The Potential of Optical Flow in Quadcopter Navigation

by
Cian Alexander Craeye

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechatronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. W.J. Smit

April 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: April 2019

Abstract

In order to determine the potential of using optical flow as an instrument for quadcopter navigation, a real-time depth estimation system was created. The system was programmed in Python and made use of: FAST algorithm to detect key points, Lucas-Kanade pyramid algorithm to calculate the optical flow, different geometrical relationships to calculate depth from translation and Euler angles to negate the effects of rotation. A simulated testing environment created in the Unity game engine was used to successfully test the overall performance, accuracy and robustness of the system. The positive results from the system and component tests proved that there is potential in using optical flow in quadcopter navigation.

Uittreksel

Ten einde die potensiaal van optiese vloei as 'n instrument vir hommeltuig navigasie te bepaal, is 'n intydse diepte skattingstelsel geskep. Die stelsel is in Python geprogrammeer en maak gebruik van: FAST-algoritme om sleutelpunte te bepaal, Lucas-Kanade piramide algoritme om optiese vloei op te spoor, verskillende geometriese verhoudings om diepte van beweging te bereken en Euler-hoeke om die effekte van rotasie te negeer. 'n Gesimuleerde toetsomgewing wat in die Unity enjin geskep is, is gebruik om die algehele prestasie, akkuraatheid en robuustheid van die stelsel suksesvol te toets. Die positiewe resultate van die stelsel- en komponenttoetse het getoon dat daar potensiaal is in die gebruik van optiese vloei in hommeltuig navigasie.

Acknowledgements

I would like to thank my supervisor, Dr. W.J. Smit for his guidance throughout the project.

I am grateful to my family and loved ones for their ever-present emotional support.

Thank you to my parents for supporting me financially and giving me the opportunity to pursue my dreams.

To Whelan Mohali for being the voice of reason in times when it was most needed.

Lastly, I would like to thank Samantha Van Heerden for always believing in me and motivating me to achieve my goals, without you this would not have been possible.

Contents

Figures

Tables

Nomenclature

Abbreviations

API	Application Program Interface
BRIEF	Binary Robust Independent Elementary
CCW	Counter Clock-Wise
CSP	Concentrated Solar Power
CW	Clock-Wise
Dual TV-L1	Duality-based Total Variation with L^1 norm
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FOE	Focus of Expansion
FOV	Field of View
FPS	Frames per Second
KLT	Kanade-Lucas-Tomasi
LIDAR	Light Detection and Ranging
OpenCV	Open Source Computer Vision
ORB	Oriented FAST and Rotated BRIEF
RANSAC	Random Sample Consensus
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
TTC	Time to Contact

Symbols

d	Distance	[m]
D	Depth	[m]
f	Focal length	[m]
t	Time	[s]
u	X-axis of the image plane	[px]
v	Y-axis of the image plane	[px]
w	Function representing a window	[s]
α	Angle between obstacle and direction of travel	[rad]
Φ	Pitch	[rad]
τ	Time period	[s]
Θ	Yaw	[rad]
Ψ	Roll	[rad]

Vectors

E	Variation of light intensity in a window
M	Matrix representation of a variation of light intensity in a window
$\overrightarrow{\mathrm{r}_x}$	Ray passing through two points
R	Rotation
T	Translation

Dimensionless Numbers

I	Light intensity	[-]
∇I	Spatial gradient	[-]
I_t	Temporal gradient	[-]
p	Point	[-]
R_c	Corner Response	[-]
s	Scale factor	[-]

1. Introduction

The topic of renewable, green electricity is one of the biggest talking points of the twenty-first century and one of the major role players in the industry is solar power. Concentrated solar power (CSP) is currently one of the leading sectors in the field of solar energy. These power plants use energy, usually carried by hot air or steam, which is then converted into electricity via turbines (Guerrero-Lemus, 2012).

Solar plants can consist of thousands of solar collectors (mirrors), and inspecting these mirrors' faces for damage, dirt and alignment is a major operation for any ground crew to complete. A solution to this time-consuming process is to introduce drones, fitted with high resolution cameras, to fly around the plant and inspect the mirror faces.

A quadcopter working on a solar plant faces multiple navigational challenges. One of these being that the area they are working in does not have a fixed surrounding. The location of all the solar collectors and the central tower may be fixed, but the position of construction or maintenance crews can be unpredictable and are not mapped on the drone's pre-planned paths. Further unique challenges faced by the quadcopter in these situations include: flying through potentially blinding solar beams between the mirrors and the central tower, and the highly reflective surfaces of the mirrors.

Systems for navigation on drones such as LIDAR, ultrasonic range-finders, and multiple cameras all have limitations and there is not yet one single system that is taken as the industry standard to be the perfect navigation solution. This has left room for alternative methods of navigation to be investigated.

Modern quadcopters fly around with a multitude of sensors, processors, cameras and other high-tech equipment. In this thesis it is theorised that a known technology (optical flow) could be adapted into a new navigation tool capable of working with the hardware already available on quadcopters. For optical flow to be seen as a useful technology in this field it would need to provide information which could be used in a navigation system.

1.1. Objectives

The main goal of this thesis is to prove or disprove the theory that optical flow has potential to be used in quadcopter navigation and validates further research. In order to answer this question, a number of objectives needed to be reached:

- Investigate the way different optical flow algorithms work.
- Explore how optical flow is currently used in quadcopters.
- Design a depth estimation system using optical flow.
- Design an environment in which to test the system.

- Put the system through a variety of tests to allow different conclusions to be drawn with regards to performance, adaptability and accuracy.

1.2. Paper Layout

In Chapter 1 the background of the reason for the research is discussed and a main objective is given with a list of sub-objectives. In Chapter 2 research is done to further understand optical flow and how it relates to the situation. This was necessary to make critical design decisions later on in the thesis.

A simulation environment is created in Chapter 3. The simulator is used throughout the project to both develop and test the system.

An in depth process of designing and testing individual components of the system is presented in Chapter 4. The reasoning behind decisions were explained and more specific research was done on the theorems and algorithms involved. The system was then tested as a whole under a variety of conditions in Chapter 5.

The main aspects of the paper is revised in Chapter 6. The chapter evaluates some of the big decisions that were made based on their outcomes and further ties together the overall outcomes by discusses the potential of the system.

In Chapter 7 the final conclusion is given by looking at what was set out to be achieved and the results obtained.

2. Literature Review

This thesis looks into optical flow navigation, we will therefore first define optical flow and where it comes from. Secondly, we define a quadcopter and give a brief overview of how it works. Thirdly we take a look at previous work done that is related to this project.

2.1. What is Optical Flow?

Optic flow can also be defined as the change of structured light in an image, due to a relative motion between the observer and the scene (Raudies, 2013). In terms of digital data, optical flow is the apparent motion of individual pixels or regions in two consecutive images. The x and y positions of pixels are determined in both images and the change in values are used to determine a movement vector which is optical flow. Considering that optical flow allows for a numerical definition of motion in a sequence of two dimensional images, it is firmly seated in the field of computer vision.

Optical flow is the basis of a system which takes the seemingly random valued pixels in an image and tries to determine where objects or useful information is hidden. The concept was first introduced by American psychologist James Jerome Gibson in the 1940's when describing how animals perceive their environments to be able to move around freely without collisions (Gibson, 1950).

Humans constantly make use of optical flow to help complete tasks, such as:

- Estimation of self-motion.
- Breaking down the visually perceived environment into moving and rigid parts.
- Determining the depth of different objects.
- Calculating an estimate for time-to-impact.

This type of information is used in various fields, for example car driving assistance to detect other cars or pedestrians, airplane guidance systems to avoid collisions, video codecs to interpolate between key frames, and in fast high resolution displays to create additional frames.

2.1.1. Assumptions

To be able to mathematically model an optical flow field, certain fundamental assumptions need to be made. The assumptions are based on the small time steps between two frames which make gradual changes appear as fixed values. The assumptions can be summed up as: object brightness consistency, spatial coherence and temporal persistence (Mammarella, et al., 2012).

The object brightness consistency assumption states that an object's change in intensity in the two images is caused only by a relative motion between the camera and the object. The intensity change due to other light sources or shadows is therefore ignored.

The spatial coherence assumption forces motion to be uniform over a small patch of pixels. These small discrepancies would otherwise give a great variety of velocity vectors for a single object. In making this assumption, the variety of velocity vectors is smoothed and stabalised.

The temporal persistence assumption states that the movement of a small area of a surface will only change gradually as time passes.

2.1.2. Different Methods Used

As per the optical flow techniques summation by Beauchemin & Barron (1995), it is widely accepted that there are four main, broad classifications of optical flow. These different areas are better defined by Mammarella, et al. (2012) as: Gradient, Phase-based, Region-matching and Feature-based methods. A short summary of the basis of each method was done to better understand the different approaches that can be taken.

Gradient Methods:

The gradient based methods are possibly the most widely used methods today. These methods are formed from the brightness conservation equation and use spatiotemporal derivatives of the intensity it defines to estimate flow. To get to the equation, we define the intensity of a pixel as $I(u, v, t)$. The pixel is located on the x and y axes of the image at u and v respectively. These plane co-ordinates represent a feature that moves δu and δv over time period δt. This leads to the following:

$$I(u, v, t) = I(u + \delta u, v + \delta v, t + \delta t) \tag{2.1}$$

When Equation 2.1 is derived with respect to time, we get:

$$I_u \dot{u} + I_v \dot{v} + I_t = 0 \tag{2.2}$$

Equation 2.2 is known as the *Brightness Conservation Equation* or the *Gradient Constraint Equation*. This is the equation which the gradient methods use to try and solve for the optical flow values $(\dot{u}, \dot{v})$. As Equation 2.2 is defined there are two unknown variables and additional assumptions needed to be included in order to solve it. The spatial coherence assumption states that all the pixels in a small surrounding region will have the same motion. This allows for multiple constraints to be placed, as well as the calculation of the region's flow. The most well-known algorithms in this field are: Lucas-Kanade (1981) and Horn and Schunck (1981).

Phase Methods:

This method was originally worked on and created by Fleet and Jepson (1990) where they proved it possible to estimate flow by using the scale, speed and orientation extracted from the image signal by linear shift-invariant filters. As in the gradient methods the spatial coherence assumption makes it possible to calculate an estimated flow value for a region by using values from a group of points.

Their algorithm determines the optical flow values by following three steps. In the first step they use a Gabor filter to obtain the spatial filtering and from there they can calculate the temporal phase gradient using the estimation of the velocity components. In the second step, if a certain corresponding filter pair's phase data is not in linear form over a given period of time, its component velocity is rejected. In the third step, the information gathered by the filters is combined and the estimation of optical flow values in the u and v directions is made using the partial velocities.

The phase methods have been found to work well at lower movement speeds, but do not give reliable values when being used at higher speeds due to the temporal aliasing.

Region-matching Methods:

For these methods, the displacement, between two consecutive images, of a certain area around a pixel is determined. This is then used for the optical flow value of that pixel. There are two different methods that are used, the difference method and the correlation method. For the difference method, different areas are assessed in a predetermined distance around the selected pixel location in the second image. These areas are then compared to the area in the first image surrounding the pixel using the sum of absolute differences. The correlation method has a similar approach, but calculates the correlation amongst the areas instead of the sum of absolute differences. Region matching-based methods have better results with faster movement than the gradient based methods, but require a large amount of computational power. Increasing the size of the region that is searched for the best matches has an exponential impact on the computing power required.

Feature-based Methods:

These methods consist of two main operations that take place in order to calculate the optical flow values. In the first operation, each of the two images is searched for unique features. The features are determined by scanning through the image, looking for areas that match a set of predetermined parameters. In the second operation, these features are put through a matching algorithm to find which features appear on both images. The matching features' data is then analysed to find the movement of those features between the two frames.

2.2. Origins of Optical Flow

In order to realise the vision the original inventors had, it was important to establish the origins of the theories used. Optical flow was first theorised by an American psychologist James J. Gibson in the 1940s to describe how animals perceive their environment in three dimensions from only the information carried by the light. The first experiments done to prove his theories were done on insects.

Most systems found in nature are far more complex, compact, efficient and accurate than any technology humans have managed to create. We therefore look to nature for inspiration when trying to find the best possible steps forward.

An earlier theory that locusts make use of the motion parallax to obtain depth information was later proven to be true (Sobel, 1990). The locusts perform a movement called "peering" before they take a leap to an intended target. This action involves the locust moving its head from side to side, and using the visual data collected to determine how far the object is and therefore how far it has to jump.

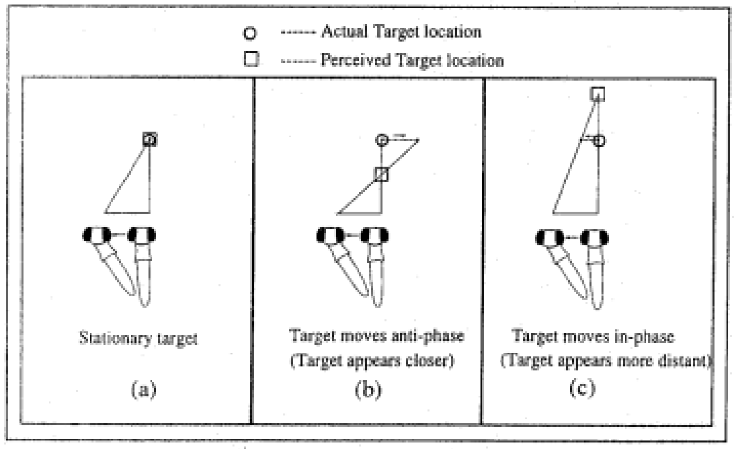

Figure 1: How locusts perceive distance by peering (Sobel, 1990).

The theory of peering is shown in Figure 1. It indicates how an object can be perceived as moving closer by taking its phase and size of motion into account. The Peering is proven to be an effective gauge of depth, but it is only applicable for stationary insects. A variety of different usages for optical flow has been found in flying insects, including stabilising themselves during flight and interpreting the three dimensional world around them (Srinivasan, 1998).

Srinivasan breaks down the features for which flying insects use optical flow, into the following categories:

- Stabilising their flight.
- Hovering.
- Flying through narrow gaps.
- Controlling their flight speed.
- Estimating their distance flown.
- Making smooth landings.
- Distinguishing objects at different distances.
- Detecting the differences between objects and background.

To better understand how such feats can be achieved using optical flow, it is further described how hovering and flying through narrow spaces is made possible.

When insects are hovering, they are essentially going through the same process as the system of a mechanical flying object. Srinivasan explains that when interference caused by wind or other sources impacts the position of an insect, it is measured by the movement of the image on the frontal retina. The amount of movement is taken into consideration and an opposite force is exerted to correct for the movement. This idea has been implemented in quadcopters using a downward facing optical flow camera to predict how much correction is required to keep the machine stable (Romero, 2009). It is the most commonly used application of optical flow in quadcopters, and preconfigured attachments to achieve this are commercially available.

Optical flow is used in a different manner when an insect navigates through small or narrow gaps. Tests were carried out on bees to determine how they use optical flow to navigate through a passageway (Kirchner & Srinivasan, 1989). It was found that bees don't measure distance when determining the middle of the passageway; instead, they balance out the flow field values on the left and right side of their vision.

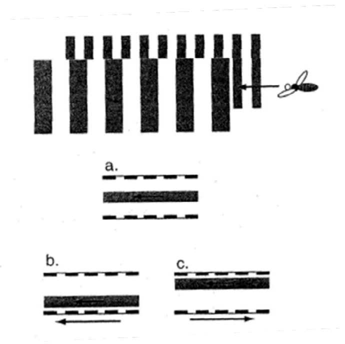

Figure 2: Bees flying through narrow passage experiment (Kirchner & Srinivasan, 1989).

In Figure 2 the thick horizontal grey bar represents the path the bees took and the arrows in b and c represent the direction in which the wall was moved. These three experiments were carried out to prove bees use optical flow to find the centre of a narrow passage.

In a, the walls are kept still and the bee flies down the middle of the passage. In b, the wall on the left of the bee is moved in the same direction as its flight. The vertical lines painted on the wall of b combined with the movement of the wall creates low optical flow values on that side, tricking the bee to think that the wall is far away and causing it to fly against it.

The opposite happens in c, when the wall is moved in the opposite direction to flight, the optical flow values are high and the bee thinks that the wall is closer than it actually is, and flies further away from it. This theory was implemented on a quadcopter by Zingg, et al. (2010) where they used it to navigate through indoor corridors.

Similarly to the previous two descriptions, the rest of the actions performed by the flying insects for using optical flow, are proved by performing carefully constructed tests. Many of the categories mentioned have been used as a foundation for research in the field of mechanical flying machines, specifically quadcopters.

2.3. Definition of a Quadcopter

A quadcopter is a mechanical flying machine with four rotors, each mounted on individual arms as shown in Figure 3. Two rotors spin clockwise and two spin anticlockwise, this balances out the torque from the motors and keeps the quadcopter stable around the z-axis (as shown in Figure 3).

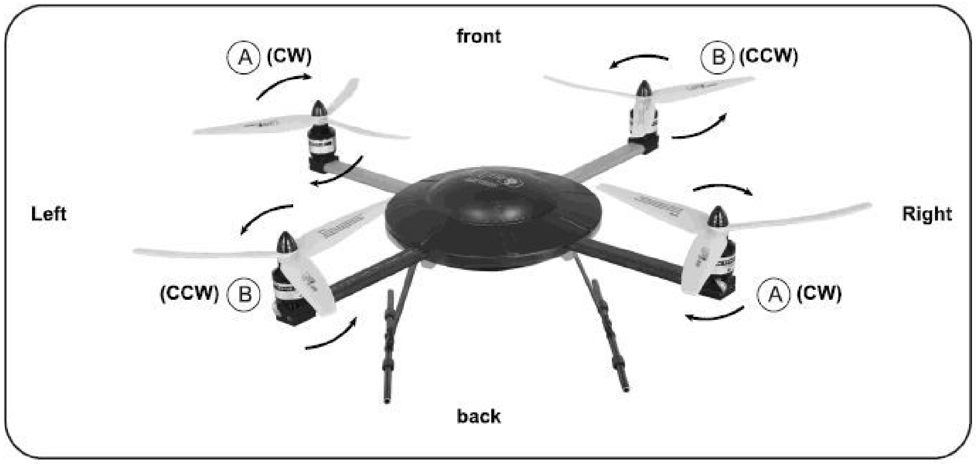

Figure 3: Rotor layout of a quadcopter.

The quadcopter is aerodynamically unstable and relies on a controller to vary the speeds of the four rotors for stability. The operator of the quadcopter sends a signal for a certain motion to the control module and the required speed adjustments are made to the rotors to perform the action. The force model in Figure 4 shows the axes and directions of motors in a quadcopter.

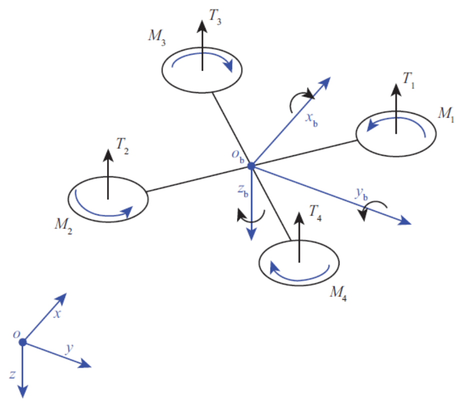

Figure 4: Force model of a quadcopter.

If the quadcopter is required to hover in position, then all motors would be rotating at roughly the same speed, creating a similar torque all around and keeping the quadcopter stationary. If any movement is required in the six degrees of freedom, then the speeds of the motors need to be adjusted as shown in the three different cases in Figure 5. To rotate counter clockwise around the x-axis and alter the pitch, motors two (M2) and three (M3) need to be sped up and motors one (M1) and four (M4) need to be kept at the same speed or slowed down (Hurd, 2013). To rotate the quadcopter clockwise about the x-axis the opposite needs to be done, where motors one (M1) and four (M4) will be sped up and the others kept the same or slowed down. Similarly, the rest of the rotations can be performed by speeding up or slowing down certain motor pairs.

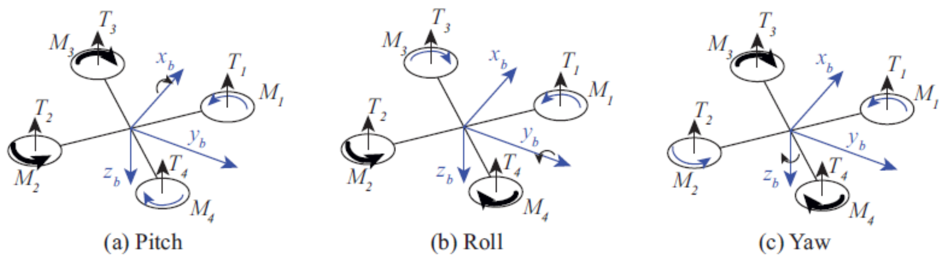

Figure 5: Rotation of a quadcopter around three axes.

2.4. Research on Optical Flow for Navigation

In this section multiple different approaches for using optical flow, to determine useful navigational information, are summarised and analysed.

2.4.1. Real Time Depth Estimation and Obstacle Detection

A front mounted car camera was used to develop and test an algorithm that detects static objects and determines their distance from the car while driving in a straight line (Wedel, et al., 2006). The depth was estimated by determining regions that represent certain obstacles and looking how those regions changed in size over time (as the car moves forward).

Wedel et al. defined the equation for depth calculation as show in Equation 2.3. This equation assumes that the speed of the vehicle is known along with the optical flow values of image points and given a large enough time scale between images. $T_Z(t,\tau)$ Represents the change in translation in the Z direction from time t to time $t+\tau$, and similarly, $s(t,\tau)$ represents the scale factor of the region representing the object over the same time period. Because the relative distance between the different levels of the surface of an object is small in comparison to the distance between the observer and the object, the surface is assumed to be a set distance d away. Using this assumption allows for multiple image points to be used, thereby more accurately determining the distance of a single object.

$$d \equiv Z(t) = \frac{s(t,\tau)}{1-s(t,\tau)} T_Z(t,\tau) \qquad (2.3)$$

In Figure 6 the results obtained by Wedel et al. are plotted against a radar measurement. The results in this case were positive when the distance of the object was less than 70 meters away. Although this experiment is not directly applicable to quadcopters, it does show that it is possible to accurately determine depth of objects in a static field when traveling in a straight line.

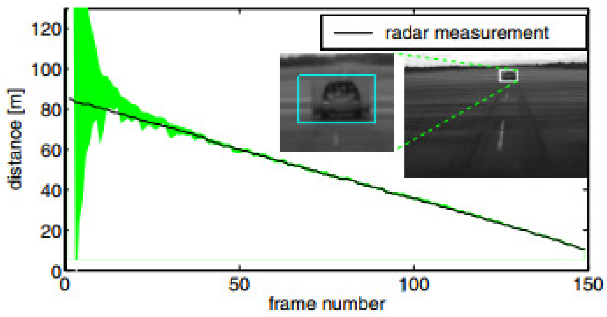

Figure 6: Distance estimation (Wedel, et al., 2006).

Their work showed that the relationship between the change in the size of the object on the image plane and the camera movement can be used to accurately determine the distance of the object. The experiment was done in a one dimensional movement scenario and only proves the theory to be possible for a single case of movement of the quadcopter's six movement dimensions.

2.4.2. Using TTC to Detect Objects with Optical Flow

The idea of using the time to contact theory in the field of obstacle avoidance with optical flow was introduced by Low and Wyeth (2005). In their experiments a ground robot with wheels was used with a single front facing camera.

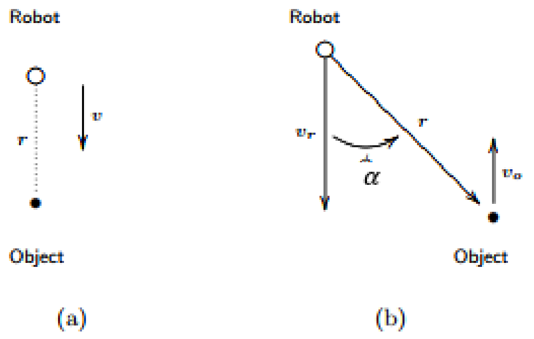

Figure 7: Test layouts (Low & Wyeth, 2005).

If the distance to an object and the velocity at which the robot is traveling are known, then the time to contact can be calculated. This describes the scenario which is found in Figure 7-a, the TTC is calculated as shown in Equation 2.4.

$$TTC = \frac{r}{v} \tag{2.4}$$

In Figure 7-b the robot is moving towards an object, but will not come into contact with it. Using the angle between the robot and object and the angular velocity at which this angle changes, it is possible to calculate the time it will take for them to pass each other. This time can be used to calculate the distance of the object. Equation 2.5 shows how the TTC is calculated for situation b.

$$TTC = \frac{cos\alpha \times sin\alpha}{\dot{\alpha}} \tag{2.5}$$

Equation 2.5 has the advantage that it only requires optical flow values to be calculated. If it is assumed that the robot travels in a straight line, then the TTC can be calculated for each point for which the optical flow was calculated in the image.

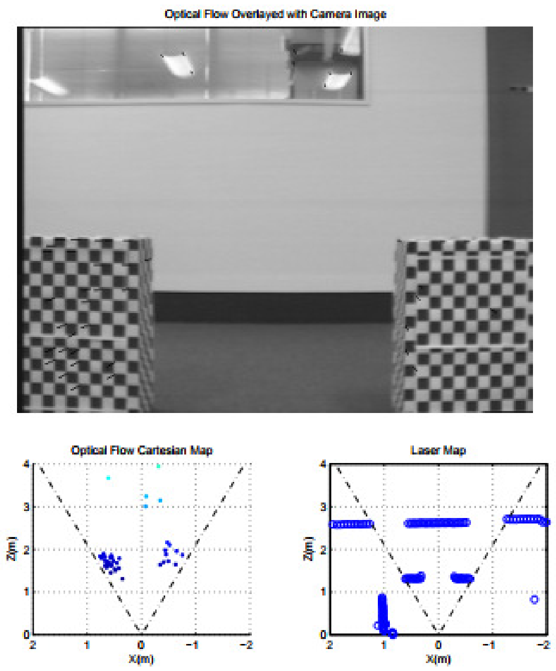

Figure 8: TTC obstacle detection results (Low & Wyeth, 2005).

Their results shown in Figure 8 indicate that the depth of the obstacles were vaguely correctly determined. The points in the graph on the left representing the optical flow estimation give a similar result to the laser map on the right for the placement of the obstacles.

Again, the results indicate to us that there is a definite link between the optical flow and the forward motion of the robot. In this case the results are not as accurate as in the previous paper, but to a human eye looking at the results it is clear that there are two distinct objects. If the accuracy could be improved to get a more definite picture of where the objects reside there is potential in this method.

2.4.3. On-Board Obstacle Avoidance in Quadcopters

The problem of running an on-board obstacle avoidance system based on optical flow was approached by Orozco (2014) in six different stages:

1. Tracking the features found in two consecutive images using both matching (SIFT, SURF and ORB) and motion (Kanade-Lucas-Tomasi (KLT)) tracking methods.
2. Removing incorrectly tracked features and outliers with RANSAC.
3. Estimate the depth of matching 2D points over multiple images using epipolar geometry.

4. Negating the accumulating errors of the 3D reconstruction using Extended Kalman Filters (EKF).
5. Generating a 3D point cloud.
6. Transforming the points into a 3D surface via Delaunay Triangulation.

In order for Orozco to achieve his goal of running his computations on-board a drone, he needed to make certain accommodations during his testing. The testing was done for only 30 tracking points, using an image stream of 320 x 180 pixels. The results successfully proved that optical flow can be used as a feature tracker in real time for a drone with only on-board processing power.

