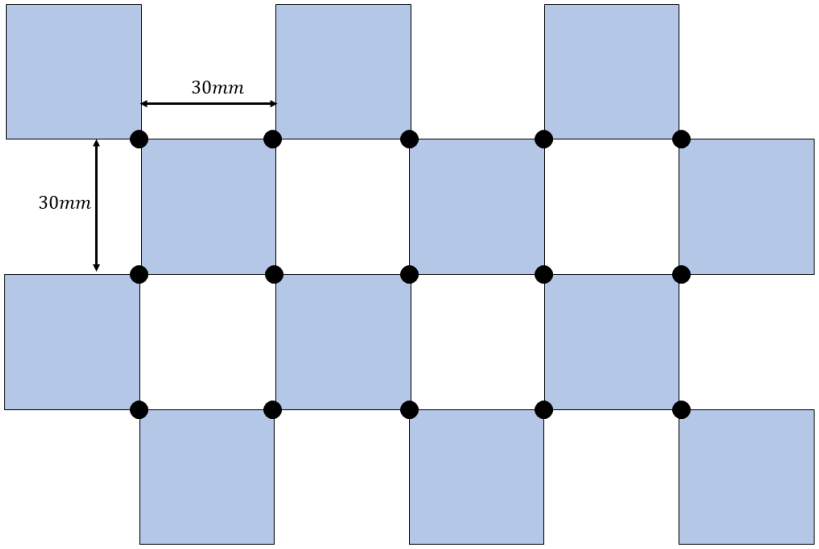


Figure 22: Expected corner locations for the purpose of camera calibration.

Now the calibration pattern has been defined, pictures of this pattern needed to be taken through the GoPro.In order to best capture the geometry of the camera, multiple views of the calibration board needed to be captured. Ensuring to capture shots of the calibration board in the edge of the image frame, as this is where the image can be most distorted. I found it most convenient to take a video of the pattern and later extracted the individual frames using *ffmpeg*, a basic command line video processing tool.

Outlined in **??**, the camera intrinsic parameters and distortion parameters are aproximated through a least squares fitting of the corner locations to the Brown-Conrady model. For this reason, multiple views of the calibration pattern are processed.

6.2 Calibration Tool

To calibrate the GoPro Hero 7 for in this project, A camera calibration tool was written in C# using WPF as a graphical user interface(see **??**)). After extracting a collection of images from the calibration video as outlined in **??**. These image are input into the calibration tool ready to be processed. The quality of the calibration images plays a large part in the resulting accuracy of the calibration routine. Using the calibration tool, the best frames from a list of frames can be selected to be calibrated against. An automatic function can also be chosen that selects 12 images with visable corners at random.

6.3 Undistorting Camera Points

Once the camera intrinsic matrix and distortion parametres were found, an algorithm to implement **??** needed to be written. This was due to the matlab implementation having a lot of overhead therefore running unnesessarily slow. The Brown Conrady model is based on a taylor series characterising the distortion of the lense, therefor it is necessary to run multiple itterations on the distorted points to obtain the most optimal undistorted coordinates possible.

For the purpose of determining the best camera distortion model, both a 3^{rd} order and a 6^{th} order radial distortion polynomial was fit to the GoPro hero 7.

Listing 4: Mex'd Undistort points function

```
void calculateUndistortion(double *xOut, double *yOut, double x,...
double y, double *k, double *p)
{
        int ii = 0;
        double x0 = x;
        double y0 = y;
        double r2, r4, r6;
        double iRaidDist; // Inverse radial distortion
        double tangDistX; // Tangental distortion in the x direction
        double tangDistY; // Tangental distortion in the y direction
        for(ii = 0; ii < ITTER; ii++) {
                r2 = x * x + y * y;
                r4 = r2 * r2;
                r6 = r4 * r2;
                iRaidDist=(1+(k[5]*r6+k[4]*r4+k[3]*r2))/(1+(k[2]*r6+k[1]*r4+k[0]*r2));
                tangDistX=2 * p[0] * x * y + p[1] * (r2 + 2 * x * x);
                tangDistY=p[0] * (r2 + 2 * y * y) + 2 * p[1] * x * y;

                x = (x0 - tangDistX) * iRaidDist;
                y = (y0 - tangDistY) * iRaidDist;
        }
        *xOut = x;
        *yOut = y;
}
```

