Inholland University of Applied Sciences &
Applied Drone Innovations Ltd.

HiPerGreen Project

Guidance Module

Improving Guidance Information by means of Metric Optical Flow Onboard an Unmanned Aerial System

Author:
Mark Ramaker

Supervisors:
Cock Heemskerk
Lucien Fesselet
Sander Gieling

October, 15 2019

Student:	Mark Ramaker
Studentnummer:	518404
Student email:	518404@student.inholland.nl
Educational Institute:	Inholland University of Applied Sciences
Domain:	Techniek, Ontwerpen en Informatica(TOI)
Degree:	Technische Informatica
Educational Supervisor:	S.Gieling
Internship Company:	ADI & Lectureship Inholland University of Applied Sciences
Company Email:	info@applieddroneinnovations.nl
Company Academic Supervisor(s):	C.Heemskerk
Company Technical Supervisor(s):	L.Fesselet
Company Supervisor(s) Email:	cock.heemskerk@inholland.nl lucien.fesselet@inholland.nl

Abstract

Currently, 5 to 25% of horticulture plants grown in greenhouses go to waste because of diseases, bacteria, fungi, damage, and other causes. Greenhouses form an optimum climate for these factors to grow and develop. The diseases can spread causing even higher product losses.

What the greenhouse growers need is a way of efficiently detecting diseases, bacteria, fungi, and other plant loss causes at a low price, higher resolution and low intervals. With this, plant loss factors can be controlled better and profits can be increased.

The HiPerGreen project is aimed at helping greenhouse farmers reach higher efficiency while using less resources, by using an autonomous scouting Unmanned Aerial System (UAS) (Nieuws Inholland.nl, 2017) [15]. This UAS carrying an array of sensors, could autonomously detect and report on diseases and other plant loss factors.

Autonomous navigation requires accurate location data to avoid static environmental factors, such as windows and posts. The goal of the research done in this thesis, is to increase the localization precision and stability through the combination of an absolute, noisy sensor called the Ultra WideBand (UWB) and a newly designed relative (drifting over time) but stable sensor based on optical flow called the Guidance Module (GM). With *relative* meaning that each measurement uses a new reference, and *absolute* meaning always using the same reference.

While the original assignment only called for the design of such a relative measurement system, this alone would not achieve the goal of increasing accuracy. This led to expanding the original assignment with the task of determining how to combine the relative sensor with the absolute sensor to increase the accuracy.

To achieve the goal of increasing the accuracy, the following research question is formulated: RQ. **How can the provided hardware architecture and an optical flow algorithm be used to increase the accuracy of the UAS?** To answer this research four sub research questions are formulated:

1. *Theory*: As little was known about the precision and theory of such systems:
   SQ.I. **Which significant parameters determine the accuracy of the GM?**
2. *Design*: Using the theory to design a workable system:
   SQ.II. **How to design the GM such that it meets all the requirements by incorporating the significant parameters?**
3. *Integration*: How to combine the relative and absolute measurements:
   SQ.III. **How can the data from the GM and UWB be combined and filtered such that useful positioning information is obtained?**
4. *Testing*: To validate the theory, design and integration:
   SQ.IV. **Does the GM with the design considerations applied realize the goal of increasing the accuracy of the UAS?**

A literature study is done in order to answer the theory question. From this study several factors are identified to have an impact on the measurement precision of the system. These factors are compiled

into a list, and their interconnectedness is explained. In answering the design question, the theory from the first sub research question is used to design and compile a list of theoretical components in exact compliance with the requirements. In answering the integration question a literature study is used in combination with analyses to answer the question of how to combine the relative and absolute measurements. The testing question is answered using specific unit tests, and one integration test. A rail system is used to create a single data set for the tests, and software simulation techniques are used to be able to base all unit tests on the same data set. Even though special care was taken to control all the testing parameters, it was found that the rail system still introduced a lot of noise into the results of the tests. This made it impossible to fully validate the theory as was the goal of the unit tests. However the results *did* find that the goal of increasing the precision and stability of the localization through the combination of the relative and absolute measurement was achieved. Thus validating the system design and sensor fusion method.

As recommendations, the author suggests that the designed relative sensor and sensor fusion method are integrated onto a UAS for further testing, and to tune the Kalman filters used for the sensor fusion in order to further improve the localization precision and stability. Further it is suggested to research three dimensional height mapping for further improving optical flow precision and collision avoidance, and to make custom hardware solutions for integrating all systems onto one Printed Circuit Board (PCB) in order to optimize weight, size and power consumption.

Acronyms

ADI	Applied Drone Innovations Ltd..
EKF	Extended Kalman Filter.
GM	Guidance Module.
GPD	Gaussian Probability Distribution.
GPS	Global Positioning System.
IMU	Inertial Measurement Unit.
INH	Inholland University of Applied Sciences.
KF	Kalman Filter.
LIDAR	LIght Detection And Ranging.
PCB	Printed Circuit Board.
RH	Relative Humidity.
SAD	Sum of Absolute Difference.
SLAM	Simultaneous Localization and Mapping.
TI	Technische Informatica.
TOI	Techniek, Ontwerpen en Informatica.
UAS	Unmanned Aerial System.
UML	Unified Modeling Language.
UWB	Ultra WideBand.

Glossary

floriculture	The cultivation of flowers.
Gaussian Probability Distribution	A curve, the area under which represents the probability of something being true. A Gaussian Probability Distribution (GPD) is a good model for certain natural processes. The total area under the curve from $-\infty$ to $+\infty$ is equal to 1.
Global Positioning System	A navigational system using satellite signals to determine the location of a radio receiver on or above the earth's surface.
h264	(Also called H.264/MPEG-4 AVC) is a video compression standard commonly used for recording, compressing, and distributing high definition video.
Inertial Measurement Unit	An electronic device that uses accelerometers and gyroscopes to measure the acceleration and angular rotation.
LIght Detection And Ranging	A remote sensing method that uses light in the form of a pulsed laser to measure ranges to objects.
Relative Humidity	The amount of water vapor which is contained in the air. With saturated air being 100% RH, and no water vapor 0% RH. The amount of water vapor which can be present in air is temperature dependent.
Simultaneous Localization and Mapping	A technique which uses sensor data to simultaneously build a map of the actor's surroundings and use the same map for navigation.
trilateration	A localization method which uses the length of the sides of (several) triangles to determine the 3D position.
Ultra WideBand	An absolute localization method using a wide radio frequency band and trilateration to determine a position.

Contents

Abstract **I**

Acronyms **III**

Glossary **IV**

1 Introduction **2**
1.1 Subject . . . 2
1.2 Problem Definition . . . 2
1.3 Relevance . . . 4
1.4 Goals . . . 4
1.5 Methods . . . 6
1.6 Overview . . . 6

2 Specifications and Requirements **7**
2.1 Specifications . . . 7
2.2 Requirements . . . 9

3 Methods **10**
3.1 Instrument for SQ.I. **Which significant parameters determine the accuracy of the GM?** . . . 10
3.2 Instrument for SQ.II. **How to design the GM such that it meets all the requirements by incorporating the significant parameters?** . . . 11
3.3 Instrument for SQ.III. **How can the data from the GM and UWB be combined and filtered such that useful positioning information is obtained?** . . . 11
3.4 Instrument for SQ.IV. **Does the GM with the design considerations applied realize the goal of increasing the accuracy of the UAS?** . . . 12

4 System Precision Factors **13**
4.1 Major System Components . . . 13
4.2 Camera Theory . . . 14
4.3 Height Measurement Theory . . . 16
4.4 Rotational Compensation Theory . . . 18
4.5 Optical Flow Estimation Theory . . . 18
4.6 Theory Discussion . . . 21
4.7 List of Factors . . . 22

5 Theoretical System Design **23**
5.1 Theoretical Optical Sensor . . . 23
5.2 Theoretical Height Sensor . . . 24
5.3 Theoretical Rotation Sensor . . . 24
5.4 Flow Estimator Design . . . 25

5.5 Theoretical Components 27

6 Sensor Fusion and State Estimation **28**
6.1 The Makings of a Kalman Filter (KF) 28
6.2 Kalman Filters and Sensor Fusion 30
6.3 System Definition 30
6.4 Data Transformation 31
6.5 Filter Initialization 32
6.6 Example Implementation 32

7 Theory Verification **35**
7.1 Methodology 35
7.2 Test Setup 36
7.3 Results 38

8 Conclusion **55**

9 Recommendations **58**

List of Figures **60**

List of Tables **61**

List of Listings **62**

Bibliography **64**

Appendices **65**

A Symmetry in Gaussian Multiplication **66**

B Unit Test Definitions **70**

C Guidance Module Sourcecode **73**

D Smearing Simulation Sourcecode **82**

E Ultra Wide Band Simulation Data **85**

F The Roll Rotation Induced Measurement Ripple **86**

G An Example of Optical Flow **87**

H Trajectory Simulation **88**

Chapter 1

Introduction

In this chapter, an introduction to the HiPerGreen project and the Guidance Module assignment is given, by providing a firm background and motivation for both HiPerGreen and the Guidance Module.

Starting with an introduction to the HiPerGreen project for context, and then introducing the Guidance Module assignment. Lastly, section Overview (1.6) will provide the reader with insight into how to read this document.

The documented research has been done in the context of a bachelor graduation assignment, conducted and authored by Mark Ramaker, student Technische Informatica (TI) at Inholland University of Applied Sciences. The author is contracted for the Guidance Module by the HiPerGreen project in the form of a trainee-ship. The timeline for the Guidance Module project is twenty weeks.

Applied Drone Innovations Ltd. (ADI) is carrying out the HiPerGreen project in cooperation with the lectureship of Inholland University of Applied Sciences, see: Problem Definition (1.2). In the context of the HiPerGreen project, the Guidance Module assignment is formulated.

1.1 Subject

The Dutch are considered one of the greatest players in the world of floriculture. To retain or solidify this position, possible venues for increasing efficiency are continuously sought. One such method which puts the Dutch at the top of floriculture exporters is the development of huge glass structures, called greenhouses. These structures capture and hold the warmth of the sun and are used to transform the usually unsuitable climate of the Netherlands into one which is very suitable for floriculture.

In greenhouses several factors play an important role in determining the financial yield of the floriculture; from the quantity of plants, and their quality, to the production costs. These production costs usually consist of materials, energy, and labor, and the yield and quality often depend on environmental conditions. The quantity and quality of plants are mostly determined by the suitability of the environment for growing them and how well they are protected from the factors that cause them to become unsuitable for selling.

1.2 Problem Definition

Currently, around 5 to 25% of plants go to waste because of diseases, bacteria, fungi, damage, and other causes. As greenhouses provide an optimum climate for these factors to grow and develop, greenhouse farmers have to scout the plants in the greenhouses for these factors in order to control

the impact. However, there are several factors which complicate the scouting. Greenhouses can be massive, sometimes covering an area of over forty thousand square meters. Combined with the fact that manual scouting is expensive, it leads to scouting usually being done on a bi-weekly interval. Further, after only about three days plants can start to show signs of disease further complicating matters, because the disease could have spread far in the time period between scoutings (Hortipoint.nl, 2016) [12].

What greenhouse growers need is a better method for detecting diseases, bacteria, fungi, and other plant loss causes at a lower price, at higher resolution and shorter intervals.

Cris Ramsey, Lucien Fesselet and Wil Simmonds from Inholland Aearonautical Engineering investigated a possible solution for the greenhouse farmers with a student project called "Drones in de Kas". The project prototyped a drone with a camera on board to help with scouting the plants for these factors. With it they won the Wij Inholland award in 2016 (Wij Inholland.nl, 2016) [19].

Together with the Inholland Lector Robotica, Cock Heemskerk, and Ragna Woodall from the Domain Technology, Design and Information Science they applied for a SIA RAAK-mkb grant, which they got approved for in July 2016. With this grant they restarted their "Drones in de Kas" project to become the HiPerGreen project on 1 Oktober 2016. The HiPerGreen project spans the domains Agri, Food & Life Sciences, Business Finance & Law, Techniek Ontwerpen & Informatica. With the HiPerGreen project they are aiming to help greenhouse farmers reach higher efficiency while using fewer resources by using a scouting Unmanned Aerial System (UAS) (Nieuws Inholland.nl, 2017) [15]. Figure 1.1 shows a demo UAS flying through a greenhouse.

Figure 1.1: A testing platform UAS flying through a demo greenhouse, indicating the scale that is involved

The HiPerGreen project is actively involved with the Inholland Robotica lectorship and involves several students from the above mentioned disciplines in a collaborative effort to further the goals of the project.

An assignment carried out by Wouter de Jong focusing on the viability of several methods of scouting, along with factors to be measured and which sensors to use. His study showed that climate measurements from a drone could help greenhouse farmers by providing them with high resolution measurements (de Jong, 2018) [7]. As a logical further development to drop human labor cost to a minimum, the HiPerGreen project wishes to develop autonomous flying, landing, recharging and data gathering capabilities for the UAS.

1.3 Relevance

While flying and taking measurements autonomously it is important to automatically tag measurement data with accurate location data. Without such a tag the UAS could detect a problem, but tracing where the problem exists would be the same as actually scouting the greenhouse, which is what the UAS is being developed for.

Autonomous navigation also requires accurate location data to avoid static environmental factors such as windows and posts. The stability of the measurement is important for the navigation system to be able to control the UAS in a stable and fluid manner. Flying fluidly and being able to hold position somewhere stably is in turn required for making pictures with the required accuracy, and for automated landing procedures.

In the report "Localisation of a UAS" by Lucien Fesselet, explorative research into the localization of a UAS inside a greenhouse is carried out. The recommendations of this research included further research into LIght Detection And Ranging (LIDAR) and Simultaneous Localization and Mapping (SLAM), further research into Ultra WideBand (UWB) systems, further research into using a range finder, and further research into using a single onboard camera with a tracking algorithm (Fesselet, 2018) [9].

After his thesis Lucien Fesselet continued the research into using UWB for localization, and that has since become the main source for localization on the UAS. The UWB system uses trilateration between base stations and a transponder to calculate an absolute position in three dimensions. It however has an accuracy of only around ten centimeters and can be affected by reflections off the greenhouse structures which further decreases the precision (Fesselet, 2018) [9].

Accordingly this gives rise to the need for an additional system, which can increase the accuracy and reliability of the overall localization.

1.4 Goals

The system which the HiPerGreen team has proposed to use, to increase the accuracy and reliability of the localization, consists of a camera, height sensor and software to determine the optical flow in the camera images. The optical flow magnitude can then be translated into the movement of the camera by using the height and other factors such as the viewing angle of the camera.

This system which is called the Guidance Module (GM) by the HiPerGreen team would perform relative measurements, meaning each measurement would use a new reference.

The demand for accuracy and the unfamiliarity with the GM system approach give rise to the following research question:

Main Research Question: RQ. **How can the provided hardware architecture and an optical flow algorithm be used to increase the accuracy of the UAS?**

Before discussing any further how a system such as the GM can influence the overall localization accuracy, the means to determine the accuracy of the GM must be investigated. This leads to the first sub research question:
SQ.I. **Which significant parameters determine the accuracy of the GM?**

The factors determining the accuracy of the GM have to be incorporated into the design both in terms of hardware and software before validating the GM any further. This leads to the second sub research question:
SQ.II. **How to design the GM such that it meets all the requirements by incorporating the significant parameters?**

With the accuracy and reliability of the relative localization measurements under control, the question remains if and how to use such measurements to increase the accuracy and reliability of the absolute measurements. This leads to the third sub research question:
SQ.III. **How can the data from the GM and UWB be combined and filtered such that useful positioning information is obtained?**

Finally with the accuracy and reliability of the GM under control, and the knowledge of how to use the relative measurement to increase the accuracy and reliability of the absolute measurement, the addition of the GM to the UAS can be validated against the requirements of the HiPerGreen team and the eventual stakeholders, the greenhouse farmers who wish to have a reliable and accurate scouting technology. This leads to the final sub research question:
SQ.IV. **Does the GM with the design considerations applied realize the goal of increasing the accuracy of the UAS?**

1.5 Methods

As the main research question is too broad, it will be answered through the results gathered from all the sub research questions.

The first research question will be answered through a literature study and logical analysis which shall be validated by peer review. This is so, because while searching for information on the subject the author could not find any sources which actually go into detail describing which parameters determine the accuracy of such systems.

The second research question will be answered by using design methods. Hardware design considerations will be determined by criteria-analysis. The software design will be done using divide and conquer over a few design iterations in Unified Modeling Language (UML).

The third question will be answered using a literature study. A great number of sources are available on this subject making literature study the most logical choice.

The last research question will be answered using a number of complementary unit tests. Each of which will serve to prove something from the theory or from the design.

1.6 Overview

The contents of this document, per chapter, are as follows:

Chapter Introduction (1) provides the reader with an introduction into the HiPerGreen project, its goals and the GM project, which is the subject of this document.

Chapter Specifications and Requirements (2) puts forth the requirements and specifications for the GM. Also, some of the scope is defined.

Chapter Methods (3) defines the methodology for answering each of the research questions and the products of each step.

Chapter System Precision Factors (4) explores which factors impact the precision of the system as described in theory through literature study, and answers the first sub research question.

Chapter Theoretical System Design (5) uses the results of the first sub research question to design a system in terms of theoretical components in exact agreement with the requirements, and answers the second sub research question.

Chapter Sensor Fusion and State Estimation (6) finds the method and implementation of how to combine the absolute and relative measurement through literature study of Kalman filters, and answers sub research question three.

Chapter Theory Verification (7) defines unit tests and an integration test, the test setup and the various results of these tests, which answer sub research question four.

Chapter Conclusion (8) discusses the results of all sub research questions and the conclusion to the main research question.

Chapter Recommendations (9) gives recommendations for continuing research in line with this research, and for continuation of the project.

Chapter 2

Specifications and Requirements

This chapter sets forth the requirements for the Guidance Module project. It then defines the system specifications and the environmental specifications. An analysis is done on the combined requirements and specifications, to determine their impact and relation to the GM prototype. This chapter also serves as a blueprint, with which to compare the final implementation and conclusion of the GM project, and this research. The requirements and specifications have been established in close consultation with the HiPerGreen team during the course of the project.

2.1 Specifications

The GM will have to function on-board a UAS and inside a greenhouse. This environment will impose design and implementation considerations. In this section, the environment and the effects of the environment will be specified and briefly matched with possible considerations regarding the design and implementation.

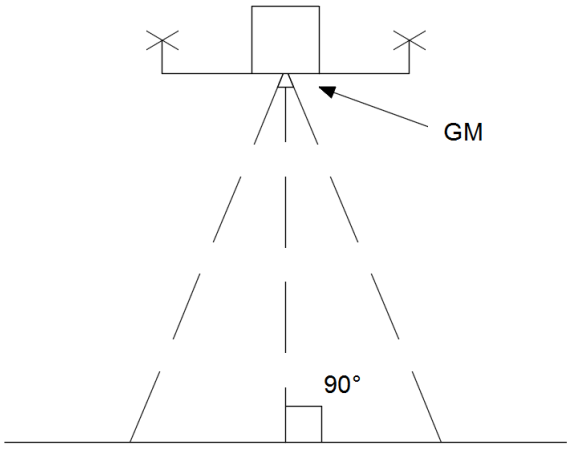

Figure 2.1: The UAS with a GM onboard.

First, the specifications with regard to the mounting of the GM, and the UAS itself:

- The GM is mounted underneath the UAS.

- The GM's optical field of view is perpendicularly aimed at the floor when the UAS is level in the horizontal plane.
- The UAS is a quadcopter, with four spinning rotors providing lift and control. These rotors create vibration.
- The UAS has a maximum rotation speed of $\pm 15°/s$.
- The UAS can accelerate with a maximum of $0.34 m/s^2$.
- The UAS has a rotational acceleration of $213°/s^2$.

Fig: 2.1 can be used to clarify some of the specifications above. Fig: 2.2 shows the basic UAS subsystems, and where the GM is placed in its subsystem architecture.

The specifications of the Greenhouse are:

- It is usually very humid ($> 90\%$RH).
- It is typically quite warm inside a greenhouse ($> 30°$).
- A greenhouse is typically mostly constructed out of glass, and thus a reasonable amount of daylight can be assumed to be present as well as an abundance of artificial lighting.
- A typical greenhouse floor can be observed in Fig: 1.1.

The GM is mounted below the UAS, and is observing the optical flow on the ground that moves underneath the UAS. The environment that is the floor of the demo greenhouse can be seen in Fig: 1.1. This environment shall be used as the target environment for the GM. The surface of the floor of the greenhouse consists of small plants with leaves, pots and potting soil, creating distinct patterns when viewed from above. This surface also has the property that it is not very reflective which is beneficial for optical flow algorithms see Fig: 1.1.

The vibrations that are created by the rotors can have an effect on the optics and electronics. During this assignment, the effects of vibrations on the electronics shall only be considered to the extent that the electronics will likely be able to withstand the vibrations. The exact effects of the vibrations on the optics will not be explored.

Electronics and humidity do not mix well. Corrosion and short circuits can cause electronics to misbehave or fail altogether. This assignment will not go into proving resilience to humidity. These concerns should be addressed in further research, or when the situation demands it.

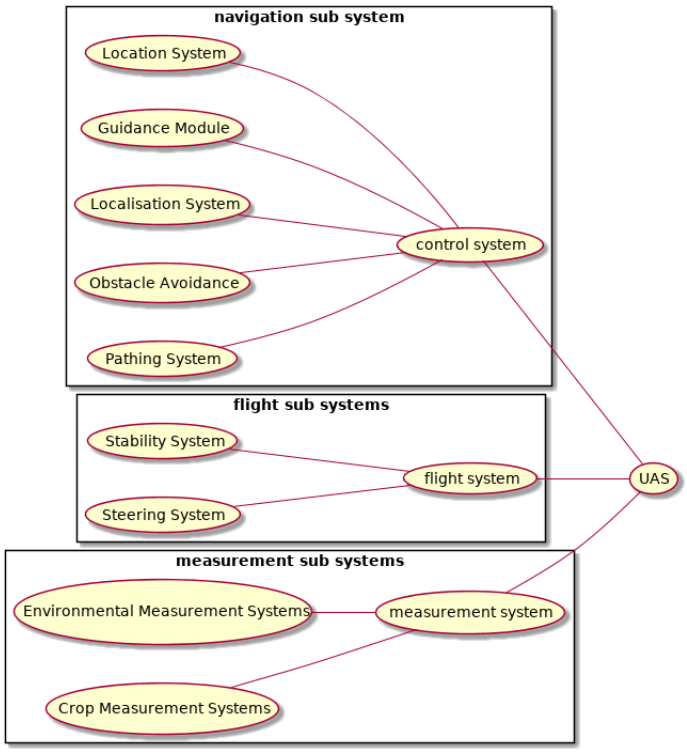

Figure 2.2: An overview of the UAS subsystems.

2.2 Requirements

The following requirements are the goals or demands that are set for the GM. They specify the functionality and physical attributes for the prototype.

- The Guidance Module shall deduce movement using optical flow.
- The Guidance Module shall report the backward/forward and left/right velocity.
- The reporting frequency of the Guidance Module shall be 40 Hertz or higher.
- The Guidance Module shall be functional between the height of 1 meter and 3 meters above the surface.
- The Guidance Module shall be able to handle speeds of up to 3 meters per second.
- The Guidance Module shall have an accuracy of better than 10 centimeters per measurement.
- An accuracy of better than 1 centimeter per measurement is preferred.

These requirements have to do with the functionality that is required of the GM, and are thus far based on estimations from the HiPerGreen team. None of these can be determined conclusively as the project is in the start up phase, and things are liable to change. As such, these requirements have been selected to likely exceed the needs of the project. This provides greater margin for future changes.

Chapter 3

Methods

In this chapter the methodology for answering the research question is set forth. The purpose of this chapter is to develop the instruments with which the questions are answered as well as the blueprint to which the answers or results must conform. The research question and sub research questions are established in section Goals (1.4).

3.1 Instrument for SQ.I. Which significant parameters determine the accuracy of the GM?

This research question will produce a list of descriptions, of the significant parameters which govern the system precision, gathered per system component.

The goal of this list is to make the precision of the complete system predictable by looking at the parameters of the individual components, and also to serve as a starting point for component selection for the HiPerGreen team. Component selection itself is outside the scope of this thesis.

This list is required as the HiPerGreen team has set requirements for the precision of the GM, and there is no specification yet which can predict the precision of the system by looking at the individual parameters of the components.

The list will be compiled by first determining which major components comprise the GM. This will be done through the assignment description from the HiPerGreen team, and through looking at Honegger et al. which describes a hardware and software system which is very similar to the GM. Next per component a literature study will be done to find the relevant parameters. Individual parameters and effects will be summarized and compiled into the list. This literature study per component and summarizing into the list will continue until via discussion with a peer the list is deemed complete enough. Finally the validity of the predictive abilities of the list will be tested in research question four by means of unit tests.

This methodology is chosen because even though there is no single specification which can be used to predict the system precision, there are plenty of sources available for the performance of the individual components. Where there is prior research, information or sources available a literature study is the default choice. The discussion with the peer will be used to determine when the list reasonably contains enough relevant parameters. The unit tests in research question four are then used to confirm or deny the predictions which can be made using the list. This is based on the assumption that when no relevant parameters are left out of the list, there also will not be any significant differences between the results of the tests and the predictions. Visa versa, when there are significant differences between

the prediction and the results it would mean that the list is incomplete or erroneous and that further research is required.

3.2 Instrument for SQ.II. How to design the GM such that it meets all the requirements by incorporating the significant parameters?

This research question will produce a single set of theoretical components, meaning they are not necessarily real, which meet the requirements when considering the list from research question one. It will also produce a UML activity diagram which is used as the behavioral design of the optical flow algorithm.

The set of theoretical components is used in research question four, for the baseline of the unit tests. The activity diagram will be used for implementing the optical flow algorithm, for future reference by the HiPerGreen team and for more clearly communicating its functioning.

For the unit tests baseline a set of viable theoretical components is needed on which variations of the baseline test can be based. This motivates the need for this set of components. The optical flow algorithm is an essential component of the overall system, and as such a design is desirable to determine the behavior of the algorithm and to make it comprehensible. The design can also be used at a later date to facilitate maintenance or adaptation of the algorithm to a different purpose.