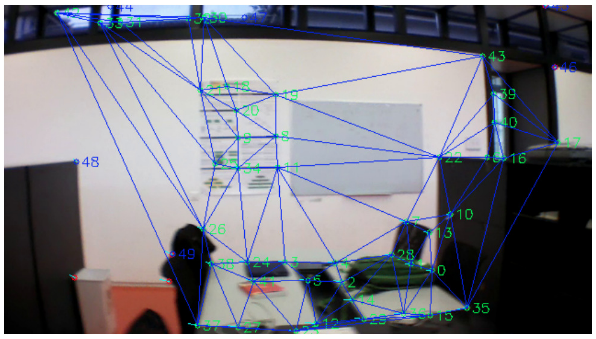

Figure 9: Point selection (Orozco, 2014).

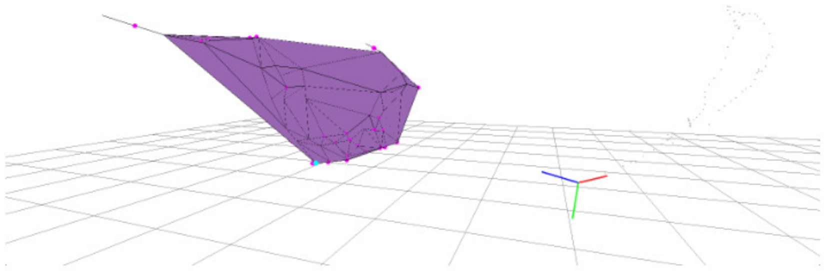

Figure 10: 3D surface reconstruction (Orozco, 2014).

The results of a successful point selection process are shown in Figure 9. Using these points, a 3D reconstruction (Figure 10) of the environment was done. Without including multiple images from different angles or the ability to rotate the 3D scene in Figure 10 it is difficult to see how well it represents the real world. The system was however able to successfully draw a fairly accurate depth model for the amount of points it was using. The main problem encountered here is that not enough points could be plotted with the available processing power to make the 3D model useful to a navigational system.

It is possible to learn from these results by taking into account that processing speed should be prioritised in the decision making process when designing for a real time quadcopter system. It is clear that accuracy or in-depth calculations would need to be swapped for basic, efficient relationships in order to create a system that can realistically operate using readily available technology.

3. Simulator

Initially it was unclear where to start with the basic implementation of optical flow and attempts were made to use a quadcopter with a mounted GoPro to record data during different test flights. It was quickly established that using real world tests for initial data was not ideal because of all the variables involved. It was impossible to determine if the positional readings correctly correlated with the video stream and whether or not any of the values could be trusted.

It was concluded that in order to thoroughly develop and evaluate the performance of any optical flow algorithm a reliable testing environment needed to be created. The environment would need to produce a video stream input to calculate the optical flow as well as the 6 dimensional movement vector of the drone to enable the calculation of the positions of the surrounding obstacles.

It was therefore decided to create a virtual world with obstacles in which a virtual camera could fly around as a drone would. All the movement vectors would be known and a ground truth of the distance from camera to obstacles would be obtainable.

3.1. Software

The idea of using a game engine to simulate the quadcopter came from the freecam mode found in many computer games, most notably in Counter Strike's death-cam. In this mode you can control your view with your mouse and four keyboard buttons to move in four out of a possible six dimensions to "fly" freely around the game's environment much like a quadcopter would.

The Unity game engine was chosen as the platform on which to create the simulator. Unity is a free cross platform game engine which can be used to create 2D or 3D games or simulators on 27 different platforms. The creation process is carried out using the Unity editor and a C# scripting Unity API.

3.2. Environment Design

The real world environment in which the worker drones operate is shown in Figure 11. The ideal surroundings consist mostly of blue skies and green grass with other objects in the distance. As an initial design, the virtual world was created in reference to the real world. The floor made of grass like material and the background set to a blue sky as shown in the image.

Figure 11: Helio 100 Test Facility.

The obstacles placed in the simulation are shown in Figure 12, these capsules had a height of 20 units and a radius of 5 units. Random preconfigured textures ranging from wood, to metal, to stone were used for each capsule. These simple objects would create a good starting point for determining if the concept had potential without any excessive designing in the unity editor. The unit value was not directly tied to any real world value because no physics engines were used, but setting 10 units = 1m gave the most relatable measurements.

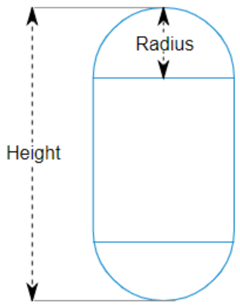

Figure 12: Simulation obstacles.

By keeping the structures simple, some important challenges that would otherwise be faced in the real world, such as reflective mirrors and solar beams, are not accounted for. The thesis is focused on determining if optical flow could provide useful depth information and the extra challenges were therefore deemed to lie outside the scope of the work being done. It is however important to note that there are other real world aspects not yet introduced to the system that would have an impact when moving forward from the simulated environment.

3.3. Freecam Script

To implement this action in the simulator, a script was created and attached directly to the main camera object. This same script was used throughout to do all the required programming. Unity's base class allows for a start function which is called once at the beginning of the program to initialize everything and an update function which gets called every frame. By adding a check in the update function for the presence of six different keyboard buttons it was possible to update a translation movement vector accordingly. This vector was used to translate the camera in all three directions (x, y and z). The mouse input was used to determine the rotation about the z- and x-axis and another keyboard button for the y-axis. The sensitivity of the rotational angles and translation distances were adjusted via trial and error to a value that was easy to control and appeared to represent the movement speed of a quadcopter.

3.4. Test Record Mode

One of the advantages of using a simulator is the ability to isolate different variables during a test run. In order to compare these variables to each other it was important to be able to repeat a test exactly with only one variable changing. To make this possible the simulator process was split into two separate modes: record and playback. The mode could be selected via one of the custom input arguments when running the program.

In the first mode the user would record a path for the camera to take during a particular test. This was done by giving the user control over the freecam and recording the camera position for every active frame. A button was assigned for the user to start, pause and stop the recording process. At the end of the recording process the camera position values were stored to an excel spreadsheet.

3.5. Test Playback Mode

In the second mode the user could use the input arguments to specify which dataset to read in and what test parameters to apply. These parameters adjusted the: brightness level, velocity (units/frame), rotational/translation vibration and rotation/translation input errors. The program would then automatically run through the list of camera locations given, recording the translation and angular rotation between frames as well as the distance from the camera to the visible objects. In addition to the movement data, the program recorded a screenshot of every frame and saved it to the specific test file. At the end of the run the data was again saved to an excel spreadsheet.

The different variables were implemented as follows:

Brightness: A lighting source was added into the unity environment. This object's intensity value was set according to the input variable during the start function.

Velocity: The path of the camera was essentially plotted out by a line of points in a 3-dimensional space, therefore it was possible to travel larger or smaller distances per frame by interpolating the path. This was done by adding an additional 100 points between every point and choosing the points on the path which would give the newly required distance value between frames.

Vibration: The vibration was implemented by using two 3-dimensional vector variables to store the previous vibration point's position (initialized to zero). Every frame the camera position was taken from the given data and adjusted by a value between –x and x where x represents the different variable input values for each of the six directions. The movement between the newly vibrated point and the two 3-dimensional vector variables was stored. Finally the new point's position was stored in the variables and then the process was repeated for all frames.

Inaccuracies: After the test was performed the resulting translation or rotation values were adjusted by a specified percentage to simulate a sensor with limited accuracy.

3.6. Automation

The compiled unity program could be executed from the command line with input variables. It was therefore possible to create two separate batch files to boot the program into either the recording or playback modes. To maximise the efficiency of creating test packages the file space in the two batch scripts were syncronised and a whole list of different tests scenarios were added to the playback script. The result of this being that the user could create a test run by executing the 'record' script and then automatically generate numerous data sets for different variables by simply executing the 'playback' script.

4. Designing an Optical Flow Navigation System

This section describes how the Optical Flow Navigation System was designed and the reason behind the different decisions taken during the design process. The main focus of the system was placed on how the optical flow data could be turned into useful information to help navigation in quadcopters. The methods used to obtain the optical flow values as well as the methods used to manipulate their outputs are further discussed.

4.1. System Parameters

As with the design of any system, the first step is to define its boundaries and goals. These two categories will paint a clearer picture of where the system fits in and how its success can be measured.

4.1.1. Boundaries

Because this thesis looks at the potential of using optical flow with quadcopters, it was decided that the system would initially operate under ideal conditions in a simulated environment. The environment was deemed to be static, perceiving all slow moving objects to not be moving at all and ignoring fast moving objects. All variables that would usually limit a drone in real world flight are controlled and would not directly impact the design of the system. The variables can later be introduced one by one to investigate their impacts.

The data stream coming in from the camera is pre-recorded and therefore no time restrictions are placed on the calculations done for every frame. The calculation time would however have an impact on the overall potential of using the system as it would need to be implemented in real time on a quadcopter at some point if the technology were to be used in the field.

The system input is limited to a single camera feed in the form of a series of images and a 6-dimensional movement vector of the drone between frame intervals. This is the only data that the system can make use of to do the navigation, no further information on the drone’s surroundings is given.

The system is required to directly or indirectly make use of optical flow methods to do its navigation.

4.1.2. Goals

Navigation in the case of this thesis is limited to moving around a virtual open grass field with “2m” tall and “1m” wide capsule shaped obstacles with different textures. The ultimate goal of the system is to accurately determine the drone’s environment and obstacles while moving in any combination of its 6 degrees of freedom.

The performance of the drone's navigation capabilities is measured by the reliability, accuracy, range and robustness. Because of the nature of the field, no exact figures were chosen to determine the level of potential of optical flow as a navigation system. The final potential rating given at the end of the thesis is based on how well the system performed individual tasks, the pros and cons of the system, how the system would perform on its own vs how it could perform as a subsystem of a bigger system, and what future work could be done with this approach.

4.2. System Design

In this subsection the main components of the system and its design process is discussed. An extensive breakdown is done on each component's theoretical background and why it was chosen. The section is in chronological order and each component was impacted by what had been done and what was still left to do.

4.2.1. Inspiration

As with most optical flow theories, the system was driven by the natural process of peering, explained earlier. The idea being that in a stationary environment it should be possible to estimate depth of any chosen pixel or pixel group by knowing the movement of the camera between images. Therefore the movement of the insect's head is similar to the movement of the drone, just at a much higher speed.

4.2.2. Software

OpenCV was chosen as the foundation of this system. OpenCV is a large open source computer vision library (including optical flow libraries) widely used in industry and at universities around the world. The library has been highly optimised and is designed specifically for real time processing. The library was originally written in C and has since moved to being developed in C++. Wrappers have been written for Python and Java allowing these languages to be used without sacrificing performance.

4.2.3. Optical Flow

The OpenCV libraries support the following optical flow algorithms:

Brox et al (2004)**:** Their energy functional is based off a combination of works to enable the usage of coarse-to-fine warping. Large displacements are handled and the algorithm responds especially well in the presence of noise. Constant brightness and gradient as well as a smoothness constraint is assumed. The library implements this as a dense optical flow function.

Farneback (2003)**:** A dense optical flow function which is implemented over two steps. Firstly it uses the polynomial expansion transform to calculate quadratic polynomials for each neighbourhood in both of the frames. Secondly it estimates the displacement from the polynomial expansion coefficients.

Zach et al (2007)**:** The Dual TV-L1 variation based function preserves the discontinuities encountered in the flow and has good results when it comes to occlusions, noise and illumination changes. It also falls under the category of dense optical flow fields.

Lucas Kanade Pyramid: Implements the classical Lucas Kanade algorithm with an extended pyramidal layer which accommodates larger movements (Bouguet, 2000). This function is given in both dense and sparse optical flows.

As the system was to be designed specifically for real time operation, the faster sparse optical flow was chosen over the dense ones. The sparse field could be fine-tuned to detect only a small group of key points on the images which could then be analysed for their optical flow. In comparison the dense flow fields would be doing a lot of unnecessary calculations on points with no useful data. Another motivation for the sparse field was the reduction in processing required on post optical flow operations. The sparse field separates the key points early on in the system's sequence; therefore all calculations after the optical flow calculations would only need to be done on the smaller data group.

The Bouguet implementation was considered to be an adequate starting point to test optical flow navigation for drones because of its good reputation in the industry. The algorithm was one of five used to produce initial baseline results for the Middlebury benchmarking and evaluation platform (Baker, et al., 2011).

The decision was made with the knowledge that the sparse field might miss out on key features in images or misinterpret objects where the dense field would have the full picture. It was decided that the increase in calculation speed would outweigh this possibility of misinterpretation when it came to evaluating the potential of the overall navigation of the drone.

4.2.4. Finding the Best Points

In order to make use of the sparse optical flow equations, a list of points need to be used as input. The function will then determine the optical flow of each point individually. Functions specifically designed to find useful points such as edges or corners of different objects are catagorised as *Corner Detection Algorithms*. In the OpenCV library there are three such algorithms which can be executed in real-time namely: Harris, Shi-Tomasi and FAST. Shi-Tomasi is based on Harris and considered to be an improved extension of it. Harris was thus ignored and a further investigation was done into the other two functions.

Shi-Tomasi:

The foundation of the Shi-Tomasi corner detection lies in the Harris algorithm published by Chris Harris and Mike Stephens (1988). In their work they make use of a function closely representing the local auto-correlation to extract the location of strong feature points from an image. They analyse a small window in the image and shift it in all directions about its origin. The variation of light intensity of the window was determined to be:

$$E(u,v) = \sum_{x,y} w(x,y)\,[I(x+u,y+v) - I(x,y)]^2 \qquad (4.1)$$

where $w(x,y)$ represents the window function and the intensity difference for each pixel is calculated in the square brackets. In order to improve the speed of the function a first order Taylor series expansion was done and after re-writing the function into matrix notation they ended up with:

$$M = w(x,y)\begin{bmatrix} \sum_{(x,y)} I_x^2 & \sum_{(x,y)} I_x I_y \\ \sum_{(x,y)} I_x I_y & \sum_{(x,y)} I_y^2 \end{bmatrix} \qquad (4.2)$$

where I_x and I_y are the derivatives of the intensity in the x and y directions respectively. A formula was determined to use the eigenvalues of matrix M to estimate a corner response:

$$R_c = Det(M) - k \times Tr(M)^2 \qquad (4.3)$$

where $(M) = \lambda_1\lambda_2$, $Tr(M) = \lambda_1 + \lambda_2$ and k is a constant. If R_c is small a flat region is indicated, if it is large it predicts a corner, and if it is negative it predicts an edge.

The difference in the Shi-Tomasi algorithm is that a region of calculated response values is taken into account when determining the final points to be used. The local maxima of each region are recorded and only the region values meeting a certain criteria are considered. It was observed that when the chosen level is reached by the smaller of the two eigenvalues, the matrix tends to pass the test as well (Shi & Tomasi, 1994). The condition for passing the requirements can therefore be given as:

$$min(\lambda_1, \lambda_2) > \lambda \qquad (4.4)$$

where λ is the lower limit.

FAST:

The FAST corner detection algorithm was introduced by Rosten & Drummond (2006) to greatly reduce computation time of the corner detection phase in systems. They recognised the lack of processing speed in the corner detection field and created an algorithm that fully prioritised time efficiency. In their method the pixel being analysed is compared to the sixteen pixels forming a circular boundary around it as shown in Figure 13. This differs to the previous methods where every pixel in the region was analysed.

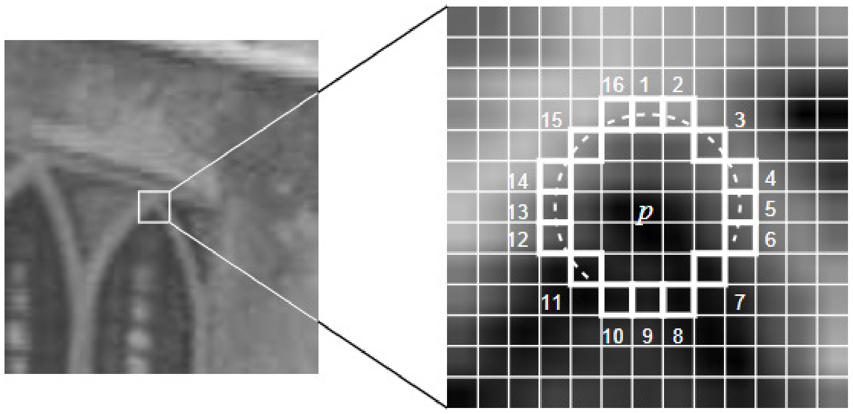

Figure 13: FAST corner detection visualisation

A corner is detected when the pixel P is found to have a difference of intensity greater than a certain threshold for at least twelve contiguous pixels of the sixteen that surround it. For this to hold true all the pixels qualifying need to be either lighter or darker, not a combination of the two. To speed up the process an extra elimination step was added where the pixel would first be compared to surrounding pixels 1, 5, 9 and 13. If at least three pixels did not meet the requirements then it would be impossible for a corner to exist, therefore saving the time it would have taken to calculate the rest of the pixels.

There are several sacrifices made in order to achieve faster performance:

1. The algorithm does not perform well when the value of pixels that have to qualify is dropped below twelve.
2. Choosing which pixels to use and their sequence, is based on assumptions about the features.
3. The four pixels calculated in the pre-test need to be recalculated during the actual test if the first section passes.
4. It is not robust to noise as it focuses mainly on speed.
5. The threshold needs to be given and can lead the algorithm to perform differently in various situations.

Comparison:

The performance of FAST, Shi-Tomasi and various other methods are compared with regards to their repeatability for varying numbers of corners per frame by Rosten & Drummond (2006). In these tests the FAST algorithm consistently outperforms Shi-Tomasi. There are however adjustable variables in each algorithm and the test data was not necessarily similar to that which would be found in our test scenario. Further testing on the two different types was done using images acquired from the simulator.

For the tests a reasonable arbitrary threshold was chosen for the FAST algorithm. The Shi-Tomasi parameters (maximum number of corners, quality threshold, minimum distance between points and size of the analysed block) were then set to generate a similar amount of points as the FAST algorithm did. In doing this it was possible to get an evaluation on the different computational speeds and the type of points chosen. All images used had a resolution of 640x480 for consistency.

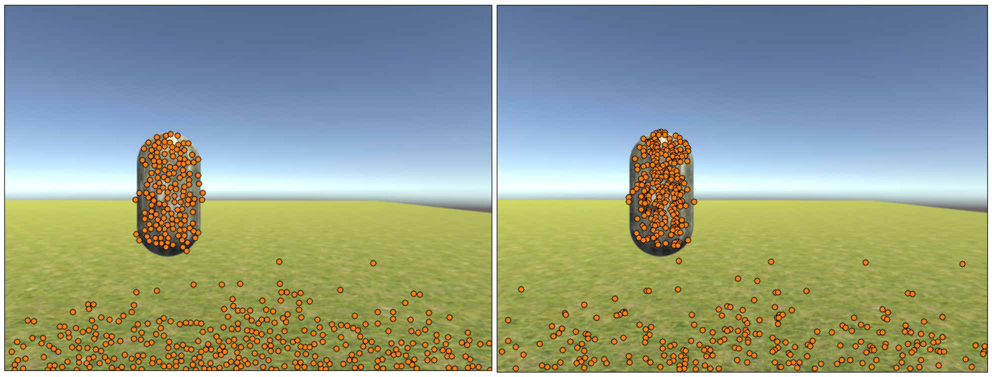
Figure 14: Point detection virtual world 1.

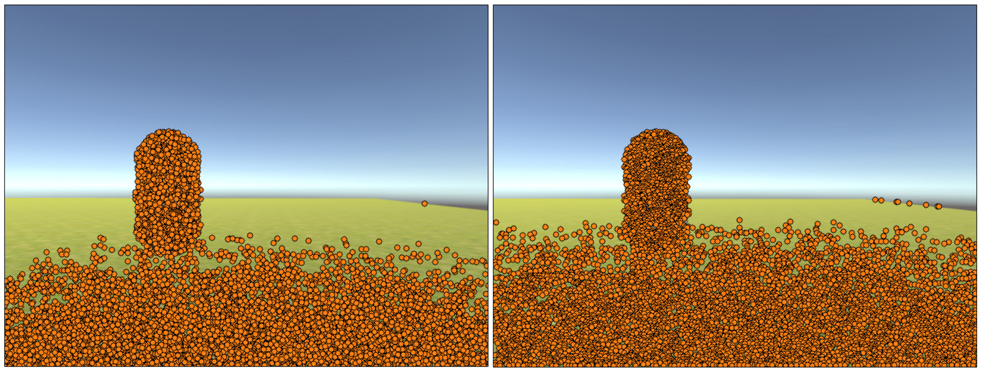
Figure 15: Point detection virtual world 2.

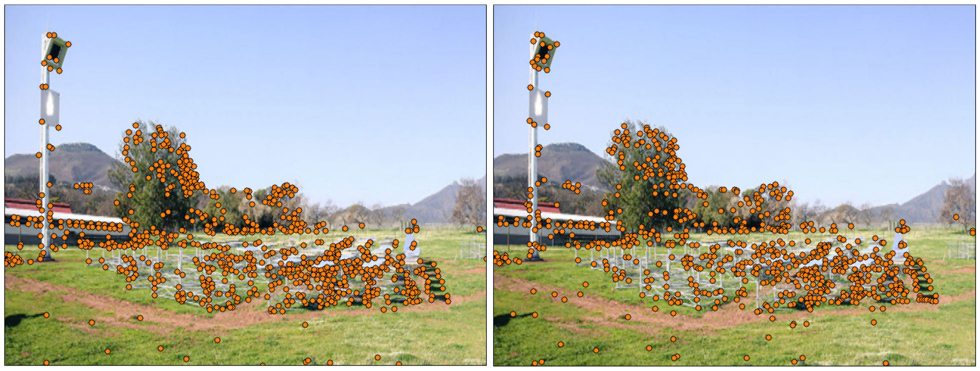

Figure 16: Point detection real world 1.

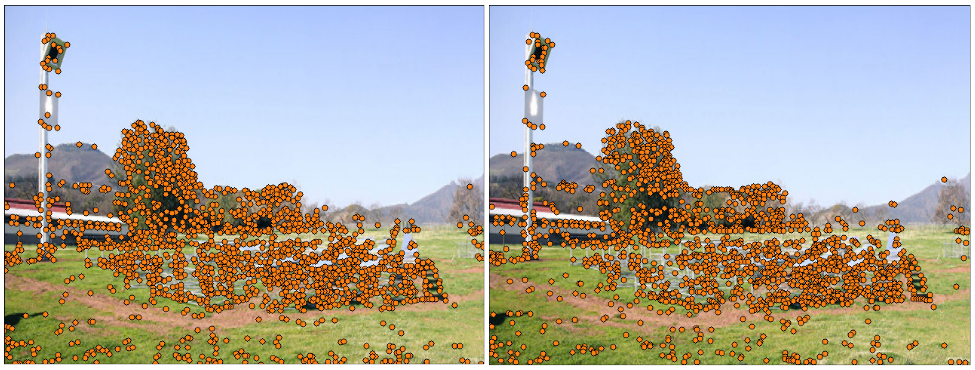

Figure 17: Point detection real world 2.

Table 1: Point detection test results.

	Shi-Tomasi		**FAST**		
Test	**Time [s]**	**Points**	**Time [s]**	**Points**	**Threshold**
Virtual world 1	0.013	475	0.002	474	20
Virtual world 2	0.017	5534	0.014	5745	3
Real world 1	0.012	779	0.003	774	54
Real world 2	0.012	1722	0.004	1720	39

Conclusions:

The results in Table 1 and Figure 14 to Figure 17 produced useful information about each method in practice, and a number of conclusions could be drawn:

1. The calculation speed is considerably faster when using FAST to find a smaller group of points. As the number of points found increases, so does the calculation time of FAST. This is due to the pre-calculation phase mentioned previously avoiding further calculation on points which don't meet the basic qualifications. As more points pass this pre-calculation phase, the processing time gets closer to that of Shi-Tomasi (as seen in the Virtual world 2 test). The time taken for Shi-Tomasi was found to be based on the minimum distance allowed between points. Indicating that all possible points are fully processed and tested. Because processing speed is of importance for this system, it was decided that the Shi-Tomasi algorithm is not fast enough to use.
2. The quality of points selected by the two algorithms was near identical with the only significant difference being the inherent ability of Shi-Tomasi to have points which were evenly distributed. An example of this can be seen in the virtual world tests where points for FAST are almost overlapping each other. This over detection could lead to multiple optical flow calculations being done on a single small area representing the same thing.
3. An interesting difference can be found between the real world and virtual world tests when considering the threshold value used vs the amount of points detected. In the first virtual world test only 474 points were detected using a threshold of 20, where in the first real world test 774 points were detected using a much higher threshold value of 54. To find the reason behind this another test was done where the virtual world threshold was set to 54, generating a result of only 4 detected points. This happened because the contrast between light and dark is much higher in the real world image, producing sharper corners and edges which are easier to detect. Increasing the rendering quality in the simulator to generate truer shading is a possible solution to this gap.

4.2.5. The Lucas Kanade Pyramid Optical Flow Algorithm

With the key points selected the next step is to take two consecutive images and determine how those key points move from image A to image B. To further understand how the data is being processed, the mathematical approach Lucas and Kanade (1981) initially took is investigated.

Lucas Kanade Algorithm:

The Lucas-Kanade algorithm predicts the movement of pixels from image to image by looking at the change in intensity in a neighbourhood of pixels. To do this an assumption is made that the two images in question are separated by a small enough time frame, to allow an insignificant displacement of objects. For a single pixel there are two unknowns and a single equation:

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [u\ v] \tag{4.5}$$

With $I_t(p_i)$ representing the temporal gradient, $\nabla I(p_i)$ representing the spatial gradient and $[u\ v]$ representing the unknown flow vector. To get more equations for each pixel a neighbourhood of pixels are assumed to have the same flow vector as the considered pixel. The mathematical interpretation of this can be given as:

$$\begin{bmatrix} I_x(p1) & I_y(p1) \\ \vdots & \vdots \\ I_x(p25) & I_x(p25) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p1) \\ \vdots \\ I_t(p25) \end{bmatrix} \tag{4.6}$$

If the first matrix is represented by A, the second matrix by d and the third by b, the, then the above problem can be solved by means of minimum least squares by determining $(A^T A)d = A^T b$. The following equation is obtained by solving as suggested:

$$\begin{bmatrix} \Sigma I_x I_x & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \Sigma I_x I_t \\ \Sigma I_y I_t \end{bmatrix} \tag{4.7}$$

Equation 4.7 can then be used to determine the flow values for a specific pixel (the pixel in the middle of the region).

Pyramid:

The Lucas Kanade algorithm on its own is limited, by only being applicable to small optical flow values. This challenge was overcome by introducing a coarse to fine approach where a low resolution version of the image was taken and optical flow calculated (Baker, et al., 2011). This was then repeated for a larger resolution image and the sum of the new optical flow values and a warped version of the old optical flow values were combined to form a more accurate estimate. This process could then be repeated for multiple levels.

OpenCV:

The Lucas Kanade Pyramid function in OpenCV follows the optimised implementation by Bouguet (2000). The parameters for the OpenCV function are as follows:

- **Previous image:** The first 8-bit image.
- **Next image:** The second 8-bit image.
- **Previous points:** The points in the first image to be mapped onto the second image using optical flow calculations (the key points found using FAST would be used here).