7 Simulation Results

Figure 23: Caption

8 Discussion

Environment navigation is an extremely broard topic, and this project contributes a small amout towards the obstacl avoidance

Figure 24: Caption

9 Conclusion

The goal of this project was to investigate the plausability of using optical flow cues as a method for avoiding obstacles in a scene. Results of the simulations show that the a cost function based on divergence of an optical flow field generates a suitable control policy, stearing the UAV in a direction away from obstacles such as walls and buildings.

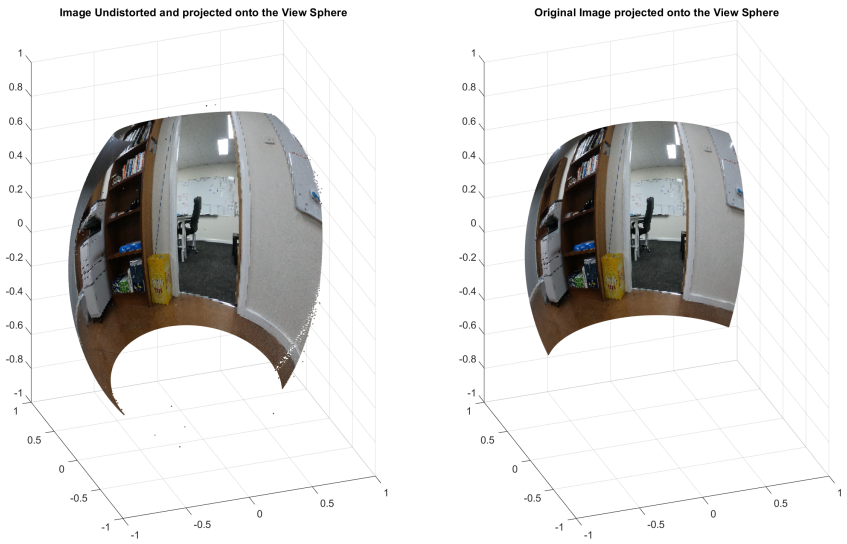


Figure 25: Caption

10 Recomendations

For the results of this project to be implemented on a robotic platform, an optical flow algoithm would need to be run in real time. for this, an image flow algorithm that can be implemented on a descrete graphic card should be implemented such as that ceated by **?**.