The set of theoretical components will be compiled using back analysis of the list of parameters per component from research question one, and the requirements. The design of the optical flow algorithm will be based on the algorithm from (Honegger et al., 2013) [11]. During the first design iteration the shortcomings from RQ1 will be addressed. A peer will be asked to verify whether the design is comprehensible, whether it addresses the shortcomings from RQ1 adequately and whether the designed algorithm accomplishes the goal of measuring optical flow. The design will be iterated until through this peer review it is deemed sufficient.

To design these theoretical components the list of factors from RQ1 and the requirements have to be combined. This combination will have to be reached through means of analyses as no prior methods of combination seem to exist. An activity diagram is a wonderful tool for designing and documenting deterministic behavior and the course of an algorithm, both of which are the goal of this design. Because a design usually needs more then one iteration, an iterative approach is chosen, and to mitigate designer tunnel vision peer review is used to determine the stopping point for the iterations.

3.3 Instrument for SQ.III. How can the data from the GM and UWB be combined and filtered such that useful positioning information is obtained?

This research question will produce a description of how a Kalman filter can be leveraged for sensor fusion, and of the advantages of using Kalman for filtering and sensor fusion (data combination).

This description is to be used to advise the HiPerGreen team on how to use the measurement of the GM, and also to offer a solution for the integration of measurement inaccuracies, which will possibly arise due to the relative measurement of the GM.

The usefulness of the GM is dependent on how the measurement is used by higher systems. The GM produces relative measurements which when integrated will start to drift, and the UWB produces jumpy measurements. Both of these characteristics are undesirable and both are inherent to the individual systems. These undesirable characteristics can be eliminated by supplementing the downsides

of each of the systems with the strengths of the other. This motivates the need for sensor fusion and thus this description.

The description will be formulated through study of common Kalman filtering literature and summarizing the relevant parts in a way which is understandable and implementable for the HiPerGreen team. The description will be validated by the HiPerGreen team. A representative of the HiPerGreen team will be asked to ascertain whether the description is usable for the team.

Kalman filters have been chosen as the subject of this research question since there was a lot of interest from the HiPerGreen team towards this subject, and as a preliminary literature study into Kalman filters showed great promise. The approach is chosen because there is a good basis of literature available on the Kalman filtering subject, making a literature study the logical choice. And since the HiPerGreen team will be the beneficiaries of this description they seem the correct party to validate the usefulness of the description.

3.4 Instrument for SQ.IV. Does the GM with the design considerations applied realize the goal of increasing the accuracy of the UAS?

This research question produces a series of unit tests and the results of these tests.

The goal of these unit tests is to ascertain whether applying the theory from research question one, the design from research question two and the sensor fusion plus filtering technology from research question three, combined increase the localization accuracy as intended.

This is desired to determine whether the list from research question one, the design from research question two and the sensor fusion plus filtering technology from research question three together answer the main research question.

A set of test data will be assembled for each of the major components. This test data is over-specified in terms of the component's properties, for instance resolution, in relation to their baseline. The exact amount of over-specification of the properties will be determined ahead of gathering the test data. Through the use of software techniques the sets of test data will be reduced to the baseline specification. For each of the parameters of the baseline a variation is reduced from the over-specified data set, creating data sets in which only one parameter is varied in relation to the baseline. Tests will be run on all the data sets excluding the over-specified data set. An additional test will be assembled using the results from the baseline test and simulated UWB data to verify the sensor fusion plus filtering performance as an integration test.

The baseline can be verified in terms of adherence to the requirements by running the test data and comparing the results against both the requirements and the predictions from the theory. By testing a variation of each parameter separately and comparing these results against the predicted variations, the theory can be validated. When there are no significant differences between the predictions and the results, it becomes reasonable to assume the theory is correct and complete enough to perform its purpose. The performance of the sensor fusion plus filtering technology can verified through use of the results from the baseline test and a simulated UWB, as the baseline test most closely resembles the eventual implementation of the GM. The UWB will be simulated to simplify the testing process. Positive results from all tests would indicate that applying the theory from research question one, the design from research question two and the sensor fusion plus filtering technology from research question three, together answer the main research question.

The methodology will be expanded in chapter Theory Verification (7), there the exact definition of the tests and the results will be given.

Chapter 4

System Precision Factors

The aim of this chapter is to describe the major components of the GM, then collect theory on those components, and describe the effects the characteristics of these components have on the precision.

4.1 Major System Components

The assignment as presented by Lucien Fesselet, was to improve on an experimental Raspberry PI system. The system was comprised of a Raspberry PI camera, a HC-SR04 sonar height sensor, and a Raspberry PI. The PI camera streamed an h264 hardware compressed video stream to the Raspberry PI. The h264 compression performing a form of block matching optical flow and sending block vectors as deltas between reference frames to compress the video stream. These vectors were used together with the height measurement to transform pixel flow to scene movement. From this system the following major system components can be distinguished: h264 compression (optical flow estimator), HC-SR04 sonar height sensor (a height measurement), the Raspberry PI camera (camera), and the Raspberry PI (an information processing system) see Fig: 4.1.

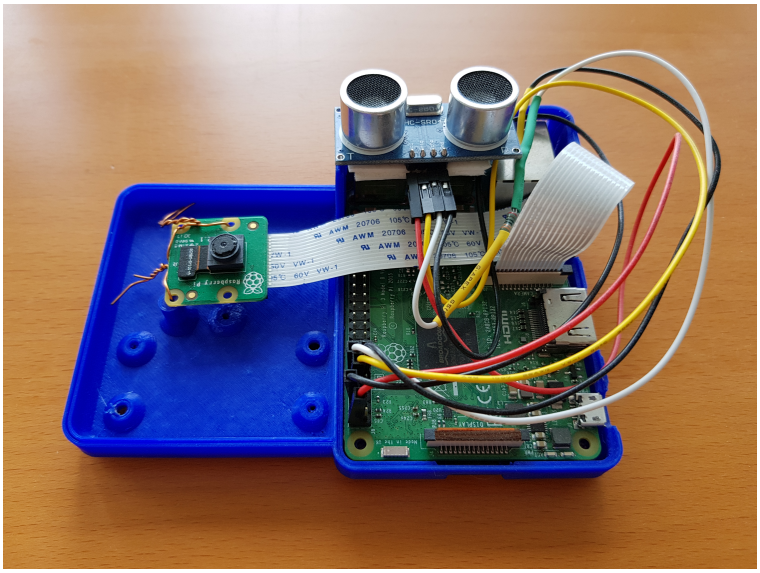

Figure 4.1: The Raspberry PI prototype. Consists of a HC-SR04 sonar height sensor, Raspberry PI Camera V2.1 and the Raspberry PI 3B.

From (Honegger et al., 2013) [11] the system architecture is described as a microprocessor system running an optical flow estimation algorithm on visual data gathered through a CMOS sensor and a 16mm lens. The flow estimation is scaled from image space to ground truth via the ultrasonic height measurement, and corrected for rotation via 3D gyro measurements. Thus the system from (Honegger et al., 2013) [11] can be summed up as the following major components: MT9V034 cmos imaging sensor plus lens (camera), HRLV-MaxSonar - EZ sonar height sensor (height measurement), L3GD20 3D Gyro (rotation sensor), PX4FLow Optical flow algorithm (optical flow estimator), Cortex M4F CPU (information processing system).

From these two sources the following major system components can be distinguished:

- A camera
- A flow estimator
- A height measurement
- A information processing system
- A rotation sensor

The information processing system has no impact on the precision of the system as long as it is capable of processing the information withing the allotted time. It will be assumed that the system chosen will be able to process the information, leading to this component having no impact on the system precision. A rotation sensor is used for compensating rotation in the frame. As the GM will be mounted on a gymbal which provides physical stabilization for mitigating rotation, another form of rotation compensation is assumed to not be required.

A flow estimator is an algorithm which can calculate the optical flow. Optical flow is the effect of regions of characteristic pixels moving a certain distance between two frames (pictures) a short length of time apart. An example of optical flow can be seen in appendix G.

4.2 Camera Theory

A lens in a camera is used to project the field of view of the camera onto the imaging sensor. The relationship between lens focal distance, object distance (height above ground), and required image distance from the lens is governed by the relationship detailed in equation (4.1). The magnification of a thin lens is governed by equation (4.2).

Equation 4.1 The lens equation. Where p is the object distance, q is the image distance and f is the focal length (Pedrotti, 2008) [16].

$$\frac{1}{p} + \frac{1}{q} = \frac{1}{f} \tag{4.1}$$

Equation 4.2 The magnification equation. Where m is the magnification (ratio of image size to object size) h_i is the transverse size of the image h_o is the transverse size of the object p and q are object and image distance respectively (Pedrotti, 2008) [16].

$$m = \frac{h_i}{h_o} = -\frac{q}{p} \tag{4.2}$$

From the lens equation follows that focusing in a camera system with a very small image distance (say a few millimeters) is relatively uncritical, because the object distances are so great that changes

in their inverse value is negligible in relation to the inverse image distance. Take for example an image distance of $q = 3mm$ and a object distance of $p = 2000mm$. This means we have a lens of $(\frac{1}{3} + \frac{1}{2000})^{-1} \approx 2.9955mm$ focal distance. Halving the object distance leaves us with $(\frac{1}{2.9955} - \frac{1}{1000})^{-1} \approx 3.0044mm$ of image distance.

The reverse projection of the imaging sensor onto the object, also called field of view, is governed by the inverse magnification factor; and the reverse projection of a pixel onto the object is also governed by the reverse magnification factor. The equation for calculating projections is given in (4.3).

Equation 4.3 Projection equation. Where h_o is the projected size, h_i is the origin size (pixel or sensor) and m is the magnification factor (4.2).

$$h_o = \frac{h_i}{m} \tag{4.3}$$

Resolution is a term of in how many discrete areas of pixels the total scene observed by the camera is divided (unspecified, 2015) [18]. This term has a direct relation in determining the Nyquist spatial frequency of the camera. The magnification factor can be used to project the Nycuist spatial frequency onto the object to determine the smallest feature frequency which is still detectable. Adapting the projection equation (4.3), the wavelength of the Nyquist spatial frequency projection can be determined by equation (4.4).

Equation 4.4 Nyquist wavelength projection equation. Where λ is the Nyquist spatial frequency wavelength, h_s is the sensor size, m is the magnification factor (4.2) and r is the resolution of the sensor.

$$\lambda = \frac{h_s}{m}\frac{2}{r} \tag{4.4}$$

Imaging sensors have a period of their operating called the *exposure time* (unspecified, 2015) [18]. During this period photons from the scene fall onto pixels and the amount of photons will determine the pixel brightness (unspecified, 2015) [18]. If there is movement in the scene in relation to the camera, pixels can receive photons from larger areas of the scene. This effect is called smearing. The magnitude of this effect is dependent on the magnitude of the movement of the scene in relation to the camera during the exposure time. This effect can generally be thought of as increasing the projection size and shape of the pixel onto the object, or in other words it can be thought of as a reduction of resolution. Equation (4.5) describes the effect smearing has on the effective resolution.

Equation 4.5 Smearing equation. Where r_e is the effective resolution after smearing is factored in, r_a is the absolute resolution, h_p is the pixel size on the sensor, and m, s, t, are magnification factor (4.2), speed and time respectively.

$$r_e = \frac{h_p r_a}{h_p + mst} \tag{4.5}$$

Another effect produced in some imaging sensors is shutter deformation. This effect is caused by differences in the integration start and end times for different pixels (unspecified, 2015) [18]. These times can differ when certain forms of pixel readout structure are applied (unspecified, 2015) [18]. The magnitude of this effect is dependent on the magnitude of movement of the scene in relation to the camera and in relation to the difference in start and end integration times.

4.3 Height Measurement Theory

The effect of the height measurement error in the change of pixel backwards projection has an effect on the measurement precision of the GM. This effect can again be determined by using the projection equation 4.3.

Height measurement for determining the object distance in flight can be done either in full three dimensions or with simple single dimension sensors. Example solutions for three dimensions are LiDAR, the XBOX Kinect sensor and image disparity calculations. The single dimension sensors are mostly based on the time of flight principle.

The three dimensional measurement systems are complex, often large in physical size, heavy, power hungry and expensive. For this reason three dimensional solutions will not be looked into further.

From the single dimensional sensors a more extensive look will be made into acoustic time of flight systems and laser time of flight systems.

Acoustic time of flight measurement systems depend on the speed of sound. The systems generate a sound pulse or set of pulses which is somewhat directional, this signal propagates through the air and is reflected off surfaces (max, 2014) [2]. The system measures the time for the signal to travel to and from the first object it is reflected off (max, 2014) [2]. Then the distance can be calculated as $d = \frac{tc}{2}$, where t is the round trip time, c is the speed of sound in m/s. The result is divided by two as the signal has to travel the distance twice [2].

The speed of sound varies based on temperature and humidity (Bohn, 1988) [5]. The magnitude of these effects can be observed in table 4.1 and table 4.2.

The temperature dependence of the speed of sound is expressed in equation 4.6. This equation can also be used to calculate the speed of sound when introducing water molecules. For this the fraction of air and water molecules need to be calculated using equation 4.9, the mean molecular weight needs to be calculated using equation 4.8 and the γ needs to be calculated using equation 4.7.

Equation 4.6 Where c is the speed of sound, γ is the ratio of the specific heat at constant pressure to that at constant volume (1.4 for dry air), R is the universal gas constant, M is the mean molecular weight and T is the temperature in Kelvin (Bohn, 1988) [5].

$$c = \sqrt{\frac{\gamma RT}{M}} \tag{4.6}$$

Equation 4.7 Where γ_w is the ratio of the specific heat at constant pressure to that at constant volume for the combined gas, d_d (5) is the degrees of freedom for dry air molecules, f_d is the fraction of dry air molecules, d_w (6) is the degrees of freedom for water molecules, and f_w is the fraction of water molecules. Adapted from (Bohn, 1988) [5].

$$\gamma_w = \frac{d_d f_d + d_w f_w + 2}{d_d f_d + d_w f_w} \tag{4.7}$$

Equation 4.8 Where M_{avg} is the average molecular weight for air, M_n, M_o, M_a, M_w, are the molecular masses for nitrogen (28), oxygen (32), argon (40) and water (18) respectively, the factors are respectively the fractions of those molecules in dry air, f_d is the fraction of non water molecules and f_w if the fraction of water molecules. Adapted from (Bohn, 1988) [5].

$$M_{avg} = (0.78M_n + 0.21M_o + 0.01M_a)f_d + M_w f_w \tag{4.8}$$

Temperature (°C)	Relative Humidity (%)									
	10	20	30	40	50	60	70	80	90	100
5	0.014	0.028	0.042	0.056	0.070	0.083	0.097	0.111	0.125	0.139
10	0.020	0.039	0.059	0.078	0.098	0.118	0.137	0.157	0.176	0.196
15	0.027	0.054	0.082	0.109	0.136	0.163	0.191	0.218	0.245	0.273
20	0.037	0.075	0.112	0.149	0.187	0.224	0.262	0.299	0.337	0.375
30	0.068	0.135	0.203	0.272	0.340	0.408	0.477	0.546	0.615	0.684
40	0.118	0.236	0.355	0.474	0.594	0.714	0.835	0.957	1.080	1.200

Table 4.1: Percentage increase in speed of sound (re 0 °C) due to moisture in air only. Temperature effects not included except as they pertain to humidity (Bohn, 1988) [5].

Temperature (°C)	Relative Humidity (%)					
	0	30	40	50	80	100
5	0.910	0.952	0.966	0.980	1.020	-
10	1.810	1.870	1.890	1.910	1.970	2.010
15	2.710	2.790	2.820	2.850	2.930	2.980
20	3.600	3.710	3.750	3.790	3.900	3.980
30	5.350	5.550	5.620	5.690	5.900	6.030
40	7.070	7.430	7.540	7.660	8.030	8.270

Table 4.2: Total percentage increase in speed of sound (re 0 °C) due to temperature and humidity combined (Bohn, 1988) [5].

Equation 4.9 Where f_w is the fraction of water molecules in air, f_d is the fraction of dry air molecules, H_r is the relative humidity, $e(t)$ is the vapor pressure of water in Pascal at temperature t and p is the ambient pressure in Pascal. Adapted from (Bohn, 1988) [5].

$$f_w = \frac{0.01 H_r e(t)}{p}$$
$$f_d = 1 - f_w \tag{4.9}$$

As acoustic time of flight systems are dependent on the reflection of the signal, some surfaces which absorb acoustic energy could limit the range or even completely make these systems unsuitable. Another factor which could introduce problems is if there is a source of acoustic energy nearby which operates near the frequency of the measurement system. In that case false reflections could be detected making the measurement completely unreliable.

Laser based time of flight systems work in a much similar way as acoustic time of flight systems. They send out a pulse of laser light and time the round trip, for the pulse to return to a receiver.

Such systems do not suffer as much from environmental conditions affecting the round trip time, as the refractive index of air is 1.00027445 for $920nm$ light (Lide, 2005) [14] a commonly used infrared wavelength (stf, 2018) [3]. This means, when the speed of light in vacuum is used as opposed to that in air, a measurement error of $\frac{1000mm}{1.00027445} \approx 999.725mm$ or less then a $0.3mm$ error per meter. The effects of differences in air density and temperature will be an order of magnitude less still.

The laser light does reduce in intensity over distance, and the reflectiveness of the surface can also reduce the signal intensity to below the environmental light (stf, 2018) [3]. Meaning that environmental light also has a big effect in the effective measurement range (stf, 2018) [3].

4.4 Rotational Compensation Theory

Rotation of the camera has as a result that the backwards projection of pixels into the scene also changes angle, and by that effect move relative to the surface. One way of thinking about this is to envision a line perpendicularly emitted from the center of the camera, and perpendicular to the surface. When this line rotates around the camera (as a result of the camera rotation) it creates a right angle triangle with the surface. From this, the opposite side - the perceived translational movement for the pixels - can be calculated using equation 4.10. Note that this assumes relatively small angles and a starting angle perpendicular to the surface.

Equation 4.10 Where v is the opposite side - perceived translation - and h is the height of the camera above the surface the rotation angle is ϕ.

$$v = h\tan(\phi) \tag{4.10}$$

As an example, a rotation between two frames of 1° and a height of $2m$ results in $v = 2\tan(\frac{0.1}{180}\pi) = 0.03491013$ or in other words $3.4cm$ of perceived translation. With frame times of $\frac{1}{30}s$ at $30fps$ this would be achieved at a rotational velocity of $30fps \times 1^\circ = 30^\circ/s$. The introduction of such a measurement error when uncompensated gives rise to the need to compensate for it.

Methods of compensating for this rotation could be to measure the rotation and deduct the calculated perceived translation which results from the rotation from the measured translation. Or to stabilize the camera such that it does not rotate relative to the surface by means of a gymbal.

A gyro can be used to measure rotation for compensation purposes. As an example the L3GD20 mems gyro from STSemiconductors specifies a $8.75mdps/lsb$ of sensitivity (gyr, 2013) [1]. Meaning the smallest rotational value it can measure is $0.00875^\circ/s$. Assuming an inter frame time of a second this would result in a $2m\tan(\frac{0.00875}{180}\pi) = 0.000305433m$ or $3mm$ of perceived and uncorrected translation. As most camera's operate at or above $30fps$ the perceived and uncorrected translation would be at least $\frac{1}{30}$ times lower.

A gymbal is a mechanical stabilization device, often controlled by gyro measurements to actuate the end effector in such a way that it is stabilized in at least two of the three rotational axes. Such a system could theoretically be used to mitigate the need for software compensation of rotation.

4.5 Optical Flow Estimation Theory

As the goal of the GM is to do it's function onboard a UAS where weight and power budgets are constrained, the computational complexity of the optical flow algorithm is constrained with it.

For this reason this thesis will only look at one class of simple and fast algorithms, the block matching algorithms.

For analysis of the structure of such an algorithm the open sourced code of the PX4Flow project is used. But it should be noted that the principles of their algorithm are similar to or the same as most other block matching algorithms.

Block matching optical flow algorithms work by dividing the reference frame (image) into discrete blocks of pixels, 8 pixels in the example (Honegger et al., 2013) [11]. The definition of a block is given formally in equation 4.11.

Equation 4.11 Definition of a block with dimensions $m \times n$ as a matrix of pixels (Honegger et al., 2013) [11].

$$\mathrm{B}_{m,n} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{pmatrix} \tag{4.11}$$

The algorithms use a function to correlate a block of pixels in the reference frame with a block within a search grid in the destination frame. One such function, that is used commonly and also in the example of (Honegger et al., 2013) [11], is the Sum of Absolute Difference (SAD) function. This function is formally defined in equation 4.12.

Equation 4.12 Definition of the SAD function. Where A and B are two $m \times n$ pixel blocks (Honegger et al., 2013) [11].

$$SAD(\mathrm{A}, \mathrm{B}) = \sum_{y=1}^{m} \sum_{x=1}^{n} \left| a_{y,x} - b_{y,x} \right|, a_{y,x} \in \mathrm{A}, b_{y,x} \in \mathrm{B} \tag{4.12}$$

The reference block is compared to every block within the destination image search grid. The minimum SAD function position within the search grid is assumed to be the best possible match, and thus the translation of that block of pixels. This algorithm is more formally defined in equation 4.17.

Equation 4.17 In equation 4.13 the search window size is define as s_y, s_x and as a range s_1, s_2. Equation 4.14 defines the reference images of A in time step k_{-1} and Z in time step k. Then equation 4.15 defines a reference block R in A where $y_\mathrm{A}, x_\mathrm{A}$ are the indexed offsets in image A. In equation 4.16 a target block is defined within the search window inside Z corresponding to the offset of R in A. Lastly equation 4.17 gives the actual algorithm as the collection O of SAD minimized target block offsets. Based on (Honegger et al., 2013) [11].

$$\begin{aligned} s_y \in \mathbb{Z}^+ \mid s_y < \frac{m-k}{2} \\ s_x \in \mathbb{Z}^+ \mid s_x < \frac{n-l}{2} \\ s_1 = \{-s_y, \cdots, s_y\} \\ s_2 = \{-s_x, \cdots, s_x\} \end{aligned} \tag{4.13}$$

$$\begin{aligned} \mathrm{A} = \{a_{i,j} \mid a_{i,j} \in [0, \cdots, 255], 1 \leq i \leq m, 1 \leq j \leq n\} \\ \mathrm{Z} = \{z_{i,j} \mid z_{i,j} \in [0, \cdots, 255], 1 \leq i \leq m, 1 \leq j \leq n\} \end{aligned} \tag{4.14}$$

$$\mathrm{R} \subset \mathrm{A}, \mathrm{R} = \{a_{i,j} \mid a_{i,j} \in \mathrm{A}, s_y < y_\mathrm{A} \leq i \leq y_\mathrm{A} + k \leq m - s_y, s_x < x_\mathrm{A} \leq j \leq x_\mathrm{A} + l \leq n - s_x\} \tag{4.15}$$

$$\begin{aligned} \mathrm{B}_{y,x} \subset \mathrm{Z}, \mathrm{B}_{y,x} = \{z_{i,j} \mid z_{i,j} \in \mathrm{Z}, 1 \leq y_\mathrm{A} - s_y \leq y_\mathrm{Z} \leq i \leq y_\mathrm{Z} + k \leq y_\mathrm{A} + s_y + k \leq m, \\ 1 \leq x_\mathrm{A} - s_x \leq x_\mathrm{Z} \leq j \leq x_\mathrm{Z} + l \leq x_\mathrm{A} + s_x + l \leq n\} \end{aligned} \tag{4.16}$$

$$\mathrm{O} = \{(y, x) \mid SAD(\mathrm{R}, \mathrm{B}_{y,x}) = min\{SAD(\mathrm{R}, \mathrm{B}_{u,z}) \mid u \in s_1, z \in s_2\}, y \in s_1, x \in s_2\} \tag{4.17}$$

It should be noted that the algorithm as defined in equation 4.17 produces a set of SAD minimized offsets. In practice only a single offset value is used per block, which is often the last value of the set. The assumption that the last set value is the correct one, or that slightly less minimized SAD values at different offsets are less valid could be incorrect. As this is how the algorithm has been defined and the function of it will be validated by means of unit tests, the validity of this assumption will not be explored.

Equation 4.17 formalizes that the reference block is matched on whole indices in the search window. As an image is a discrete representation of a continuous phenomenon aliasing of discrete values is present. Meaning that the best match can be biased to match on whole indices always with a plus and minus half a index uncertainty. This gives an inherent limit to the accuracy of the simple block matching algorithm. This limited accuracy can again be translated to a backwards projection into the scene with equation 4.3.

As can be seen in equation 4.17 the search window exhaustive search has a complexity of $O(n^2)$, which grows with the square of the window size. The same however can be said about the number of individual blocks contained, and thus calculated, within a frame grows with $O(n^2)$ of the resolution. This results in $O(n^4)$ worst case complexity.

Lastly a reference block with a local gradient in only one direction can only measure the displacement in that direction (Beauchemin and Barron, 1995) [4]. It's best matching position can be anywhere along a line perpendicular the that gradient (Beauchemin and Barron, 1995) [4]. This means that local intensity structures have to be sufficiently complex in order to be able to measure multi direction optical flow (Beauchemin and Barron, 1995) [4].

4.6 Theory Discussion

From section Optical Flow Estimation Theory (4.5) and equation 4.17 it can be concluded that the optical flow algorithm can in theory match up to half the wavelength of the nyquist spatial frequency in precision. But aliasing of continuous phenomena might make this half wavelength precision unattainable. For this the author proposes that an effective precision of a full nyquist wavelength be specified. As can be seen in section Camera Theory (4.2) and equation 4.4 the nyquist wavelength is dependent on resolution, sensor size, height and the magnification factor. The effective resolution was determined to be affected by the smearing equation 4.5, which is itself dependent on height, speed and Δt. The rotation of the camera sensor, and the effect of this on the backwards projection seen in equation 4.10 is only dependent on the height and the angle.

From the above it becomes clear that height plays as a factor in each of the other factors of accuracy, but as determined in section Height Measurement Theory (4.3) the measurement of height itself is only dependent on its own factors. Meaning that the height measurement function has a factor in each other function, but the height measurement function is only determined by itself.