- **Next points:** The resulting calculation points.
- **Status:** Indicates whether each point was calculated.
- **Error:** Indicates errors for each point calculated.
- **Window size:** The size of each pyramid level's search window is set here.
- **Max level:** The maximum pyramid level to which the algorithm will search.
- **Criteria:** The parameter indicating when the iterative search is deemed successful.
- **Flags:** Describes the initial estimation input method and error measure method.
- **Minimum Eigen Threshold:** This value is compared to the minimum Eigen value of the spatial gradient matrix (calculated in the step before starting pyramid calculations) to reduce computational requirements by filtering out bad points at the earliest possible stage. If the flag for this parameter is not set the error value is calculated by dividing the distance between the points by the number of pixels in a window.

By understanding the theory behind the optical flow calculations and what each parameter represented, it was possible to change the input values until the function produced results with minimal to no clearly miscalculated results. The starting points for the variables were taken from the example given in the OpenCV documentation. The left and right sides of Figure 18 to Figure 20 shows the initial and final outputs respectively.

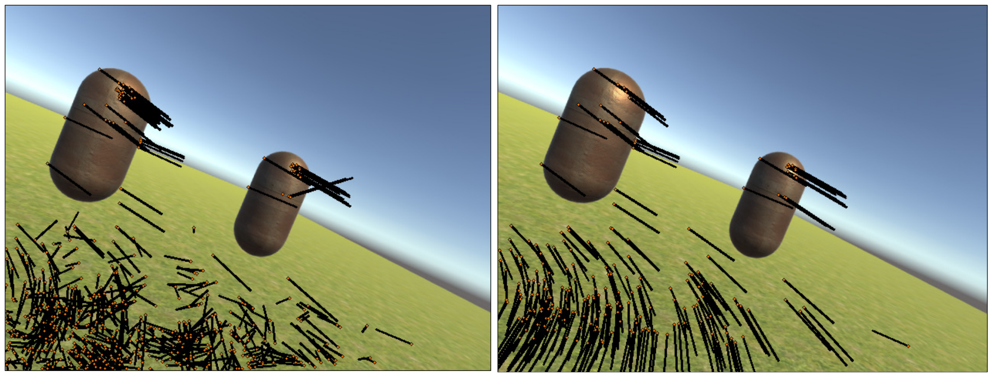

Figure 18: Optical flow correction 1.

In Figure 18 the quadcopter is moving forward and tilting its angle upwards and to the left. In Figure 19 the quadcopter is traveling in a straight line and the optical flow is moving outward from a central focal point. In Figure 20 the quadcopter is flying toward the ground while tilting upwards, causing the focal point to shift up.

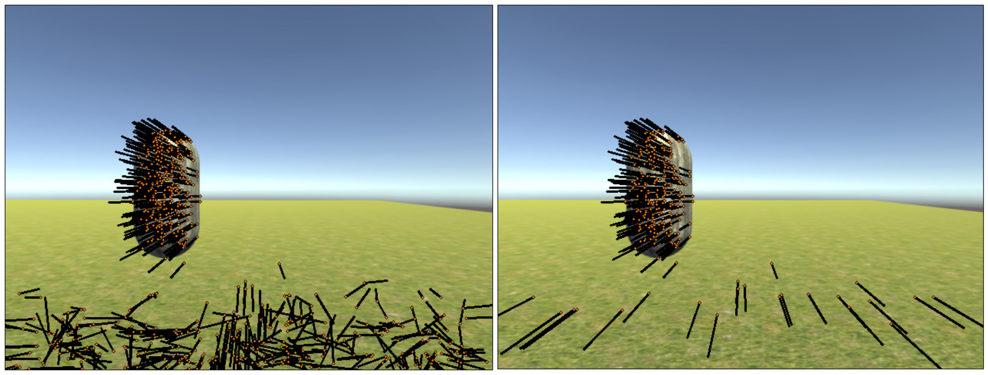

Figure 19: Optical flow correction 2.

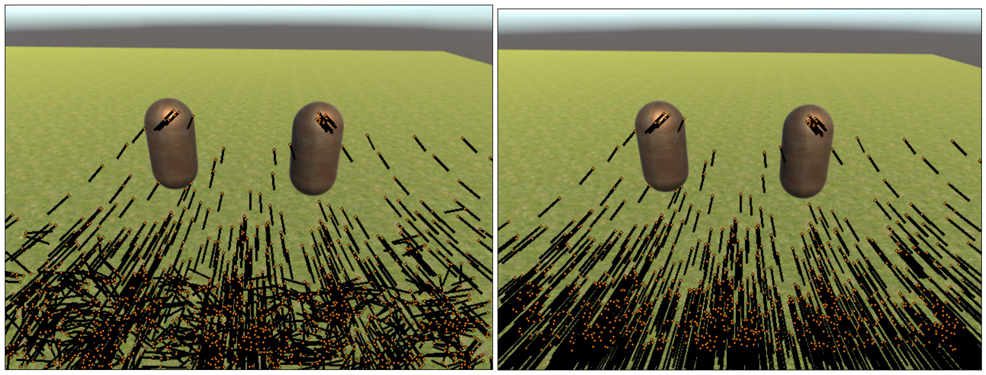

Figure 20: Optical flow correction 3.

To achieve the final outputs the max pyramid level was increased from 2 to 3, the window size decreased from 15x15 to 10x10, and an extra sifting phase was added to exclude points with an error value greater than 5. It was found that the rest of the adjustable variables had very little to no impact on the performance of the calculation in the case of the given scenario.

4.2.6. Depth from Optical Flow

Scene flow can be described as the change in 3-Dimentional location of each single point in an environment and optical flow can be interpreted as the 2-Dimensional projection of this (Vedula, et al., 1999). Because the location of the camera is known for both images, it is possible to geometrically predict the location of the points of the static scene from the optical flow data.

This method was used for navigating a passageway using a quadcopter by Zingg, et al. (2010). This subsubsection investigates and adapts their work onto the virtual quadcopter in the open simulated environment.

Depth from One Dimensional Movement:

In their paper they fist discuss depth estimation when traveling in a straight line. They end up with the following equation to predict the depth value (D) of a single pixel:

$$D = \frac{v}{OF} \cdot \sin\alpha \tag{4.8}$$

The velocity of the camera is indicated by v and the angle between the camera's movement and the obstacle is given as α. Equation 4.8 depicts an alternate method of determining the depth from the geometry of the situation visualized in Figure 21.

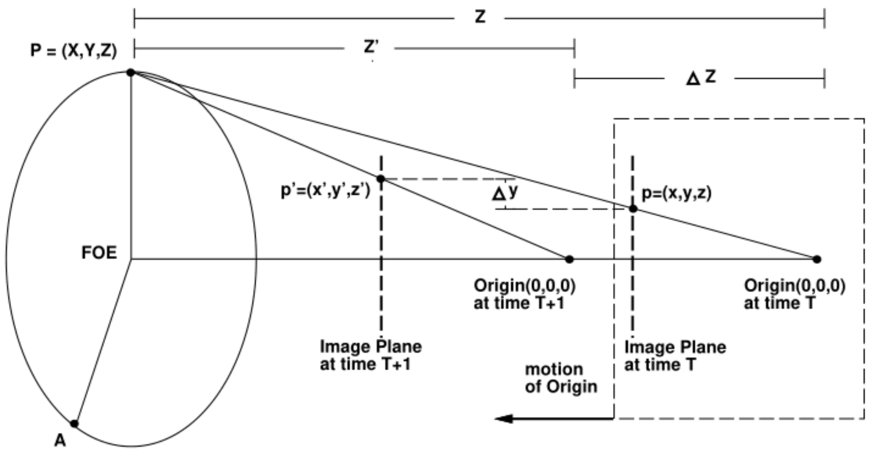

Figure 21: Geometry of time to contact in optical flow (Camus, 1995).

Because the angle between the camera and the obstacle is not directly known we make use of the original TTC derivation by Camus (1994) where similar triangles gives:

$$\frac{y}{z} = \frac{y}{1} = \frac{Y}{Z} \tag{4.9}$$

The equation is differentiated with respect to time, to get:

$$\dot{y} = \frac{\dot{Y}}{Z} - Y\left(\frac{\dot{Z}}{Z^2}\right) \tag{4.10}$$

The point P remains constant causing the first term to equal to zero. From the similarity Y can be substituted with yZ leaving us with:

$$\dot{y} = -y\left(\frac{\dot{Z}}{Z}\right) \qquad (4.11)$$

This resembles a ratio of change in the Z direction compared to the depth of the point and the change in the pixel location in the y direction compared to the pixel distance from the centre of the image. Rearranging to solve for depth:

$$Z = -\dot{Z}\left(\frac{y}{\dot{y}}\right) \qquad (4.12)$$

Assuming the same behaviour in the x direction, the equation was applied to both the x, and y version of Equation 4.12 to test the ability of determining depth when traveling in a straight line. For this scenario a single object was placed in-front of the drone as shown previously in Figure 14 and the drone moved in a straight line past it. The average value of the depth of the points was plotted against the ground truth of the distance between the object's plane and the drone.

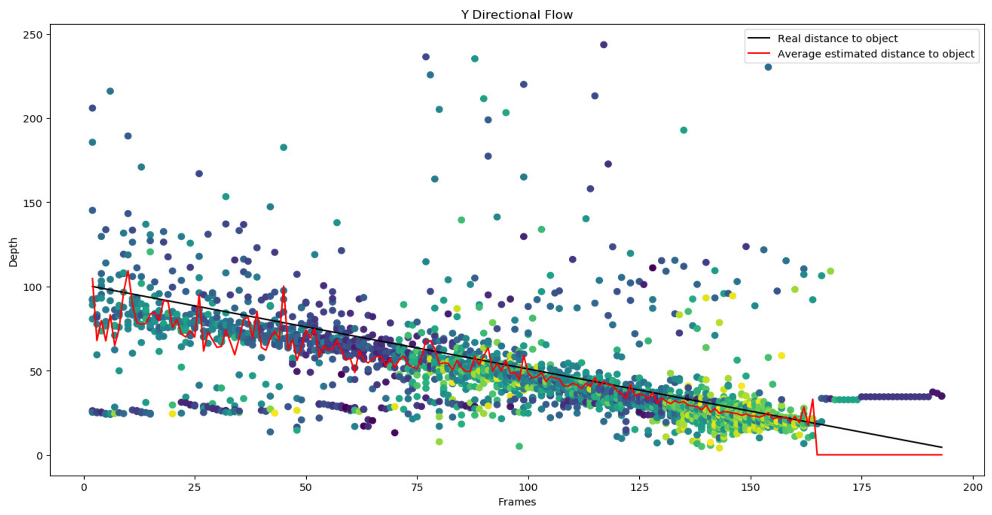

Figure 22: Depth from y-directional flow.

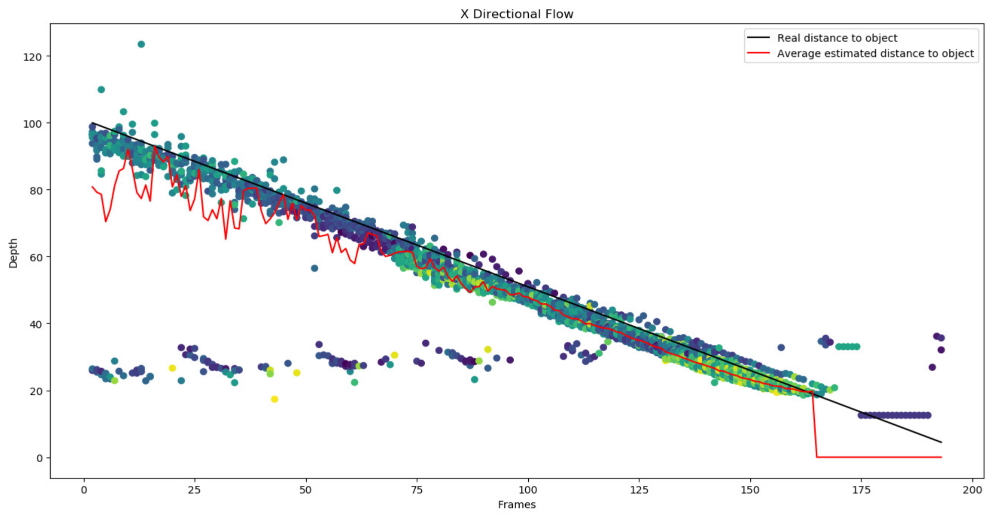

Figure 23: Depth from x-directional flow.

In Figure 22 and Figure 23 the dots represent each individual point with an optical flow value that was used to determine a depth at a specific frame. From Figure 23 it is clear that the points that represent the obstacle give a fairly accurate estimate of its depth for the flow in the x direction, with few outliers. The average distance is found to be slightly below the actual distance, this is partly due to the surface of the object not being exactly the same distance from the camera as the objects plane which passes through its centre. The points are spaced out in the early frames when the obstacle is far away and get more compact and precise as it moves closer.

In Figure 22 the points also produce an average value of depth that closely resembles the actual value, but it contains many outliers and the points remain scattered throughout. A reason for the difference in the accuracy of the two flows can be due to the object making larger movements in the x direction on the image plane. This is backed up by the results shown in Figure 24 where each point is evaluated by its largest flow direction. Most of the points representing the object are taken directly from the x flow and the points on the grass by the y flow. The graph shows an improved version of the two singular flows and therefore confirms the theory.

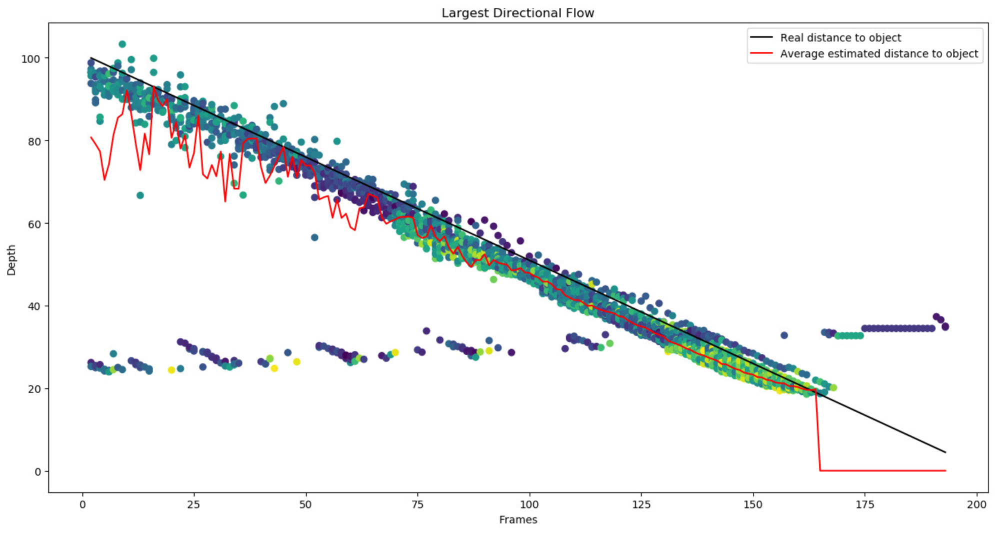

Figure 24: Depth from bi-directional flow.

The points plotted at around 25 to 35 units of depth in the first 120 frames and last 30 frames are points that were detected on the grass. An example of such points are the two grey markers near the bottom of Figure 25.

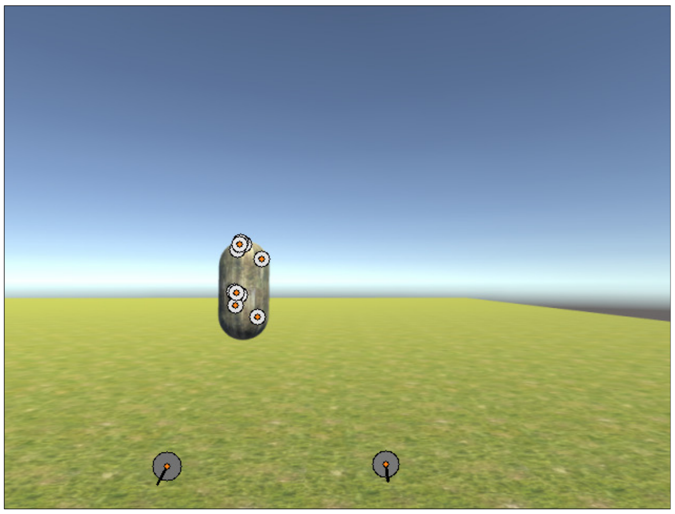

Figure 25: Straight movement depth visualisation frame 25

.

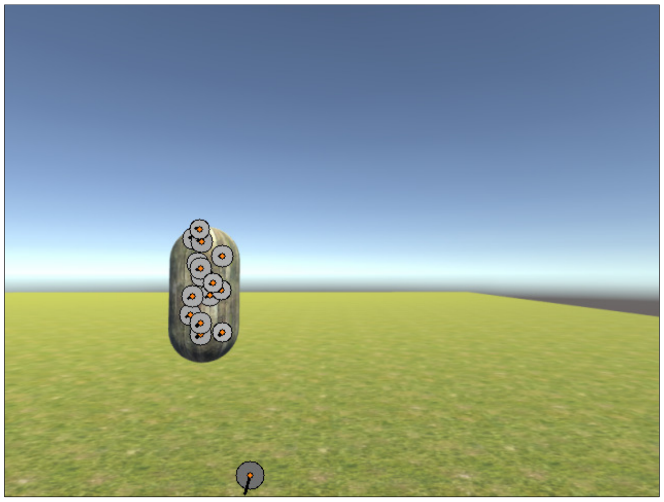

Figure 26: Straight movement depth visualisation frame 75.

In Figure 25, Figure 26 and Figure 27 the points detected are indicated by the orange dots. The size and colour of the circles surrounding the points increase and shifts from white to dark grey as the estimated depth decreases. The black lines coming from the orange points represent the magnitude and direction of their optical flow.

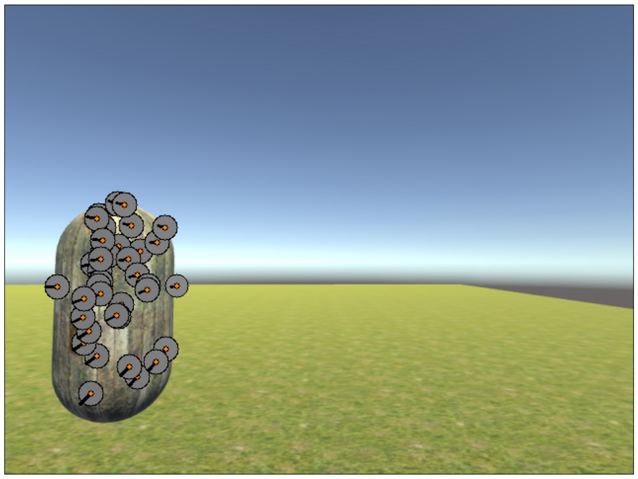

Figure 27: Straight movement depth visualisation frame 125.

4.2.7. Compensating for Rotation

Although the straight line method accurately detects how far away an obstacle at a certain pixel is, a new or adapted method needed to be introduced in order to accommodate the five other degrees of freedom a quadcopter has. One method of allowing for an angular change is to mathematically inverse the effects that each of the three rotational directions have on every pixel. This method is applied as an optical flow filter, fitting between the steps of optical flow calculation and depth estimation.

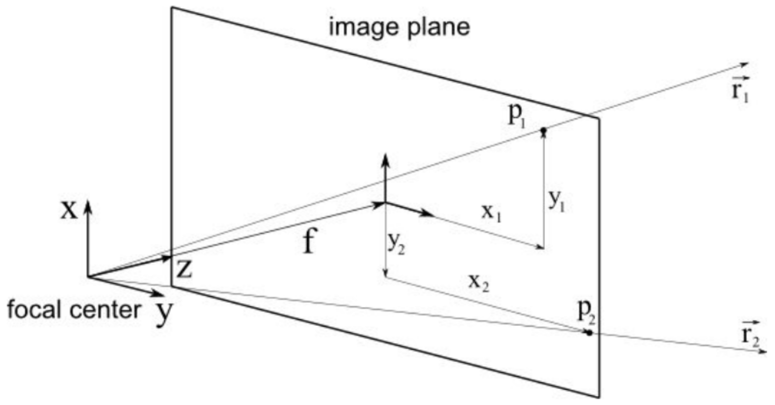

Figure 28: Pinhole camera model (Zingg, 2010).

The problem is solved by Zingg, et al. (2010) by analysing two rays passing through the two points on the pinhole camera model shown in Figure 28. These two points represent a matching pair of points in the two consecutive images of an optical flow calculation. They predict that by applying the Euler angles of the drone's movement to the second ray it is possible to mathematically estimate where the ray would have been if there was no rotation. The estimated ray could then be traced to the point on the image plane. The initial two rays representing the points are given as:

$$\vec{r_1} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \lambda_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ f \end{pmatrix} \tag{4.13}$$

And

$$\vec{r_2} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \lambda_2 \cdot \begin{pmatrix} x_2 \\ y_2 \\ f \end{pmatrix} \tag{4.14}$$

Where f represents the focal length as depicted in Figure 28. The Euler angles applied to the second ray gives us:

$$\overrightarrow{r'_2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\Delta\Phi) & -sin(\Delta\Phi) \\ 0 & sin(\Delta\Phi) & cos(\Delta\Phi) \end{pmatrix} \cdot \begin{pmatrix} cos(\Delta\Theta) & 0 & sin(\Delta\Theta) \\ 0 & 1 & 0 \\ -sin(\Delta\Theta) & 0 & cos(\Delta\Theta) \end{pmatrix} \cdot \begin{pmatrix} cos(\Delta\Psi) & -sin(\Delta\Psi) & 0 \\ sin(\Delta\Psi) & cos(\Delta\Psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \overrightarrow{r_2} \qquad (4.15)$$

where Φ, Θ and Ψ are the roll, pitch and yaw angles of the drone respectively. To test the accuracy of this filter the simulated drone was restrained in the x, y and z directions and only allowed to rotate around the three axes. In theory this scenario should cause the filter to completely negate the movement of the pixels, meaning that the adjusted second rays would be in the exact same position as ray one.

Before this test could be done the focal length of the camera needed to be calculated. The default Unity Field Of View angle was used (60 degrees) and the horizontal image width in pixels was set to 640. Using geometry from Figure 28 it is possible to calculate the focal length using the following equation:

$$f = \frac{\frac{x}{2}}{tan\left(\frac{FOV}{2}\right)} = \frac{\frac{640}{2}}{tan\left(\frac{60}{2}\right)} \approx 554 \qquad (4.16)$$

In Figure 29 to Figure 31 the purple points represent the original pixel location (p1) and the orange points represent the new pixel location (p2) determined via optical flow calculations. The slightly smaller cyan points (p2') are the result of mapping the recalculated ray two (r'_2) onto the images. The expected result is for the cyan points to shift from their original position on the orange points to the position of the purple points.

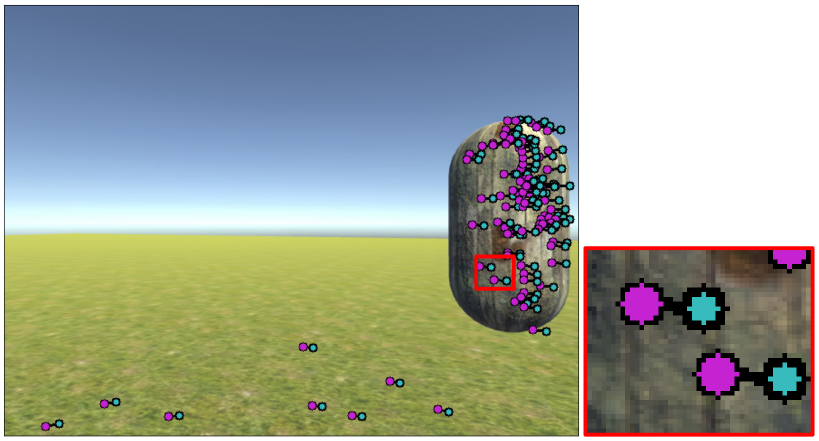

Figure 29: Rotation with no angular compensation.

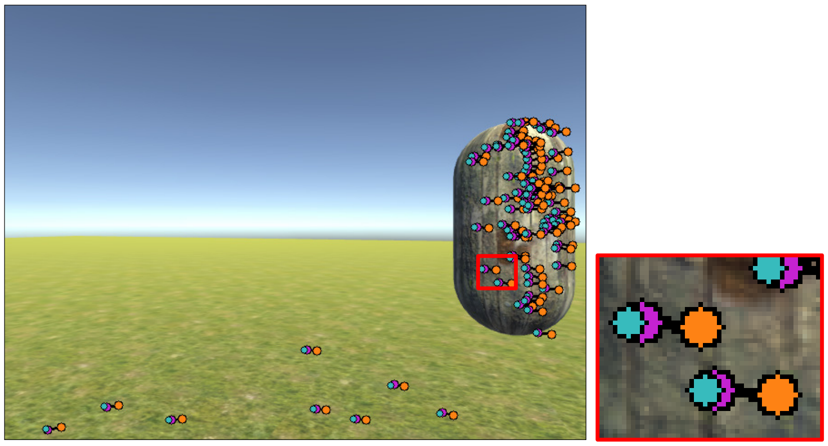

Figure 30: Rotation with mathematical angular compensation.

Figure 29 is the result of zero compensation for rotation, and leaves the cyan points perfectly covering the orange points. In Figure 30 a rotation of 1.05 degrees about the y-axis is compensated for. This compensation moved the cyan points to roughly the position of the purple points. Zooming in on specific points in the image (marked by the red block) it is clear that there is a slight miscalculation somewhere in the process.

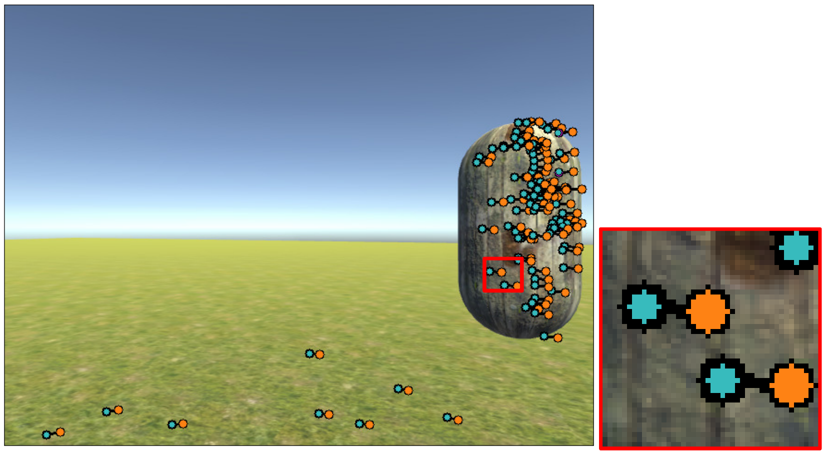

Figure 31: Rotation with adjusted angular compensation.

Table 2: Rotation compensation test results.

Test	Average Dist. (Incl. and excl. outliers)		Number of Points (incl. outliers, excl. outliers)		Rotation around Axes		
	Incl.	**Excl.**	**Incl.**	**Excl.**	**x[°]**	**y [°]**	**z [°]**
1	0.778	0.329	321	316	0	0	-8.02
2	0.565	0.277	328	322	0	0	8.02
3	0.057	0.057	376	376	1.4	0	0
4	0.065	0.065	440	440	-1.72	0	0
5	0.428	0.428	410	410	0	1.05	0
6	3.207	1.518	283	274	0	5.6	0
7	2.730	2.656	66	65	-0.7	0	-8.02
8	3.757	3.090	222	169	-1.4	3.84	0
9	12.07	-	408	0	-1.4	-4.9	0

By iteratively adjusting the focal length to compensate for a possible inaccurate Field of View rendering by the simulator, the results in Figure 31 were achieved with a value of $f = 417$. The zoomed-in side image shows a near perfect match for this situation. To further analyse the accuracy of the compensation for rotation, the average distance between the newly calculated point (p2') and the original point (p1) was taken for a variety of scenarios showed in Appendix A, the results are shown in Table 2.