A Journal

B Calibration Tool

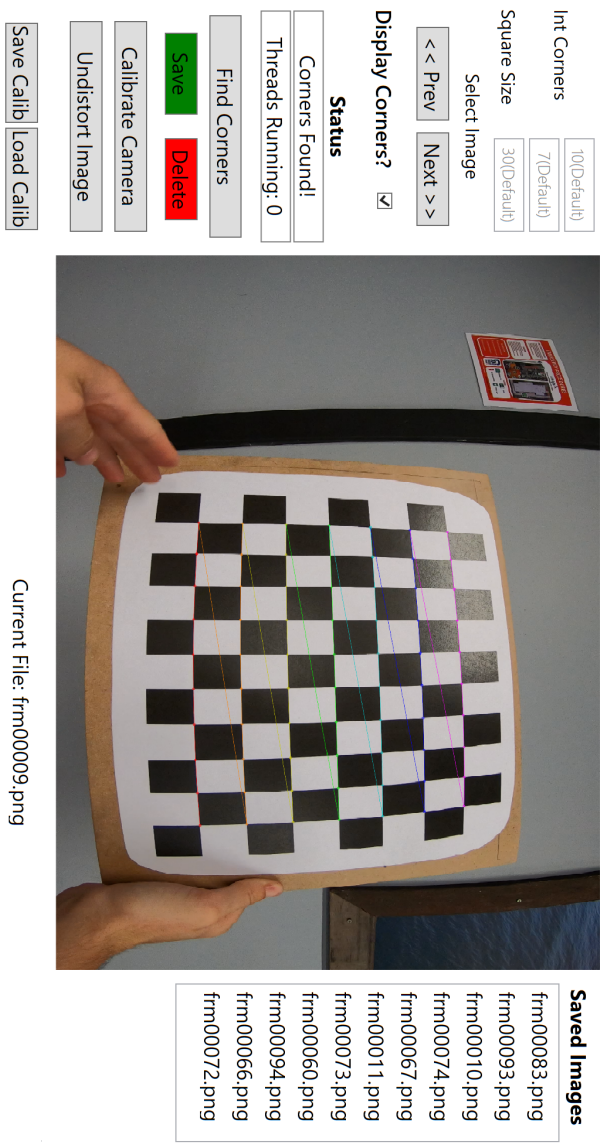


Figure 26: Write something here

C Calibration Board

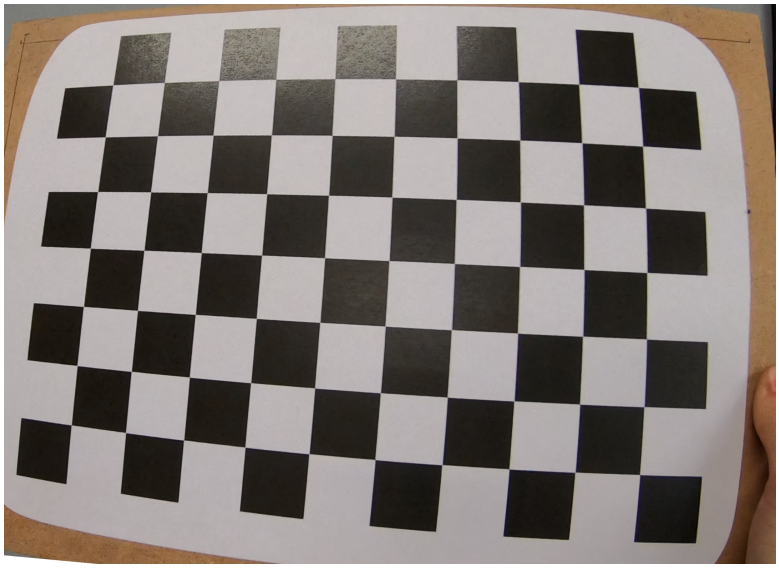


Figure 27: Calibration pattern mounted to a rigid MDF board, creating a rigid pattern the calibration process can reference

D Unity environment

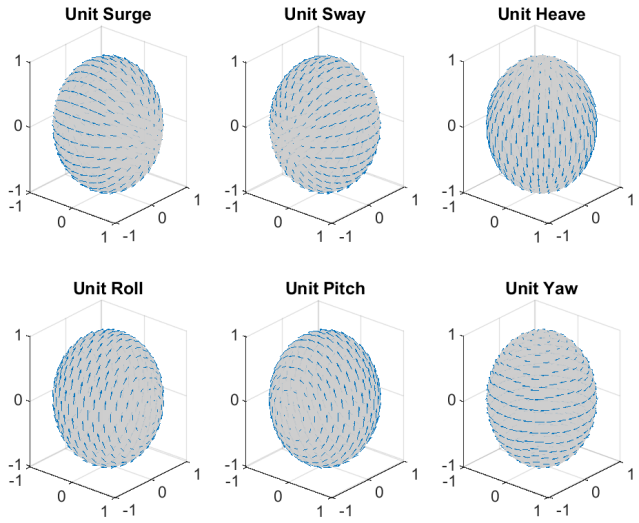


Figure 28: Write something here