The optical flow algorithm accuracy is mostly reliant on the existence and complexity of local image intensity structures. When these structures are missing or missing in one or more directions, the measurement values can be erroneous. As the optical flow algorithm needs to work on a drone the computational complexity (the amount of work) needs to be kept within certain limits. As the pixel projection size should be kept low to minimize the nyquist spatial wavelength, and the sensor size should be large enough that frames overlap even at high speeds, the resolution should preferably be high. This is in direct competition with the computational complexity as smaller pixels mean a larger search window for the same displacement, and more pixels mean more blocks to calculate. Making this algorithm tenable in terms of computation should be considered as a factor.

4.7 List of Factors

Factor	Description	Function	dependency
	Camera		
1	Magnification factor: determines the backwards projection (or size) of a pixel on the ground see equation: 4.2.	$m = \frac{h_i}{h_o} = -\frac{q}{p}$	4
2	Smearing equation: determines the effective resolution based on speed, height, exposure time, magnification factor and full resolution see equation: 4.5.	$r_e = \frac{h_p r_a}{h_p + mst}$	1, 4
3	Nyquist spatial wavelength: determines the smallest wavelengths or small structures which will not be aliased see equation: 4.4.	$\lambda = \frac{h_s}{m}\frac{2}{r}$	1, 2, 4
	Height Measurement		
4	Round trip time: the measurement method itself.	$d = \frac{tc}{2}$	5
5	Temperature and humidity see equation: 4.6.	$c = \sqrt{\frac{\gamma RT}{M}}$	-
6	Surface reflectiveness, distance and environmental noise factors.	-	-
	Optical Flow		
7	Local intensity complexity: is the algorithm able to accurately match the block between reference frames.	-	-
8	Image resolution and pixel size: will the algorithm be able to be executed onboard a constrained environment.	-	1, 3
	Rotation		
9	Perceived translation: the shift of the visual plane due to rotation see equation: 4.10.	$v = h\tan(\phi)$	4

Table 4.3: List of factors

Chapter 5

Theoretical System Design

In the chapter Specifications and Requirements (2), the requirement for the precision is set at better then $10cm$, and preferred at better then $1cm$ per measurement. For the theoretical design of the components the aim will be the $1cm$ precision. For this the individual components will be budgeted at the following:

- Optical sensor: 35%
- Height sensor: 20%
- Rotation sensor: 30%
- Unforseen factors: 15%

This distribution of the error budget is chosen semi randomly. As the author assumes the height measurement has a far smaller impact on the overall precision, the height measurement is allocated less of the error budget. There were no such considerations for the budgets of the optical and rotational sensors. A part of the total precision budget has been reserved for unforseen factors.

The optical flow estimator doesn't have its own inherent precision, it is directly dependent on the optical sensor precision. This eliminates it from the budgeting. It however still has real performance implications for which a design will be made in this chapter.

5.1 Theoretical Optical Sensor

The optical sensor is budgeted at 35% of the $1cm$ accuracy, this equates to $0.35 \times 1 = 0.35cm$ of ground truth. Factor 3 from the List of factors (4.3) specifies that there need to be 2 pixels for the smallest structures which need to be accurately detected. This means that a pixel projection of $\frac{0.35}{2} = 0.175cm$ is required. This pixel projection value should be valid for the worst case scenario of the maximum flight height of $3m$ (Specifications and Requirements (2)). If this projection value is valid for the maximum flight height it also is valid for any lower flight height as the pixel projection will only shrink when descending.

The maximum required operating speed is $3m/s$ and an operating frequency of $40Hz$ is required (Specifications and Requirements (2)). This equates to a maximum of $\frac{300}{40} = 7.5cm$ displacement per measurement interval. A frame overlap of at least 50% is arbitrarily chosen, which means the minimum projected frame width $\frac{7.5}{0.5} = 15cm$ is required. This minimum projected frame width should be valid for the worst case scenario of the $1m$ flight height requirement. If it is valid for the minimum flight height, it is also valid for all ascending heights as the frame size will only increase.

When taking this minimum projection up to the maximum flight height of $3m$, the width of the projection increases to $15 \times 3 = 45cm$. At the maximum flight height the pixel projection size is already determined to be no bigger then $0.175cm$, which gives a resolution for the frame of $\frac{45}{0.175} = 258$ pixels (rounded up). The minimum viewing angle of the theoretical optical sensor is then $\arctan(\frac{15/2}{100}) \times 2 = 8.578°$. Conversely a camera with a frame width of 1024 pixels should not have a viewing angle greater than roughly $\frac{1024}{258} \times 8.578 = 34°$.

If the maximum allowed amount of smearing is set to be equal to the budgeted precision of the optical sensor, using the maximum speed, the maximum allowable exposure time can be calculated as $\frac{0.175cm}{300cm/s} = 0.00058s$.

Taking all results together, the theoretical optical sensor has the following characteristics:

- **Resolution**: 258 pixels
- **Viewing angle**: 8.578°
- **Exposure time**: $< 0.00058s$
- **Frame rate**: $40Hz$

5.2 Theoretical Height Sensor

From the List of factors (4.3) factor 1 it can be seen that the height directly influences the magnification factor, and therefore also the projection equation 4.3. The error in the measured height can therefore be seen as creating a error in the projection. For an error in the height measurement of +1% of the total height an exact error of +1% in the projection will be observed.

The projection error caused by the height error is expressed as a percentage. This percentage of error carries over onto the measurement magnitude, meaning the absolute error for small displacements will be small, but the absolute error will grow for larger displacements. With the maximum displacement per measurement interval of $7.5cm$, obtained through the maximum flying speed and the operating frequency (Specifications and Requirements (2)), the maximum height error percentage can be calculated as the percentage of the error in relation to the maximum displacement. The height sensor is budgeted at 20% of the total error of $1cm$, making the error budget of the height measurement $0.20 \times 1cm = 0.2cm$ of displacement. So the height measurement can have a maximum error of $\frac{0.2cm}{7.5cm} \times 100 = 2.666\%$. This means a absolute error for the minimum flight height of $100cm \times 0.02666 = 2.666cm$ and an absolute error for the maximum flight height of $300cm \times 0.02666 = 7.998cm$.

The theoretical height sensor has the following characteristics:

- **Measurement precision**: $< 2.666\%$

5.3 Theoretical Rotation Sensor

The rotation sensor has been budgeted at 30% of the $1cm$ total error budget, giving an error budget of $0.30 \times 1cm = 0.3cm$. From the List of factors (4.3) factor 9 and equation 4.10 it can be determined that the increase of height has an increase in the effect of perceived translation from rotation. The error of the rotation, when compensation is applied is likewise affected by the height. For this reason, if the maximum allowable rotation error is given for the maximum flight height, it is also valid for any height below that.

For the maximum flight height of $3m$, the maximum of uncompensated angle can be calculated by a variation of equation 4.10, $\phi = \arctan(\frac{v}{h})$ or $\arctan(\frac{0.3}{300}) = 0.038°$ degrees of maximum error per interval.

The theoretical rotation sensor has the following characteristics:

- **Measurement precision**: $< 0.038°/interval$

5.4 Flow Estimator Design

From the List of factors (4.3) can be observed that the flow estimator has no inherent or direct role in the precision of the system. However it does have a performance impact which must be controlled in order to be applicable in the constrained environment of a UAS.

Most notably the complexity of the number of blocks, combined with the search grid size when unconstrained, would present an algorithm with $O(n^4)$ complexity, with n being the number of pixels vertically and horizontally. Further more the local image intensity structure quality needs to be determined to get good quality block tracking.

These problems are addressed thusly:

- **Image Intensity Structure Quality**: In section 4.5 it was determined that flow can only be determined in the direction of a gradient. This means that the image intensity structure quality is equal to the amount of gradient in all directions. As such structures are less common and also more costly to verify, the algorithm design will be limited to verifying a gradient threshold in two directions (in the image line directions). After the search has been done and a best fit location for the block has been found through the SAD function, it is still possible to determine something about the quality of this match. The SAD function value for the block match is a measure for how precise the match is, with smaller values being better. This value can be used with a threshold to discard improper matches.
- **Number of Blocks**: The assumption is made that with a fixed amount of blocks a sufficiently generalized motion vector can be deduced. As such not always all possible blocks are required. Then the question arises which blocks to use and which ones to discard. The assumption is made that the height measurement most accurately corresponds to the center of the frame, making blocks more valuable for tracking the closer they are situated to the center of the frame. As block locations can be rejected by the image intensity structure quality criteria, just using fixed block locations might produce an insufficient number of tracked blocks. Finally a spiral search pattern from the center of the image outwards until the block count criteria has been satisfied or the image has been exhausted has been selected as the design compromise.
- **Search Grid Size**: In section 4.5 the algorithm was allowed a search grid the size of the remaining image. This is impractical and in some instances even wrong as similar patterns could cause false positives. In practice the search grid is limited to some value which allows the desired maximum displacement to be tracked. In that case the search grid needs to cover all possible magnitudes of displacement. This can be further reduced in some cases when a prediction of the amount of movement is known. Then for small movements the search grid can be kept small and grow dynamically when displacement is expected to be larger. The design compromise solution which has been decided on for this problem is to move the tracking problem to higher orders (like acceleration, jerk, etc). By offsetting the search grid by an expected displacement, the search effectively needs to cover acceleration when speed is used as the prediction factor, or jerk if speed and acceleration combined are used as the prediction factor. As these derivatives are often smaller in magnitude the search grid can be further minimized.
- **Frame Size and Position**: As a larger magnitude of displacement is expected, the edges of the frame opposite the direction of displacement are almost certainly not going to overlap in the next

frame. This means that effectively the frame size gets smaller and offset for larger displacements. Earlier in the number of blocks discussion it was decided that blocks should be taken from the center of the frame outwards, but for large displacements this could mean that part of the spiral search pattern does not overlap in the next image. To counteract this it has been decided to effectively change the frame size and position to center around the expected overlapping parts of the frames. This is done by using the predicted displacement plus the search grid size, since the search grid also should be fully overlapping in the next image in order to minimize edge artifacts.

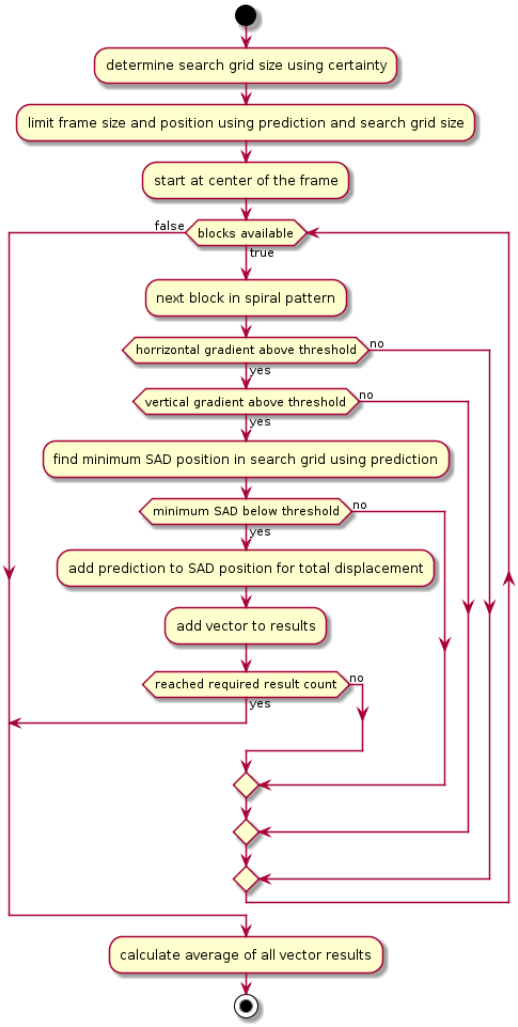

Figure 5.1: Activity design for the flow estimator.

Figure 5.1 shows the UML activity diagram which implements the design considerations. In this diagram purposely a high level is maintained as this leaves the programmer with the freedom to implement the design in whichever language and with whichever implementation details are deemed necessary, while giving a solid foundation and guideline as to which behavior is required for estimating flow. In the diagram is referred to: the certainty which is an estimate of how the prediction is expected to vary, the prediction which is used as an offset, the horizontal gradient threshold which is the amount of horizontal gradient that needs to be present in order to meet the intensity structure quality criteria, the vertical gradient threshold which is the amount of vertical gradient that needs to be present in order to meet the intensity structure quality criteria, the SAD threshold which is the maximum allowable

SAD value - being a measure of how close the match was. The definition of the SAD function is given in equation 4.12; it should be noted however that in this context the definition is changed, to limit the search field significantly using the determined search grid size.

5.5 Theoretical Components

The set of designed components:

- **Camera**:
  - **Resolution**: 258 pixels
  - **Viewing angle**: $8.578°$
  - **Exposure time**: $< 0.00058s$
  - **Frame rate**: $40Hz$
- **Height Sensor**:
  - **Error**: $< 2.666\%$
- **Rotation Sensor**:
  - **Error**: $< 0.038°/interval$

Chapter 6

Sensor Fusion and State Estimation

Kalman Filters (KFs) have become popular in use since their invention in the 1960's, mostly due to the fact that they actually worked, were easy to implement in discrete logic and that they could be applied without necessarily understanding or caring about the intricacies of their derivation (Zarchan and Musoff, 2013) [20].

Since their invention, KFs have been used in applications that include providing estimates for navigating the Apollo spacecraft, predicting short-term stock market fluctuations, and estimating user location with relatively inexpensive hand-held Global Positioning System (GPS) receivers (Zarchan and Musoff, 2013) [20].

One such application as described in (Gaylor and Lightsey, 2003) [10] uses an Extended Kalman Filter (EKF) to accurately estimate position and velocity metrics from two data sources in a space environment while working with both absolute and relative measurements and intermittent performance of one of the sensors. Which is in many ways similar to what the goal is to accomplish with the GM and UWB combination. For this reason the KF will be the subject of this chapter.

In (Zarchan and Musoff, 2013) [20] it is stated that often the hardest part of implementation is: the setting up problem. This will be the focus of this chapter, explaining the problem and deriving the solution together with the benefits and potential pitfalls as described in literature.

6.1 The Makings of a KF

This thesis will not attempt to explain the derivations or even low working level details of a KF, but for the reader's understanding a short introduction is contained in this section.

A KF functions on two basic steps. The first, in which the system state x and accompanying certainty (probability distribution) P is projected forward in time through a linear combination of the system state and a process model F. Here the process model F can be seen as a model description of how the state varies over time. The second step is to update the forward projected system state $\hat{x}$ (where the hat denotes a model based prediction) with a new measurement z. The predict step as it is called can be seen in equation 6.1, and the update step can be seen in equation 6.2. These examples assume a discrete time implementation using matrices. An overview of the matrices in equations 6.1 and 6.2 can be seen in list 6.1.

List 6.1: The Kalman filter matrices (Roger, 2015) [17]

- x The system state. *A column vector with dimensions* $(n, 1)$ *which contains the means of the Gaussian Probability Distributions (GPDs). These means represent the system state.*
- P The system state probability. *A square matrix with dimensions* (n, n)*. It describes the (co-) variance of the GPDs of the system state* x.
- F The state transition function. *A square matrix with dimensions* (n, n)*. It is the system model which propagates* x_k *to* $\hat{\mathrm{x}}_{k+1}$ *through linear combination of the system state variables (the hat denotes a model based prediction).*
- Q Process noise. *A square matrix with dimensions* (n, n)*. As no model is absolute, the propagation of* x_k *to* $\hat{\mathrm{x}}_{k+1}$ *through* F *increases the probability space. This prediction error is modeled as a noise by adding a zero mean (co-) variance to the system state probability.*
- H The measurement function. *A row vector with dimensions* $(1, n)$*. Used to select a single state variable and transform it between filter state space and measurement space.*
- z The measurement. *In the form of a matrix with dimensions* $(1, 1)$.
- R The measurement noise. *Variance of* z*, represented as a matrix with dimensions* $(1, 1)$.
- y The residual. *Difference between the transformed state space of* $\hat{\mathrm{x}}$ *and* z*, in the form of a matrix with dimensions* $(1, 1)$.
- S The state uncertainty. *An uncertainty of the tracked state variable after incorporating* R*, represented as a matrix of dimensions* $(1, 1)$.
- K The Kalman gain. *A scaling factor used on* y *to update the state mean* x*. It is based on* P *and* R *and is a column vector of dimensions* $(n, 1)$.

Equation 6.1 The Kalman Filter Predict function (Roger, 2015) [17]. The hat indicates a prediction and the subscript denotes the time step. An overview of the involved matrices can be found in list 6.1.

$$\begin{aligned} \hat{\mathrm{x}}_{k+1} &= \mathrm{F}\mathrm{x}_k \\ \hat{\mathrm{P}}_{k+1} &= \mathrm{F}\mathrm{P}_k\mathrm{F}^T + \mathrm{Q} \end{aligned} \tag{6.1}$$

Equation 6.2 The Kalman Filter Update function (Roger, 2015) [17]. Where I is an identity matrix of the correct dimensions. The hat indicates a prediction and the subscript denotes the time step. An overview of the involved matrices can be found in list 6.1.

$$\begin{aligned} \mathrm{y}_{k+1} &= \mathrm{z}_{k+1} - \mathrm{H}\hat{\mathrm{x}}_{k+1} \\ \mathrm{S}_{k+1} &= \mathrm{H}\hat{\mathrm{P}}_{k+1}\mathrm{H}^T + \mathrm{R} \\ \mathrm{K}_{k+1} &= \hat{\mathrm{P}}_{k+1}\mathrm{H}^T\mathrm{S}_{k+1}^{-1} \\ \mathrm{x}_{k+1} &= \hat{\mathrm{x}}_{k+1} + \mathrm{K}_{k+1}\mathrm{y}_{k+1} \\ \mathrm{P}_{k+1} &= (\mathrm{I} - \mathrm{K}_{k+1}\mathrm{H})\hat{\mathrm{P}}_{k+1} \end{aligned} \tag{6.2}$$

A kalman filter represents the state variables it tracks as multivariate GPDs, with the mean stored in the x vector and the multivariate co-variances stored in the P matrix (Roger, 2015) [17]. The accumulated estimate mean and certainty is propagated through time by means of the the predict function 6.1. From (Roger, 2015) [17] it is stated that the update can be seen as GPD multiplication. Multiplying two GPDs together creates a new GPD with a weighted mean and a greater certainty or smaller variance (Roger, 2015) [17].

In the predict step the variance and mean of the previous state is propagated to the next time step.

As the process model almost certainly will not exactly represent the process, the prediction will have some error introduced. This error is modeled by adding a zero mean GPD to the variance and mean, effectively this is decreasing the certainty of the prediction, or increasing the probability space of the prediction. In the update step a new measurement is incorporated through GPD multiplication, which increases the certainty, or decreases the state probability space (Roger, 2015) [17].

These steps are repeated each time step, propagating the estimate and incorporating measurements. Repeating the predict and update steps, the KF will eventually converge to a degree of state certainty, which in the case that the tracked phenomenon and process noise are Gaussian is the most optimal state certainty and prediction attainable (Kalman, 1960) [13].

6.2 Kalman Filters and Sensor Fusion

The KF update function is identical to Gaussian multiplication (Roger, 2015) [17]. When sensor measurements can be represented as Gaussians, the measurement is the mean and the measurement error is Gaussian distributed, then fusion of two sensors via Gaussian multiplication produces the best possible estimate (Kalman, 1960) [13]. The assumption is then that Gaussian multiplication can be sequentially done on two or more sensors to produce the best possible estimate. This assumption is true if Gaussian multiplication is symmetrical for all multiplications. This has been proven to be the case in appendix A. Thus as Gaussian multiplication is identical to the KF update function, the KF update function can be used to sequentially fuse multiple sensors. This is the method also used in (Caron et al., 2006) [6].

It should be noted that when different sensors or sensors with different noise profiles are used, that the R sensor noise factor should also be changed per update. Furthermore it could be necessary to change the H measurement function in order to target derivative or anti-derivative values of that tracked value.

The generalization for an i'th sensor fusion update function is presented in 6.3.

Equation 6.3 The Kalman Filter Update function for an i'th sensor fusion. Where I is an identity matrix of the correct dimensions. The hat indicates a prediction and the subscript denotes the time step with the i denoting the sensor in the fusion sequence. An overview of the involved matrices can be found in list 6.1.

$$\begin{aligned}
\mathrm{y}_{k+1,i} &= \mathrm{z}_{k+1,i} - \mathrm{H}_i\hat{\mathrm{x}}_{k+1,i-1} \\
\mathrm{S}_{k+1,i} &= \mathrm{H}_i\hat{\mathrm{P}}_{k+1,i}\mathrm{H}_i^T + \mathrm{R}_i \\
\mathrm{K}_{k+1,i} &= \hat{\mathrm{P}}_{k+1,i}\mathrm{H}_i^T\mathrm{S}_{k+1,i}^{-1} \\
\mathrm{x}_{k+1,i} &= \hat{\mathrm{x}}_{k+1,i-1} + \mathrm{K}_{k+1,i}\mathrm{y}_{k+1,i} \\
\mathrm{P}_{k+1,i} &= (\mathrm{I} - \mathrm{K}_{k+1,i}\mathrm{H}_i)\hat{\mathrm{P}}_{k+1,i}
\end{aligned} \tag{6.3}$$

6.3 System Definition

Kalman filters track state variables, but as of yet the state variables are undefined for the UAS. The filter tracks and feeds information concerning navigation and flight stability to the navigation subsystem, see Fig: 2.2. In chapter Specifications and Requirements (2) the information of interest is defined as the absolute position, speed and acceleration of the UAS. Thus the filter states will comprise position, speed and acceleration. These state values will be defined in a Cartesian coordinate system, with some absolute relation to the environment - greenhouse - where the UAS is to operate.

With the values of interest known the dynamic behavior of the system - UAS - can be defined in terms of changes of position, speed, and acceleration in the Cartesian coordinate space. In Cartesian coordinate space the changes of position and speed are defined by classical mechanics. A simple function for the system model is given in 6.4.

Equation 6.4 The state variables of each Cartesian dimension defined as a linear combination of the previous state plus some noise factor η. Where the noise factor η models the otherwise unmodeled variations in the state variables.

$$\begin{aligned}
\{p_{x,k+1}\ ,\ v_{x,k+1}\ ,\ a_{x,k+1}\} &= \{p_{x,k} + v_{x,k}\Delta t + a_{x,k}\frac{\Delta t^2}{2} + \eta_{x,p}\ ,\ v_{x,k} + a_{x,k}\Delta t + \eta_{x,v}\ ,\ a_{x,k} + \eta_{x,a}\} \\
\{p_{y,k+1}\ ,\ v_{y,k+1}\ ,\ a_{y,k+1}\} &= \{p_{y,k} + v_{y,k}\Delta t + a_{y,k}\frac{\Delta t^2}{2} + \eta_{y,p}\ ,\ v_{y,k} + a_{y,k}\Delta t + \eta_{y,v}\ ,\ a_{y,k} + \eta_{y,a}\} \\
\{p_{z,k+1}\ ,\ v_{z,k+1}\ ,\ a_{z,k+1}\} &= \{p_{z,k} + v_{z,k}\Delta t + a_{z,k}\frac{\Delta t^2}{2} + \eta_{z,p}\ ,\ v_{z,k} + a_{z,k}\Delta t + \eta_{z,v}\ ,\ a_{z,k} + \eta_{z,a}\}
\end{aligned} \tag{6.4}$$

The model defined in 6.4 does not describe the behavior of the UAS with regards to any random time interval. It is an approximation, which is valid for small time steps Δt which follow consecutively, and a correct noise factor. In the real world this noise factor is basically the dynamic nature of the system, and is therefor not model-able as a known change. It is however model-able as a uncertainty factor, which is exactly what can be applied in a KF. In other words the noise factor from equation 6.4 needs to fall inside the probability space of the Q process noise matrix.

6.4 Data Transformation

In section System Definition (6.3) the assumption was made that the UWB and GM produce measurements as vectors inside a Cartesian coordinate space. While this assumption stands, it does not mean that the Cartesian coordinate spaces of two sensors are the same. The spaces can still be offset, scaled and rotated in reference to each other. In order for the KF to work with these different measurement values they need to be inside the same Cartesian reference frame, meaning the rotation and scale need to be the same, and in case of absolute measurements the offset should also be the same. The offset is not important in the case of a relative measurement because this offset can be corrected in the filter itself.

For the UAS there are three possible information sources which can be incorporated into the filter, the UWB, GM and the Inertial Measurement Unit (IMU). The UWB provides absolute measurements so it is assumed to already work in the world reference frame, thus requiring no transformations. The GM measures speeds along the image lines of the optical sensor. As these lines are orientated at some angle in respect to the UAS frame, and the UAS frame has a free orientation in respect to the world frame, the vectors are most often rotated in respect to the world frame. As the optical sensor remains relatively perpendicular to the ground, and internal compensation can be applied, the GM measurement is assumed to be unaffected by pitch and roll rotations. This leaves yaw rotation which can be transformed back using an euler angle rotation matrix, shown in equation 6.5, via multiplication with a column vector.

Equation 6.5 The rotation matrix for the z axis (Diebel, 2006) [8]. Note that α is defined as clockwise rotation.

$$\begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.5}$$

The IMU's UAS frame can however be rotated around all three axes which complicates the reverse transformation. The solution to this problem is outside the scope of this thesis and will be left as an exercise to the reader, if the IMU measurement is desired in the filter.

6.5 Filter Initialization

KFs require an initial value from which to start and eventually converge to the actual system state (Roger, 2015) [17]. Both the x and P state and state co-variance matrices need to be initialized at the start of filtering.

The filter state x can be initialized at zero but then the uncertainty of the filter should be large to be able to converge quickly. A more optimal and the recommended way is to use the first measurement as the starting value. The state co-variance matrix P can be initialized with the sensor specific R measurement variance values on the diagonals. For values which are not measured directly the maximum assumed variance for this value can be used. The author notes that this is not the only possible approach nor that it is the best, but it should function for most situations, and further research is outside the scope of this thesis.

As mentioned earlier, KFs converge to the actual system state. For this reason it can be advisable to let the filter run for a certain amount of time before using the filter results. This could be done to ensure the filter is converged and stable before other systems start to rely on it.

6.6 Example Implementation

An example implementation of the process state x is given in equation 6.6.