In the nine tests in Table 2 any point where the distance from p1 to p2' was greater than 5 was considered to be an outlier. In the first five tests the results are consistently accurate with the exception of blatant outlying points. Upon closer inspection the outliers were detected as incorrect matchings from the optical flow calculation process.

The difference between tests 4 and 5 indicates how the magnitude of rotation affects the accuracy. In test 5 the larger rotation angle causes the optical flow calculation to have lower accuracy and produce more mistakes (in the form of outliers). It makes sense that with lower point position confidence from optical flow, the distance between the mathematical prediction and the intended target point would increase. This is confirmed in the increased *Average Distance* value in test 5.

Tests 7, 8 and 9 have rotation in more than one direction. From the results it is clear that the error size is a lot greater than the sum of the two errors in similar single angle tests. It is difficult to give an exact reason for these unexpectedly large error values from the data in the table. An interesting observation can be made in test number 9 where all the points appear to be wrong by a similar amount, meaning that more accurate results could probably be obtained by adjusting the formulae with some value.

As the goal of the thesis is to test the general potential of optical flow in this system, it was decided that the overall results for compensating for rotation was sufficient to give a fair result in later full system tests.

4.2.8. X and Y Translation

Quadcopters are able to move in any combination of their 6 dimensions of freedom therefore the X and Y translations were also analysed. Translating solely in one of these directions makes it possible for the quadcopter to predict the distance of the different pixels that are being tracked similarly to the Z axis translation described earlier. These translations are however simpler because of their straightforward flow and translation size to depth ratio.

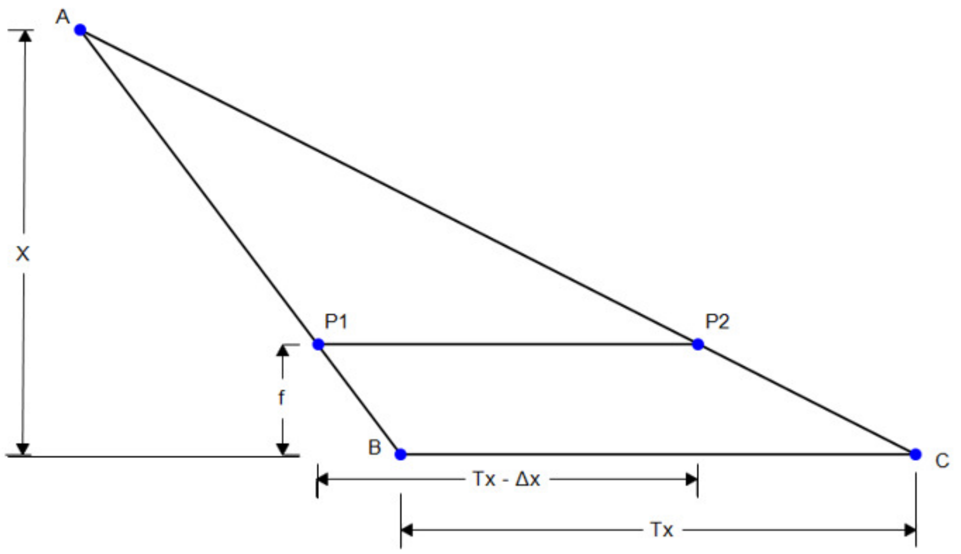

Figure 32: Geometry for x-y translation depth calculation.

In Figure 32 points B and C are the focal centres of the image planes containing points P1 and P2 respectively. Both P1 and P2 are projections of the same real world point A at different times and locations. In this situation the camera only moves on the X axis (from B to C) and the change in location between the two images is given as T_x. When the camera translates on the X axis in a static environment, optical flow is only observed in the X direction opposite to that of the camera motion. The distance between the two points P1 and P2 can therefore be given as $T_x - \Delta x$. In Figure 32 f indicates the focal length from the image planes to the focal centres and X indicates the distance from the focal centres to the real world point's plane. Using similar triangles it is possible to extract the following ratio:

$$\frac{f}{X} = \frac{\Delta x}{T_x} \tag{4.17}$$

Reorganising to find a value for X gives:

$$X = \frac{T_x f}{\Delta x} \tag{4.18}$$

To test the performance of this equation a single object was placed in a simulated environment and a virtual quadcopter was flown first left and right (on the X axis) and then up and down (on the Y axis). The results are shown in Figure 33 and frames 15 and 290 are shown in Figure 34 respectively. In the first 200 frames the movement was only on the X axis and in the rest of the frames the movement was only on the Y axis. The points show great consistency and clearly depict a stable distance for the object. It is not clear from the results exactly how far off the precision of the distance estimates are because of the difference between the distance to the object plane (ground truth value) and the surface of the object (measured by optical flow). These questions could be more concerning at a later stage of implementing this technology, but at the early stages the high consistency and moderate accuracy are good enough to move onto further testing.

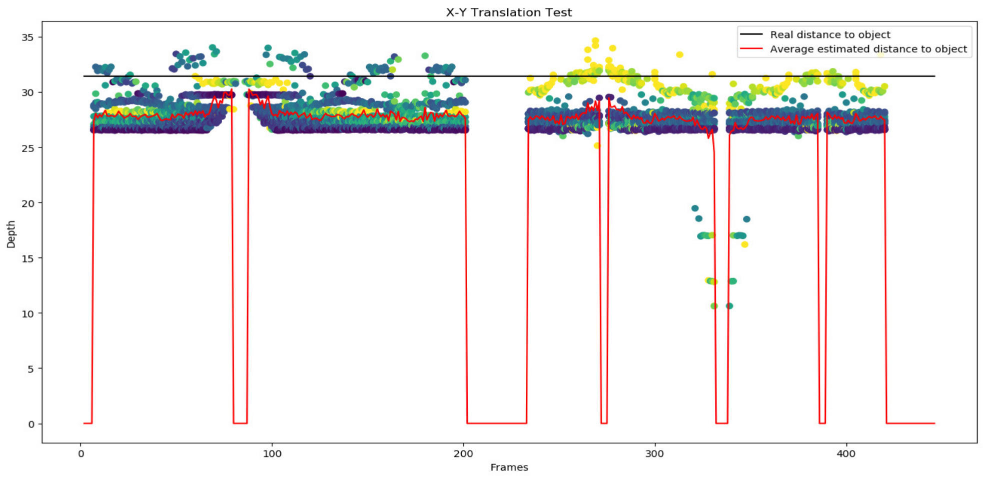

Figure 33: Depth translation test 1 in x and y directions.

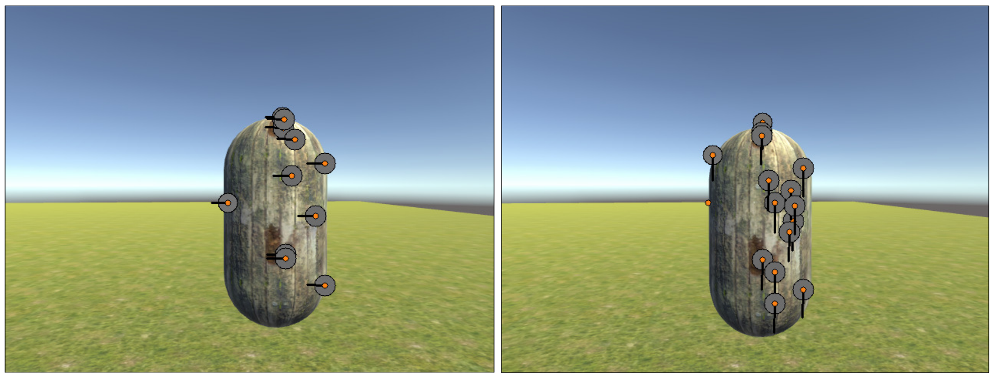
Figure 34: Depth translation test 1 visualisation.

The next test environment contained four different obstacles at varying depths and with varying textures. A similar flight path to the previous test was taken and the results are shown in Figure 35 and the visualisation of frames 97 and 430 in the left and right of Figure 36 respectively. The overall results were the same as with the single obstacle test for each individual obstacle, with each obstacle's depth estimation consistent and slightly inaccurate. It was found that estimation of the obstacles' depth could be improved by increasing the focal length value, but due to the difference between ground truth and expected optical flow results mentioned earlier, the focal length was left at 417.

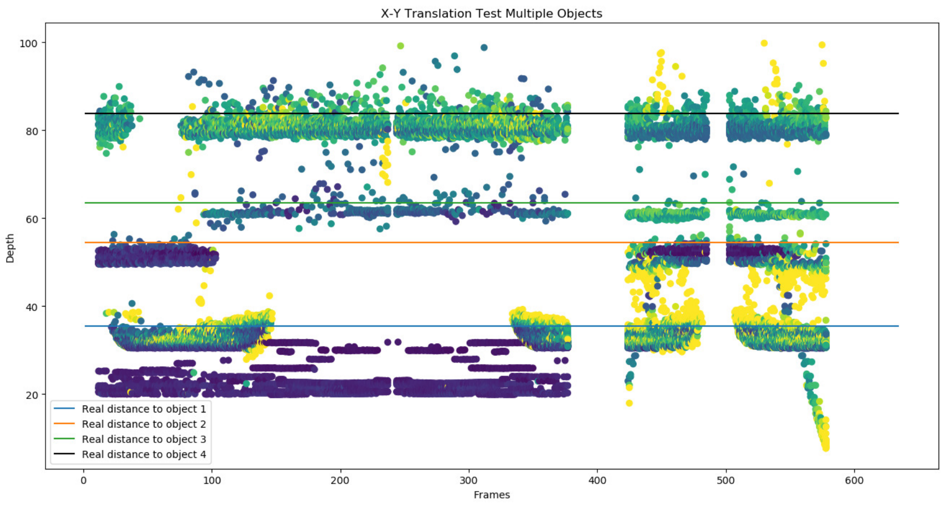

Figure 35: Depth translation test 2 in x and y directions.

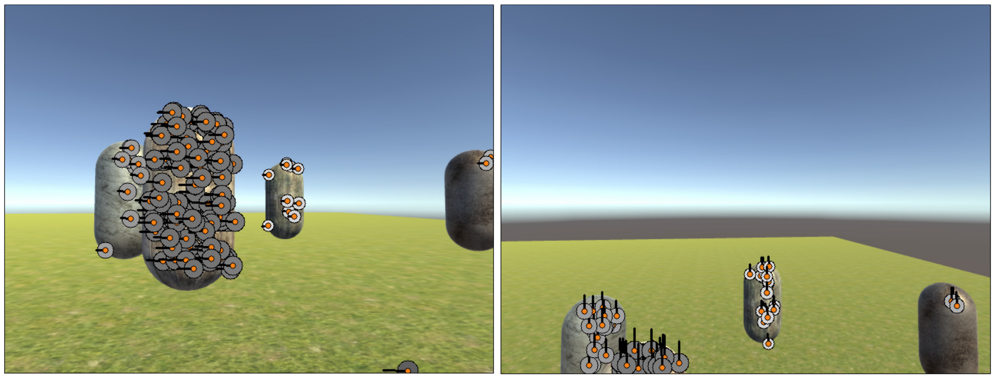

Figure 36: Depth translation test 2 visualisation.

5. Testing the System

In this section the system designed in Chapter 4 would be tested in a range of aspects. All tests were done in two stages. In the first stage they were created in the simulated environment and all test data was stored to a folder. In the second stage the data in the folder was used as input to the system. The system recorded all its calculations and plotted it on a graph against the predicted results from the data file. The main goals of the tests were:

- To determine if the system could function in a simple environment.
- To establish a baseline of the system's performance capabilities.
- To determine how different variables would affect the system and what limitations they could potentially place on it.

5.1. Performance Test

In order to later apply this technology in a real environment the system would need to reach a certain level of processing speed, accuracy and robustness. The tests were performed on a laptop with an Intel Core i7-4702MQ CPU (2.2 GHz) processor.

In the first test, the simulated quadcopter flew in a random path in front of a single object. The goal was to determine what processing speed could be achieved at different stringency settings while visually analysing the accuracy. This would allow us to determine a solid baseline which could be used in further testing to draw comparisons.

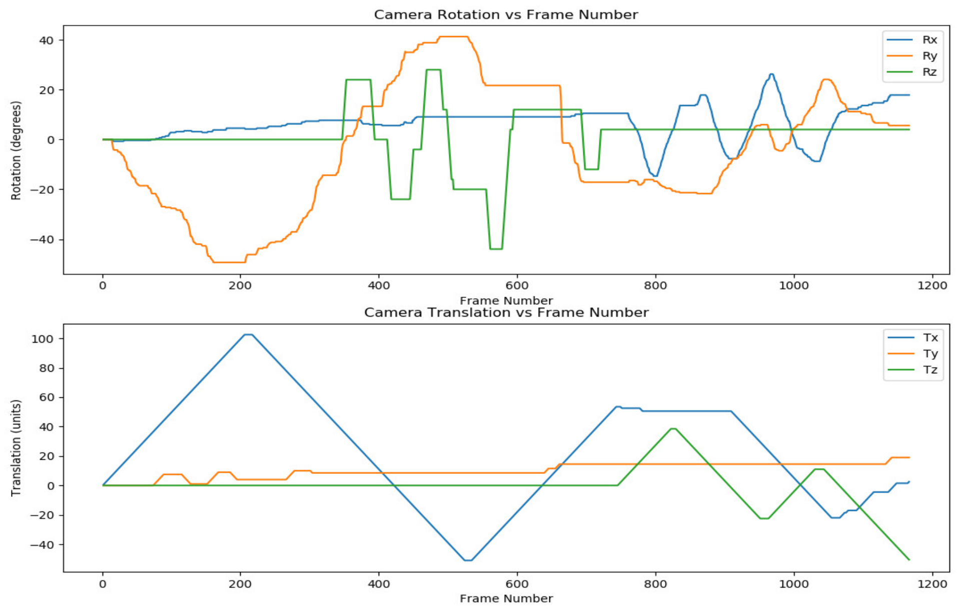

Figure 37: Random motion test orientation per frame.

The movement vector input of the random motion was recorded and is shown in Figure 37. The random movements were created in the simulator by a human operator, trying to imitate manoeuvres typical of a quadcopter in normal operation. The movements were kept fairly conservative, with no rotation going past 50 degrees and no abrupt turns.

In Figure 38 the resulting depth points are plotted for every frame, on the same graph as the real distance to the object. By analysing this graph one can clearly see an object being tracked at roughly the same distance as the target. The points around the target line all fall within a small band, this could again be due to the surface having a different distance to the object's measured plane. Similar to before, the lower points represent locations on the grass. Arguably the only major discrepancy could be found at frames 875 to 900 where the distance is overestimated. It is uncertain from the results whether the mistake is because of an incorrectly tracked group of points or a certain combination of movements.

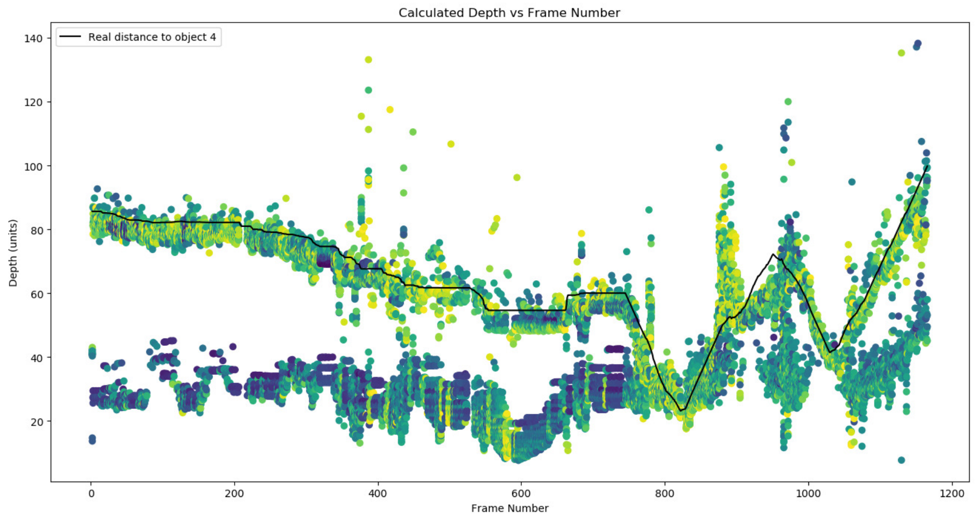

Figure 38: Random motion test results.

The rest of the results can be found in Appendix B. Although the results became more densely populated as more points are allowed through the filters, the depth estimation of the object remained consistent and true. It was decided that the accuracy of this first overall system test was good enough to use its data as a baseline.

In Table 3 a detailed breakdown of the performance over the variations of the test is given. In each variation either the threshold or maximum allowable error values were adjusted. This in turn allowed the FAST algorithm to find more points for the Optical Flow calculations to pair up. A large variety of tests were done to determine how fast the system could perform with a small number of good points on the one end, while also establishing the trade-off on performance to calculate an increased number of points on the other.

In the first two tests only the best points are selected and in turn the calculations reach speeds of over 140 FPS. In this ideal case where there was only one object and a neutral, bland environment, the limited number of points was enough to predict the depth of the obstacle. This information is useful because it showed that if the object could be isolated and only a limited amount of points placed on it, there would be great potential for fast cycle times.

Table 3: Performance test results.

Test	**Thresh. and Max Error Val.**		**Time (average per frame) (s)**				**Points found per frame**	**Points used per frame**	**Avg. FPS**
	Th	**Er**	**FAST**	**O.F.**	**Depth**	**Total**			
1	53	4	0.0032	0.0027	0.0005	0.006	8.1	4.5	153
2	45	4	0.0033	0.0033	0.0007	0.007	17.1	10.5	140
3	40	3	0.0034	0.0035	0.0010	0.007	29.8	14.8	123
4	40	4	0.0034	0.0035	0.0013	0.008	29.8	20.0	122
5	40	5	0.0034	0.0035	0.0016	0.008	29.8	23.4	120
6	35	4	0.0034	0.0040	0.0031	0.010	54.2	38.9	95.4
7	30	4	0.0037	0.0048	0.0049	0.013	98.0	75.6	74.5
8	25	4	0.0039	0.0057	0.0064	0.016	144	119	62.4
9	20	4	0.0040	0.0056	0.0069	0.016	151	128	60.7
10	15	4	0.0050	0.0067	0.0093	0.021	204	177	47.5
11	10	4	0.0071	0.0111	0.0198	0.038	436	388	26.3
12	5	4	0.0110	0.0183	0.0385	0.067	830	746	14.8
13	4	4	0.0148	0.0202	0.0456	0.080	921	842	12.4
14	3	4	0.0140	0.0228	0.0520	0.088	1008	923	11.3
15	2	4	0.0146	0.0240	0.0557	0.094	1092	997	10.6
16	1	4	0.0158	0.0247	0.0575	0.097	1169	1056	10.2

In tests 3, 4 and 5 the threshold was held constant and the maximum allowed error value adjusted. There was a slight difference in calculation time due to less/more time being spent doing depth calculations, depending on the allowed error level. In these tests the depth calculation times were much faster than the other calculations and therefore not a big difference in performance was seen. This would however be different in later tests where the depth calculation time is the longest of the three phases. Selecting a single, correct, error value limit would not be optimal as the ideal target changes for different environments and scenarios. The best solution would therefore be to determine a dynamic threshold and error value limit combination which would give you the desired amount of points and thus the desired FPS.

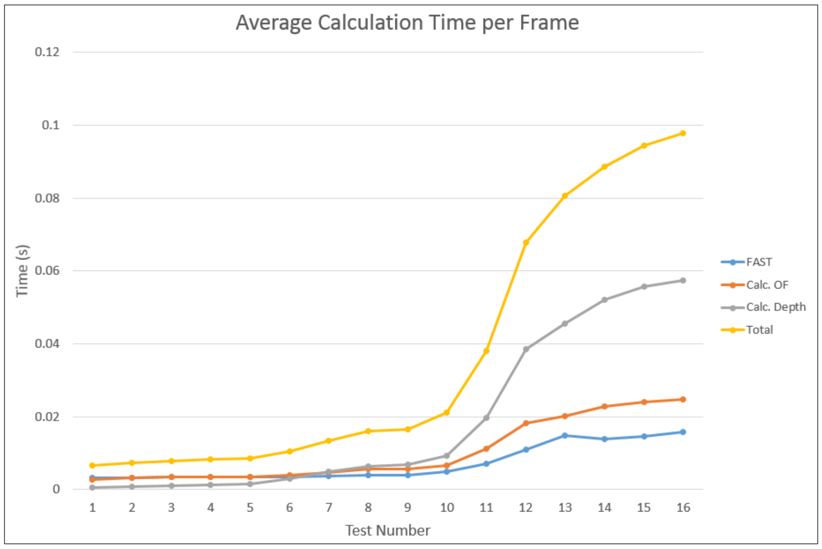

Figure 39: Graph of average calculation time per frame.

For the rest of the tests the impact of an increase in points was investigated. In Figure 39 and Figure 40 the shapes of the total time and the number of point's lines are similar. Closer inspection of Figure 39 reveals that the total time-shape follows the depth calculation line-shape. The values of the other two lines also increase as the points increase, but with a much more gradual slope. Considering that the FAST and Lucas and Kanade Pyramid functions are part of a highly optimised library it makes sense that they efficiently handle larger amounts of data. On the other hand the depth calculations were done in Python where each point was individually calculated. It is estimated that by fully optimising the depth calculations it will be possible to get the duration down to below that of FAST and OF.

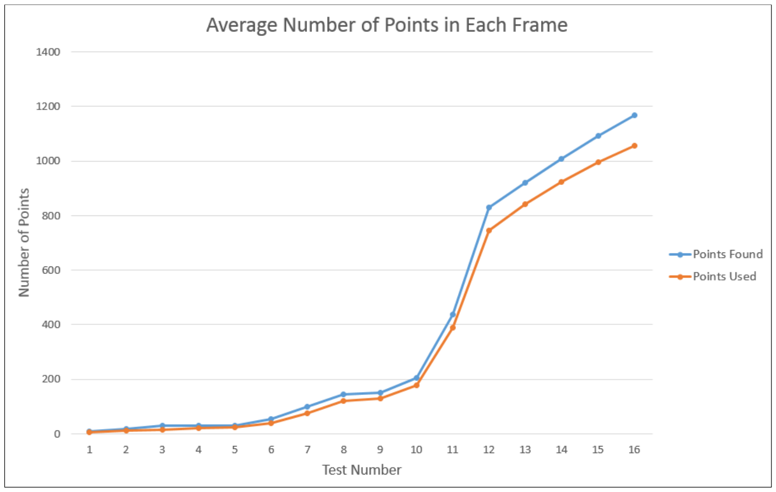

Figure 40: Graph of average number of points per frame.

5.2. Manipulating Test Variables

In this section a single controlled test was created with three objects at varying distances as shown in Figure 41. In the base test the quadcopter moved in straight lines on the three movement axes as shown in Figure 42. Slight variations were then introduced to simulate problems that could be encountered in the real world. These variations included: angular instability, velocity, brightness levels, and inaccurate input movement variables. The results of these variations were recorded and compared to the results of the base test in Figure 43. The impact of each variable was then further discussed.

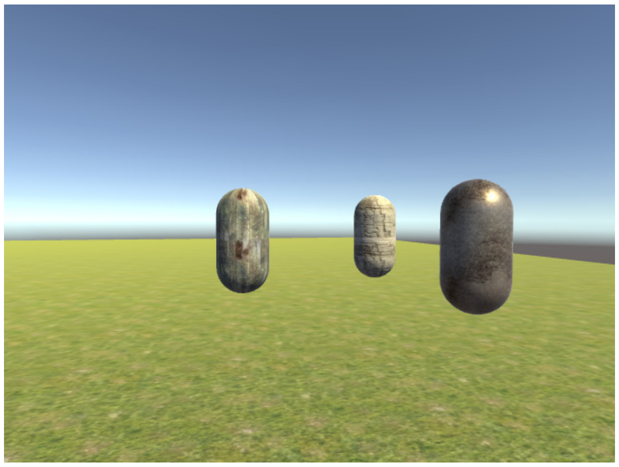

Figure 41: Manipulating variables test visualisation.

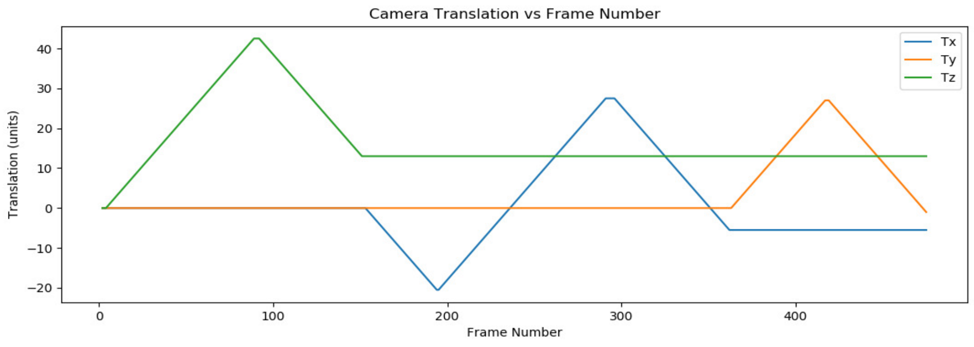

Figure 42: Manipulating variables test movement.

A few things could already be noted from the results of the base test in Figure 43:

1. The different textures of the objects had an impact on how many points were detected on them. For this reason the middle and furthest object had substantially more points than that of the closest object.
2. The distance of the object was directly related to the accuracy of the depth prediction. The furthest object had a wide band of points around the predicted distance where the closer objects had noticeably tighter bands.
3. Another observation was that the forward and backward motion on the z-axis in the first 150 frames produced less accurate predictions than the other directions. This is due to multiple optical flow directions being involved in the calculation of z-axis motion, where only one direction is involved when moving parallel to the image plane.

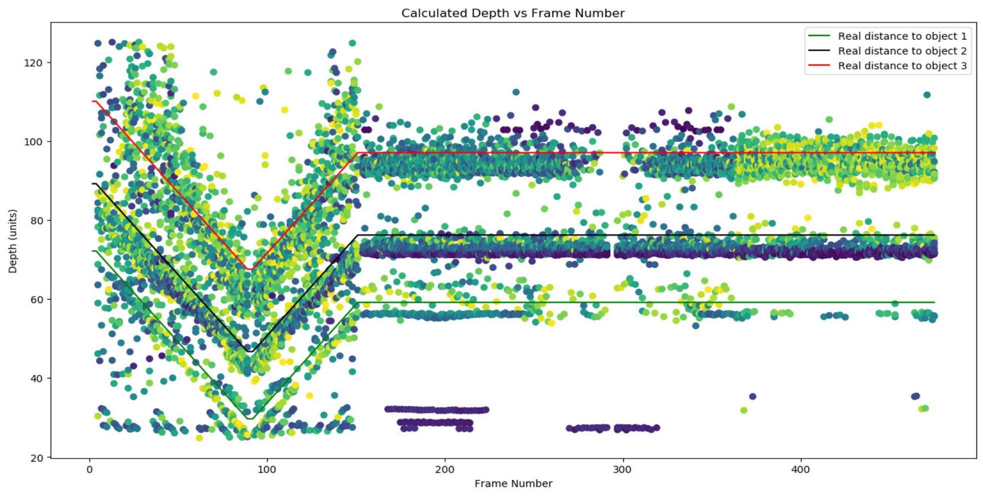

Figure 43: Manipulating variables test base results.