Equation 6.6 An example implementation for the x process state. p, v and a with their subscripts are respectively position, velocity and acceleration in their respective axis.

$$\mathrm{x} = \begin{pmatrix} p_x & v_x & a_x & p_y & v_y & a_y & p_z & v_z & a_z \end{pmatrix}^T \tag{6.6}$$

An example implementation of process model F for the process state of equation 6.6 is given in equation

Equation 6.7 An example implementation for the F process model with respect to the state definition of equation 6.6. It is also an implementation in three dimensions of the model described in equation 6.4.

$$\mathrm{F} = \begin{pmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.7}$$

As can be seen in equation 6.7 the process model F needs to be defined with respect to the time step Δt. The process noise matrix Q is defined in equation 6.8.

Equation 6.8 The process noise matrix Q can be calculated as defined here. The process model F is taken from equation 6.7. The process noise spectral density $\Phi_s \in \mathbb{R}$ cannot be determined right now, and in all cases is something which should be experimentally defined to find the best working value. It represents the white noise error of the process prediction. Calculation taken from (Roger, 2015) [17].

$$\mathrm{Q}_c = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \Phi_s$$

$$\mathrm{Q} = \int_0^{\Delta t} \mathrm{F}(t)\mathrm{Q}_c\mathrm{F}^T(t)dt$$

$$\mathrm{Q} = \begin{pmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{pmatrix} \Phi_s \tag{6.8}$$

Equation 6.17 Example implementations for the H measurement function matrices. The measurement function is defined per axis and variable type, distinguished by the subscript, where x, y and z are the respective axis and p, v and a are the position, velocity and acceleration of each axis respectively.

$$\mathrm{H}_{x,p} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.9}$$

$$\mathrm{H}_{x,v} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.10}$$

$$\mathrm{H}_{x,a} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.11}$$

$$\mathrm{H}_{y,p} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.12}$$

$$\mathrm{H}_{y,v} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.13}$$

$$\mathrm{H}_{y,a} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{6.14}$$

$$\mathrm{H}_{z,p} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{6.15}$$

$$\mathrm{H}_{z,v} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{6.16}$$

$$\mathrm{H}_{z,a} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.17}$$

The implementations of the x system state and P system co-variance can be extrapolated from sections Filter Initialization (6.5), The Makings of a KF (6.1) and Example Implementation (6.6). The individual R sensor variance values can be experimentally defined by first setting it to the expected (or tested) variance value, and then adjusting it to make the filter perform better. There might be a way to unequivocally determine the values for R, P and Q, but the author has not found any such method in research. Thus the implementation of these leaves some experimentation to get the optimal result.

The filter update function can be taken from equation 6.1 and the relevant update function is described in equation 6.3.

Chapter 7

Theory Verification

In this chapter the unit tests for research question four will be defined and analyzed. With the goal of either validating or invalidating the theory discussed in chapters 4 and 6.

7.1 Methodology

In this section the methodology for research question four as described in chapter 3 will be supplemented with the exact definition of the unit tests to be performed.

Each unit test will produce the following results:

- **The maximum relative error**: *the maximum relative deviation for a single measuring step*
- **The average relative error**: *a measure of the overall single measurement step precision*
- **The standard deviation of the relative error**: *a measure of the overall single measurement step precision*
- **The maximum absolute error**: *the largest deviation from the actual path*
- **The average absolute error**: *a measure of the overall integrated measurement precision*
- **The loop closure distance**: *the distance between the start of the path and the end of the path*
- **The distribution of the relative measurement error**: *a heat map scatter plot of the relative error, showing the error distribution*
- **The integration of the measurements**: *a picture or graph which shows the path integration verses the actual path*

A total of eight unit tests have been devised, of which the last one is an integration test:

1. Testing the baseline components at maximum height
2. Testing the baseline components at maximum speed
3. Testing the effect of resolution
4. Testing the effect of exposure
5. Testing the effect of height error
6. Testing the effect of rotation error
7. Testing double precise specification

8. Testing sensor fusion - Integration test

These tests have been devised to each verify or invalidate one conclusion from the theory. They have been included here in abbreviated form but the full test specifications can be seen in appendix B.

7.2 Test Setup

The test setup is aimed at simulating the flight of the UAS while controlling most environmental variables.

Camera Setup

To simulate the camera on board of the UAS, a rail system is used inside the green house (see Fig: 7.1) to translate a camera at a fixed height, fixed angle and fixed speed.

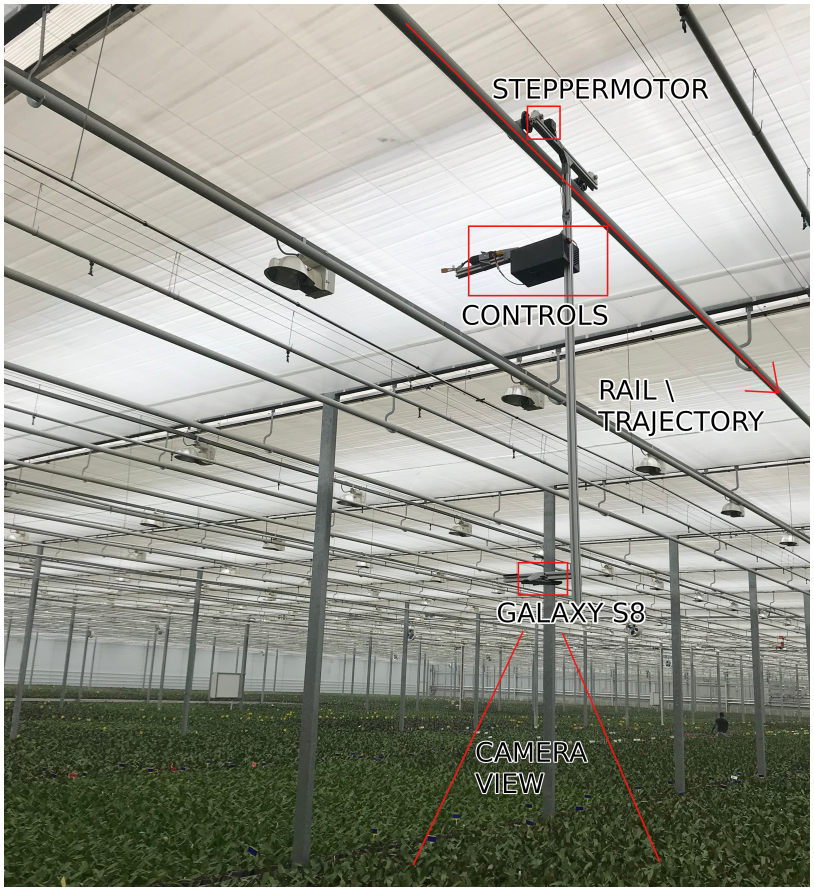

Figure 7.1: The testsetup rail system. In the red squares from top to bottom: a steppermotor for consistent speed, rail system control box, the Galaxy S8 phone used as a camera.

The camera used is the rear mounted camera module of a Samsung Galaxy S8 phone with the following specifications:

- **Sensor type**: Samsung S5K2L2

- **Resolution**: UHD $3840x2160$ pixels
- **Frame rate**: $30Hz$
- **Pixel pitch**: 1.4μ
- **Focal length**: $4.2mm$
- **Viewing angle**: $\approx 40°$

The specifications of the rail system and mounting locations are as follows:

- **Translation speed**: $14.073cm/s$
- **Image depth (scene to lens)**: $1.97m$
- **Height of plants**: $\pm 15cm$
- **Trajectory**: a line of $45m$
- **Flow algorithm**: a custom implementation of the flow estimator design from section Flow Estimator Design (5.4) (source code provided in appendix C)

With this setup the height, angle and speed are assumed to be constant or invariable. With these variables controlled they can be reintroduced via simulation in a controlled way to verify their individual effects.

Uncontrolled in this setup is the exposure time and thus smearing effect. As the rail system will be moving at a comparatively slow speed and plenty of light will be available, the dynamic exposure time of the used camera is assumed to be negligible.

Different speeds can be simulated by skipping frames in regular intervals, thus creating the illusion of achieving a multiple of the base speed of $14.073cm/s$.

Height, Angle & UWB Simulation

As discussed in the previous section, the height and angle of the camera setup are assumed to be constant and invariable. The effects of these controlled variables can then be manipulated through adding a simulated amount of each to the tests.

For the height this can be achieved by taking the base height and adding a zero offset noise with the required variance to the calculations.

For the angular error this can be achieved by applying the theory from section Rotational Compensation Theory (4.4). Which states that the uncompensated rotation manifests itself in the measurement as a translation. Thus by taking a zero mean noise with the required variance as the angular error, calculating the perceived translation for this error and adding this to the measurement, the angular error can be simulated.

The UWB can be simulated by taking the absolute position of the rail system and adding a zero mean noise with the required variance to it.

Camera Specifications Simulation

As the camera is over-specified, the exactly required specifications for the tests will need to be generated from the video footage. This is done in the following ways:

- **Viewing angle**: cropping the image using OpenCV

- **resolution**: scaling (sub sampling) the image using OpenCV
- **Exposure time**: calculating the smearing distance in pixels and performing the smearing by combining pixels in a path using custom software (source code provided in appendix D)
- **Frame rate**: by redefining the frame time, and thereby in effect changing the base speed the frame rate can be changed

Trajectory Simulation

The rail system follows a straight line trajectory. For the tests the trajectory needs to follow the sides of a square. To achieve this the video frames will be rotated 90° every 4th part of the image sequence. By rotating an additional 90° every 4th part of the image sequence the line trajectory is effectively transformed into a square. All simulated variables can be adjusted through the same method. Refer to appendix H for a picture detailing this.

Possibly Unaccounted Variables

As mentioned earlier, the height, speed and rotation of the rail system are assumed to be constant and invariable. But the surface of the greenhouse (the image object) itself does vary in height. Plant heights and table angle in relation to the rail are possibly going to affect the tests.

In running the rail system it was noted that it was sometimes able to rotate in the roll direction slightly.

It is as of yet unknown if and how much these factors will influence the tests. But if deviations in the predictions are found, these effects will be taken into account in an attempt to explain the deviations.

7.3 Results

Unit Test 1 - the baseline components at maximum height

Maximum height is $3m$, see appendix: B.

- **The maximum relative error**: $1.6984cm$
- **The average relative error**: $0.0028cm$
- **The standard deviation of the relative error**: $0.1444cm$
- **The maximum absolute error**: $50.8374cm$
- **The average absolute error**: $22.4502cm$
- **The loop closure distance**: $36.5122cm$

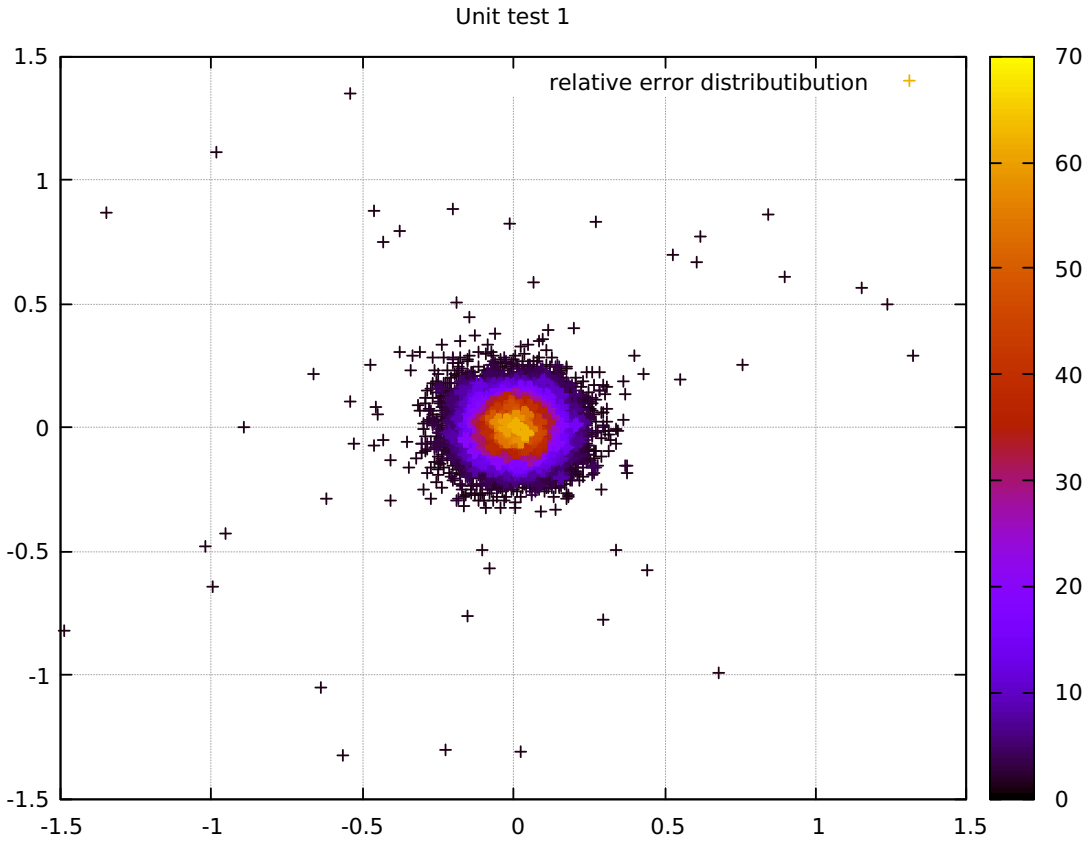

Figure 7.2: Unit test 1 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points.

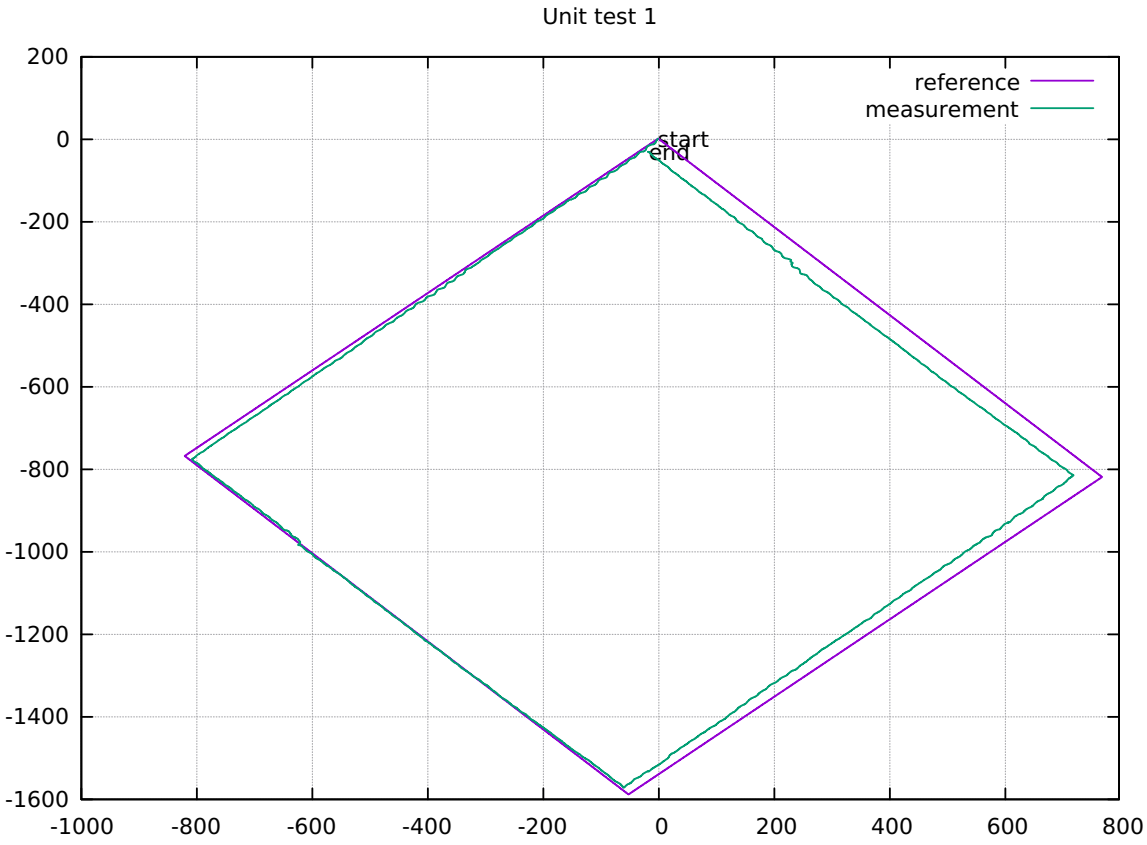

Figure 7.3: Unit test 1. Scale in centimeters.

Unit Test 2 - the baseline components at maximum speed

Maximum speed is $4.2m/s$, see appendix: B.

- **The maximum relative error**: $10.5591cm$
- **The average relative error**: $0.0824cm$
- **The standard deviation of the relative error**: $1.5630cm$
- **The maximum absolute error**: $41.1311cm$
- **The average absolute error**: $27.8082cm$
- **The loop closure distance**: $35.1193cm$

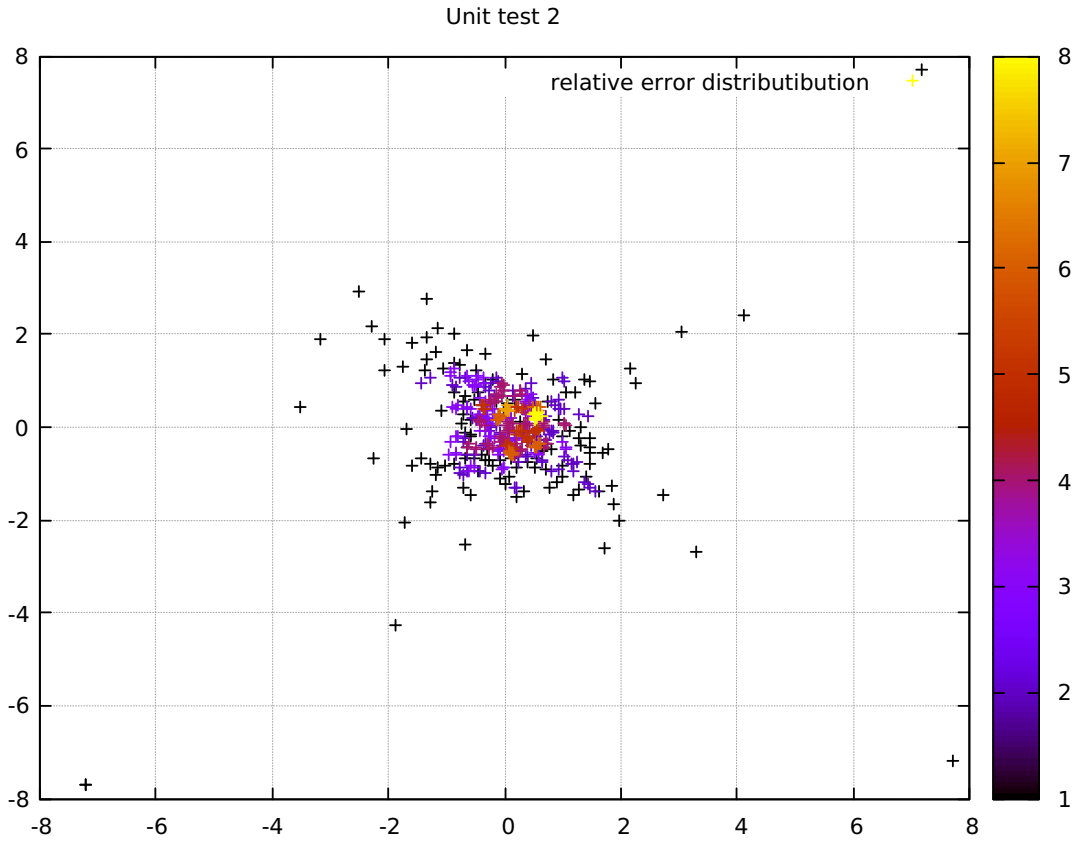

Figure 7.4: Unit test 2 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0064cm^2$, and a total of 426 points.

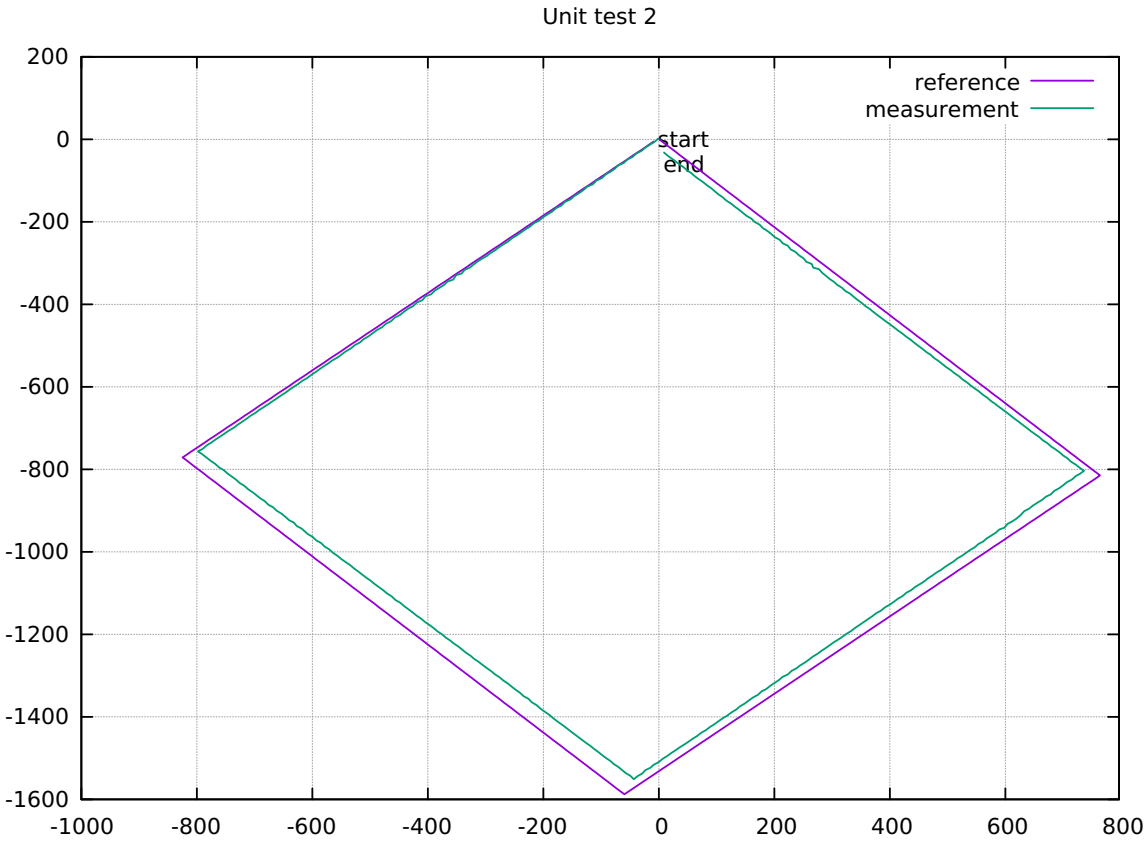

Figure 7.5: Unit test 2. Scale in centimeters.

Unit Test 3 - the effect of resolution

The resolution is halved to 129 pixels, see appendix: B.

- **The maximum relative error**: $5.4411cm$
- **The average relative error**: $0.0049cm$
- **The standard deviation of the relative error**: $0.1938cm$
- **The maximum absolute error**: $159.4568cm$
- **The average absolute error**: $100.7407cm$
- **The loop closure distance**: $63.0357cm$

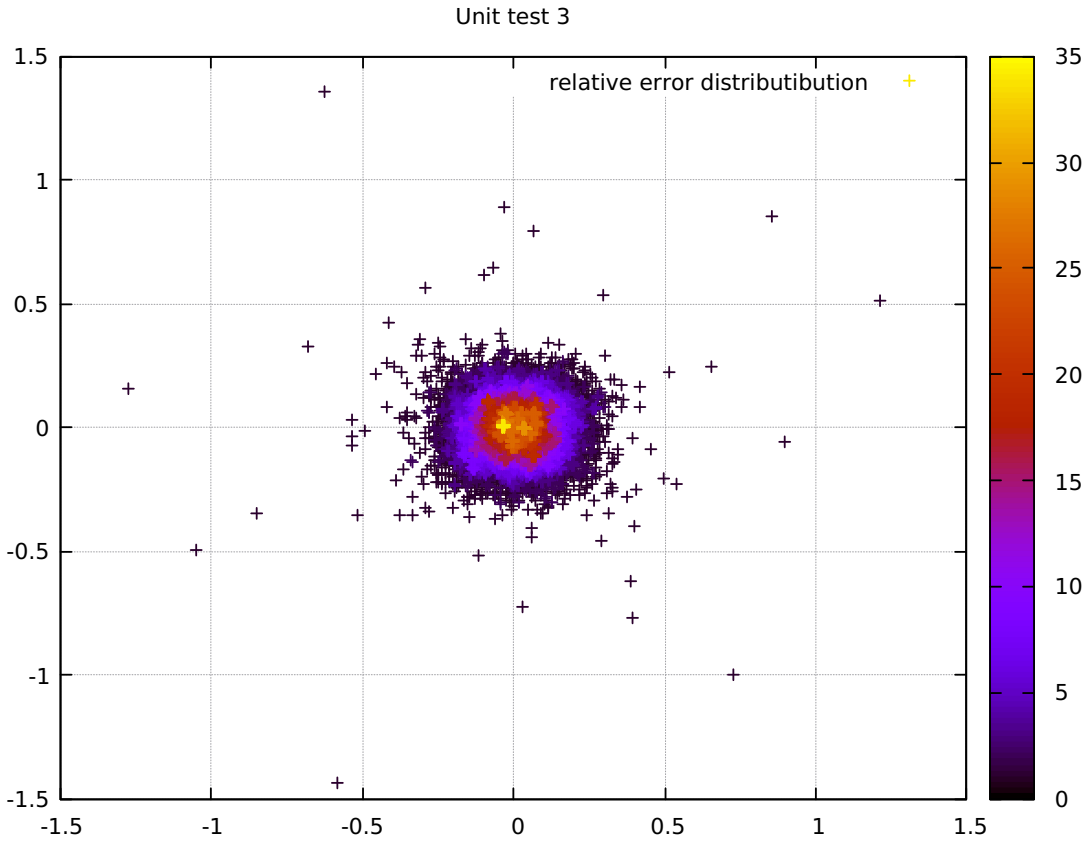

Figure 7.6: Unit test 3 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points.