5.2.1. Angular Instability

In these tests the three angles of the camera were offset by a random number from zero to x degrees in every frame. These small angles were recorded and given as input to the system. The values for x in tests 1 to 6 (shown in Appendix C) are: 0.00°(base), 0.5°, 1.0°, 1.5°, 2.0° and 2.5° respectively. To put this into perspective, the stability of a quadcopter was analysed by Kuantama, et al. (2018) and their results showed a variation in the pitch and roll angles of less than 0.2° while hovering, and 0.5° while at 80% throttle.

In the tests there was a clear pattern of accuracy deterioration as the angular instability increased. Even though the real angular movement of the quadcopter between frames was known, the optical flow calculations became excessively inaccurate with the fast jittering motion from test 4 (1.5°) onwards.

The overall results of these tests were considered to be extremely positive. It was shown that by correcting for the unstable motions, it was possible to generate acceptable depth predictions of the objects at instabilities greater than that experienced by a quadcopter at 80% throttle. It should also be taken into account that these instabilities do not consider any camera stabilisation and could be considered worst case scenario.

5.2.2. Distance between Frames

In optical flow, distance travelled over time is irrelevant. The optical flow between two images will remain the same as long as their camera positions stay static, irrespective of the time taken to get from position A to position B. In the first set of tests the distance between frames was repeatedly increased, and in the second set they were made smaller.

Increased Distance between Frames:

The distances per frame in test 1 to 6 (shown in Appendix D) were: 0.5, 1.0, 1.5, 2.0, 2.5 and 3.0 units respectively. These values can be put into perspective by Figure 44 which shows how the distances relate to speeds in km/h for different frame rates.

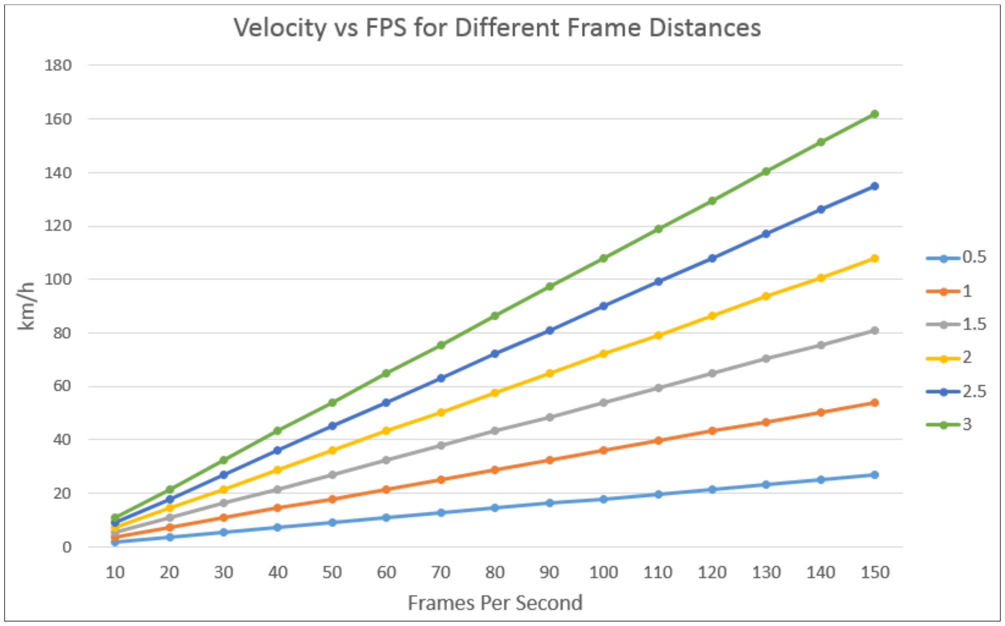

Figure 44: Velocity vs FPS for different frame distances.

The results suggest that the accuracy of the optical flow calculations that were used remained true, however the amount of successful matches greatly decreased as the distance increased. It can be argued that out of the results shown, the more accurate depth predictions came from the tests with larger distances. This happened as the outliers from tests one became unmatchable, or matched with too great an error, leaving only the strong matches behind.

From these tests we learn that to keep optical flow stable and predictable the distance between frames needs to be controlled. In order to do this at varying speeds the frames per second need to be adjustable. This could be achieved by recording at the highest desired frequency and then skipping frames at lower speeds to stabilise the movement per frame.

Decreased distance between frames:

The distances per frame in tests 1 to 5 (shown in Appendix E) are: 0.3, 0.25, 0.2, 0.15 and 0.05 units respectively. Because of how slowly the camera had to move, and the limited amount of ram to cache all the images beforehand, a similar but shorter test was created for this scenario. The flight path of this test is shown in Figure 45.

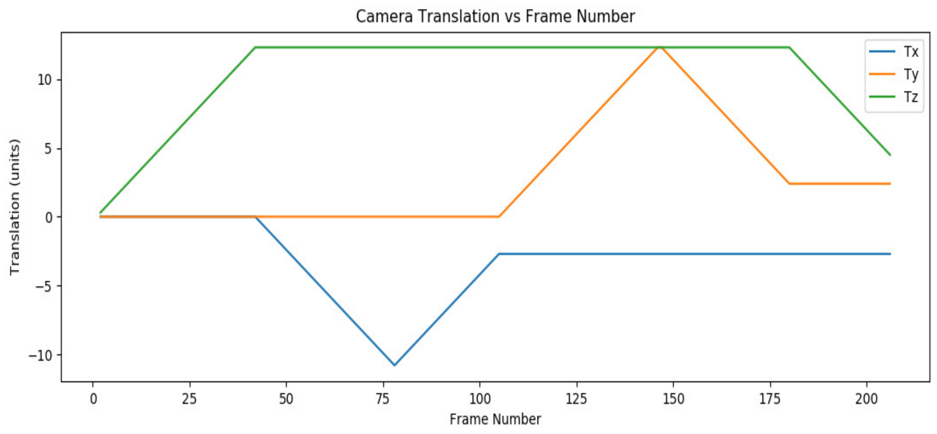

Figure 45: Decreased distance between frames test movement.

The results from these tests show that there are limitations to optical flow with very small movements between frames. The motion on the z-axis at the beginning and end of the tests gives very little useful data with only the middle object visibly being detected and a high amount of points with random depth values. The motion on the other axes gave fairly accurate depth estimations up to speeds of 0.2 units per frame. After this point the data started to spread out and no longer convincingly predicted the distance of the objects.

Comparing this set of results with the larger inter-frame distance results, it is apparent that optical flow operates better at higher distances for these given scenarios. The optimal distance is somewhere between 0.5 and 1.5 units per frame.

5.2.3. Different Brightness

In these tests the level of light was reduced repetitively to determine how strongly lower lighting conditions affected the system. To achieve these lighting levels the light source in Unity was adjusted to produce the brightness as indicated below each of the six sets of results in Appendix F.

A similar result of increased accuracy due to increased difficulty was achieved in the first few tests. In test 4 the limit is reached and the quality of depth predictions starts to decrease.

Given that optical flow is directly calculated from difference in light intensity it was predictable that these tests would fail at a certain point. We can however still learn from the manner in which they start failing and use it to generate ideas when coming up with solutions for handling each variable.

5.2.4. Inaccuracy in Rotation

Using a simulator which can perfectly predict the accuracy of the movements made by the camera, between frames, helps us to learn about the capabilities of optical flow. It is however not realistic to assume that using this technology in real world applications would be that easy. There are currently no devices that can measure quadcopter movements with pin-point accuracy and these tests were performed to determine how inaccurate measurements would affect the system's ability to estimate the depth of pixels.

In order to test for rotation the 0.5° angular instability test was used as a base. Tests 1 to 6 in Appendix G had random rotational input errors of up to: 0%, 2%, 4%, 6%, 8% and 10% respectively. There was a clear gradual movement from accurate to inaccurate predictions for each point as the error values were increased. Once the errors were at 8% the depth estimations were spaced out to the point of no longer producing accurate information. These results pose a strong threat to the ability of optical flow being useful in the field, as it requires the orientation measurements to be very accurate.

5.2.5. Inaccuracy in Translation

The reasoning behind the translation tests was the same as the rotation. The only difference between the two sets of tests is that the base used in this set was the same as the original base test. The error percentages were kept the same as with the rotational inaccuracy tests and the results are shown in Appendix H.

The values again gradually moved from their correct positions to a widening band around the initial prediction lines. As expected the results of this test-set closely resembled that of the rotational inaccuracy with a slightly greater resemblance of a direct 2-10% variation on the base results. These results further highlight a possible hurdle for optical flow in the real world, with its stringent requirements on accuracy for normal operation.

6. Discussion and Estimating Potential

In this section all the important findings and decisions throughout the system design and test phases are reviewed. The different aspects that could lead to success or failure are compared and weighted, to develop a case for the overall potential of this system to succeed as a navigation tool for a drone.

Dense vs Sparse:

Early in the design phase the decision was made to use sparse optical flow functions in the system. This decision resulted in sacrificing scope for performance. During the testing phase it was determined that this choice was the correct one, as the strong points (with high confidence in accurate optical flow values) provided the useful data for depth estimation. The weaker points had lower accuracy levels and oversaturated the data, drowning out the correct estimations.

FAST:

The decision to make use of FAST over Shi-Tomasi for detecting points to analyse, was again based on calculation speed. The result of the speed tests in Table 3 confirms the FAST algorithm is the correct one to use when working with a small number of points. Unlike Shi-Tomasi, it scales with points and allows for extremely fast calculations with low point numbers. Shi-Tomasi would greatly slow down the overall performance when only a low amount of points are involved.

As more points get selected the optical flow and depth calculations take much longer. When the other calculation times begin to increase, there is no longer an urgent need for such fast point detection times.

During the system test process it was found that there is often a case of too many points being selected for specific objects and no points being selected on others. Although not optimal, the points selected were good enough to produce useful test results and allow for conclusions to be drawn based on the optical flow and depth equation performances.

Optical Flow Calculations:

The optical flow calculations were done using the Lucas Kanade Pyramid algorithm. During the design phase it was shown that by carefully selecting the parameters of this algorithm it was possible to obtain highly accurate results, as reflected in Figure 18 to Figure 20. The processing speed of the algorithm also performed adequately during the speed tests presented in Table 3. The calculation times matched those of the FAST algorithm for a small amount of points and increased gradually as the number of points increased. No real areas for improvement or negative impacts were found in this section.

Depth Calculations:

Great success was found during the design phase of the depth calculations. It was firstly proven that by using the ratio in Equation 4.12 in both the x and y direction it was possible to fairly accurately determine real world depth of individual pixels when travelling on the z-axis (Figure 24).

Secondly it was proven that by applying the Euler angles in the opposite direction to the actual rotation experienced between the two frames, it was possible to negate the effects of rotation on the optical flow values (Figure 29 to Figure 31). It was therefore possible to carry on measuring depth when travelling in the z-axis and rotating the view at the same time.

Thirdly, it was proven that by using the ratio in Equation 4.18 (derived from the geometry in Figure 32) it was possible to accurately predict the real world depth of pixels when travelling in the direction of the x- or y-axis (Figure 33 and Figure 35). It was further shown that for a simple situation where a quadcopter is randomly flying around in front of a single obstacle the distance of that obstacle is constantly known regardless of the combination of movements carried out (Figure 38).

Although these tests were carried out under perfectly simulated conditions with completely accurate input data it was important for the system to perform well at this stage to have any chance of moving forward with the project. The positive results gave the system a foundation from which to work and conduct analysis.

Testing:

After the system was created it was put through a series of tests to determine where it could potentially reach points of failure.

In the first test-set described above, with the quadcopter flying in a random path in front of an obstacle, the run was performed multiple times with different threshold settings. This produced results showing how much the frames calculated per second dropped for the amount of points calculated. In Table 3 the frame rate ranged from 153 to 10, as the amount of points detected increased from 8 to 1169. At a later point during the different velocity tests it was shown that the accuracy was greatest for 0.5 to 1.5 units per frame. Using the simulator description for a ratio of 10 units = 1m, and the middle value of 1 unit per frame, it can be predicted that the system would work with maximum speeds of 55 km/h down to 3.6km/h depending on the number of points being calculated. This set of tests proved the system had potential for real time application, without the need for optimisation when only calculating small amounts of points. If a large number of points need to be calculated, some form of optimisation might be necessary, but real time operation should still be possible.

The rest of the test-sets involved isolating variables to determine their impact on results. In the first set, angular instability was added to the tests, creating a wobbling effect. The results of these tests showed that the compensation for angular rotation could in theory negate effects greater than that normally experienced by quadcopters, provided the exact disturbances were known. This means it may be possible to achieve reasonably accurate optical flow depth estimations from a camera mounted directly to a quadcopter, without any complex camera stabilisation systems.

In the second test-set, larger and smaller distances between frames were tested. It was found that in general the larger distances lead to more accurate points, but only a limited amount passed the conditions. When analysing the shorter distances there were a lot more points meeting the requirements, but the overall accuracy was reduced. The results did not produce a single, perfect distance which had the optimal performance; instead it produced multiple different trends which could be used to construct a dynamic system. It is clear that the optical flow algorithms perform differently at different distances between frames and threshold setting combinations.

In the third test-set, the impact of different brightness levels was investigated. It was determined that although the system could function at all different light levels tested, the reduction in brightness did have a negative impact. At weaker lighting conditions the system struggled to detect a large amount of points, but accuracy stayed high. At stronger lighting conditions the accuracy also started reducing. The results therefore suggest that a system operating in a real world environment would need to accommodate for differing light levels. This could be done by adjusting threshold levels or adjusting the brightness of the incoming video feed.

In the fourth and fifth test-sets the reaction to inaccurate positional inputs was found. The results of both sets were moderately accurate for small deviations (0-4%) and quickly decreased to very inaccurate at 10% deviations. This would definitely be one of the main challenges to overcome when applying this system to a real world scenario. Because the drone is ideally moving only 1 unit per frame, which translates to 0.1m, a 4% error means a required accuracy of 4mm. Similarly for the angular measurements, the 4% error margin would leave little room for miscalculations.

Overall Performance:

The overall standard of performance of the system, was good under ideal conditions. When the conditions were altered and individual challenges introduced, the system held its own for smaller changes before reaching failure at some point in the test-sets. This proved that there was a certain level of robustness in the system and that when applied to a real world situation there would be potential for success.

Future Work:

Along with analysing its potential, the first steps towards applying optical flow in quadcopter navigation has been taken in this thesis. The next logical step would be to take the theoretical work done and apply it in a real world scenario. From there the different challenges identified in this thesis would need to be addressed. Once the system has been optimised for real world operation it can be tested as a tool for navigation.

As the system is not a fully functioning navigational system, a separate process would need to be undertaken is converting the depth values of the individual pixels into useful navigation information. This could be done directly with the depth information calculated or by merging it into a larger system.

Optical flow is directly related to the pixels in an image and a great example of integrating the system with other technology is therefore to use image recognition to determine which pixels belong to certain objects. A search algorithm could then be applied to the sub-images and an average of the pixel depth results taken to determine the distance to the detected object.

The nature of the data creates potential for a solution to be found using machine learning or neural networks. In doing so shifting into the very relative field of autonomy.

There are a range of different research paths which opened up from this project, ranging from integrating sensors, to real world application testing, to path planning using only the output data of the system, to artificial intelligence. A strong sense of future opportunities surrounding the project could be seen as a very positive indication of potential for the technology.

7. Conclusion

The objective of this thesis was to measure the potential of using optical flow in quadcopter navigation. This potential was measured by designing and testing an optical flow-based depth estimation system, using a combination of the latest optical flow research. The potential for the technology in this field therefore directly depended on the success of the system and the ability for its output data to be useful in quadcopter navigation.

The system proved through multiple simulated tests that it was capable of correctly estimating the depth of obstacles by analysing the optical flow and its own movement. The system could operate in all three movement directions and was able to compensate for rotation in all three directions. The system was able to perform its calculations in real time, running on the laptop as described. When individual non-ideal variables were introduced the system showed robustness by handling smaller deviations and only starting to fail when these variables reached certain stages.

The output of the system provided depth information to specific pixels of an image. This could be used in a navigation system to draw a depth map of its surroundings or to predict the depth of specific detected obstacles. It was therefore concluded that the depth estimation system has potential to be further developed in a real world application.

References

Baker, S. et al., 2011. A Database and Evaluation Methodology for Optical Flow. *International Journal of Computer Vision,* 92(1), pp. 1-31.

Beauchemin, S. & Barron, J., 1995. The Computation of Optical Flow. *ACM Computing Surveys,* 27(3), pp. 433-467.

Bouguet, J., 2000. *Pyramidal implementation of the affine Lucas Kanadefeature tracker description of the algorithm.* Intel Corporation Micro-processor Research Labs: s.n.

Brox, T., Bruhn, A., Papenberg, N. & Weickert, J., 2004. High Accuracy Optical Flow Estimation Based on a Theory for Warping. *Lecture Notes in Computer (Springer),* Volume 3024, pp. 25-36.

Camus, T., 1994. *Real-Time Optical Flow.* PhD Thesis, Brown University, USA: s.n.

Farnebäck, G., 2003. Two-Frame Motion Estimation Based on Polynomial Expansion. *SCIA. LNCS,* Volume 2749, pp. 363-370.

Fleet, D. J. & Jepson, A. D., 1990. Computation of Component Image Velocity from Local Phase Information. *International Journal of Computer Vision, 5:1,* pp. 77-104.

Gibson, J. J., 1950. *The perception of the visual world.* Oxford, England: Houghton Mifflin.

Guerrero-Lemus, R., 2012. *Concentrated Solar Power.* London: Springer.

Harris, C. & Stephens, M., 1988. A Combined Corner and Edge Detector. *4th ALVEY Vision Conference,* Volume 23, pp. 147-151.

Horn, B. & Schunck, B., 1981. Determining optical flow. *Artificial Intelligence,* Volume 17, pp. 185-203.

Hurd, B. M., 2013. *Control of a Quadcopter Aerial Robot Using Optic Flow Sensing, Unpublished master's thesis.* s.l.:s.n.

Kirchner, W. H. & Srinivasan, M. V., 1989. Freely flying honeybees use image motion to estimate object distance. *Naturwissenschaften, 76,* pp. 281-282.

Kuantama, E. et al., 2018. Flight Stability Analysis of a Symmetrically-Structured Quadcopter Based on Thrust Data Logger Information. *Symmetry (MDPI),* 291(10).

Low, T. & Wyeth, G., 2005. *Obstacle Detection using Optical Flow,* St. Lucia, Australia: s.n.

Lucas, B. & Kanade, T., 1981. An iterative image registration technique with an application to stereo vision.. *Proceedings of the International Joint Conference on Artificial Intelligence,* pp. 674-679.

Mammarella, M., Campa, G., Fravolini, M. L. & Napolitano, M. R., 2012. Comparing Optical Flow Algorithms Using 6-DOF. *IEEE Transactions on Systems, Man and Cybernetics,* 42(6), pp. 1752-1762.

Orozco, J. N. V., 2014. *Monocular camera based real-time on-board obstacle avoidance for unmanned aerial vehicles.* Master's Thesis, Università della Svizzera Italiana, Switzerland: s.n.

Patel, D. & Upadhyay, S., 2013. Optical flow measurement using Lucas Kanade Method. *International Journal of Computer Applications,* 61(10), pp. 6-10.

Raudies, F., 2013. Optic flow. *Scholarpedia,* p. 30724.

Romero, H., 2009. Real-time Stabilization of an Eight-rotor UAV Using Optical FLow. *IEEE Transactions on Robotics,* 25(4), pp. 809-817.

Rosten, E. & Drummond, T., 2006. Machine learning for high-speed corner detection. *European Conference on Computer Vision,* Volume 1, pp. 430-443.

Shi, J. & Tomasi, C., 1994. Good Features to Track. *IEEE Conference on Computer Vision and Pattern Recognition,* pp. 593-600.

Sobel, E. C., 1990. The locust's use of motion parallax to measure distance. *Journal of Comparative Physiology A, 167,* pp. 579-588.

Srinivasan, M. V., 1998. Insects as Gibsonian Animals. *Ecological Psychology, 10(3-4),* pp. 251-270.

Vedula, S. et al., 1999. Three-Dimensional Scene Flow. *7th International Conference on Computer Vision,* pp. 722-729.

Wedel, A. et al., 2006. Realtime Depth Estimation and Obstacle. *In K. F. et al., editor, Pattern Recognition,* pp. 475-484.

Zach, C., Pock, T. & Bischof, H., 2007. A Duality Based Approach for Realtime TV-L1. *Lecture Notes in Computer Science (Springer),* Volume 4713, pp. 214-223.

Zingg, S., Scaramuzza, D., Weiss, S. & Siegwart, R., 2010. MAV Navigation through Indoor Corridors Using. *IEEE International Conference on Robotics and Automation (ICRA).*

Appendix A: Rotation Compensation Tests

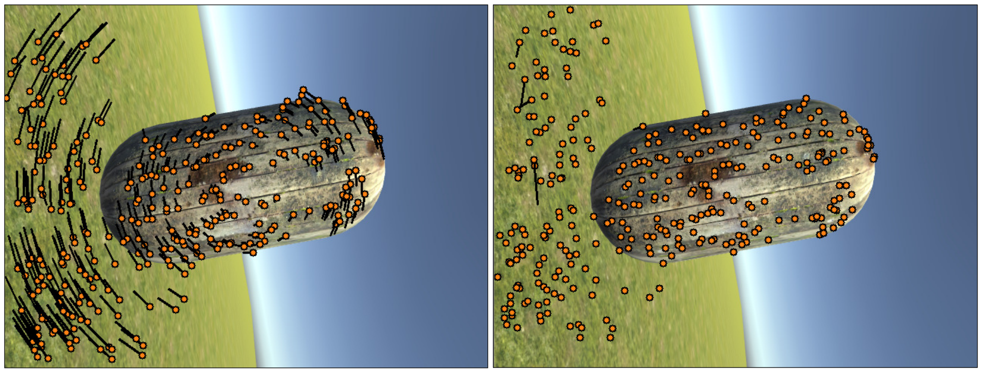

Figure 46: Rotation compensation test 1.

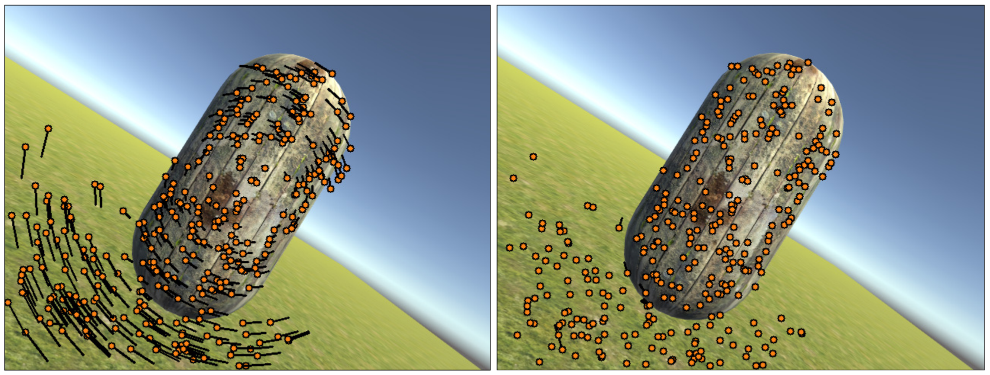

Figure 47: Rotation compensation test 2.

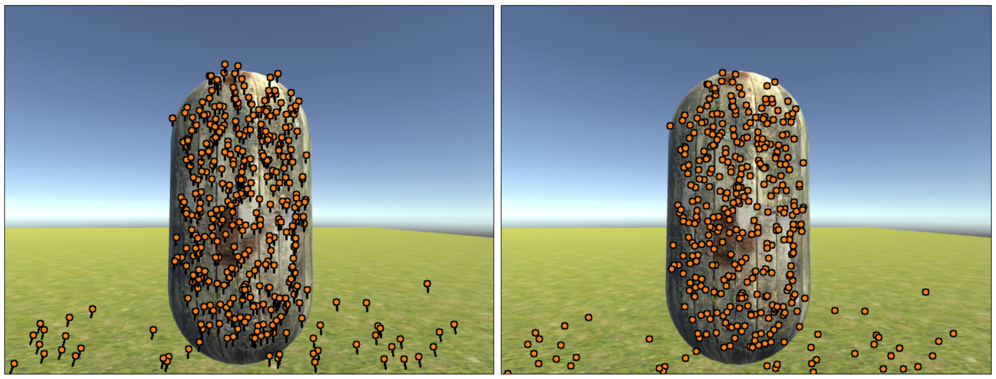

Figure 48: Rotation compensation test 3.

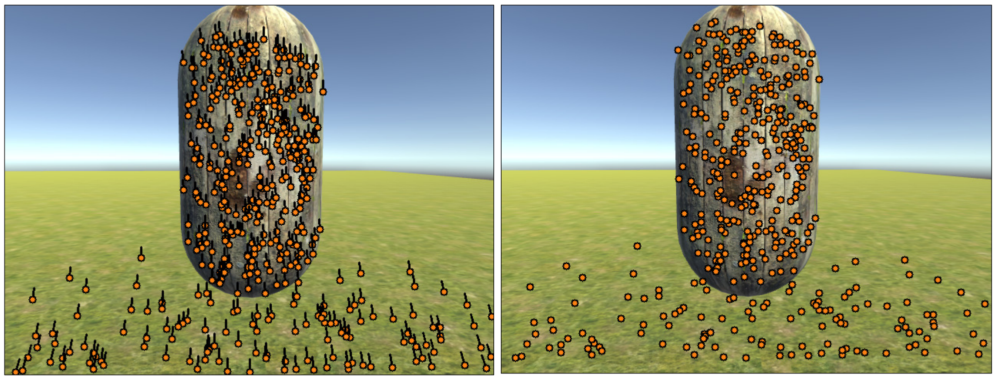
Figure 49: Rotation compensation test 4.

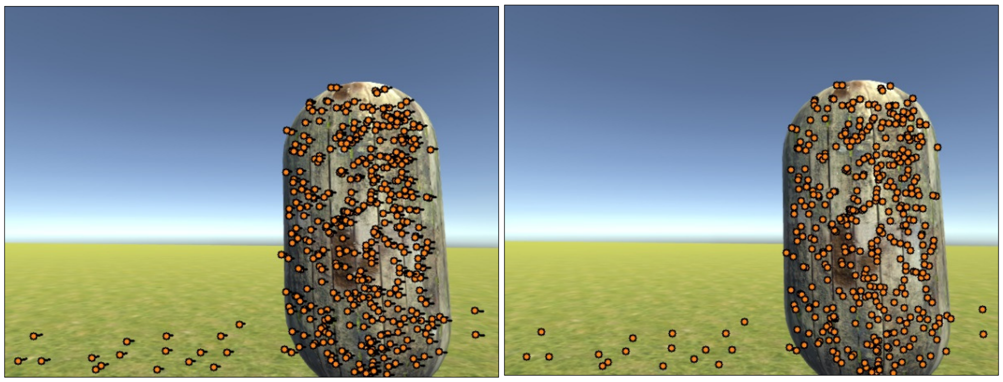
Figure 50: Rotation compensation test 5.