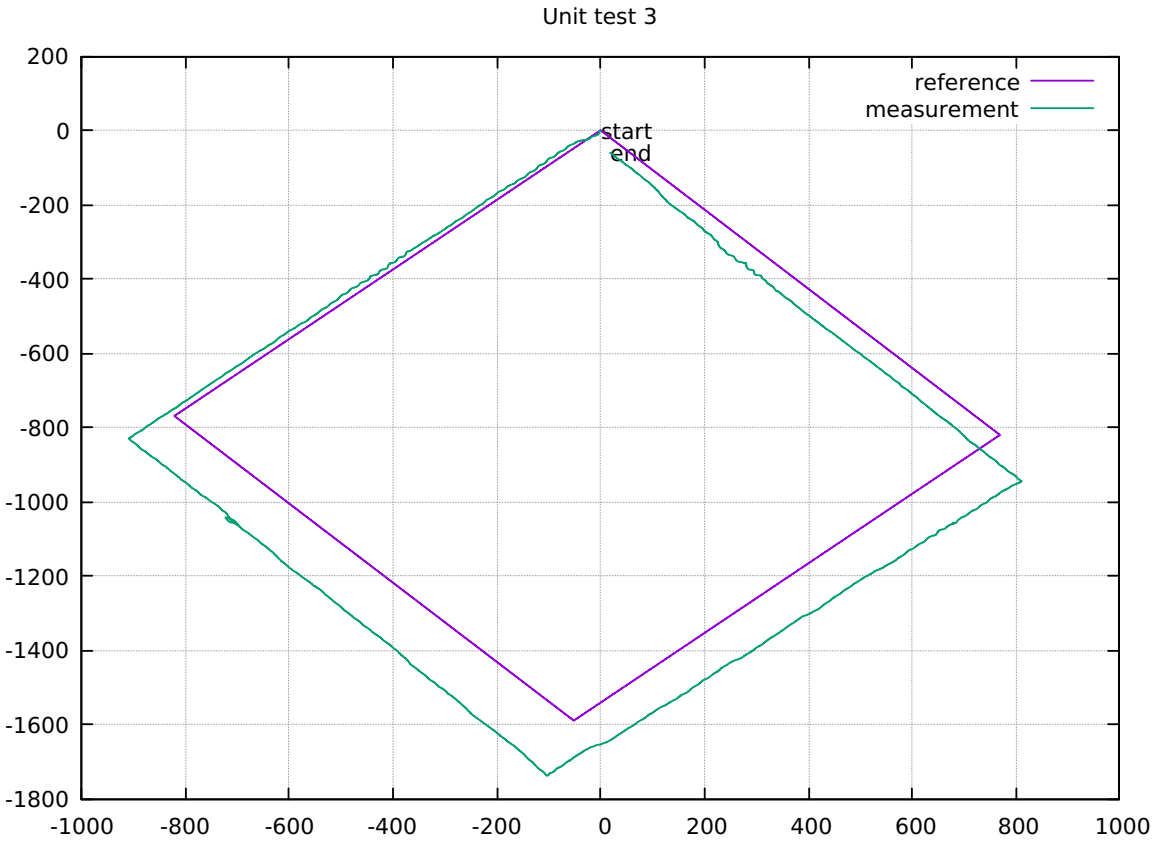

Figure 7.7: Unit test 3. Scale in centimeters.

Unit Test 4 - the effect of exposure

The exposure time is doubled to $1.16ms$, see appendix: B.

- **The maximum relative error**: $8.2545cm$
- **The average relative error**: $0.0832cm$
- **The standard deviation of the relative error**: $1.2844cm$
- **The maximum absolute error**: $37.0516cm$
- **The average absolute error**: $17.4654cm$
- **The loop closure distance**: $35.4495cm$

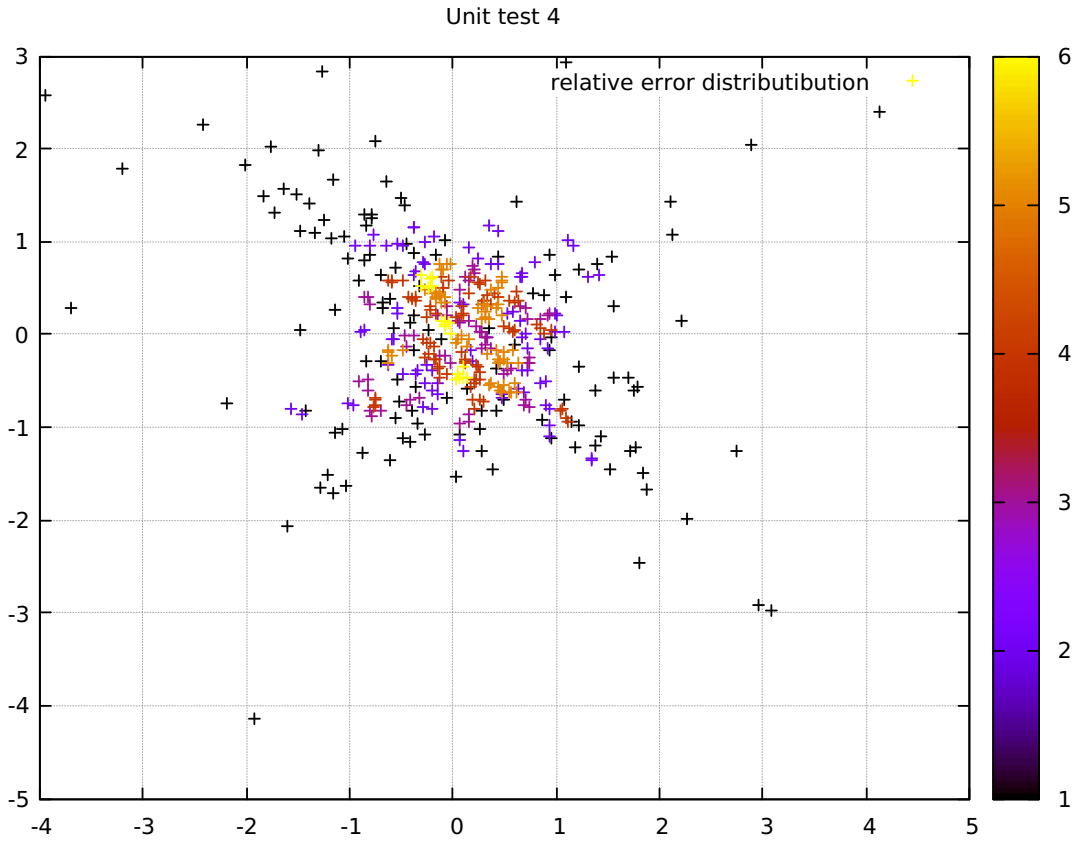

Figure 7.8: Unit test 4 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0064cm^2$, and a total of 426 points.

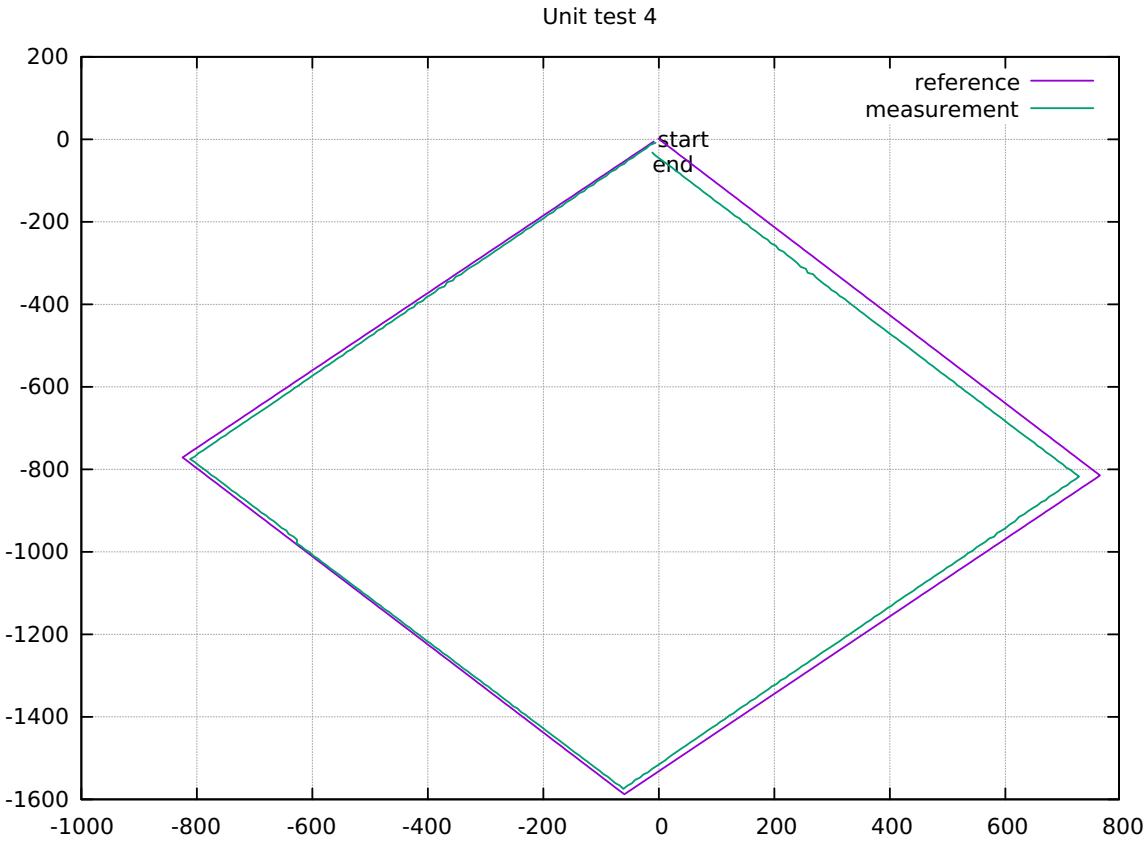

Figure 7.9: Unit test 4. Scale in centimeters.

Unit Test 5 - the effect of height error

The height error is doubled to $\pm 5.333\%$, see appendix: B.

- **The maximum relative error**: $1.5651cm$
- **The average relative error**: $0.0029cm$
- **The standard deviation of the relative error**: $0.1453cm$
- **The maximum absolute error**: $45.1463cm$
- **The average absolute error**: $19.8176cm$
- **The loop closure distance**: $37.2534cm$

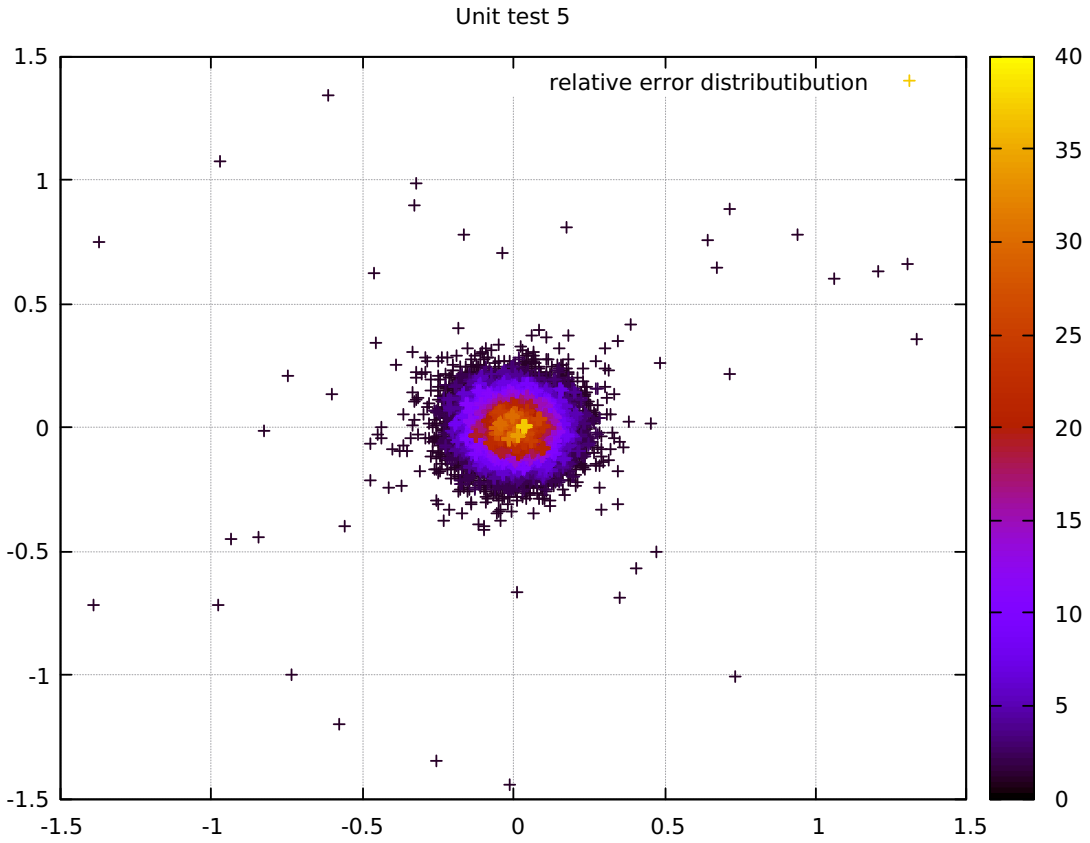

Figure 7.10: Unit test 5 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points.

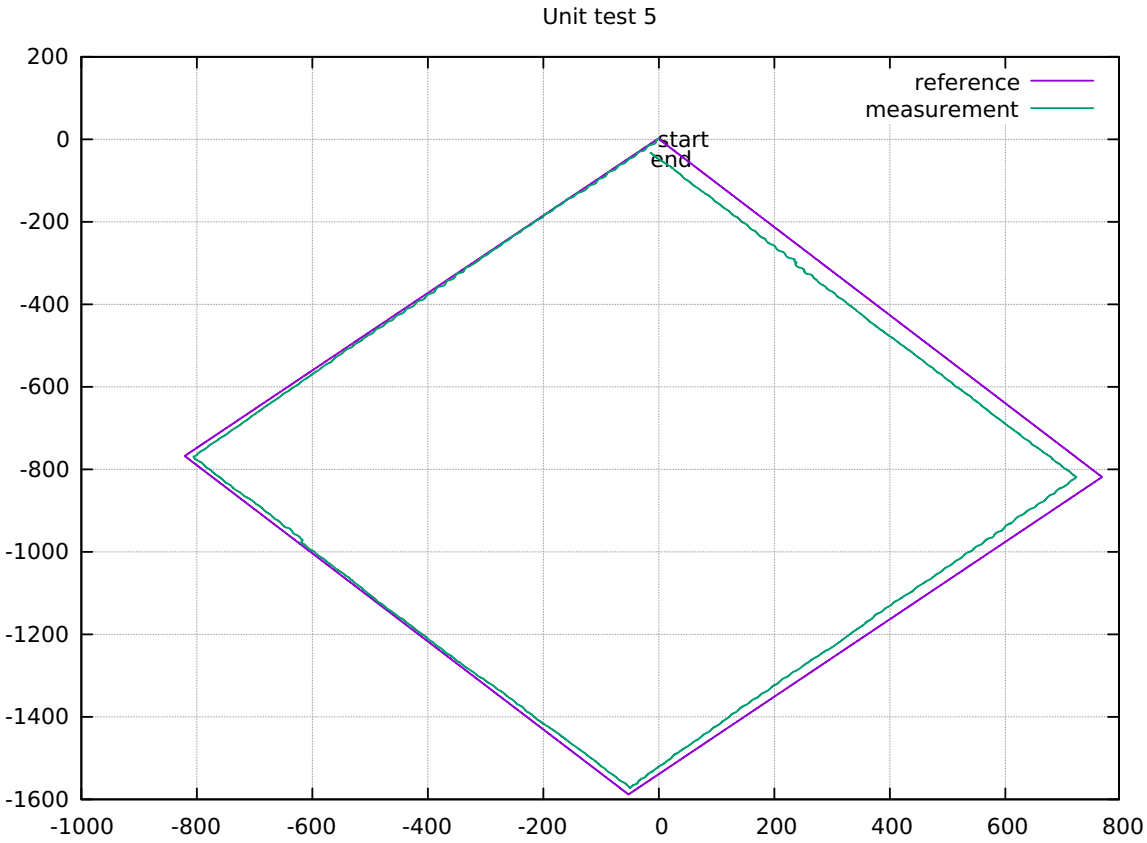

Figure 7.11: Unit test 5. Scale in centimeters.

Unit Test 6 - the effect of rotation error

The rotation error is doubled to $\pm 0.076°$, see appendix: B.

- **The maximum relative error**: $1.6216cm$
- **The average relative error**: $0.0024cm$
- **The standard deviation of the relative error**: $0.2176cm$
- **The maximum absolute error**: $66.4305cm$
- **The average absolute error**: $29.9082cm$
- **The loop closure distance**: $31.4260cm$

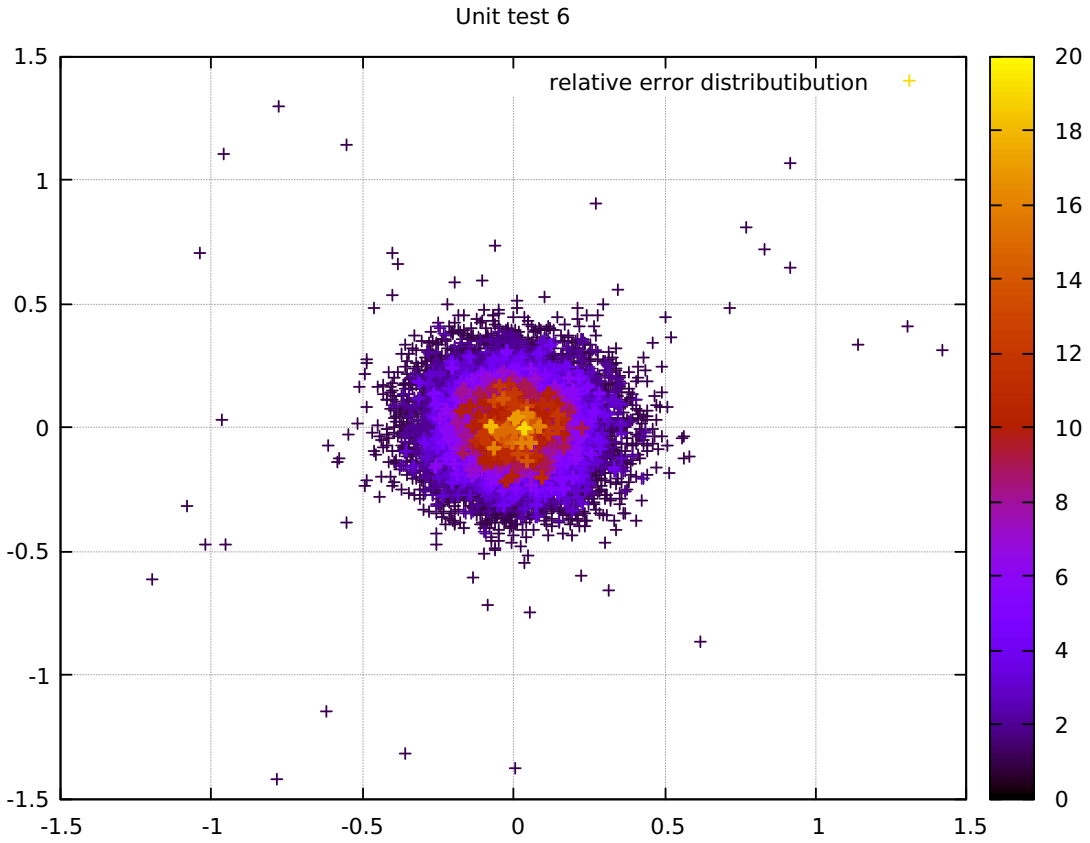

Figure 7.12: Unit test 6 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points.

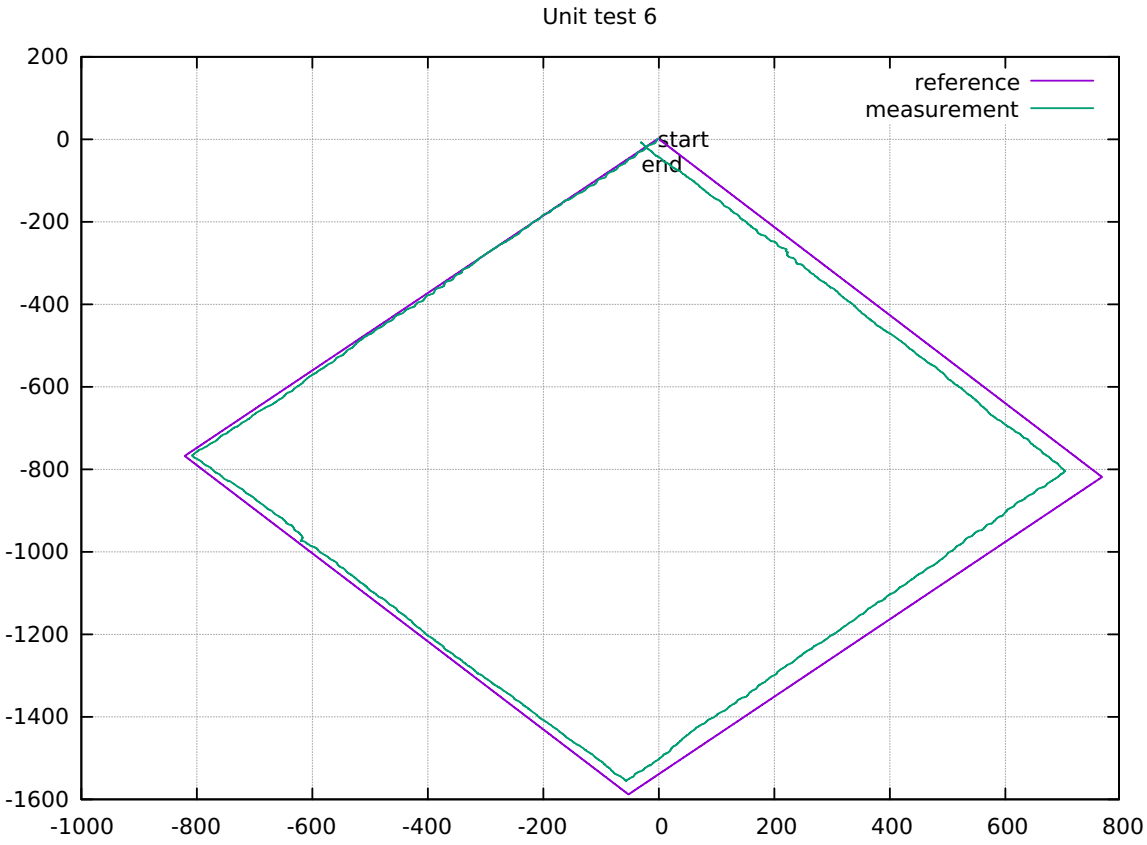

Figure 7.13: Unit test 6. Scale in centimeters.

Unit Test 7 - double precise specification

All errors halved and resolution doubled, see appendix: B.

- **The maximum relative error**: $0.7841cm$
- **The average relative error**: $0.0033cm$
- **The standard deviation of the relative error**: $0.1003cm$
- **The maximum absolute error**: $42.6928cm$
- **The average absolute error**: $19.4199cm$
- **The loop closure distance**: $42.6928cm$

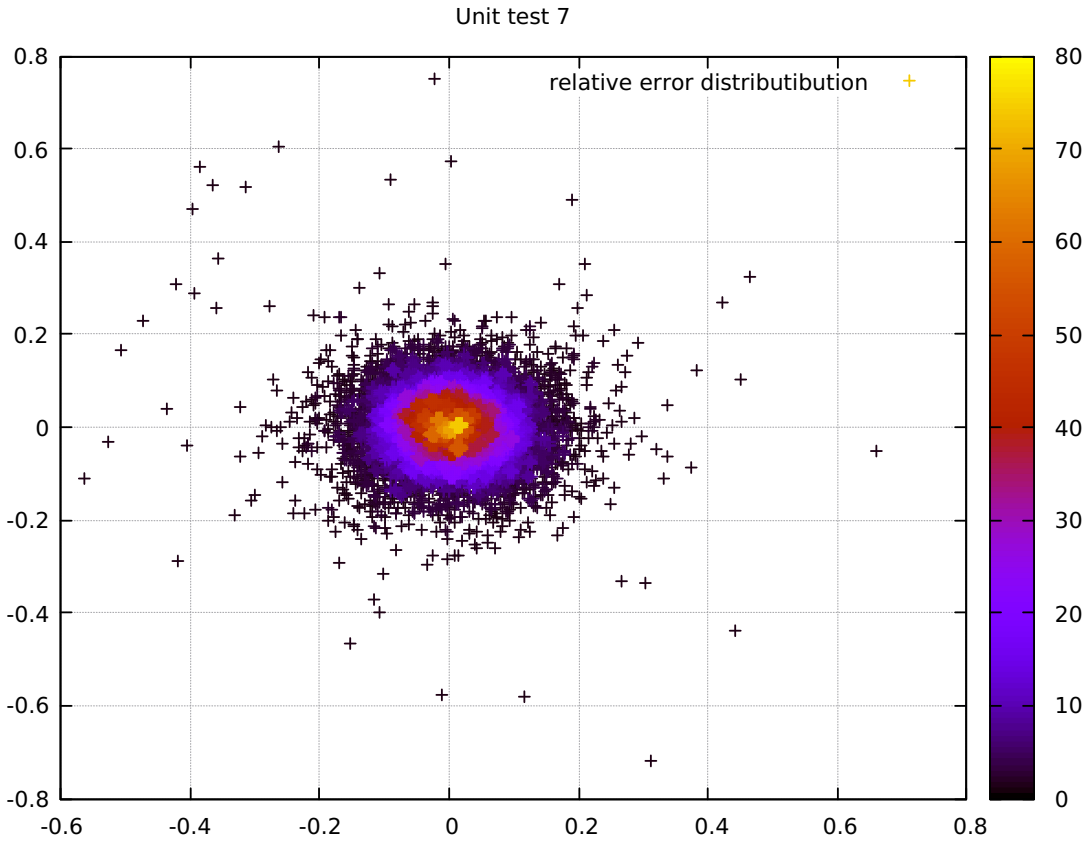

Figure 7.14: Unit test 7 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points.