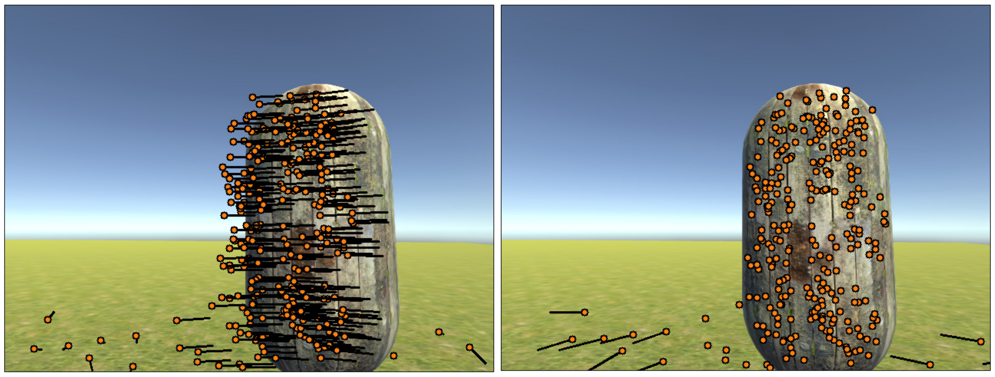
Figure 51: Rotation compensation test 6.

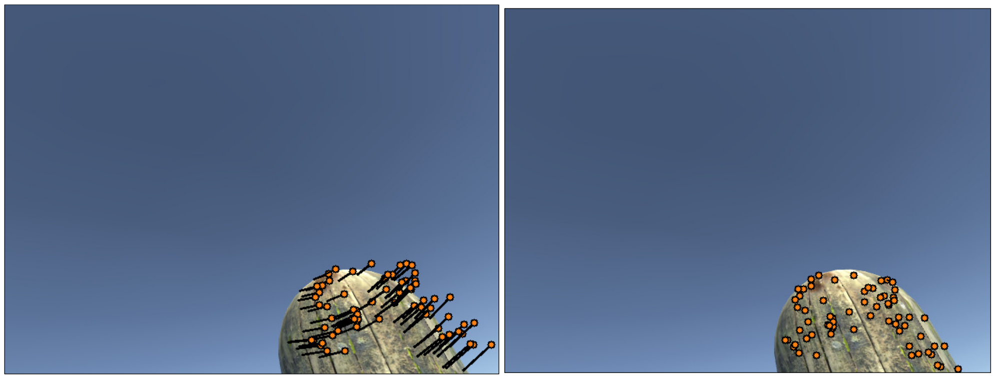

Figure 52: Rotation compensation test 7.

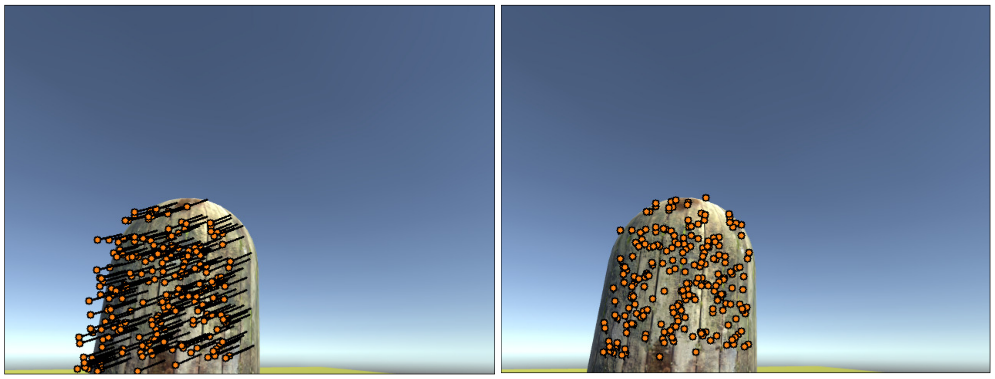

Figure 53: Rotation compensation test 8.

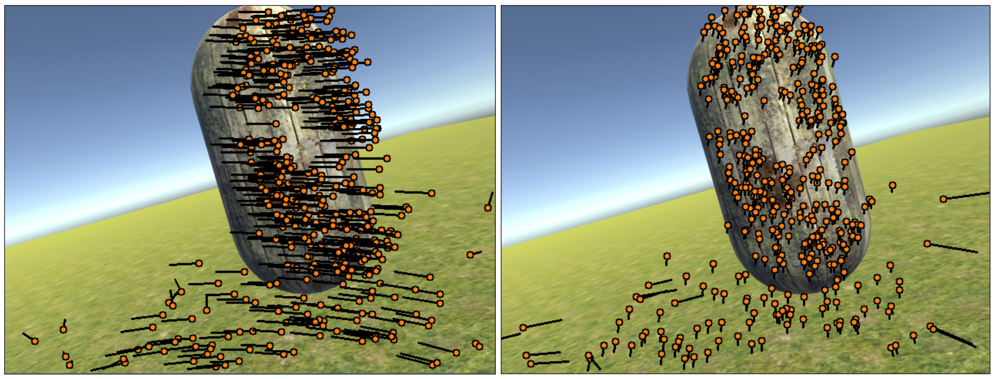

Figure 54: Rotation compensation test 9.

Appendix B: Random Motion Performance Tests

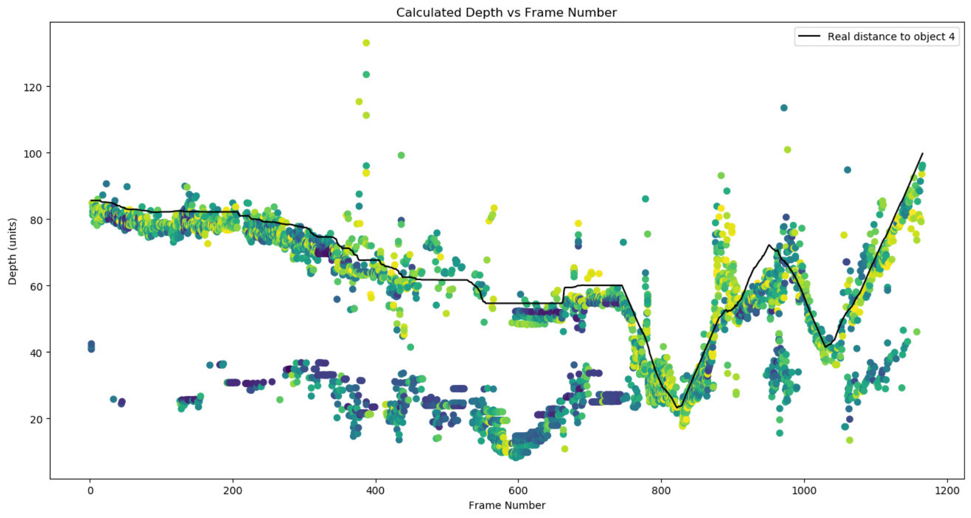

Figure 55: Random motion performance test 1.

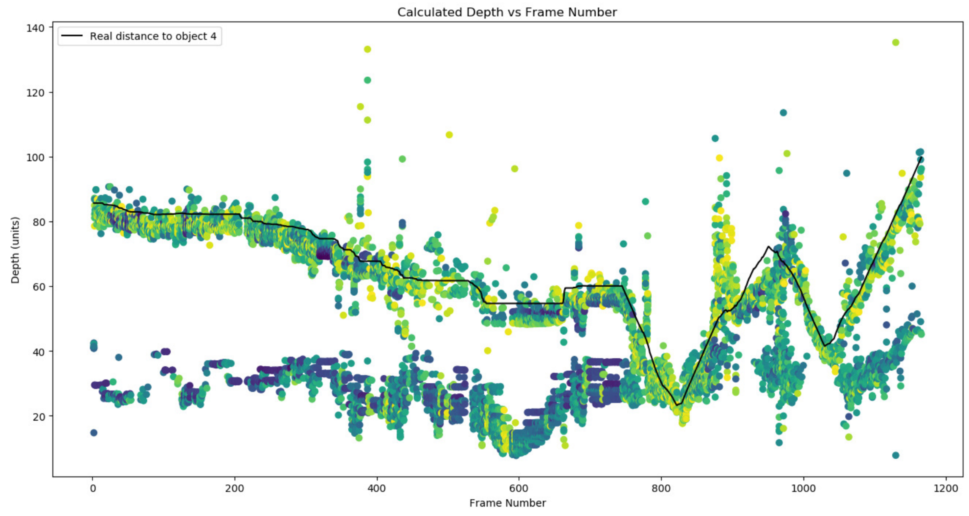

Figure 56: Random motion performance test 2.

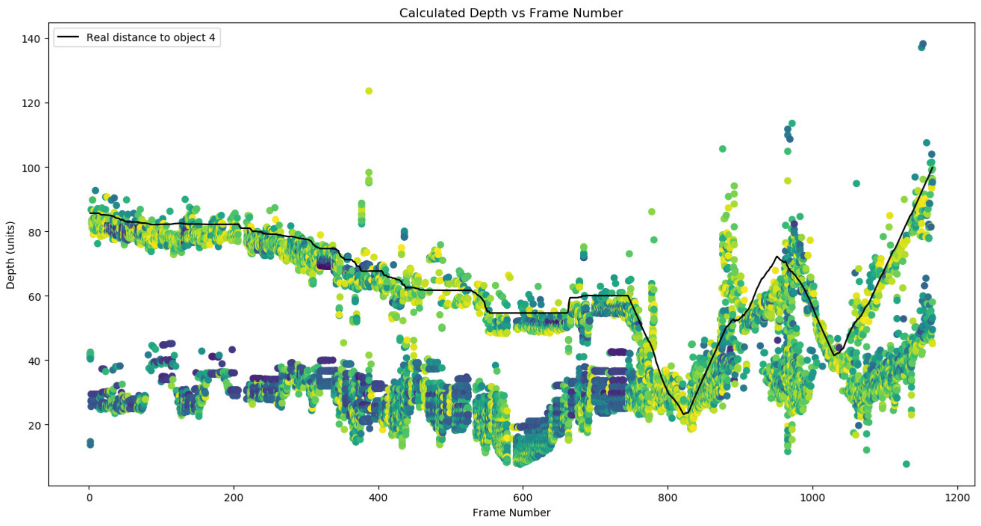

Figure 57: Random motion performance test 3.

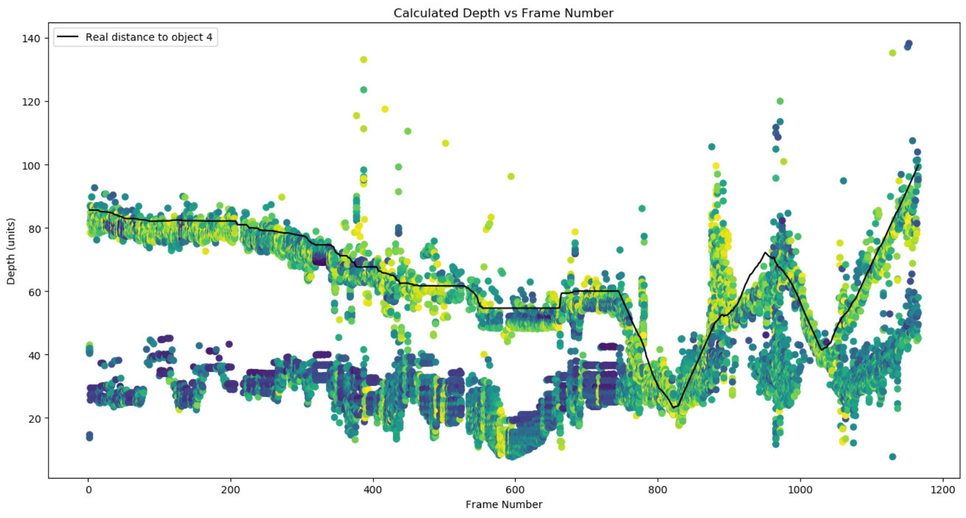

Figure 58: Random motion performance test 4.

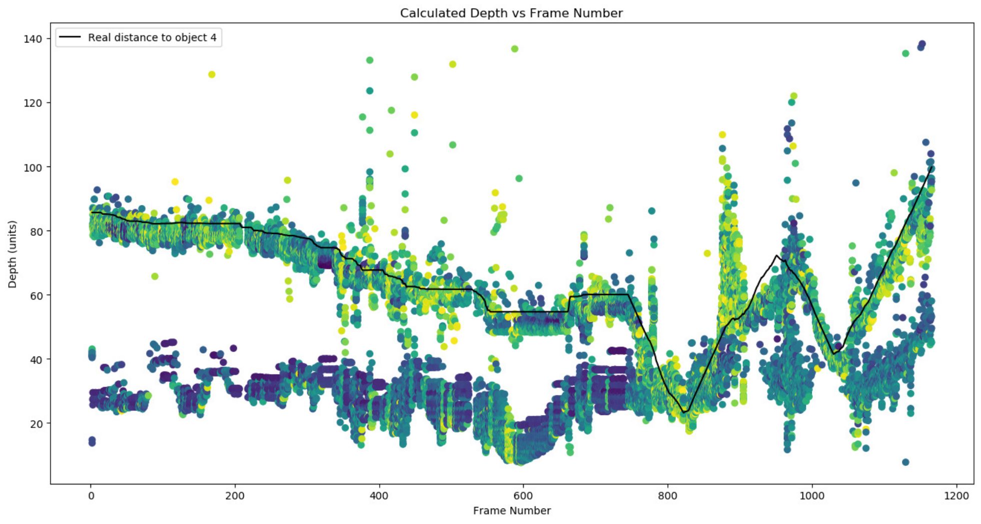

Figure 59: Random motion performance test 5.

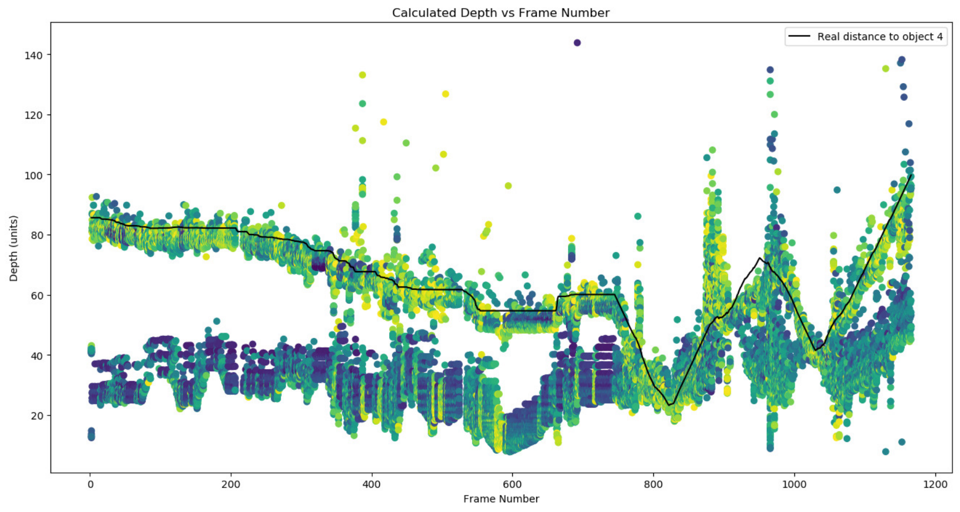

Figure 60: Random motion performance test 6.

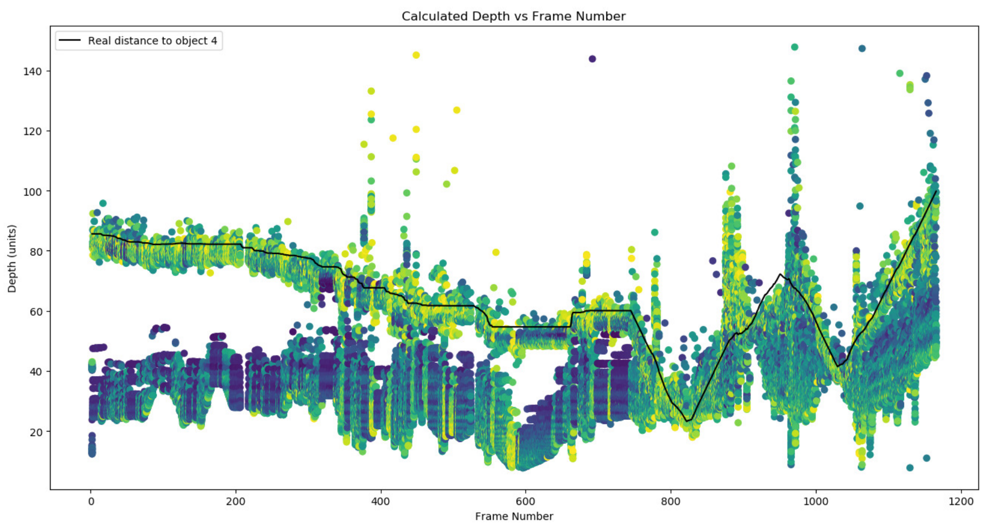

Figure 61: Random motion performance test 7.

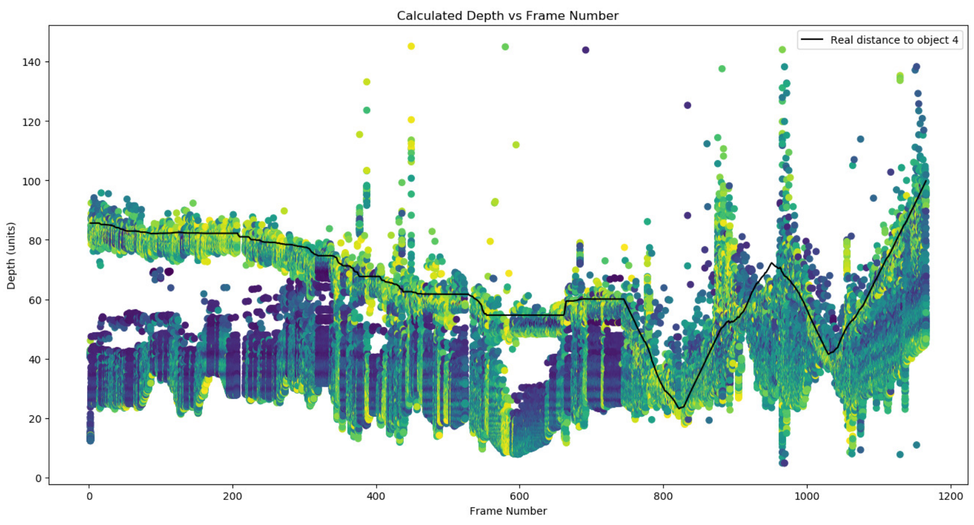

Figure 62: Random motion performance test 8.

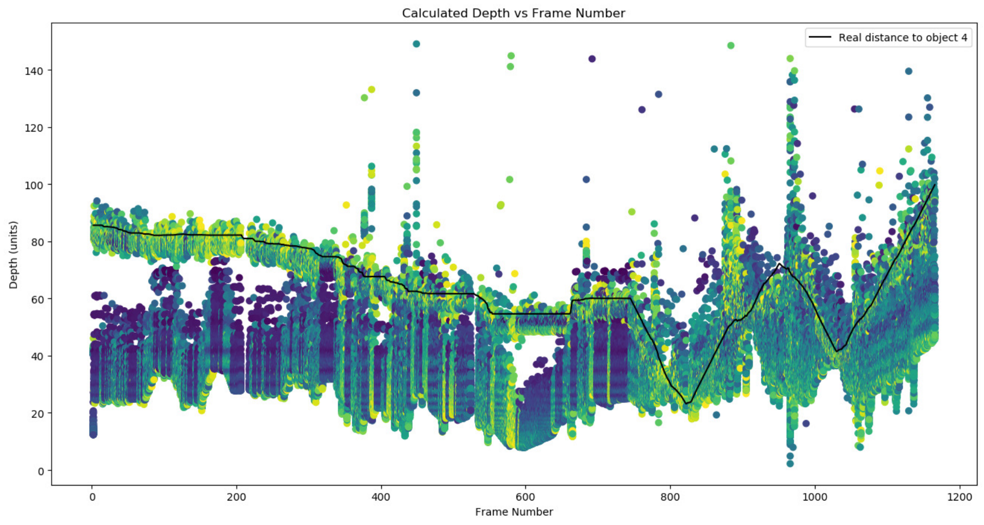

Figure 63: Random motion performance test 9.

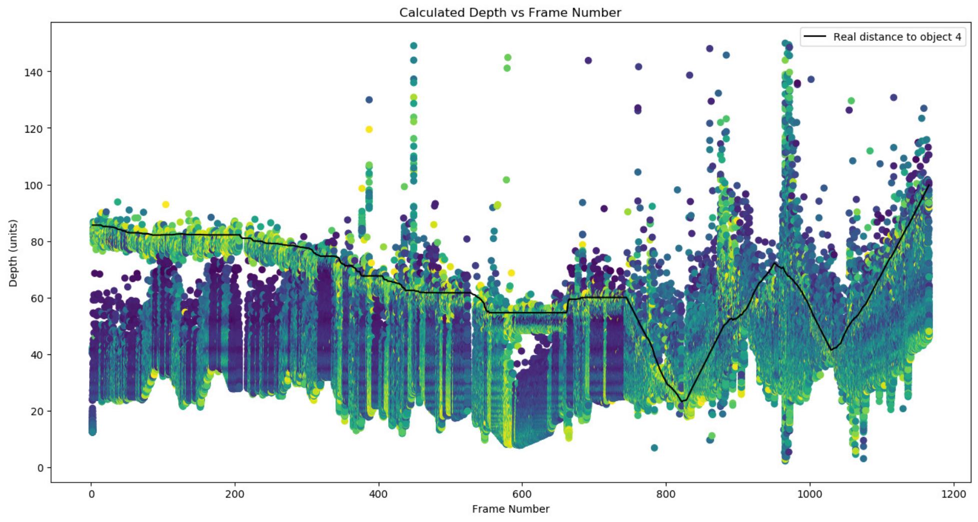

Figure 64: Random motion performance test 10.

Appendix C: Angular Instability Tests

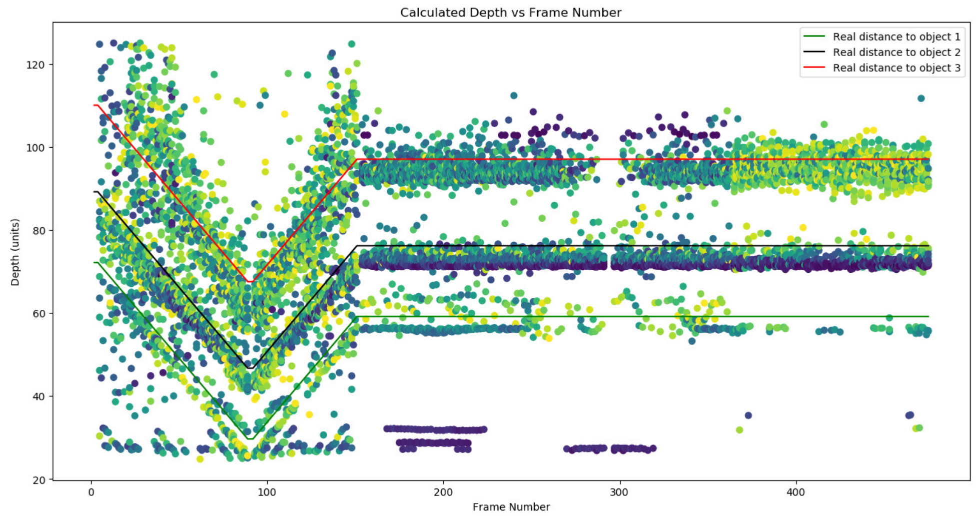

Figure 65: Angular instability test 1.

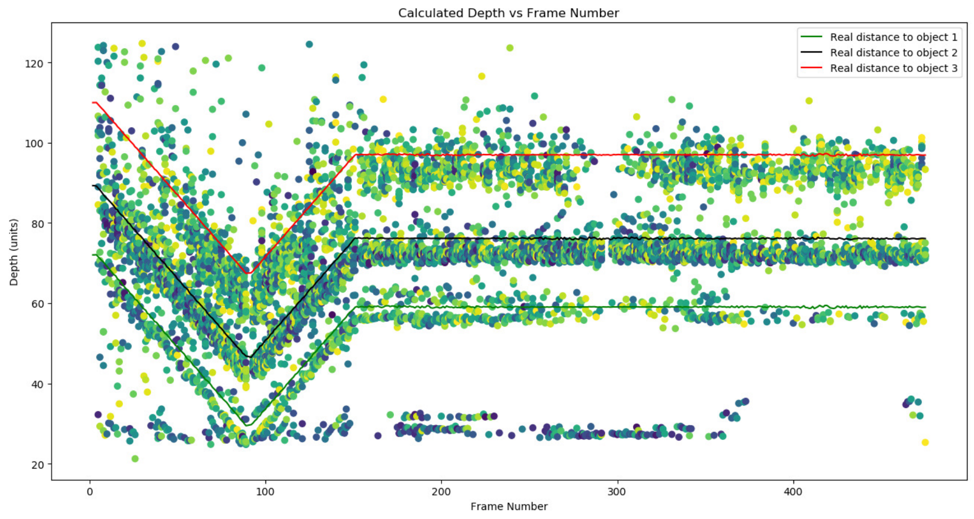

Figure 66: Angular instability test 2.

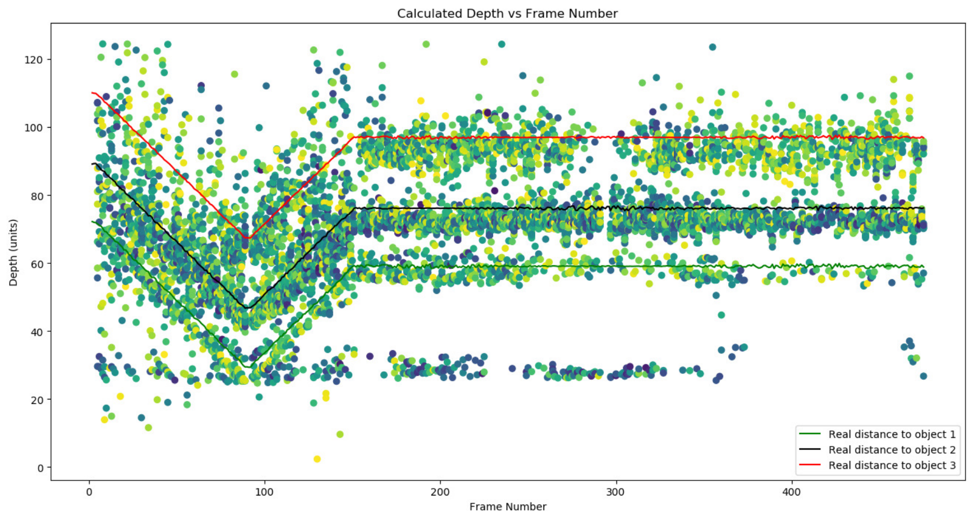

Figure 67: Angular instability test 3.