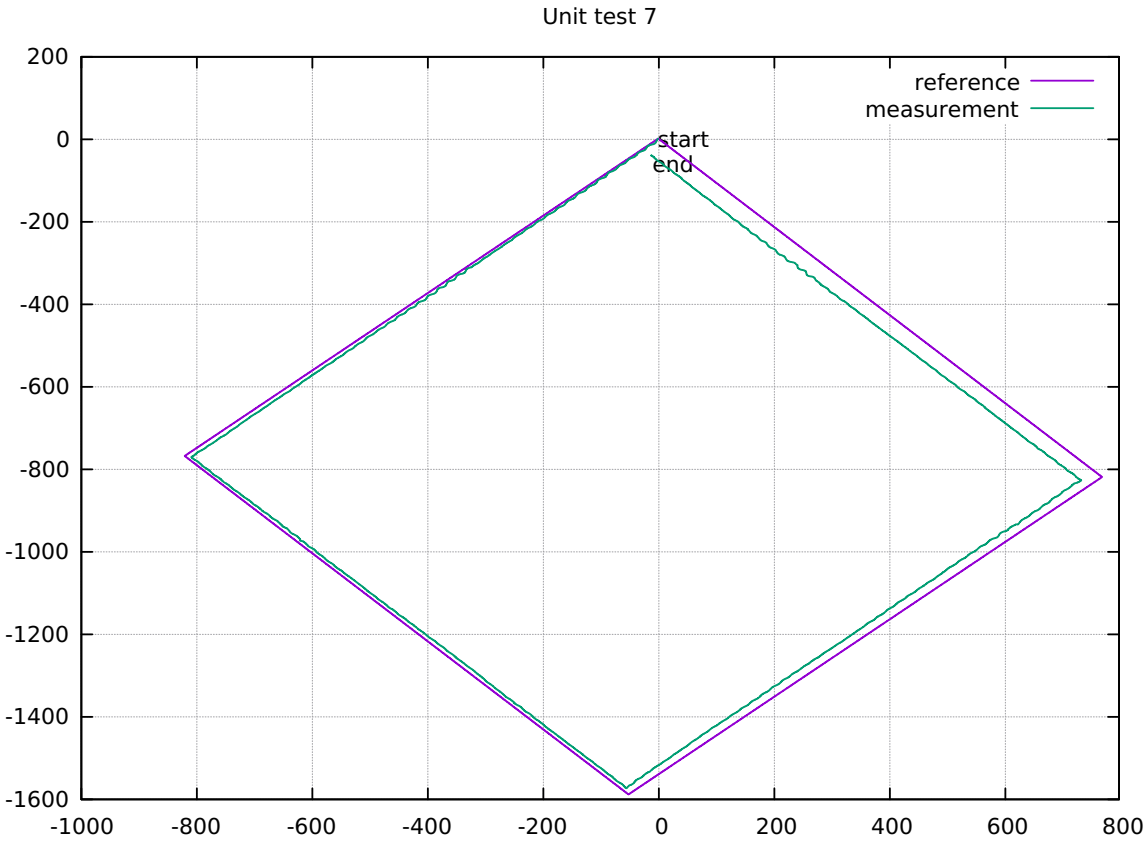

Figure 7.15: Unit test 7. Scale in centimeters.

Integration Test - sensor fusion

Simulation data for the UWB can be seen in appendix: E.

- **The maximum relative error**: 1.0650cm
- **The average relative error**: 0.0028cm
- **The standard deviation of the relative error**: 0.0976cm
- **The maximum absolute error**: 14.7383cm
- **The average absolute error**: 5.4987cm
- **The loop closure distance**: 11.9363cm

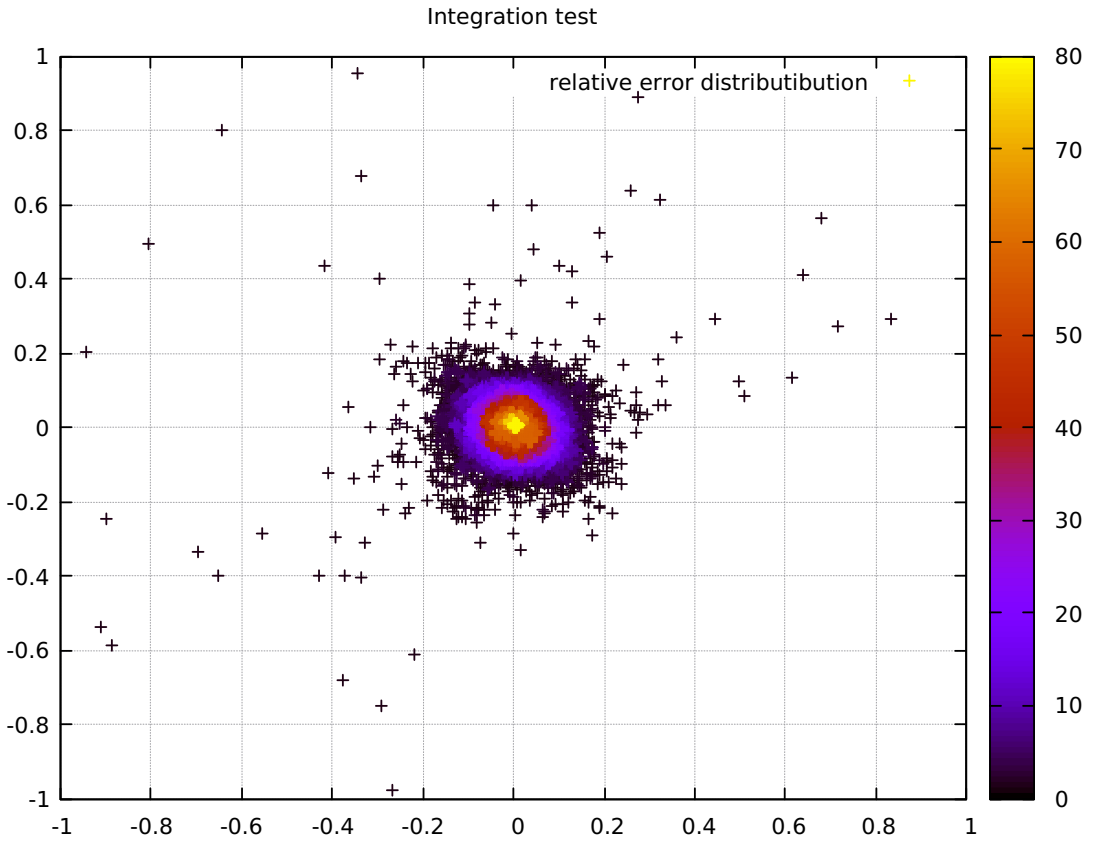

Figure 7.16: Integration test relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points.

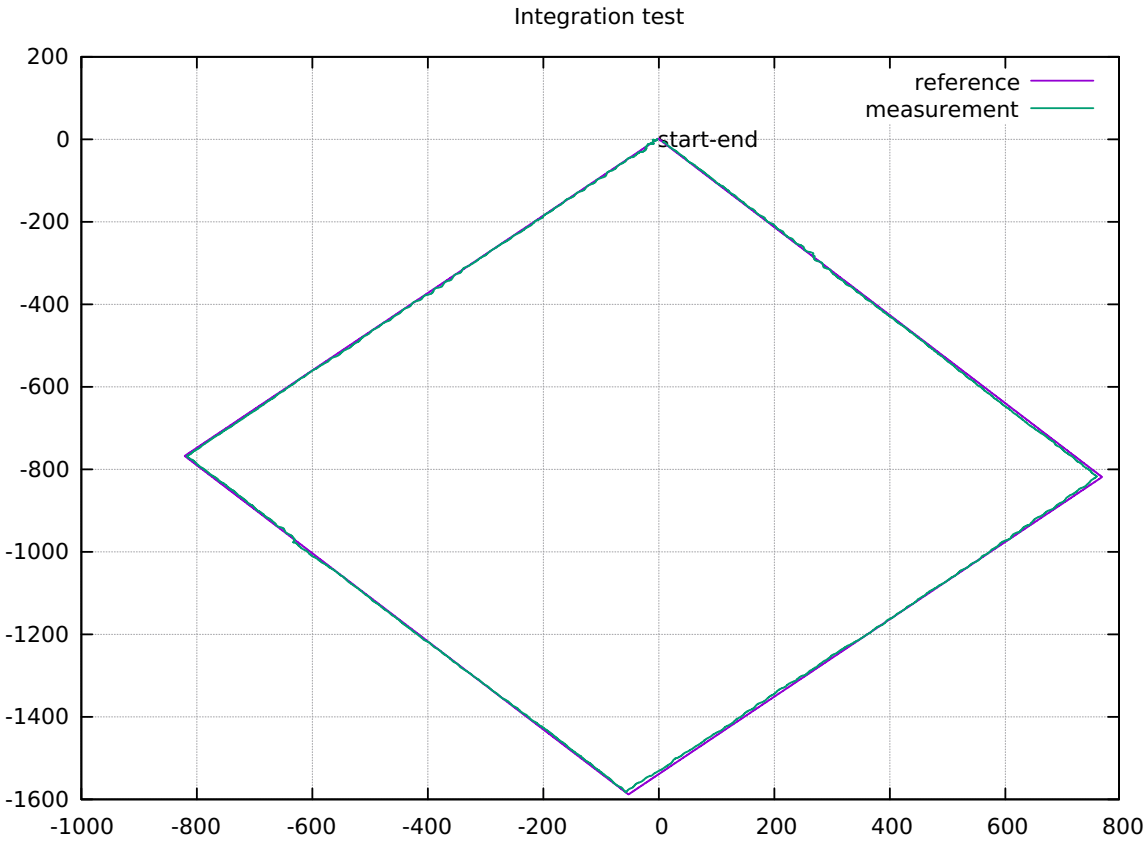

Figure 7.17: Integration test. Scale in centimeters.

Chapter 8

Conclusion

From SQ1 and chapter System Precision Factors (4) it can be seen that several parameters factor into the precision of a system such as the GM. Chapter Theory Verification (7) which answers SQ4, makes it highly plausible that by taking the factors from SQ1 into account for SQ2, as has been done in chapter Theoretical System Design (5), and using the sensor fusion techniques from SQ3 and chapter Sensor Fusion and State Estimation (6), that a precise and stable positioning for the UAS can be obtained.

SQ.I. Which significant parameters determine the accuracy of the GM?

First the major system components were identified as the following: a camera, a flow estimator, a height measurement, a rotation sensor and an information processing system.

Per component, literature was studied to find relevant characteristics of the components with regard to the system measurement precision.

For the camera it was identified that the magnification factor, the smearing amount and nyquist spatial wavelength play a role in the system precision.

For the height measurement, the measurement itself plays a role. Two examples of measurement techniques have also been examined to compare benefits and downsides.

For the flow estimator it was determined that the local intensity complexity and the camera parameters matter in terms of its precision.

Lastly the rotation sensor was looked at to see what kind of impact it has on the measurement precision.

Control over these parameters results in a very stable and precise relative (local) measurement that has a tendency to drift when observed over longer periods of time.

SQ.II. How to design the GM such that it meets all the requirements by incorporating the significant parameters?

The major components and parameters from SQ1 have been used in this research question to form a definition of theoretical components. These components are aimed at fitting the requirements exactly.

This was achieved by first making a semi arbitrary division of the error budget from the requirements, and then per component applying the theory to determine the specifications.

For the optical sensor a 35% slice of the error budget of $1cm$ was taken. This resulted in a resolution of 258 pixels, a viewing angle of 8.578°, a maximum exposure time of $0.00058s$ and a frame rate of $40Hz$.

The height measurement was budgeted at 20% of the $1cm$ total budget. This worked out to a component which has a maximum of 2.666% of measurement inaccuracy.

A slice of 30% of the total error budget of $1cm$ was given to the rotation sensor. This worked out to a measurement error per time step that must be smaller than 0.038°.

The last 15% of the error budget has been kept for unforseen factors.

The flow estimator has not been given a slice of the error budget, as it essentially is dependent on the camera for its contribution to the precision. The general algorithm discussed in chapter System Precision Factors (4) has however been expanded on to improve on quality of block matching and limiting the algorithm run time. This has been accomplished by making a design description for image intensity structure quality control, controlling the number of blocks which are needed, limiting the search grid size and making the frame size and position the most effective. Lastly an UML activity diagram was used to give a high level overview of the algorithm flow and behavior.

SQ.III. How can the data from the GM and UWB be combined and filtered such that useful positioning information is obtained?

Kalman filtering was selected to be the focus of this question. This was done as interest in this subject was expressed from the HiPerGreen project, and as a preliminary literature study concluded that this technique was a great fit for this question.

As this question was only posed to supplement the original assignment, the emphasis was put on informing instead of researching.

Chapter Sensor Fusion and State Estimation (6) provides a short introduction to Kalman Filtering. Sensor fusion as a method of combining sensor data using Kalman Filters has been explored; and it has been found that the Kalman Filter update function works for this purpose. Then a process model was designed, and all matrices for a functioning Kalman Filter were given. Thus an example design for the project has been provided.

The Kalman filter was designed to use cartesian coordinate space and a newtonian process model. A cartesian coordinate space was used because all sensors for which the filter was designed function in such a space. The linearity of newtonian physics allowed for the use of a standard (linear) Kalman filter, which fits a cartesian coordinate space and simplifies design when compared to non linear filtering techniques.

SQ.IV. Does the GM with the design considerations applied realize the goal of increasing the accuracy of the UAS?

A test setup which was meant to control all parameters exactly was first specified, and then used to carry out the generation of a test data set.

A total of eight unit tests were defined to test most of the parameters from chapter System Precision Factors (4).

Through simulation techniques the test data set was adapted to each of the individual unit tests.

Before the tests were done it was noted that the rail system seemed to rotate in the roll direction. When observing the path graphs of unit test 1 these roll movements can clearly be identified as a sinusoidal ripple in the path (see appendix: F). However since all measurements have been compared to an ideal (absolutely straight) path, this ripple caused by the roll of the rail system has an unidentified impact on several of the results. Furthermore it has been identified that the rail system at times ran over welding seams on the heating pipe (rail). This caused a jolt to the system and a (relatively) violent shuddering movement which could have caused measurement errors in places.

As higher speeds were simulated from the same data set, the shudders and rolls were also sped up and thus amplified in terms of their effect per time step. In the high speed tests this had a detrimental effect on some of the results.

Unit test 1 was meant as the baseline at maximum height and lowest speed, and was intended to exactly fit the requirements. The standard deviation of the relative error of $0.1444cm$ fell easily within the designed and expected standard deviation of $0.3333cm$. In this matter the baseline components outperformed the expectations. The average relative error also was very small at $0.0028cm$, meaning that the relative error has very little bias. This can also be observed by looking at Fig: 7.3. It can be seen that the integrated path of measurements does not deviate a great deal. The maximum relative error was larger then desired, most likely due to the rail system jolts or rolls.

Unit test 2 was meant as the baseline at lowest height and maximum speed, and was intended to exactly fit the requirements. The standard deviation of the relative measurement error was larger than in unit test 1, and also larger than the design requirements had predicted. As this is a sped up unit test, these deviations are likely due to the jolts and rolls being amplified. As can be seen from the integrated path, Fig: 7.5, the resulting measurement is still performing roughly identical to unit test 1.

Unit test 3 was meant to test the effect of halving the resolution. It was predicted that this would have a detrimental effect on the relative measurement error standard deviation, this effect was barely seen. This is likely due to all unit tests being tainted by the roll and jolt effects. One interesting and unexplained phenomenon is the integrated path which seems to indicate a scaling error, see Fig: 7.7.

Unit test 4 was meant to test the effect of exposure time. It is a sped up test again, which basically invalidates most relative results. The integrated path followed exceptionally well see Fig: 7.9. The effect of exposure can't be validated nor dismissed through these results.

Unit test 5 was meant to test the effect of a greater variance of the height measurement. Due to the effects of the roll and jolts and the normally distributed nature of the height error, almost no additional effect can be observed.

Unit test 6 was meant to test the effect of a greater variance of the rotation error. Due to the effects of the roll and jolts and the normally distributed nature of the rotation error, only a small effect can be observed.

Unit test 7 was meant to test the effect of making all components doubly precise. While a small improvement can be observed, most of the benefit was likely lost due to the roll and jolt problems of the rail system.

The integration test was meant to test the sensor fusion techniques discussed in chapter Sensor Fusion and State Estimation (6). The fusion was performed on the results from unit test 1 and a simulation of a UWB. The UWB simulation was created by taking the ideal path and adding a $30cm$ standard deviation to each step. The results from the test improve on all results from unit test 1. The most notable result can be seen in Fig: 7.17. It is obvious that the sensor fusion eliminated the noise of the UWB and effectively eliminated the integration error over time from the GM. Although the filter was designed to be able to fuse the UWB, GM and IMU, in this test only the UWB and GM were used as IMU data was unavailable.

Chapter 9

Recommendations

Further research and development is required for the GM. The recommendations have been split into first and second priority. The first priority recommendations are needed to fully implement the GM and the second priority recommendations are for further refinement and improvement.

First priority:

1. Fully validate the optical flow accuracy in a real UAS environment:
   To fully validate the GM system performance it should be integrated onto an actual UAS and tested further.
2. Implement and tune the Kalman filters:
   The Kalman filter used to run the unit tests was not extensively tuned. When integrating and testing the systems on a functioning and real UAS the tuning of such a filter can perhaps further improve the location precision.

Second priority:

1. Research into creating height maps on board a UAS:
   As the height sensor is one dimensional, it only measures the height of a certain area after which the whole scene is assumed to conform to this height. This however is not always the case. A height map, produced by using the disparity technique with two camera's, structured light or possibly LIDAR, can be used to give a height to each flow vector, possibly improving on accuracy and especially in situations where more depth variation is expected. These height maps could also be used for things such as object detection, collision avoidance and SLAM.
2. Hardware Printed Circuit Board (PCB) design for the GM:
   The hardware which was selected for the GM can be connected to each other using wires, but a better solution would be to have a full PCB design which combines all components. This could allow for better vibration resistance, better resistance to humidity through the use of conformal coating and more reliable connections.
3. Research into integrating all UAS flight systems into one module:
   Aggregating all flight systems into a single module could save on weight and space which on the UAS are at a premium.

List of Figures

1.1 A testing platform UAS flying through a demo greenhouse, indicating the scale that is involved . . . 3

2.1 The UAS with a GM onboard. . . . 7
2.2 An overview of the UAS subsystems. . . . 9

4.1 The Raspberry PI prototype. Consists of a HC-SR04 sonar height sensor, Raspberry PI Camera V2.1 and the Raspberry PI 3B. . . . 13

5.1 Activity design for the flow estimator. . . . 26

7.1 The testsetup rail system. In the red squares from top to bottom: a steppermotor for consistent speed, rail system control box, the Galaxy S8 phone used as a camera. . . . 36
7.2 Unit test 1 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points. . . . 39
7.3 Unit test 1. Scale in centimeters. . . . 40
7.4 Unit test 2 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0064cm^2$, and a total of 426 points. . . . 41
7.5 Unit test 2. Scale in centimeters. . . . 42
7.6 Unit test 3 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points. . . . 43
7.7 Unit test 3. Scale in centimeters. . . . 44
7.8 Unit test 4 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0064cm^2$, and a total of 426 points. . . . 45
7.9 Unit test 4. Scale in centimeters. . . . 46
7.10 Unit test 5 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points. . . . 47
7.11 Unit test 5. Scale in centimeters. . . . 48
7.12 Unit test 6 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points. . . . 49
7.13 Unit test 6. Scale in centimeters. . . . 50
7.14 Unit test 7 relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points. . . . 51
7.15 Unit test 7. Scale in centimeters. . . . 52
7.16 Integration test relative measurement error distribution. Scale in centimeters, color scale in points per $0.0001cm^2$, and a total of 12790 points. . . . 53
7.17 Integration test. Scale in centimeters. . . . 54

E.1 Ultra Wide Band Simulation data. Scale in centimeters. Generated by taking the reference and adding a noise having a standard deviation of $30cm$ to each measurement interval. . . . 85

F.1 The ripple effect in the integrated measurement of unit test 1 (Fig: 7.3), as produced by the rail system's sinusiodal rotations in the roll axis. 86

G.1 An example of optical flow where the right image is translated 140 pixels down and to the left compared to the left image. The *green* blocks on the left can be seen as the reference features, and the *purple* blocks on the right the corresponding feature locations in the translated image. 87

H.1 The frames of the straight line trajectory are rotated 90° every quarter of the total trajectory thus forming a square trajectory. 88

List of Tables

4.1 Percentage increase in speed of sound (re 0 °C) due to moisture in air only. Temperature effects not included except as they pertain to humidity (Bohn, 1988) [5]. 17
4.2 Total percentage increase in speed of sound (re 0 °C) due to temperature and humidity combined (Bohn, 1988) [5]. 17
4.3 List of factors 22

Listings

C.1 flow.c 73
C.2 flow_interface.c 80
D.1 pixelsmearing.c 82

Bibliography

[1] (2013). *MEMS motion sensor: three-axis digital output gyroscope*. STMicroelectronics. Rev. 2.

[2] (2014). *HRLV-MaxSonar ® - EZ TM Series*. MaxBotix ® Inc.

[3] (2018). *VL53L1X*. STMicroelectronics.

[4] Beauchemin, S. S. and Barron, J. L. (1995). The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466.

[5] Bohn, D. A. (1988). Environmental effects on the speed of sound. *Journal of Audio Engineering Society*, 36(4):9.

[6] Caron, F., Duflos, E., Pomorski, D., and Vanheeghe, P. (2006). Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information fusion*, 7(2):221–230.

[7] de Jong, W. (2018). Climate sensor module for greenhouse drone. *HiPerGreen*, page 88. unpublished.

[8] Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35.

[9] Fesselet, L. (2018). Localisation of a uas. unpublished.

[10] Gaylor, D. and Lightsey, E. G. (2003). Gps/ins kalman filter design for spacecraft operating in the proximity of international space station. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 5445.

[11] Honegger, D., Meier, L., Tanskanen, P., and Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *2013 IEEE International Conference on Robotics and Automation*, pages 1736–1741.

[12] Hortipoint.nl (2016). Drones conquer horticulture. `https://www.hortipoint.nl/floribusiness/drones-conquer-horticulture/`. Last checked on Oct 04, 2018.

[13] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45.

[14] Lide, D. R., editor (2005). *CRC Handbook of Chemistry and Physics*. CRC Press LLC, 85 edition. Internet version.

[15] Nieuws Inholland.nl (2017). Wij inholland-winnaars drones in de kas maken doorstart met hipergreen. `http://nieuws.inholland.nl/wij-inholland-winnaars-drones-in-de-kas-maken-doorstart-met-hipergreen/`. Last checked on Oct 04, 2018.

[16] Pedrotti, L. S. (2008). Basic geometrical optics. *Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, accessed Dec*, 5:2017.

[17] Roger, R. L. J. (2015). *Kalman and Bayesian Filters in Python*. Self, 1 edition.

[18] unspecified (2015). Ccd fundamentals. Application Note AND9188/D, ON Semiconductor, P.O. Box 5163, Denver, Colorado 80217 USA. Rev 3.

[19] Wij Inholland.nl (2016). Drones in de kas. `http://wijinholland.nl/nl/winnaar-2016/`. Last checked on Oct 04, 2018.

[20] Zarchan, P. and Musoff, H. (2013). *Fundamentals of Kalman filtering: a practical approach.* American Institute of Aeronautics and Astronautics, Inc.

Appendices

Appendix A

Symmetry in Gaussian Multiplication

To be proven is the symmetry of multiple Gaussian multiplications. Given the formulas of Gaussian multiplication in equation. A.1

Equation A.1 The product of two Gaussians is expressed as a new Gaussian. From (Roger, 2015) [17].

$$\begin{aligned} \mu_{1\cdot 2} &= \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \\ \sigma_{1\cdot 2}^2 &= \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \end{aligned} \tag{A.1}$$

The general structure for n sequentially multiplied Gaussians is assumed to be equation A.2.

Equation A.2 The general structure of n sequential Gaussian multiplications. The bar here is used to denote the product, as a differentiation to the n'th σ or μ.

$$\begin{aligned} D_{n,i} &= \prod_{k=1,k\neq i}^{n} \sigma_k^2 \\ \bar{\mu}_n &= \frac{\sum_{i=1}^{n} \mu_i D_{n,i}}{\sum_{i=1}^{n} D_{n,i}} \\ \bar{\sigma}_n &= \frac{\prod_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{n} D_{n,i}} \end{aligned} \tag{A.2}$$

Equation A.1 applied to the the $n+1$ case yields equation A.3.

Equation A.3 The $n+1$'th Gaussian multiplication.

$$\begin{aligned} \bar{\mu}_{n+1} &= \frac{\bar{\mu}_n \sigma_{n+1}^2 + \mu_{n+1} \bar{\sigma}_n^2}{\bar{\sigma}_n^2 + \sigma_{n+1}^2} \\ \bar{\sigma}_{n+1} &= \frac{\bar{\sigma}_n^2 \sigma_{n+1}^2}{\bar{\sigma}_n^2 + \sigma_{n+1}^2} \end{aligned} \tag{A.3}$$

Expanding the definitions of D in equation A.4.

Equation A.4 Expanded definitions for D.

$$\begin{aligned} D_{n+1,n+1} &= \prod_{k=1}^{n} \sigma_k^2 \\ D_{n+1,i} &= D_{n,i} \sigma_{n+1}^2 \mid i < n+1 \end{aligned} \tag{A.4}$$

Then in equation A.8 the terms of equation A.2 are substituted into equation A.3 and simplified to show that the structure is equivalent to equation A.2, and symmetrical.