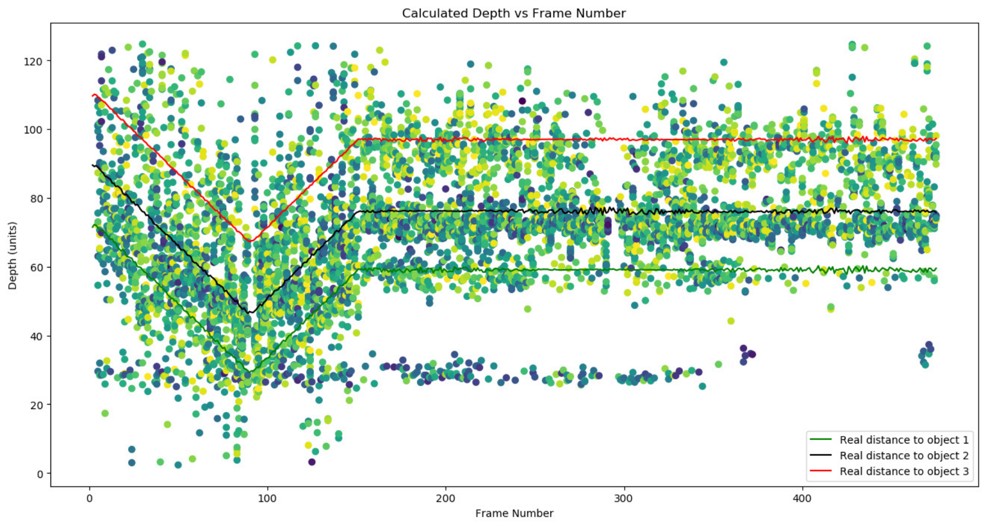

Figure 68: Angular instability test 4.

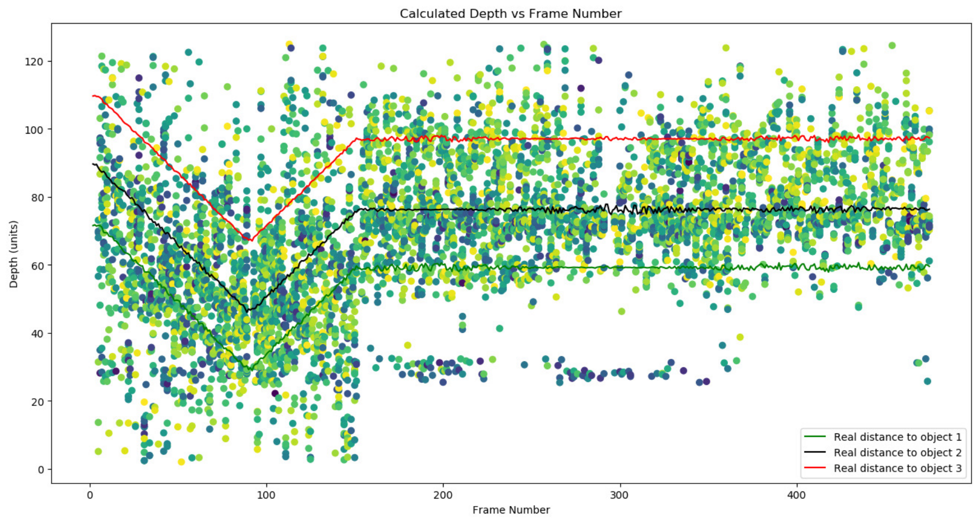

Figure 69: Angular instability test 5.

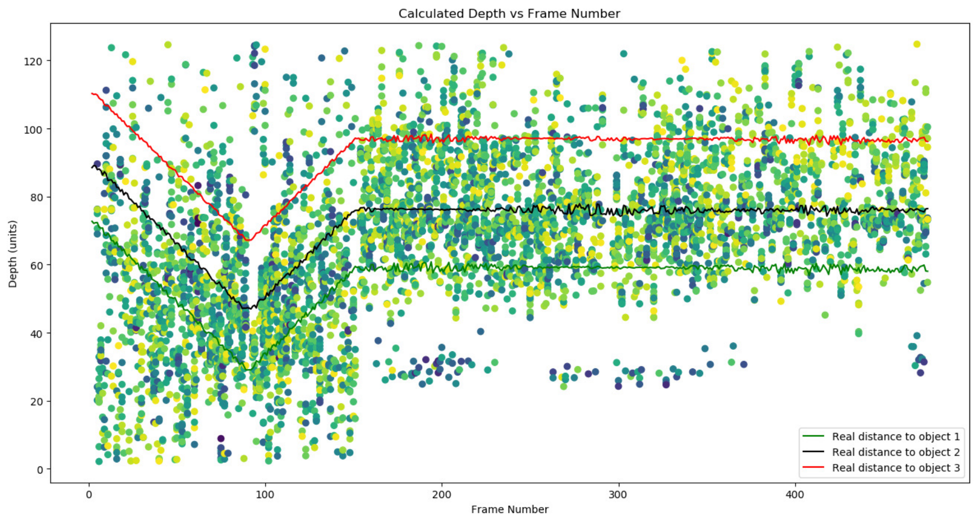

Figure 70: Angular instability test 6.

Appendix D: Increased Distance between Frames Tests

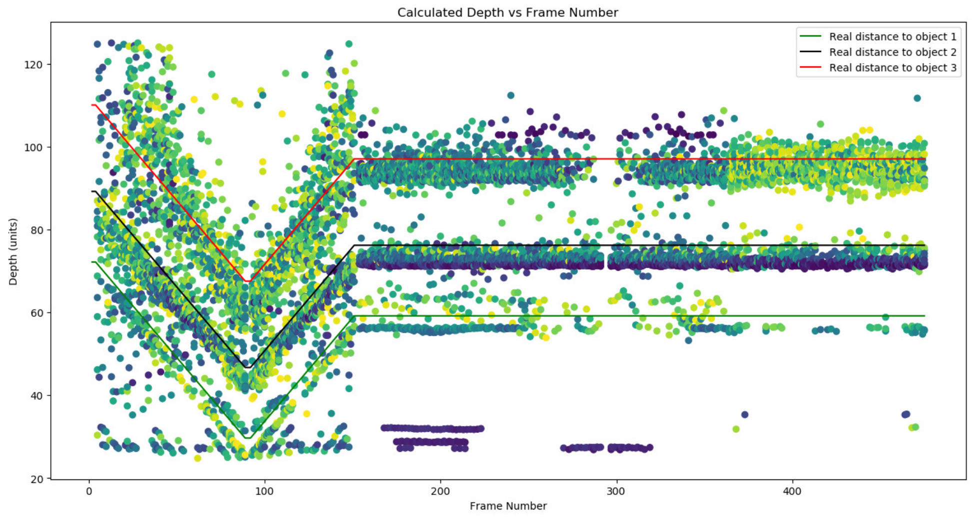

Figure 71: Increased distance between frames test 1.

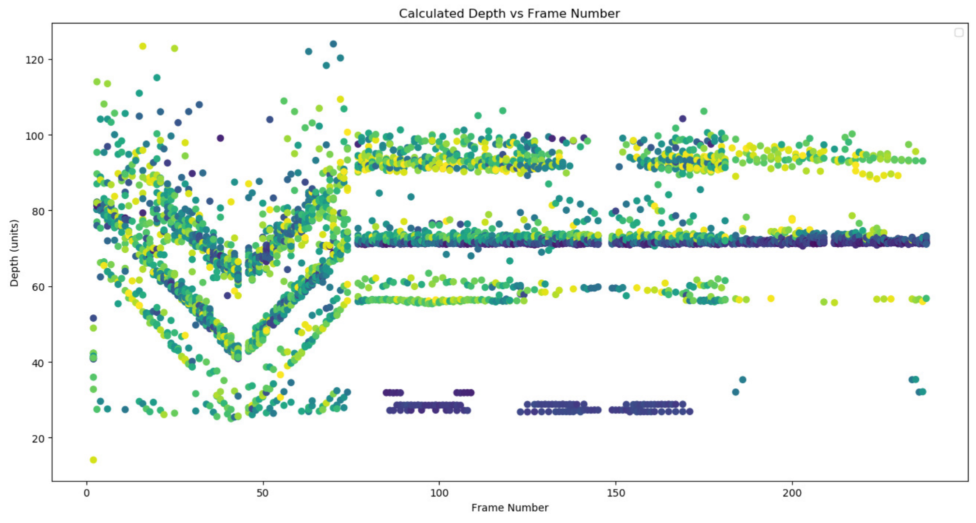

Figure 72: Increased distance between frames test 2.

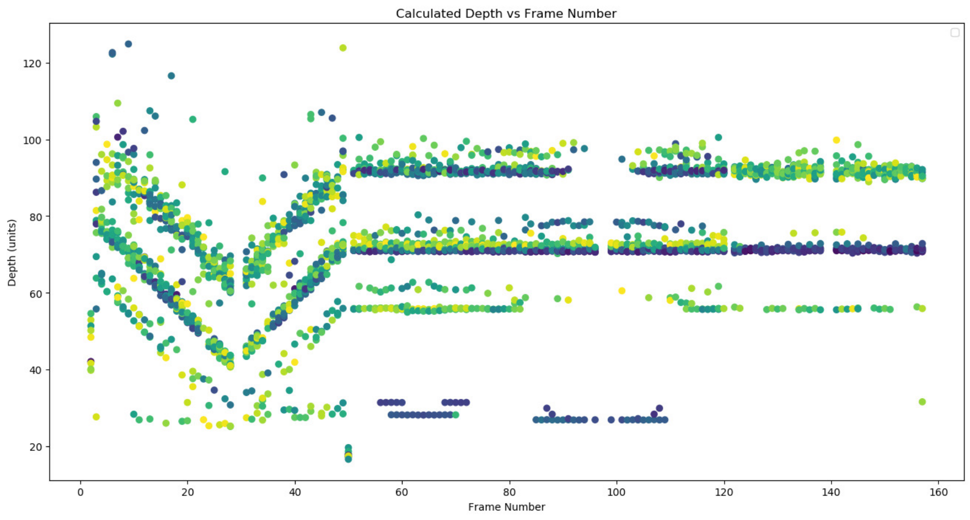

Figure 73: Increased distance between frames test 3.

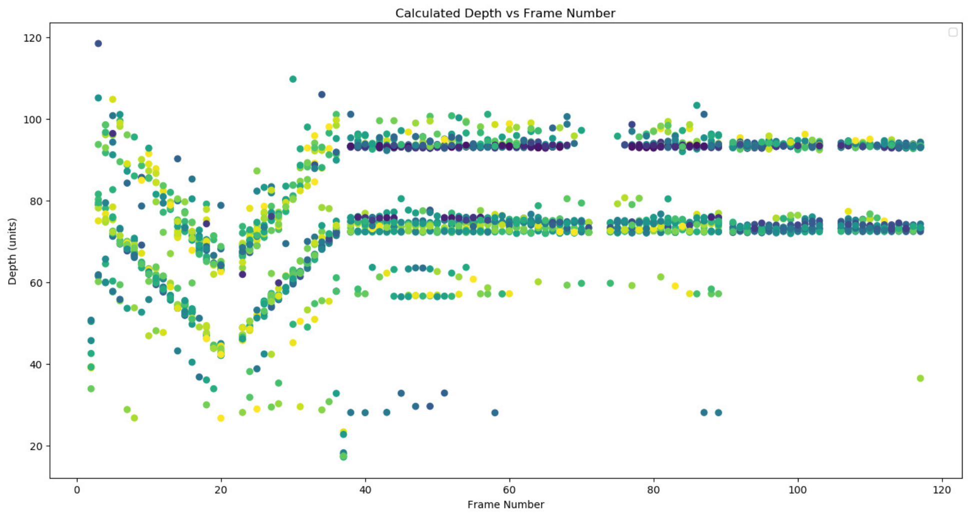

Figure 74: Increased distance between frames test 4.

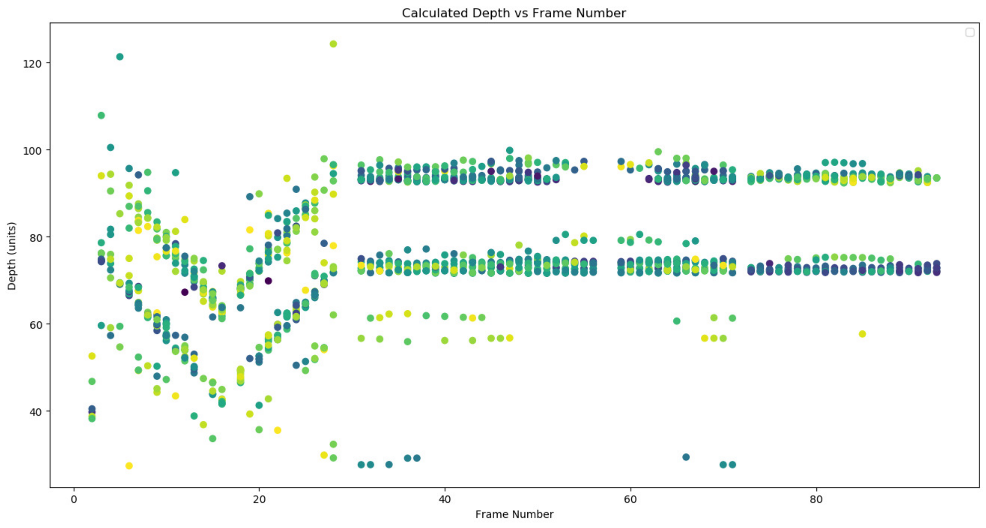

Figure 75: Increased detween frames test 5.

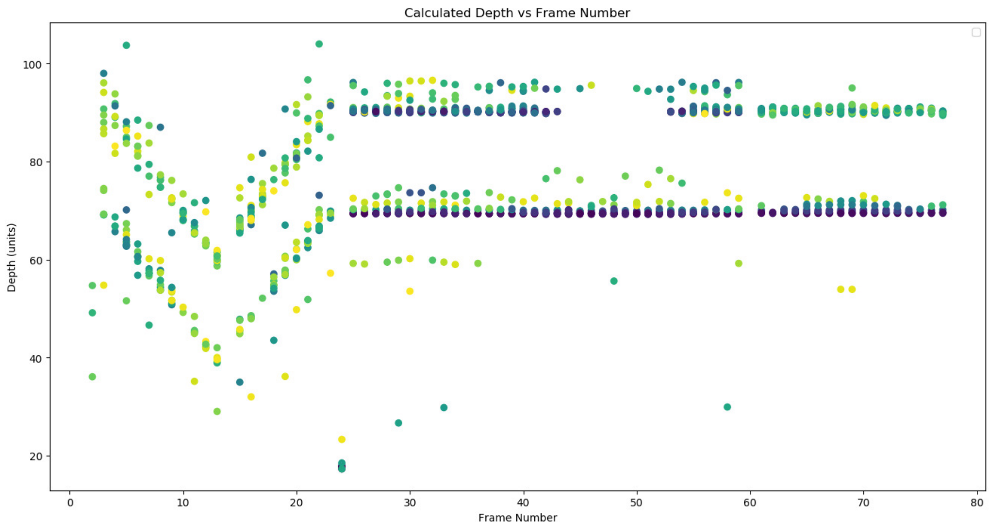

Figure 76: Increased distance between frames test 6.

Appendix E: Decreased Distance between Frames Test

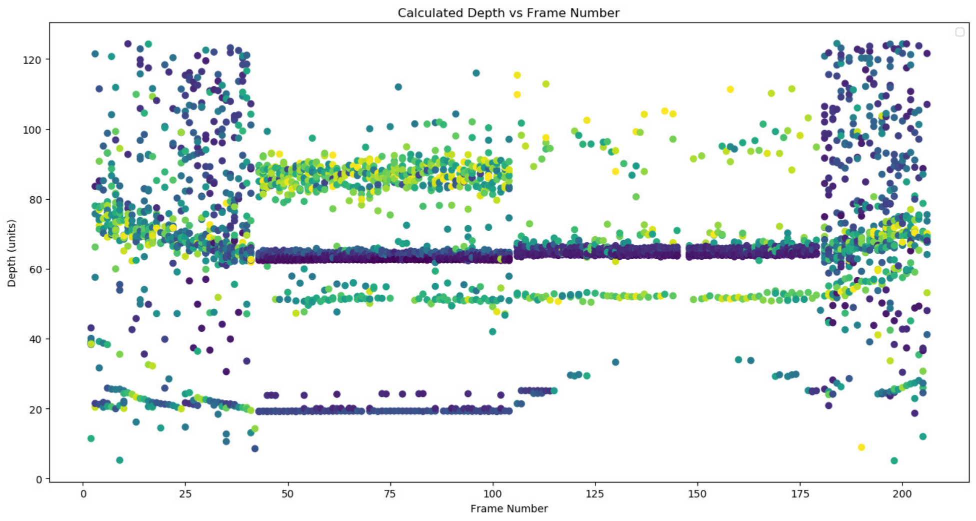

Figure 77: Decreased distance between frames test 1.

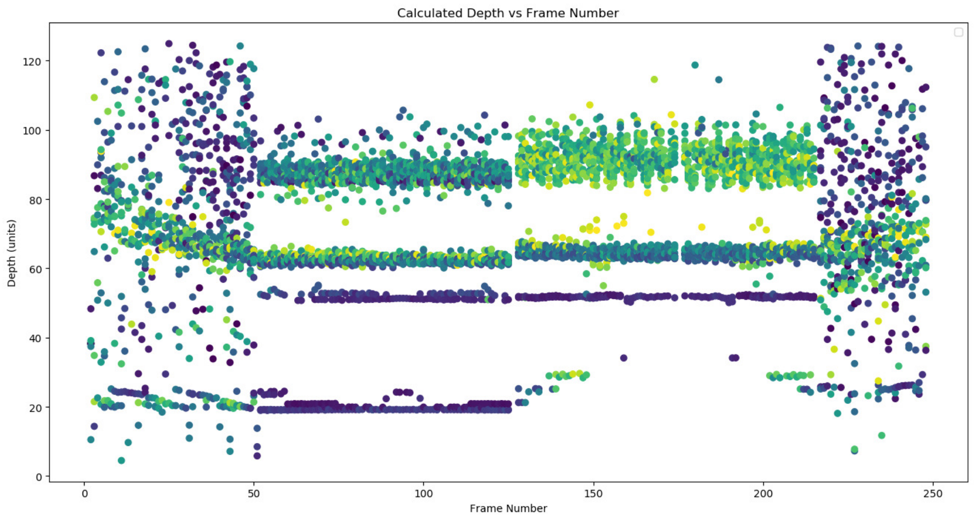

Figure 78: Decreased distance between frames test 2.

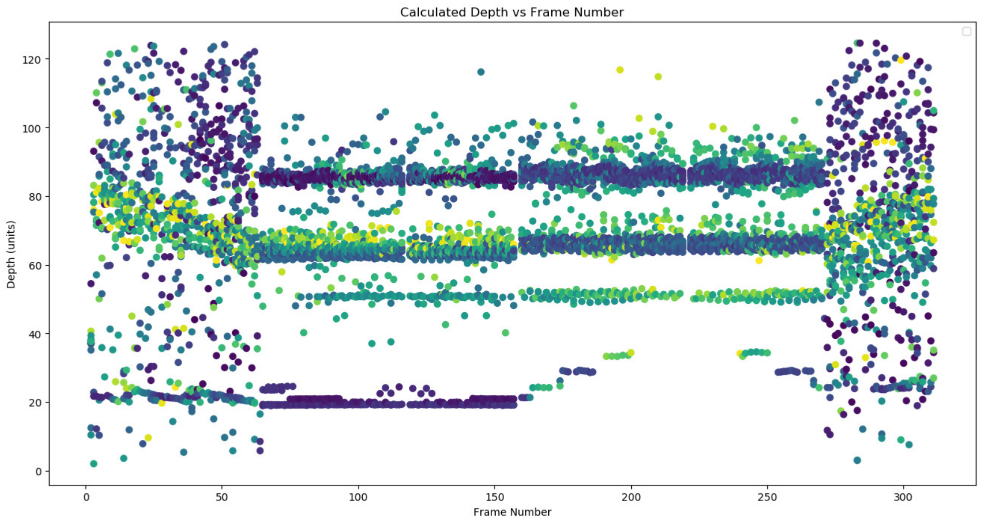

Figure 79: Decreased distance between frames test 3.

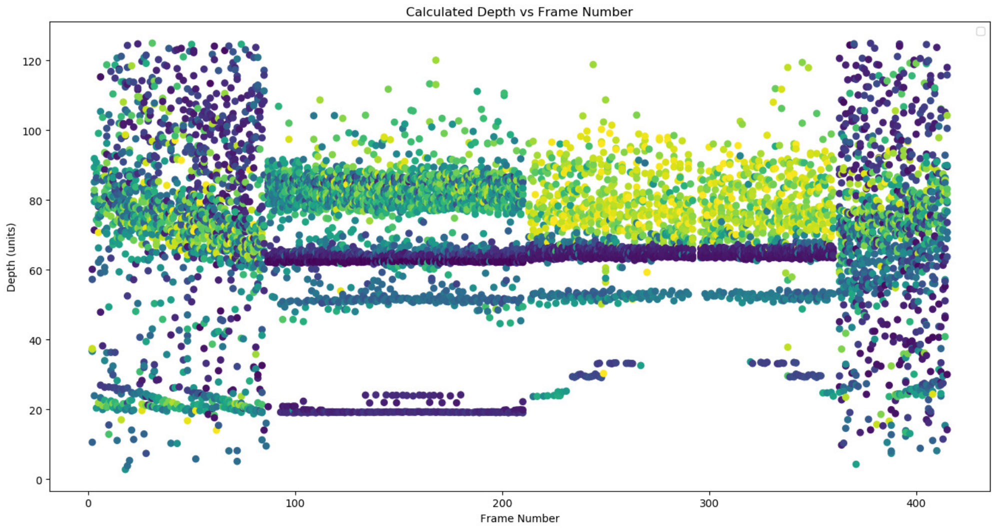

Figure 80: Decreased distance between frames test 4.

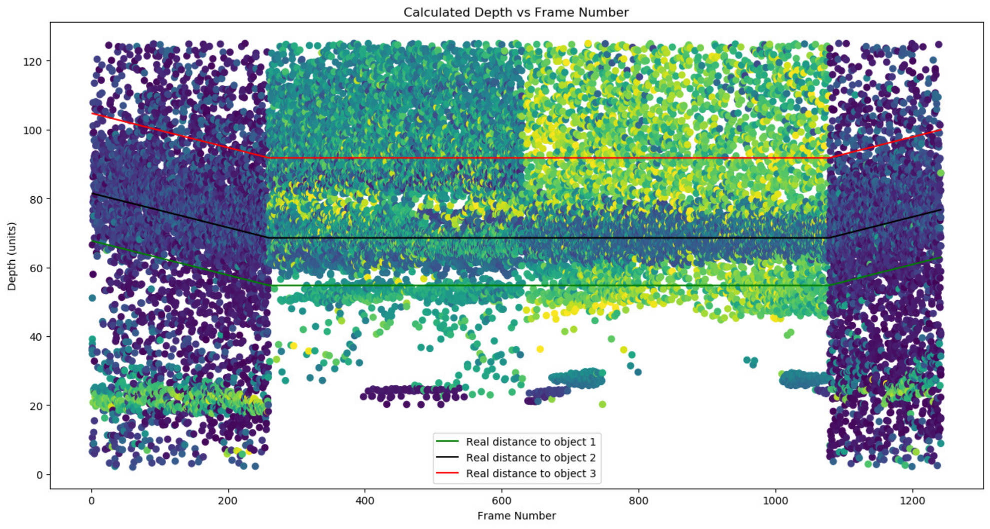

Figure 81: Decreased distance between frames test 5.

Appendix F: Brightness Tests

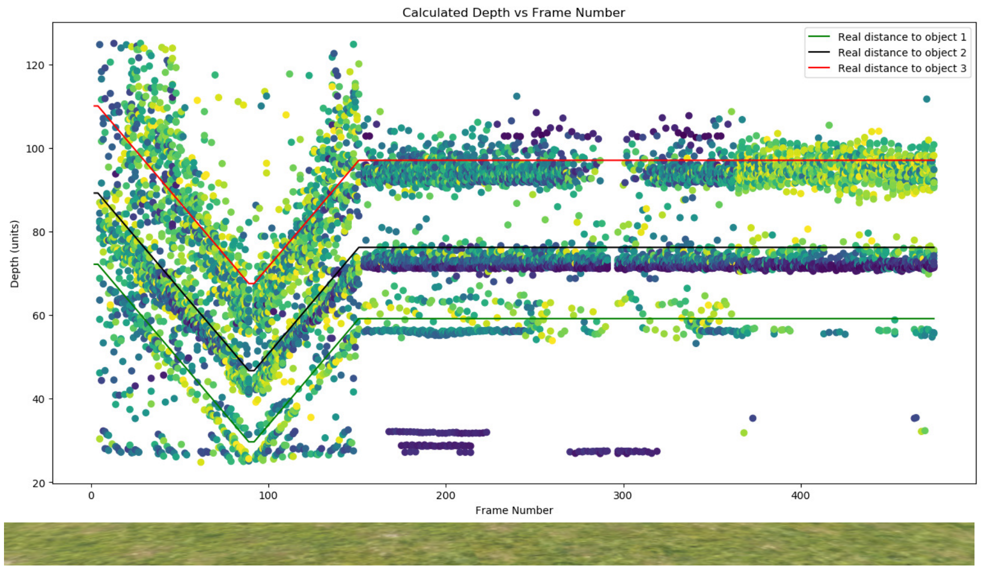

Figure 82: Brightness test 1.

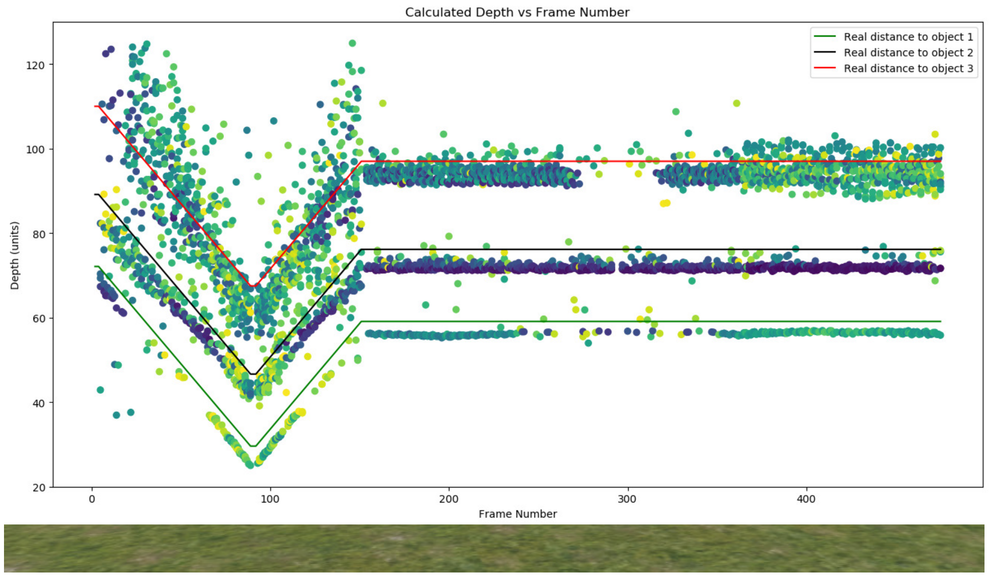

Figure 83: Brightness test 2.

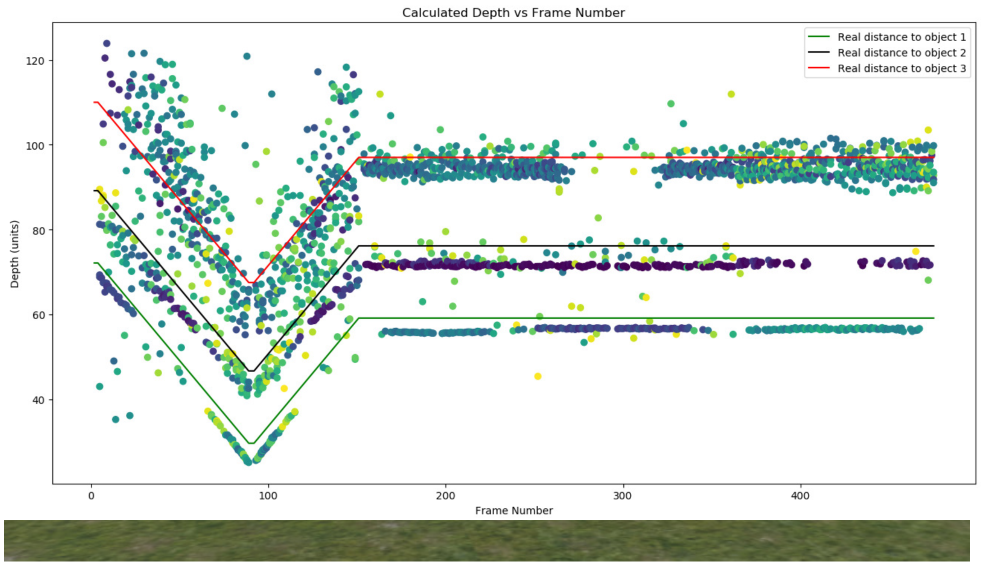

Figure 84: Brightness test 3.

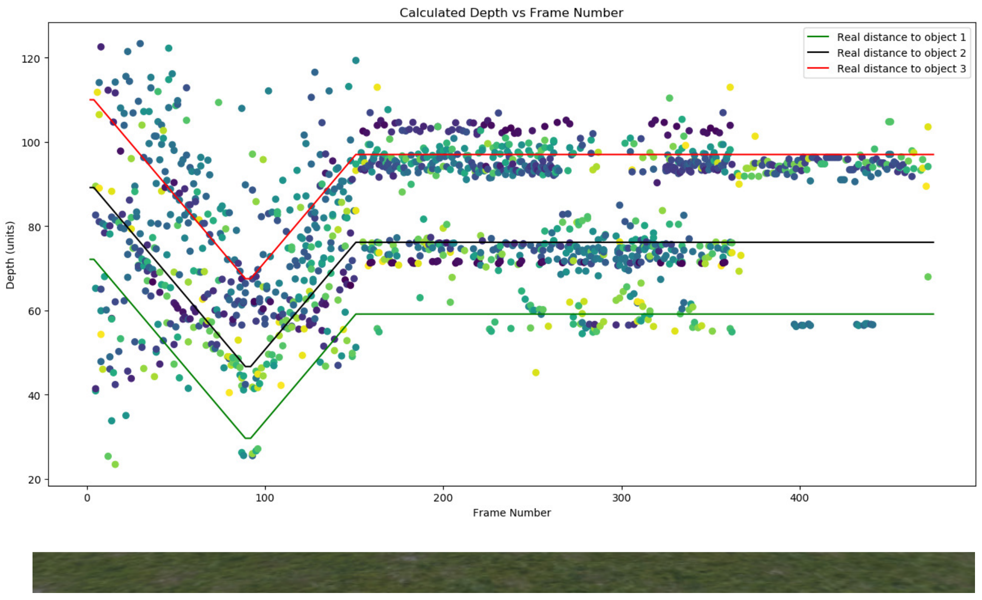

Figure 85: Brightness test 4.

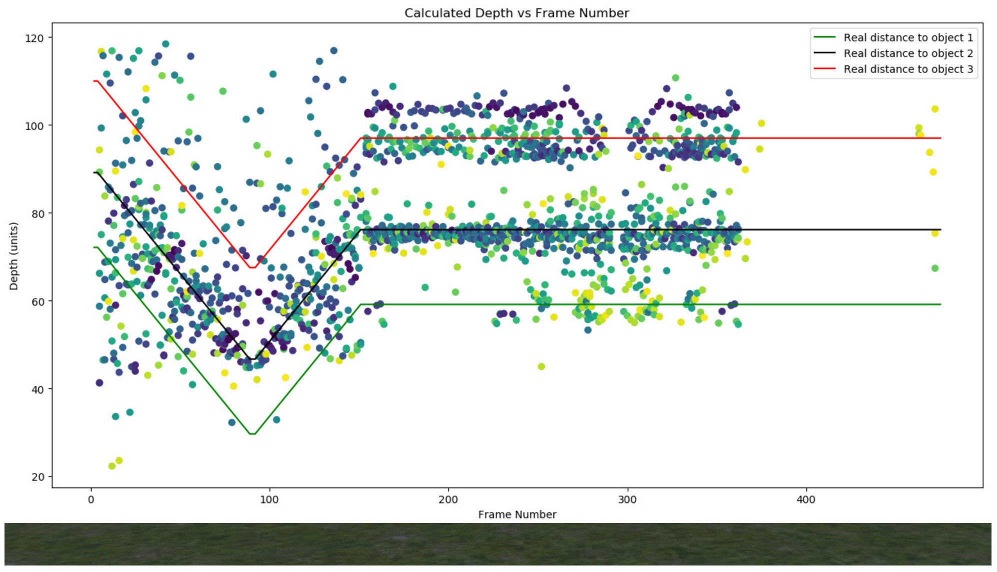

Figure 86: Brightness test 5.

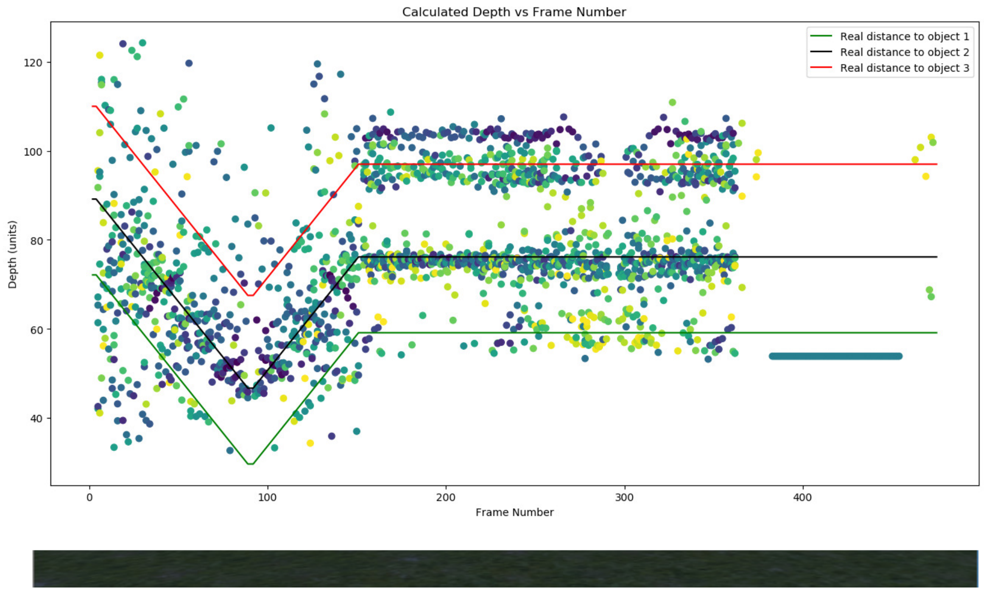

Figure 87: Brightness test 6.

Appendix G: Inaccuracy in Rotation Tests

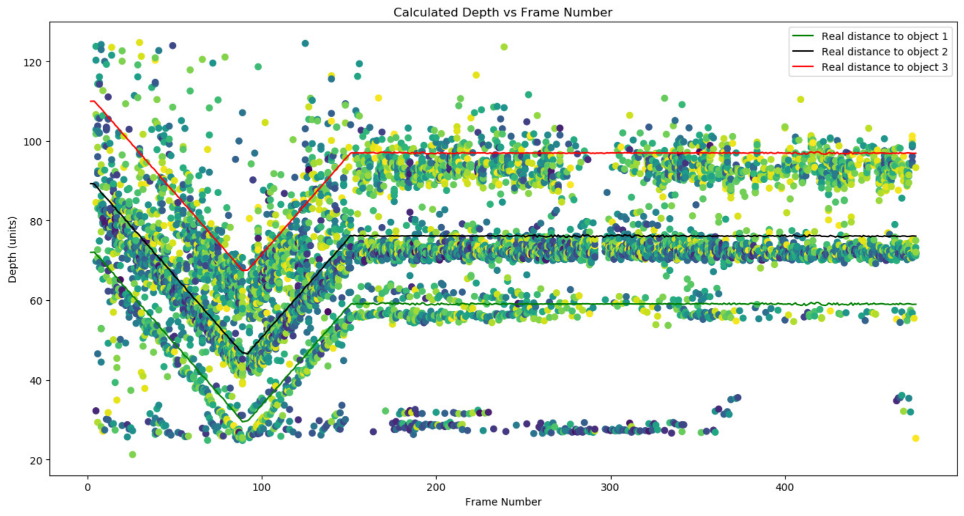

Figure 88: Inaccuracy in rotation test 1.

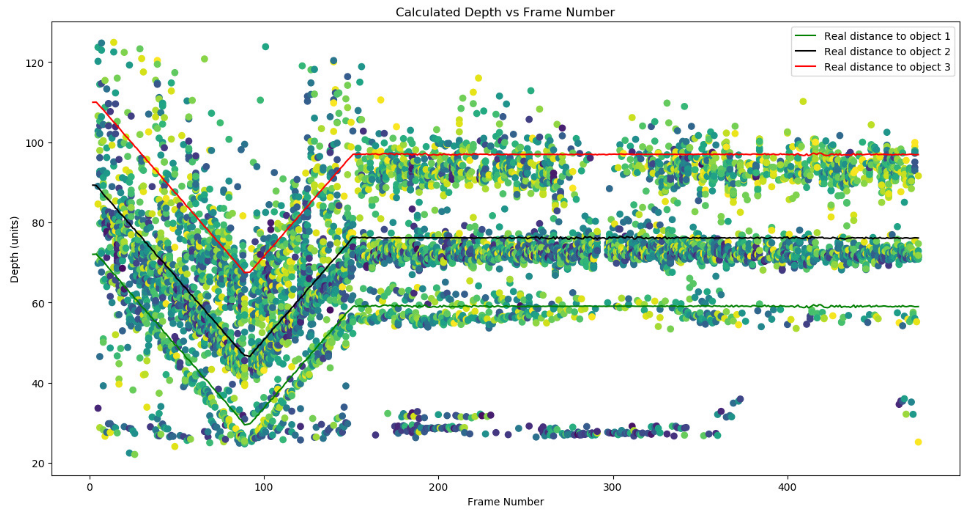

Figure 89: Inaccuracy in rotation test 2.

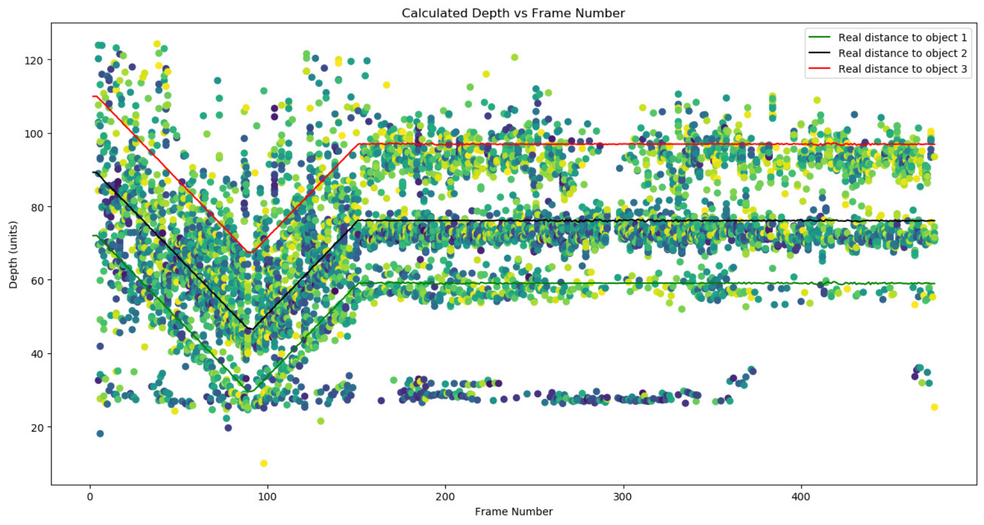

Figure 90: Inaccuracy in rotation test 3.

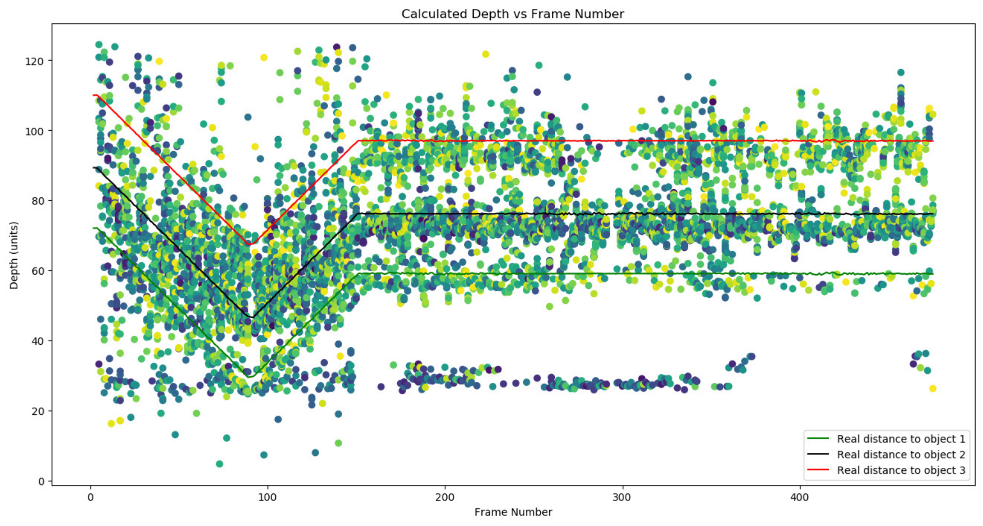

Figure 91: Inaccuracy in rotation test 4.

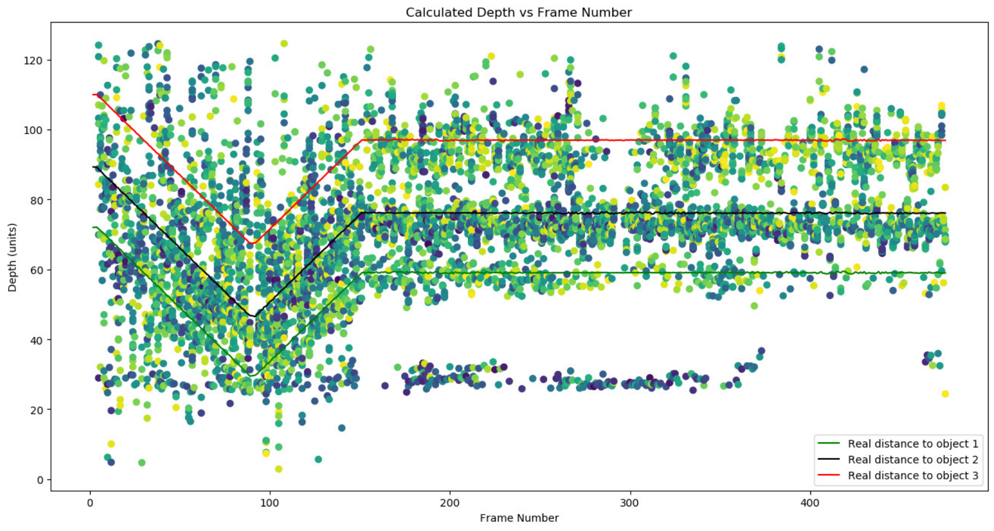

Figure 92: Inaccuracy in rotation test 5.

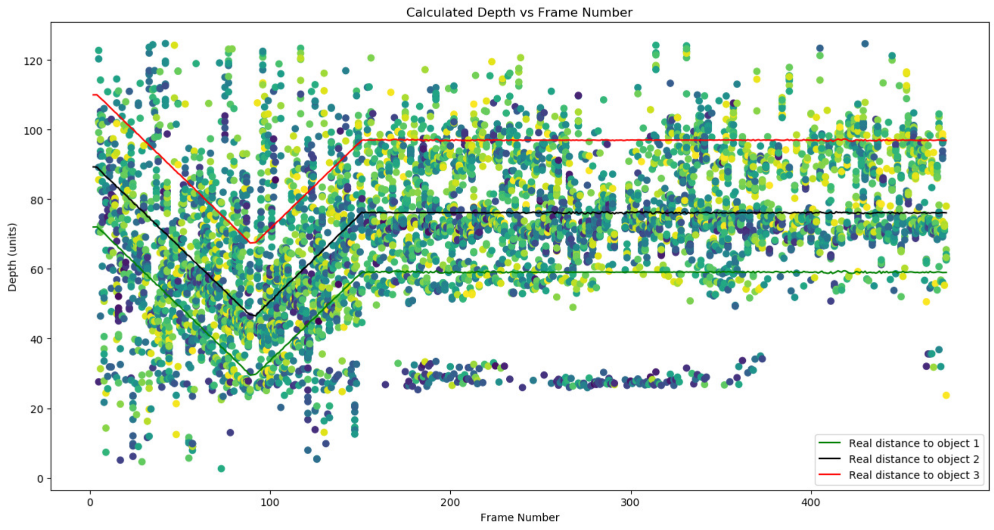

Figure 93: Inaccuracy in rotation test 6.

Appendix H: Inaccuracy in Translation Tests

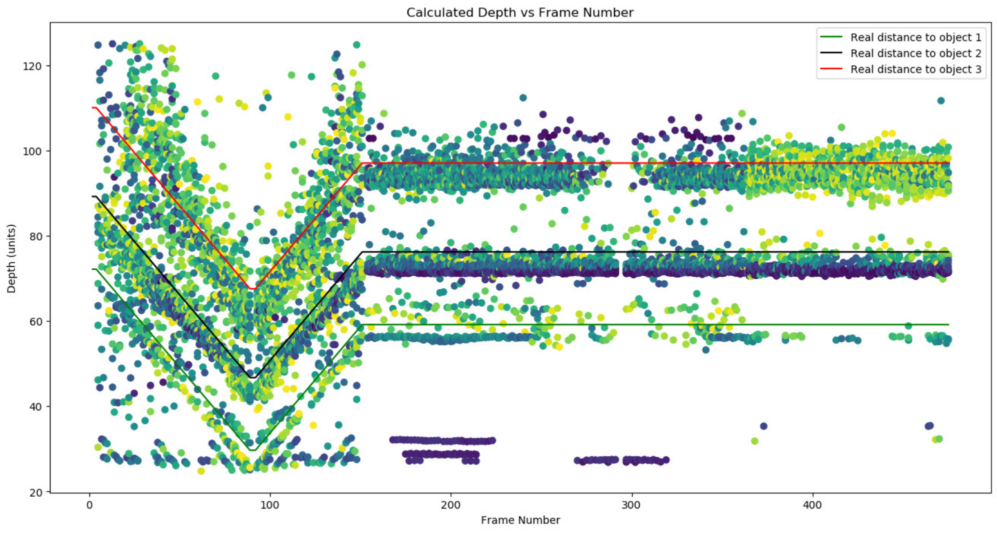

Figure 94: Inaccuracy in translation test 1.

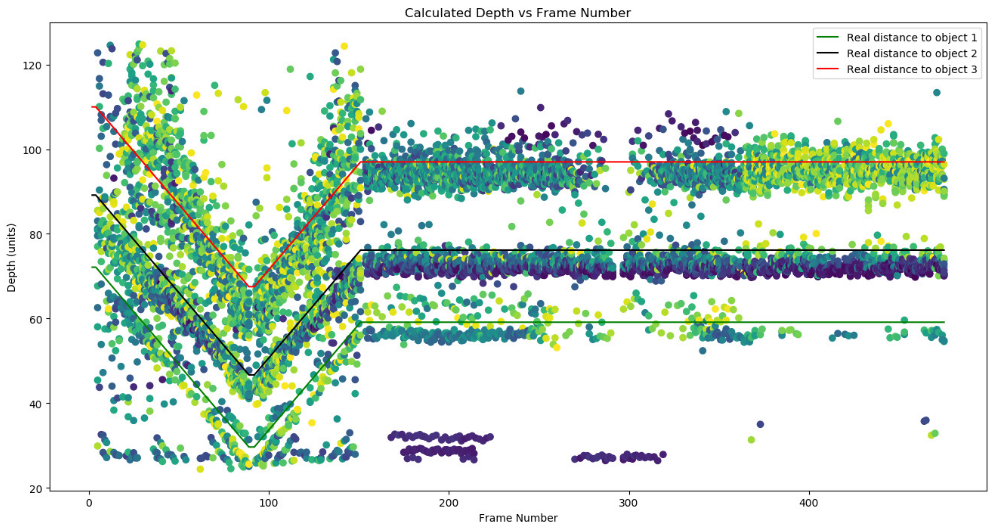

Figure 95: Inaccuracy in translation test 2.

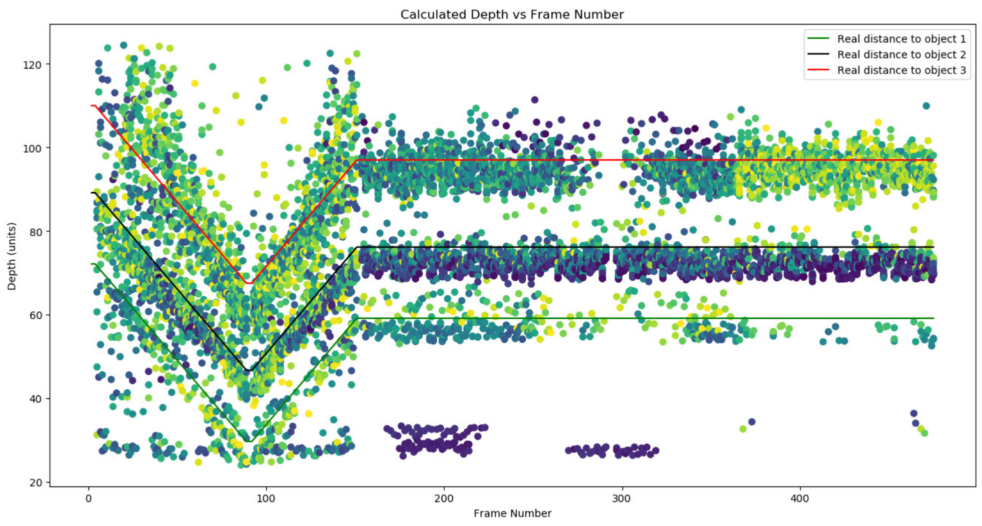

Figure 96: Inaccuracy in translation test 3.

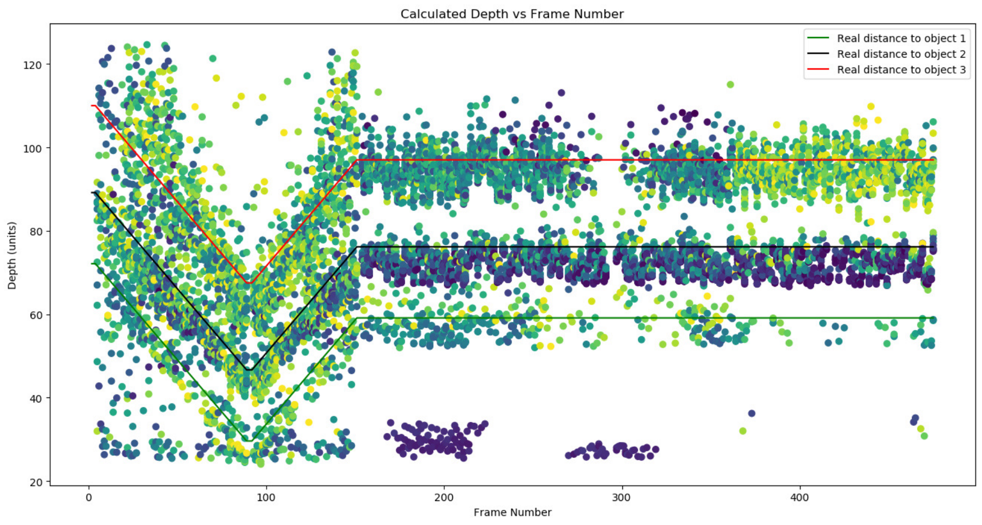

Figure 97: Inaccuracy in translation test 4.

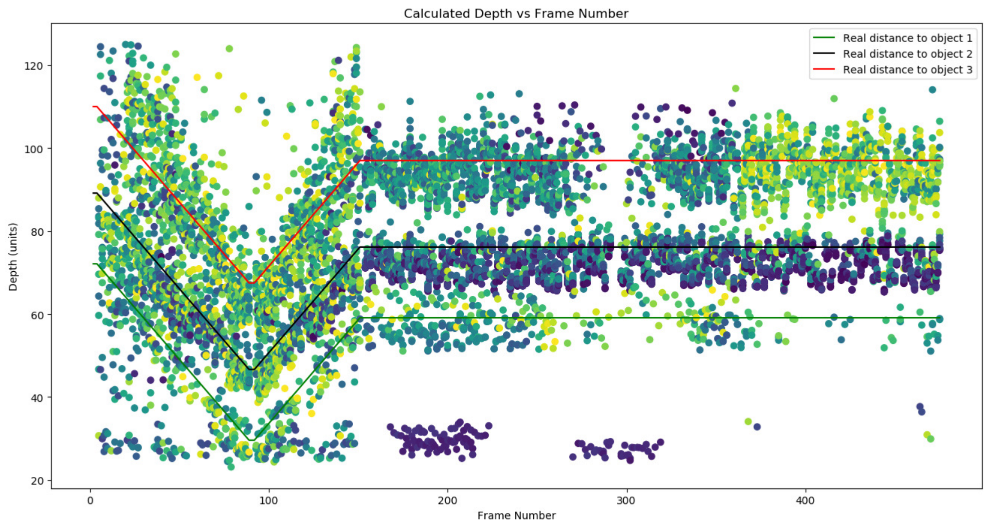

Figure 98: Inaccuracy in translation test 5.

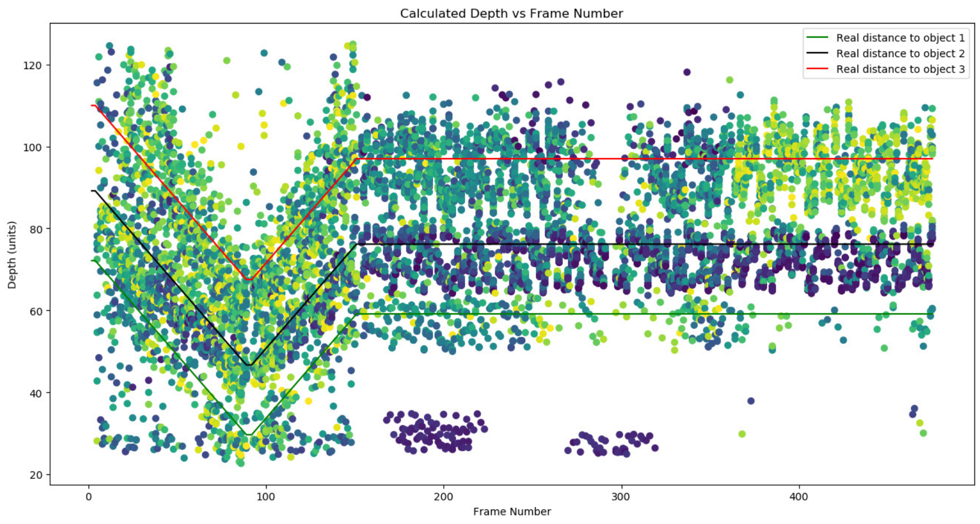

Figure 99: Inaccuracy in translation test 6.