Equation A.8 Equation A.5 substitutes the terms of equation A.2 into equation A.3. Equation A.6 simplifies terms by multiplying with $\sum_{i=1}^{n} D_{n,i}$. Next in equation A.7 the terms are further simplified using the definition of D from equation A.4. In equation A.8 the terms are congregated and the result is equivalent to equation A.2 and thus symmetrical.

$$\begin{aligned}\bar{\mu}_{n+1} &= \frac{\frac{\sum_{i=1}^{n} \mu_i D_{n,i}}{\sum_{i=1}^{n} D_{n,i}} \sigma_{n+1}^2 + \mu_{n+1} \frac{\prod_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{n} D_{n,i}}}{\frac{\prod_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{n} D_{n,i}} + \sigma_{n+1}^2} \\ \bar{\sigma}_{n+1} &= \frac{\frac{\prod_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{n} D_{n,i}} \sigma_{n+1}^2}{\frac{\prod_{i=1}^{n} \sigma_i^2}{\sum_{i=1}^{n} D_{n,i}} + \sigma_{n+1}^2}\end{aligned} \tag{A.5}$$

$$\begin{aligned}\bar{\mu}_{n+1} &= \frac{\sum_{i=1}^{n} \mu_i D_{n,i} \sigma_{n+1}^2 + \mu_{n+1} \prod_{i=1}^{n} \sigma_i^2}{\prod_{i=1}^{n} \sigma_i^2 + \sum_{i=1}^{n} D_{n,i} \sigma_{n+1}^2} \\ \bar{\sigma}_{n+1} &= \frac{(\prod_{i=1}^{n} \sigma_i^2) \sigma_{n+1}^2}{\prod_{i=1}^{n} \sigma_i^2 + \sum_{i=1}^{n} D_{n,i} \sigma_{n+1}^2}\end{aligned} \tag{A.6}$$

$$\begin{aligned}\bar{\mu}_{n+1} &= \frac{\sum_{i=1}^{n} \mu_i D_{n+1,i} + \mu_{n+1} D_{n+1,n+1}}{D_{n+1,n+1} + \sum_{i=1}^{n} D_{n+1,i}} \\ \bar{\sigma}_{n+1} &= \frac{\prod_{i=1}^{n+1} \sigma_i^2}{D_{n+1,n+1} + \sum_{i=1}^{n} D_{n+1,i}}\end{aligned} \tag{A.7}$$

$$\begin{aligned}\bar{\mu}_{n+1} &= \frac{\sum_{i=1}^{n+1} \mu_i D_{n+1,i}}{\sum_{i=1}^{n+1} D_{n+1,i}} \\ \bar{\sigma}_{n+1} &= \frac{\prod_{i=1}^{n+1} \sigma_i^2}{\sum_{i=1}^{n+1} D_{n+1,i}}\end{aligned} \tag{A.8}$$

Lastly equation A.1 is expanded to three multiplications and shown to satisfy equation A.2 which through full induction proves symmetry for all sequential Gaussian multiplications (QED).

Equation A.11 Gaussian multiplication applied to three iterations. It is equivalent to equation A.2.

$$\begin{aligned}\mu_{1\cdot2\cdot3} &= \frac{\frac{\mu_1\sigma_2^2+\mu_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}\sigma_3^2 + \mu_3\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}+\sigma_3^2} \\ \sigma_{1\cdot2\cdot3}^2 &= \frac{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}\sigma_3^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}+\sigma_3^2}\end{aligned} \tag{A.9}$$

$$\begin{aligned}\mu_{1\cdot2\cdot3} &= \frac{(\mu_1\sigma_2^2+\mu_2\sigma_1^2)\sigma_3^2+\mu_3\sigma_1^2\sigma_2^2}{\sigma_1^2\sigma_2^2+\sigma_3^2(\sigma_1^2+\sigma_2^2)} \\ \sigma_{1\cdot2\cdot3}^2 &= \frac{\sigma_1^2\sigma_2^2\sigma_3^2}{\sigma_1^2\sigma_2^2+\sigma_3^2(\sigma_1^2+\sigma_2^2)}\end{aligned} \tag{A.10}$$

$$\begin{aligned}\mu_{1\cdot2\cdot3} &= \frac{\mu_1\sigma_2^2\sigma_3^2+\mu_2\sigma_1^2\sigma_3^2+\mu_3\sigma_1^2\sigma_2^2}{\sigma_1^2\sigma_2^2+\sigma_1^2\sigma_2^2+\sigma_2^2\sigma_3^2} = \frac{\sum_{i=1}^3\mu_i D_{3,i}}{\sum_{i=1}^3 D_{3,i}} \\ \sigma_{1\cdot2\cdot3}^2 &= \frac{\sigma_1^2\sigma_2^2\sigma_3^2}{\sigma_1^2\sigma_2^2+\sigma_1^2\sigma_2^2+\sigma_2^2\sigma_3^2} = \frac{\prod_{i=1}^3\sigma_i^2}{\sum_{i=1}^3 D_{3,i}}\end{aligned} \tag{A.11}$$

Appendix B

Unit Test Definitions

1. **Theoretical component test, maximum height**:
   - **Resolution**: 258 pixels
   - **Viewing angle**: 8.5°
   - **Exposure time**: $0.58ms$
   - **Frame rate**: $40Hz$
   - **Height**: $3m \pm 2.666\%$
   - **Rotation**: $0° \pm 0.038°$
   - **Speed**: $0.14m/s$
   - **Trajectory**: four $11.25m$ sides of a square

   Component specifications in concordance with the theoretical components. *This test specifically tests the theoretical components as the baseline, at the maximum height required, which should perform in concordance with the requirements.*
2. **Theoretical component test, maximum speed**:
   - **Resolution**: 258 pixels
   - **Viewing angle**: 8.5°
   - **Exposure time**: $0.58ms$
   - **Frame rate**: $40Hz$
   - **Height**: $1m \pm 2.666\%$
   - **Rotation**: $0° \pm 0.038°$
   - **Speed**: $\approx 4.2m/s$
   - **Trajectory**: four $11.25m$ sides of a square

   Component specifications in concordance with the theoretical components. *This test specifically tests the theoretical components as the baseline, at the maximum speed required, which should perform in concordance with the requirements.*
3. **Half resolution test**:
   - **Resolution**: 129 pixels

- **Viewing angle**: $8.5°$
- **Exposure time**: $0.58ms$
- **Frame rate**: $40Hz$
- **Height**: $3m \pm 2.666\%$
- **Rotation**: $0° \pm 0.038°$
- **Speed**: $0.14m/s$
- **Trajectory**: four $11.25m$ sides of a square

Component specifications in concordance with the theoretical components, except resolution. *This test specifically tests the impact of resolution (nyquist spatial wavelength) on the precision of the system.*

4. **Double exposure time**:
   - **Resolution**: 258 pixels
   - **Viewing angle**: $8.5°$
   - **Exposure time**: $1.16ms$
   - **Frame rate**: $40Hz$
   - **Height**: $3m \pm 2.666\%$
   - **Rotation**: $0° \pm 0.038°$
   - **Speed**: $\approx 4.2m/s$
   - **Trajectory**: four $11.25m$ sides of a square

   Component specifications in concordance with the theoretical components, except exposure time. *This test specifically tests the impact of exposure time on the precision of the system.*

5. **Height error doubled**:
   - **Resolution**: 258 pixels
   - **Viewing angle**: $8.5°$
   - **Exposure time**: $0.58ms$
   - **Frame rate**: $40Hz$
   - **Height**: $3m \pm 5.333\%$
   - **Rotation**: $0° \pm 0.038°$
   - **Speed**: $0.14m/s$
   - **Trajectory**: four $11.25m$ sides of a square

   Component specifications in concordance with the theoretical components, except the height precision. *This test specifically tests the impact of the height measurement on the precision of the system.*

6. **Rotation error doubled**:
   - **Resolution**: 258 pixels
   - **Viewing angle**: $8.5°$
   - **Exposure time**: $0.58ms$

- **Frame rate**: $40Hz$
- **Height**: $3m \pm 2.666\%$
- **Rotation**: $0° \pm 0.076°$
- **Speed**: $0.14m/s$
- **Trajectory**: four $11.25m$ sides of a square

Component specifications in concordance with the theoretical components, except for the rotation compensation precision. *This test specifically tests the impact of the rotation error on the precision of the system.*

7. **Double precision**:
   - **Resolution**: 516 pixels
   - **Viewing angle**: 8.5°
   - **Exposure time**: $0.29ms$
   - **Frame rate**: $40Hz$
   - **Height**: $3m \pm 1.333\%$
   - **Rotation**: $0° \pm 0.019°$
   - **Speed**: $0.14m/s$
   - **Trajectory**: four $11.25m$ sides of a square

   This test specifically tests if the system would be doubly as precise when all factors are scaled to be doubly as precise.

8. **Sensor fusion - Integration test**:
   - **Resolution**: 258 pixels
   - **Viewing angle**: 8.5°
   - **Exposure time**: $0.58ms$
   - **Frame rate**: $40Hz$
   - **Height**: $3m \pm 2.666\%$
   - **Rotation**: $0° \pm 0.038°$
   - **Speed**: $0.14m/s$s
   - **Trajectory**: four $11.25m$ sides of a square

   Component specifications in concordance with the theoretical components. *This test specifically tests the fusion of the theoretical components with a simulated UWB.*

Appendix C

Guidance Module Sourcecode

Listing C.1: flow.c

```
/****************************************************************************
 *
 *   Copyright (C) 2018 ADI. All rights reserved.
 *   Author: Mark Ramaker <markramaker@outlook.com>
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, is prohibited unless written consent from the copyright holders
 * is obtained.
 *
 * This software is based on and modified from open source software.
 * Parts the parts of this software which are based on the open source code
 * are available from https://github.com/PX4/Flow and are marked in this document.
 * And their copyright disclaimer (see the disclaimer below) applies solely to the code
 * which can be obtained from the provided link. Any modifications contained within this
 * file belong to the copyright holder ADI.
 *
 * When written permission is obtained the following conditions apply:
 *
 * 1. Redistributions of source code must retain the all copyright
 *    notices, this list of conditions and the all disclaimers.
 * 2. Redistributions in binary form must reproduce the all copyright
 *    notices, this list of conditions and all disclaimers in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name ADI nor the names of its contributors may be
 *    used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 ****************************************************************************/
// In accordance with the PX4 copyright disclaimer, this modified code contains their orriginal disclaimer below.
/****************************************************************************
 *
 *   Copyright (C) 2013 PX4 Development Team. All rights reserved.
 *   Author: Petri Tanskanen <tpetri@inf.ethz.ch>
 *           Lorenz Meier <lm@inf.ethz.ch>
 *           Samuel Zihlmann <samuezih@ee.ethz.ch>
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name PX4 nor the names of its contributors may be
 *    used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
```

```
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 ****************************************************************************/

#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

#ifdef __DEBUG_DISPLAY
#include <window.h>
#define DEBUG_DISPLAY_ACCTION(x) x
#else
#define DEBUG_DISPLAY_ACCTION(x) /* NOTHIN */
#endif

#include "flow_interface.h"
extern struct flow_store_t data;

#ifdef __GM_ARMSIMD
#define __INLINE inline
#define __ASM asm
#include "core_cm4_simd.h"
#else
#define GET_ABS8(x, y) ((x) > (y) ? (x) - (y) : (y) - (x))
#endif

#ifdef __GM_ARMSIMD
//================================================================================
// Code in this section is from the PX4 Development Team.
// Copyright (C) 2013 PX4 Development Team. All rights reserved.
// compliments of Adam Williams
#define ABSDIFF(frame1, frame2, row_size) \
({ \
 int result = 0; \
 asm volatile( \
  "mov %[result], #0\n"           /* accumulator */ \
 \
  "ldr r4, [%[src], #0]\n"        /* read data from address + offset*/ \
  "ldr r5, [%[dst], #0]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #4]\n"        /* read data from address + offset */ \
  "ldr r5, [%[dst], #4]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 1)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 1)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 1 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 1 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 2)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 2)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 2 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 2 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 3)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 3)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 3 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 3 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 4)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 4 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 4 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 5)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 5)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 5 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 5 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 6)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 6)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 6 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 6 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  "ldr r4, [%[src], #(64 * 7)]\n"         /* read data from address + offset*/ \
  "ldr r5, [%[dst], #(64 * 7)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
  "ldr r4, [%[src], #(64 * 7 + 4)]\n"         /* read data from address + offset */ \
  "ldr r5, [%[dst], #(64 * 7 + 4)]\n" \
  "usada8 %[result], r4, r5, %[result]\n"       /* difference */ \
 \
  : [result] "+r" (result) \
  : [src] "r" (frame1), [dst] "r" (frame2) \
  : "r4", "r5" \
  ); \
  \
 result; \
})
// End of the PX4 Development code section.
//================================================================================
```

```
178
179 #elif __GM_X64SIMD
180 // X64 version of ABSDIFF.
181 __attribute__((always_inline)) inline uint32_t ABSDIFF(uint8_t * frame1, uint8_t * frame2, uint32_t row_size
        ) {
182   uint64_t sad = 0;
183   #define INT64_M64(x)  *(__m64*)(&(x))
184   uint8_t (*image1_buff) [8][row_size] = (uint8_t (*) [8][row_size]) (frame1);
185   uint8_t (*image2_buff) [8][row_size] = (uint8_t (*) [8][row_size]) (frame2);
186   for(uint8_t yi = 0; yi < 8; yi++) {
187     INT64_M64(sad) += _m_psadbw(INT64_M64((*image1_buff)[yi][0]), INT64_M64((*image2_buff)[yi][0]));
188   };
189   return (uint32_t) sad;
190 }
191 #else
192 // Platform independant version of ABSDIFF.
193 __attribute__((always_inline)) inline uint32_t ABSDIFF(uint8_t * frame1, uint8_t * frame2, uint32_t row_size
        ) {
194   uint32_t sad = 0;
195   uint8_t (*image1_buff) [8][row_size] = (uint8_t (*) [8][row_size]) (frame1);
196   uint8_t (*image2_buff) [8][row_size] = (uint8_t (*) [8][row_size]) (frame2);
197   for(uint8_t yi = 0; yi < 8; yi++) for(uint8_t xi = 0; xi < 8; xi++) {
198     sad += GET_ABS8((*image1_buff)[yi][xi], (*image2_buff)[yi][xi]);
199   };
200   return sad;
201 }
202 #endif
203
204 /**
205  * @brief Compute the average pixel gradient of all horizontal steps
206  *
207  *
208  * @param image (I) image pointer
209  * @param offX (I) x coordinate of upper left corner of 8x8 pattern in image
210  * @param offY (I) y coordinate of upper left corner of 8x8 pattern in image
211  */
212 static inline uint32_t compute_horizontal_gradient (uint8_t *image, uint32_t row_size) {
213   uint32_t acc = 0;
214   uint8_t (*image_buff) [4][row_size] = (uint8_t (*) [4][row_size]) (image);
215
216   #ifdef __GM_ARMSIMD
217   /* we need to get columns */
218   uint32_t col1 = (*image_buff)[0][0] << 24 | (*image_buff)[1][0] << 16 | (*image_buff)[2][0] << 8 | (*
        image_buff)[3][0];
219   uint32_t col2 = (*image_buff)[0][1] << 24 | (*image_buff)[1][1] << 16 | (*image_buff)[2][1] << 8 | (*
        image_buff)[3][1];
220   uint32_t col3 = (*image_buff)[0][2] << 24 | (*image_buff)[1][2] << 16 | (*image_buff)[2][2] << 8 | (*
        image_buff)[3][2];
221   uint32_t col4 = (*image_buff)[0][3] << 24 | (*image_buff)[1][3] << 16 | (*image_buff)[2][3] << 8 | (*
        image_buff)[3][3];
222
223   /* calc column diff */
224   acc = __USADA8(col1, col2, acc);
225   acc = __USADA8(col2, col3, acc);
226   acc = __USADA8(col3, col4, acc);
227   #else // Platform independant version of the pixel gradient computation.
228   // Calculate sidewards gradient.
229   for (uint8_t yi = 0; yi < 4; yi++) acc += GET_ABS8((*image_buff)[yi][0], (*image_buff)[yi][1]) +
230                                             GET_ABS8((*image_buff)[yi][1], (*image_buff)[yi][2]) +
231                                             GET_ABS8((*image_buff)[yi][2], (*image_buff)[yi][3]);
232   #endif
233   return acc;
234 }
235
236 /**
237  * @brief Compute the average pixel gradient of all vertical steps
238  *
239  *
240  * @param image (I) image pointer
241  * @param offX (I) x coordinate of upper left corner of 8x8 pattern in image
242  * @param offY (I) y coordinate of upper left corner of 8x8 pattern in image
243  */
244 static inline uint32_t compute_vertical_gradient (uint8_t *image, uint32_t row_size){
245   uint32_t acc = 0;
246   uint8_t (*image_buff) [4][row_size] = (uint8_t (*) [4][row_size]) (image);
247
248   #ifdef __GM_ARMSIMD
249   /* calc vertical gradient */
250   acc = __USADA8(*((uint32_t*) &(*image_buff)[0][0]), *((uint32_t*) &(*image_buff)[1][0]), acc);
251   acc = __USADA8(*((uint32_t*) &(*image_buff)[1][0]), *((uint32_t*) &(*image_buff)[2][0]), acc);
252   acc = __USADA8(*((uint32_t*) &(*image_buff)[2][0]), *((uint32_t*) &(*image_buff)[3][0]), acc);
253
254   #else // Platform independant version of the pixel gradient computation.
255   // Calculate gradient downwards.
256   for (uint8_t xi = 0; xi < 4; xi++) acc += GET_ABS8((*image_buff)[0][xi], (*image_buff)[1][xi]) +
257                                             GET_ABS8((*image_buff)[1][xi], (*image_buff)[2][xi]) +
258                                             GET_ABS8((*image_buff)[2][xi], (*image_buff)[3][xi]);
259   #endif
260   return acc;
261 }
262
263 /**
264  * @brief Compute SAD distances of subpixel shift of two 8x8 pixel patterns.
265  *
266  * @param image1 (I) ...
267  * @param image2 (I) ...
268  * @param dir (O) return value of the direction where the lowest SAD value was found.
269  * @param row_size (I) the number of pixels in one row.
270  *
271  * @return the lowest SAD value.
272  */
273 static inline uint32_t compute_subpixel(uint8_t *image1, uint8_t *image2, uint8_t *dir, uint32_t row_size)
274 {
275   /* calculate position in image buffer */
276   uint16_t off1 = 0; // image1
277   uint16_t off2 = 0; // image2
```

```
278   uint32_t acc[8];
279   uint32_t s0, s1, s2, s3, s4, s5, s6, s7, t1, t3, t5, t7;
280   for (uint16_t i = 0; i < 8; i++)
281   {
282     acc[i] = 0;
283   }
284   #ifdef __GM_ARMSIMD
285   //================================================================================
286   // Code in this section is from the PX4 Development Team.
287   // Copyright (C) 2013 PX4 Development Team. All rights reserved.
288   for (uint16_t i = 0; i < 8; i++)
289   {
290     /* compute average of two pixel values */
291     s0 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 1 + (i
       +0) * row_size])));
292     s1 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i+1) * row_size]), *((uint32_t*) &image2[off2 + 1 + (i
       +1) * row_size])));
293     s2 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 0 + (i
       +1) * row_size])));
294     s3 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i+1) * row_size]), *((uint32_t*) &image2[off2 - 1 + (i
       +1) * row_size])));
295     s4 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i+0) * row_size]), *((uint32_t*) &image2[off2 - 1 + (i
       +0) * row_size])));
296     s5 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i-1) * row_size]), *((uint32_t*) &image2[off2 - 1 + (i
       -1) * row_size])));
297     s6 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 0 + (i
       -1) * row_size])));
298     s7 = (__UHADD8(*((uint32_t*) &image2[off2 +  0 + (i-1) * row_size]), *((uint32_t*) &image2[off2 + 1 + (i
       -1) * row_size])));
299     /* these 4 t values are from the corners around the center pixel */
300     t1 = (__UHADD8(s0, s1));
301     t3 = (__UHADD8(s3, s4));
302     t5 = (__UHADD8(s4, s5));
303     t7 = (__UHADD8(s7, s0));
304     /* fill accumulation vector */
305     acc[0] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), s0, acc[0]);
306     acc[1] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), t1, acc[1]);
307     acc[2] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), s2, acc[2]);
308     acc[3] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), t3, acc[3]);
309     acc[4] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), s4, acc[4]);
310     acc[5] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), t5, acc[5]);
311     acc[6] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), s6, acc[6]);
312     acc[7] = __USADA8 ((*((uint32_t*) &image1[off1 + 0 + i * row_size])), t7, acc[7]);
313     s0 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 5 + (i
       +0) * row_size])));
314     s1 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i+1) * row_size]), *((uint32_t*) &image2[off2 + 5 + (i
       +1) * row_size])));
315     s2 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 4 + (i
       +1) * row_size])));
316     s3 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i+1) * row_size]), *((uint32_t*) &image2[off2 + 3 + (i
       +1) * row_size])));
317     s4 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 3 + (i
       +0) * row_size])));
318     s5 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i-1) * row_size]), *((uint32_t*) &image2[off2 + 3 + (i
       -1) * row_size])));
319     s6 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i+0) * row_size]), *((uint32_t*) &image2[off2 + 4 + (i
       -1) * row_size])));
320     s7 = (__UHADD8(*((uint32_t*) &image2[off2 + 4 + (i-1) * row_size]), *((uint32_t*) &image2[off2 + 5 + (i
       -1) * row_size])));
321     t1 = (__UHADD8(s0, s1));
322     t3 = (__UHADD8(s3, s4));
323     t5 = (__UHADD8(s4, s5));
324     t7 = (__UHADD8(s7, s0));
325     acc[0] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), s0, acc[0]);
326     acc[1] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), t1, acc[1]);
327     acc[2] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), s2, acc[2]);
328     acc[3] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), t3, acc[3]);
329     acc[4] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), s4, acc[4]);
330     acc[5] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), t5, acc[5]);
331     acc[6] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), s6, acc[6]);
332     acc[7] = __USADA8 ((*((uint32_t*) &image1[off1 + 4 + i * row_size])), t7, acc[7]);
333   }
334   // End of the PX4 Development code section.
335   //================================================================================
336
337   #else
338   // Platform independant version of the sum of absolute difference computation.
339   uint8_t (*image1_buff) [8][row_size] = (uint8_t (*) [8][row_size]) image1;
340   uint8_t (*image2_buff) [8][row_size] = (uint8_t (*) [8][row_size]) image2;
341
342   // Loop row wise. Handle one row each time.
343   // Note: negative and out of bounds array indexes are used. This is fine since the array is positioned
344   // in the middle of an image. Thus we are never really looking out of bounds.
345   for (uint8_t yi = 0; yi < 8; yi++) for (int8_t xi = 0; xi < 8; xi++) {
346     // acc[0] is between ref[0][0] and ref[0][1] etc.
347     acc[0] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi + 0][xi + 1]) >>
         1));
348
349     // acc[1] is between ref[0][0] and ref[1][1] etc. This is slightly different than the orriginal
       implementation.
350     acc[1] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi + 1][xi + 1]) >>
         1));
351
352     // acc[2] is between ref[0][0] and ref[1][0] etc.
353     acc[2] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi + 1][xi + 0]) >>
         1));
354
355     // acc[3] is between ref[0][0] and ref[1][-1] etc. This is slightly different than the orriginal
       implementation.
356     acc[3] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi + 1][xi - 1]) >>
         1));
357
358     // acc[4] is between ref[0][0] and ref[0][-1] etc.
359     acc[4] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi + 0][xi - 1]) >>
         1));
360
```

```
361     // acc[5] is between ref[0][0] and ref[-1][-1] etc. This is slightly different than the orriginal
        implementation.
362     acc[5] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi - 1][xi - 1]) >>
          1));
363
364     // acc[6] is between ref[0][0] and ref[-1][0] etc.
365     acc[6] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi - 1][xi + 0]) >>
          1));
366
367     // acc[7] is between ref[0][0] and ref[-1][1] etc. This is slightly different than the orriginal
        implementation.
368     acc[7] += GET_ABS8((*image1_buff)[yi][xi], (((*image2_buff)[yi][xi] + (*image2_buff)[yi - 1][xi + 1]) >>
          1));
369   }
370
371
372   #endif
373   // Initial value for dir is zero.
374   *dir = 0;
375   /* Run through the accumulator array to find the lowest SAD direction. */
376   for (uint16_t i = 1; i < 8; i++) {
377     if (acc[i] < acc[0]) {
378       acc[0] = acc[i]; // Keep the lowest SAD in acc[0].
379       *dir = i; // Set the direction.
380     }
381   }
382   return acc[0];
383 }
384
385
386 /**
387  * @brief Computes pixel flow from image1 to image2
388  *
389  * Searches the corresponding position in the new image (image2) of max. 64 pixels from the old image (
        image1)
390  * and calculates the average offset of all.
391  *
392  * @param image1 (I) previous image buffer
393  * @param image2 (I) current image buffer (new)
394  * @param flow_x (O) output the x flow value
395  * @param flow_y (O) output the y flow value
396  * @param search_x (I) search window for x
397  * @param search_y (I) search window for y
398  *
399  * @return quality of flow calculation
400  */
401  static inline uint32_t compute_fullpixel(uint8_t *image1, uint8_t *image2,
402                     int32_t *flow_x, int32_t *flow_y,
403                     uint8_t search_x, uint8_t search_y, uint32_t width) {
404     /* Variables */
405     uint32_t sad_accumulator = 0xFFFFFFFF; // set initial distance to "infinity"
406     uint32_t temp_sad;
407     //  Convert the image2 pointer to a defined column and row sized pointer, makes for code which is
        cleaner in implementation.
408     uint8_t (*image2_buff) [][width] = (uint8_t (*) [][width]) image2;
409
410     /* Itterate from -search_x and -search_y to search_x and search_y to compute the minimum SAD in the
        search window. */
411     for (int8_t yi = -(int8_t)search_y; yi <= search_y; yi++) for (int8_t xi = -(int8_t)search_x; xi <=
        search_x; xi++) {
412       temp_sad = ABSDIFF(image1, &(*image2_buff)[yi][xi], width); // Calculate SAD for this xi, yi position.
413       if (temp_sad < sad_accumulator) { // Is the new position a better match?
414         sad_accumulator = temp_sad; // Store the SAD value.
415         *flow_x = xi; // Store the x flow position.
416         *flow_y = yi; // Store the y flow position.
417       }
418     }
419     return sad_accumulator;
420   }
421
422 /**
423  * @brief Computes pixel flow from src to dst
424  *
425  * Searches the corresponding position in the new image (image2) of max. 64 pixels from the old image (
        image1)
426  * and calculates the average offset of all.
427  *
428  * @param dst (I) current image buffer (new)
429  * @param src (I) previous image buffer
430  * @param x_variance (I/O) the certainty of pixel_flow_x.
431  * @param y_variance (I/O) the certainty of pixel_flow_y.
432  * @param y_variance (O) the certainty of pixel_flow_r.
433  * @param pixel_flow_x (I/O) the predicted flow x, and out the measured flow x.
434  * @param pixel_flow_y (I/O) the predicted flow y, and out the measured flow y.
435  * @param pixel_flow_r (O) the measured flow rotation "build with __FLOW_ROT".
436
437  *
438  * @return count of accepted block flow measurements.
439  */
440 uint32_t compute_flow(uint8_t *dst, uint8_t *src,
441                     float *x_variance, float *y_variance, float *r_variance,
442                     float *pixel_flow_x, float *pixel_flow_y, float *pixel_flow_r) {
443   /* constants */
444   const uint8_t BLOCK_SIZE    = 8;
445
446   /* variables */
447   uint8_t MAX_SEARCH          = data.max_search;
448   uint8_t MIN_SEARCH          = data.min_search;
449   uint32_t BIN_SIZE           = data.max_count * 2;
450   uint32_t IMAGE_COLUMN_COUNT= data.image_width;
451   uint32_t IMAGE_ROW_COUNT    = data.image_height;
452   uint32_t MIN_FLOW           = data.gradient_threshold;
453   uint32_t MAX_SAD_VAL        = data.sad_threshold;
454   uint32_t MAX_COUNT          = data.max_count; // Total number of flow results quit when we reach this.
455   int32_t xbin [BIN_SIZE]; // Here we will store all computed and accepted x flow results.
456   int32_t ybin [BIN_SIZE]; // Here we will store all computed and accepted y flow results.
```

```
457   int32_t xflow_predicted = (int32_t) *pixel_flow_x; // Store the predicted x flow as integer.
458   int32_t yflow_predicted = (int32_t) *pixel_flow_y; // Store the predicted y flow as integer.
459   float radius [BIN_SIZE]; // Here we will store all offsets of the computed and accepted x flow results.
460   uint32_t bin_counter = 0; // Counts how many flow results are stored.
461   uint8_t xsearch_window = (uint8_t) (data.search_scalar * (*x_variance)) + MIN_SEARCH > MAX_SEARCH ?
462                            MAX_SEARCH : (uint8_t) (data.search_scalar * (*x_variance)) + MIN_SEARCH; //
         Defines the maximum full pixel ofset for the x -> distance.
463   uint8_t ysearch_window = (uint8_t) (data.search_scalar * (*y_variance)) + MIN_SEARCH > MAX_SEARCH ?
464                            MAX_SEARCH : (uint8_t) (data.search_scalar * (*y_variance)) + MIN_SEARCH; //
         Defines the maximum full pixel ofset for the y -> distance.
465   uint8_t subpixel_dir; // Temp store for the subpixel direction.
466   uint32_t min_sad = 0; // Store the intermidiate SAD results.
467   uint32_t temp_sad = 0; // Another temporary SAD store.
468
469   #define ROUND_UP(x, y) ((((x) + (y) - 1) / (y)) * (y)) // Round up to a y multiple, works only for
         positive numbers!
470   #define ROUND_DOWN(x, y) (((x) / (y)) * (y)) // Round down to a y multiple, works only for positive
         numbers!
471   // Constrain the image to blocks we can still track considering the initial offset and search window.
472   uint32_t xmin = xflow_predicted > 0 ? // When the predicted flow is positive the block will move away from
          the min index.
473                   (xflow_predicted > xsearch_window ? 0 : // When the predicted flow is bigger than the
         search window we can start at zero.
474                   ROUND_UP(0 + xsearch_window, BLOCK_SIZE)) : // A positive flow is predicted so we just
         have to take care of the search window.
475                   ROUND_UP(0 - xflow_predicted + xsearch_window, BLOCK_SIZE); // A negative flow is
         predicted so also take flow prediction into account.
476   uint32_t xmax = xflow_predicted > 0 ? // When the predicted flow is positive the block will towards the
         max index.
477                   ROUND_DOWN(IMAGE_COLUMN_COUNT - xflow_predicted - xsearch_window, BLOCK_SIZE) : // A
         positive flow is predicted so we have to take image width, flow prediction and search window into
         account.
478                   (xflow_predicted < xsearch_window ? IMAGE_COLUMN_COUNT : // When the predicted flow is
         smaller than the search window we can start at image edge.
479                   ROUND_DOWN(IMAGE_COLUMN_COUNT - xsearch_window, BLOCK_SIZE)); // A negative flow is
         predicted so we only take image width and search window into account.
480   uint32_t ymin = yflow_predicted > 0 ? // When the predicted flow is positive the block will move away from
          the min index.
481                   (yflow_predicted > ysearch_window ? 0 : // When the predicted flow is bigger than the
         search window we can start at zero.
482                   ROUND_UP(0 + ysearch_window, BLOCK_SIZE)) : // A positive flow is predicted so we just
         have to take care of the search window.
483                   ROUND_UP(0 - yflow_predicted + ysearch_window, BLOCK_SIZE); // A negative flow is
         predicted so also take flow prediction into account.
484   uint32_t ymax = yflow_predicted > 0 ? // When the predicted flow is positive the block will towards the
         max index.
485                   ROUND_DOWN(IMAGE_ROW_COUNT - yflow_predicted - ysearch_window, BLOCK_SIZE) : // A positive
          flow is predicted so we have to take image width, flow prediction and search window into account.
486                   (yflow_predicted < ysearch_window ? IMAGE_ROW_COUNT : // When the predicted flow is
         smaller than the search window we can start at image edge.
487                   ROUND_DOWN(IMAGE_ROW_COUNT - ysearch_window, BLOCK_SIZE)); // A negative flow is predicted
          so we only take image width and search window into account.
488
489   //  Convert the dst pointer to a defined column and row sized pointer, makes for code which is cleaner in
         implementation.
490   uint8_t (*image1_buff) [IMAGE_ROW_COUNT][IMAGE_COLUMN_COUNT] = (uint8_t (*) [IMAGE_ROW_COUNT][
         IMAGE_COLUMN_COUNT]) dst;
491   //  Convert the src pointer to a defined column and row sized pointer, makes for code which is cleaner in
         implementation.
492   uint8_t (*image2_buff) [IMAGE_ROW_COUNT][IMAGE_COLUMN_COUNT] = (uint8_t (*) [IMAGE_ROW_COUNT][
         IMAGE_COLUMN_COUNT]) src;
493
494   // Sanity check, we cannot track flow larger than the image size. Use the underflow characteristic of
         uint32_t.
495   // This assumes that we would never use a resolution which approaches the maximum size of uint32_t.
496   if (xmin > IMAGE_COLUMN_COUNT || xmax > IMAGE_COLUMN_COUNT || ymin > IMAGE_ROW_COUNT || ymax >
         IMAGE_ROW_COUNT) {
497     // We cannot do anything with this flow... return failure. FIXME is this how we indicate failure?
498     *x_variance    = 0;
499     *y_variance    = 0;
500     *pixel_flow_x = 0;
501     *pixel_flow_y = 0;
502     return 0;
503   }
504
505   /* Itterate over every block within the constrained image. */
506   int8_t dxi; // Collumn itterator.
507   int8_t dyi; // Row itterator.
508   uint32_t xi = ((xmax - xmin) / BLOCK_SIZE) >> 1; // Calculate the halfway point.
509   uint32_t yi = ((ymax - ymin) / BLOCK_SIZE) >> 1; // Calculate the halfway point.
510
511   // Start moving to the right.
512   dxi = BLOCK_SIZE;
513   dyi = 0;
514
515   uint32_t m = xi > yi ? yi : xi; // Square up the search area.
516   m = (m << 1) * (m << 1); // Total amount of blocks contained.
517   xi = xmin + xi * BLOCK_SIZE; // Get the actual position.
518   yi = ymin + yi * BLOCK_SIZE; // Get the actual position.
519
520   uint8_t switches = 0; // Switch count, every two direction switches the subcount is incremented.
521   uint32_t subcount = 2; // Number of blocks in this line.
522
523   // For all contained blocks.
524   for (uint32_t t = 0; t < m && bin_counter < MAX_COUNT; t += subcount) {
525     // Ending condition check.
526     if (t + subcount >= m) {
527       subcount = m - t;
528     }
529     // For a single side.
530     for (uint32_t c = 1; c < subcount; c++, xi += dxi, yi += dyi) {
531       // Test block for flow features and accept it or reject it.
532       if (compute_vertical_gradient(&(*image1_buff)[yi + 2][xi + 2], IMAGE_COLUMN_COUNT) < MIN_FLOW)
       continue;
533       if (compute_horizontal_gradient(&(*image1_buff)[yi + 2][xi + 2], IMAGE_COLUMN_COUNT) < MIN_FLOW)
       continue;
```

```
534       DEBUG_DISPLAY_ACCTION(draw_source(yi, xi));
535
536       // Now compute the best SAD value and respective flow values for this block.
537       min_sad = compute_fullpixel(&(*image1_buff)[yi][xi], &(*image2_buff)[yi + yflow_predicted][xi +
      xflow_predicted],
538                                   &xbin[bin_counter], &ybin[bin_counter],
539                                   xsearch_window, ysearch_window, IMAGE_COLUMN_COUNT);
540
541       /* Minumum SAD value acceptance. */
542       if (min_sad > MAX_SAD_VAL) continue;
543
544       /* Process results with the predicted offset. */
545       xbin[bin_counter] += xflow_predicted;
546       ybin[bin_counter] += yflow_predicted;
547       DEBUG_DISPLAY_ACCTION(draw_dest(ybin[bin_counter] + yi, xbin[bin_counter] + xi));
548       DEBUG_DISPLAY_ACCTION(draw_path(yi, xi, ybin[bin_counter] + yi, xbin[bin_counter] + xi));
549
550       /* Now calculate subpixels. */
551       temp_sad = compute_subpixel(&(*image1_buff)[yi][xi],
552                                   &(*image2_buff)[yi + ybin[bin_counter]][xi + xbin[bin_counter]],
553                                   &subpixel_dir, IMAGE_COLUMN_COUNT);
554
555       /* Make the accumulated flows half pixels. */
556       xbin[bin_counter] *= 2;
557       ybin[bin_counter] *= 2;
558
559       /* Test subpixel SAD to see if we improve with subpixel directions. */
560       if (temp_sad < min_sad) {
561         switch (subpixel_dir) {
562           case 0:
563             xbin[bin_counter] +=  1;
564             break;
565           case 1:
566             xbin[bin_counter] +=  1;
567             ybin[bin_counter] +=  1;
568             break;
569           case 2:
570             ybin[bin_counter] +=  1;
571             break;
572           case 3:
573             xbin[bin_counter] += -1;
574             ybin[bin_counter] +=  1;
575             break;
576           case 4:
577             xbin[bin_counter] += -1;
578             break;
579           case 5:
580             xbin[bin_counter] += -1;
581             ybin[bin_counter] += -1;
582             break;
583           case 6:
584             ybin[bin_counter] += -1;
585             break;
586           case 7:
587             xbin[bin_counter] +=  1;
588             ybin[bin_counter] += -1;
589             break;
590           default:
591             break;
592         }
593       }
594       #ifdef __FLOW_ROT
595       {
596         // Get the distance from the center of the image, which we assume we rotate around.
597         int32_t xoff = xi * 2 - IMAGE_COLUMN_COUNT; // X dinstance for this flow vector
598         int32_t yoff = yi * 2 - IMAGE_ROW_COUNT; // Y distance for this flow vector.
599         radius [bin_counter] = sqrt(xoff * xoff + yoff * yoff); // Use pythagorean theorem to get the radius
      .
600         if (xoff < 0) radius[bin_counter] *= -1; // Invert the radius for minus x offsets to distinguish
      negative rotation.
601       }
602       #endif
603       bin_counter++;
604     }
605     // Switch direction.
606     int8_t temp = dxi;
607     dxi = -dyi;
608     dyi = temp;
609
610     // Modify the run length of the sub loop.
611     if (++switches > 1) {
612       switches = 0;
613       subcount++;
614     }
615   } // End of block loop.
616
617   /* Reset mean and variance of flows. */
618   *x_variance   = 0;
619   *y_variance   = 0;
620   *r_variance   = 0;
621   *pixel_flow_x = 0;
622   *pixel_flow_y = 0;
623   *pixel_flow_r = 0;
624
625   // Only do flow result calculation if we have flow results.
626   if (bin_counter) {
627     /* Calculate mean of xbin and Calculate mean of ybin. */
628     for(uint32_t xi = 0; xi < bin_counter; xi++) {
629       *pixel_flow_x += xbin[xi];
630       *pixel_flow_y += ybin[xi];
631     }
632     *pixel_flow_x /= bin_counter;
633     *pixel_flow_y /= bin_counter;
634
635     /* Calculate variance for xbin and Calculate variance for ybin. */
636     for(uint32_t xi = 0; xi < bin_counter; xi++) {
```

```
637       float diff = xbin[xi] - *pixel_flow_x; // Difference from the mean.
638       diff /= 2; // Bring back to full pixel value.
639       diff *= diff; // Square the difference.
640       *x_variance += diff; // Accumulate.
641       diff = ybin[xi] - *pixel_flow_y; // Difference from the mean.
642       diff /= 2; // Bring back to full pixel value.
643       diff *= diff; // Square the difference.
644       *y_variance += diff; // Accumulate.
645     }
646     *x_variance /= bin_counter; // Devide to get the variance.
647     *y_variance /= bin_counter; // Devide to get the variance.
648     #ifdef __FLOW_STD
649     *x_variance = sqrt(*x_variance); // Sqrt to get std deviation. This might need to change... ATTENTION
650     *y_variance = sqrt(*y_variance); // Sqrt to get std deviation. This might need to change... ATTENTION
651     #endif
652
653     #ifdef __FLOW_ROT
654     uint32_t stop = (uint32_t)(sqrt((float)bin_counter) * 4) - 4; // Implement a stop condition which limits
          us to only the outer flow results.
655     for(uint32_t xi = bin_counter - 1; xi > stop; xi--) {
656       /* Attempt to calculate rotation here. */
657       float rx = (float)xbin[xi] - *pixel_flow_x; // The movement without its lateral part is considered to
        be rotational.
658       float ry = (float)ybin[xi] - *pixel_flow_y; // The movement without its lateral part is considered to
        be rotational.
659       rx = sqrt(rx * rx + ry * ry); // Get our rotation movement length.
660       if (ry < 0) rx *= -1; // Invert length in case of other direction.
661       radius[xi] = atan(rx / radius[xi]); // Calculate the angle.
662       radius[xi] *= 57.2957795131; // Radials to degrees.
663       *pixel_flow_r += radius[xi]; // Accumulate
664     }
665     *pixel_flow_r /= bin_counter - stop; // Mean of rotation.
666     *pixel_flow_r *= 9./7.; // Magnitude adjustment FIXME
667
668     DEBUG_DISPLAY_ACCTION(
669       // Draw an integrated line representing the change of measured angle for debug purposes.
670       static float angle = 0; // Integrator.
671       float d = (IMAGE_ROW_COUNT / 2 - 10); // Distance of the line.
672       angle += *pixel_flow_r; // Integrate.
673       float angler = angle * 3.141592 / 180.; // To radials.
674       uint32_t sourcex = IMAGE_COLUMN_COUNT / 2 - 5;
675       uint32_t sourcey = IMAGE_ROW_COUNT / 2 - 5;
676       uint32_t destx = sourcex + d * cosf(angler);
677       uint32_t desty = sourcey + d * sinf(angler);
678       draw_path(sourcey, sourcex, desty, destx) // Draw line.
679     );
680
681     /* Calculate variance for rotation. */
682     for(uint32_t xi = bin_counter - 1; xi > stop; xi--) {
683       float diff = radius[xi] - *pixel_flow_r; // Difference from the mean.
684       diff *= diff; // Square the difference.
685       *r_variance += diff; // Accumulate.
686     }
687     *r_variance /= bin_counter - stop; // Devide to get the variance.
688     #ifdef __FLOW_STD
689     *r_variance = sqrt(*r_variance); // Sqrt to get std deviation. This might need to change... ATTENTION
690     #endif
691     #endif
692   }
693
694   /* Bring the flow values back to full pixels. */
695   *pixel_flow_x /= 2;
696   *pixel_flow_y /= 2;
697
698   /* Return the amount of results we had. */
699   return bin_counter;
700 }
```

Listing C.2: flow_interface.c

```
1  /*****************************************************************************
2   *
3   *   Copyright (C) 2018 ADI. All rights reserved.
4   *   Author: Mark Ramaker <markramaker@outlook.com>
5   *
6   * Redistribution and use in source and binary forms, with or without
7   * modification, is prohibited unless written consent from the copyright holders
8   * is obtained.
9   *
10  * When written permission is obtained the following conditions apply:
11  *
12  * 1. Redistributions of source code must retain the all copyright
13  *    notices, this list of conditions and the all disclaimers.
14  * 2. Redistributions in binary form must reproduce the all copyright
15  *    notices, this list of conditions and all disclaimers in
16  *    the documentation and/or other materials provided with the
17  *    distribution.
18  * 3. Neither the name ADI nor the names of its contributors may be
19  *    used to endorse or promote products derived from this software
20  *    without specific prior written permission.
21  *
22  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
23  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
24  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
25  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
26  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
27  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
28  * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
29  * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
30  * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
31  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
32  * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
33  * POSSIBILITY OF SUCH DAMAGE.
34  *
```

```
35  ***************************************************************************/
36
37 #include "flow_interface.h"
38
39 struct flow_store_t data = {
40   .max_search = 6,
41   .min_search = 3,
42   .search_scalar = 3,
43   .image_width = 260,
44   .image_height = 260,
45   .gradient_threshold = 65,
46   .sad_threshold = 700,
47   .max_count = 100
48 };
49
50 /**
51  * __STANDALONE_DATA forces the input to be decoupled from the output pointers.
52  */
53 #ifdef __STANDALONE_DATA
54 static float x_variance_bin = data.max_search;
55 static float y_variance_bin = data.max_search;
56 static float x_flow_bin = 0;
57 static float y_flow_bin = 0;
58 #endif
59
60 #ifdef __IPL_IMAGE_INTERFACE
61 /**
62  * @brief Computes pixel flow from src to dst
63  *
64  * Searches the corresponding position in the new image (image2) of max. 64 pixels from the old image (
       image1)
65  * and calculates the average offset of all.
66  *
67  * @param dst (I) current image buffer (new)
68  * @param src (I) previous image buffer
69  * @param x_variance (I/O) the certainty of pixel_flow_x.
70  * @param y_variance (I/O) the certainty of pixel_flow_y.
71  * @param y_variance (O) the certainty of pixel_flow_r.
72  * @param pixel_flow_x (I/O) the predicted flow x, and out the measured flow x.
73  * @param pixel_flow_y (I/O) the predicted flow y, and out the measured flow y.
74  * @param pixel_flow_r (O) the measured flow rotation "build with __FLOW_ROT".
75
76  *
77  * @return count of accepted block flow measurements.
78  */
79 uint32_t compute_flow_handler (IplImage *dst, IplImage *src,
80                                float *x_variance, float *y_variance, float *r_variance,
81                                float *pixel_flow_x, float *pixel_flow_y, float *pixel_flow_r) {
82   #ifdef __STANDALONE_DATA
83   data.image_width = src->width;
84   data.image_height = src->height;
85   uint32_t count = compute_flow((uint8_t *)dst->imageData, (uint8_t *)src->imageData, &x_variance_bin, &
       y_variance_bin, r_variance, &x_flow_bin, &y_flow_bin, pixel_flow_r);
86   *x_variance = x_variance_bin;
87   *y_variance = y_variance_bin;
88   *pixel_flow_x = x_flow_bin;
89   *pixel_flow_y = y_flow_bin;
90   return count;
91   #else
92   data.image_width = src->width;
93   data.image_height = src->height;
94   return compute_flow((uint8_t *)dst->imageData, (uint8_t *)src->imageData, x_variance, y_variance,
       r_variance, pixel_flow_x, pixel_flow_y, pixel_flow_r);
95   #endif
96 }
97 #else
98 /**
99  * @brief Computes pixel flow from src to dst
100 *
101 * Searches the corresponding position in the new image (image2) of max. 64 pixels from the old image (
       image1)
102 * and calculates the average offset of all.
103 *
104 * @param dst (I) current image buffer (new)
105 * @param src (I) previous image buffer
106 * @param x_variance (I/O) the certainty of pixel_flow_x.
107 * @param y_variance (I/O) the certainty of pixel_flow_y.
108 * @param y_variance (O) the certainty of pixel_flow_r.
109 * @param pixel_flow_x (I/O) the predicted flow x, and out the measured flow x.
110 * @param pixel_flow_y (I/O) the predicted flow y, and out the measured flow y.
111 * @param pixel_flow_r (O) the measured flow rotation "build with __FLOW_ROT".
112
113 *
114 * @return count of accepted block flow measurements.
115 */
116 uint32_t compute_flow_handler (uint8_t *dst, uint8_t *src,
117                                float *x_variance, float *y_variance, float *r_variance,
118                                float *pixel_flow_x, float *pixel_flow_y, float *pixel_flow_r) {
119   #ifdef __STANDALONE_DATA
120   uint32_t count = compute_flow(dst, src, &x_variance_bin, &y_variance_bin, r_variance, &x_flow_bin, &
       y_flow_bin, pixel_flow_r);
121   *x_variance = x_variance_bin;
122   *y_variance = y_variance_bin;
123   *pixel_flow_x = x_flow_bin;
124   *pixel_flow_y = y_flow_bin;
125   return count;
126   #else
127   return compute_flow(dst, src, x_variance, y_variance, r_variance, pixel_flow_x, pixel_flow_y, pixel_flow_r
       );
128   #endif
129 }
130 #endif
```

Appendix D

Smearing Simulation Sourcecode

Listing D.1: pixelsmearing.c

```
/*****************************************************************************
 *
 *    Copyright (C) 2018 Mark Ramaker. All rights reserved.
 *    Author: Mark Ramaker <markramaker@outlook.com>
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, is prohibited unless written consent from the copyright holders
 * is obtained.
 *
 * When written permission is obtained the following conditions apply:
 *
 * 1. Redistributions of source code must retain the all copyright
 *    notices, this list of conditions and the all disclaimers.
 * 2. Redistributions in binary form must reproduce the all copyright
 *    notices, this list of conditions and all disclaimers in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name Mark Ramaker nor the names of its contributors may be
 *    used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 *****************************************************************************/

#include <math.h>
#include "pixelsmearing.h"

/**
 * @brief Helper function for computing the distance between points.
 *
 *
 * @param src (I) point one
 * @param dst (I) point two
 */
float point_point_dst (smearing_pixel_t * src, smearing_pixel_t * dst) {
  return sqrtf(powf(dst->xoff - src->xoff, 2) + powf(dst->yoff - src->yoff, 2));
}

/**
 * @brief Helper function for computing the distance between a point and a line.
 * Math courtesy of Ton Ramaker.
 *
 *
 * @param dst (I) the point
 * @param A (I) part A of the line
 * @param B (I) part B of the line
 */
float point_line_dist (smearing_pixel_t * dst, float A, float B) {
  float C = dst->yoff + dst->xoff / A;
  float ix = (C - B) / (A + 1 / A);
  float iy = ix * -1 / A + C;
  return sqrtf(powf(dst->xoff - ix, 2) + powf(dst->yoff - iy, 2));
}

/**
 * @brief Helper function for computing the distance between a point and a line for when the line is
     horizontal.
 *
 *
```

```
 * @param dst (I) the point
 * @param A (I) part A of the line
 * @param B (I) part B of the line
 */
float point_line_dist_horz (smearing_pixel_t * dst, float A, float B) {
  return dst->yoff > 0 ? dst->yoff : -dst->yoff;
}

/**
 * @brief Helper function for computing the distance between a point and a line for when the line is vertical.
 *
 *
 * @param dst (I) the point
 * @param A (I) part A of the line
 * @param B (I) part B of the line
 */
float point_line_dist_vert (smearing_pixel_t * dst, float A, float B) {
  return dst->xoff > 0 ? dst->xoff : -dst->xoff;
}

/**
 * @brief Helper function creates a list of pixel points (offsets) and their incorporation cooeficient
 * Math courtesy of Ton Ramaker.
 *
 *
 * @param return_buffer (O) the buffer for the collected points
 * @param xvec (I) the x component of the smearing vector
 * @param yvec (I) the y component of the smearing vector
 * @param cutoff (I) the distance of pixels along the vector to be incorporated (between 0.5 and 1 should be exected)
 */
int collect_points (smearing_pixel_t * return_buffer, float xvec, float yvec, float cutoff) {
  int rc = 0;
  float (* point_line_dist_func) (smearing_pixel_t * dst, float A, float B);
  float vd = sqrtf(powf(xvec / 2, 2) + powf(yvec / 2, 2));
  float A;
  float B;
  float wa = 0;
  unsigned int sizx = xvec >= 0 ? (unsigned int)(xvec) + 5 : (unsigned int)(-xvec) + 5;
  unsigned int sizy = yvec >= 0 ? (unsigned int)(yvec) + 5 : (unsigned int)(-yvec) + 5;
  if (sizx % 2) sizx++;
  if (sizy % 2) sizy++;
  smearing_pixel_t src = {.xoff = 0, .yoff = 0, .dist = 0};
  smearing_pixel_t pend = {.xoff = xvec / 2, .yoff = yvec / 2, .dist = 0};
  smearing_pixel_t nend = {.xoff = -xvec / 2, .yoff = -yvec / 2, .dist = 0};

  if (xvec == 0 && yvec == 0) return 0;
  else if (xvec == 0) point_line_dist_func = &point_line_dist_vert;
  else if (yvec == 0) point_line_dist_func = &point_line_dist_horz;
  else {
    point_line_dist_func = &point_line_dist;
    A = yvec / xvec;
    B = yvec - A * xvec;
  }

  int limy = (sizy / 2);
  int limx = (sizx / 2);
  for (int yi = -(sizy / 2); yi <= limy; yi++)
  for (int xi = -(sizx / 2); xi <= limx; xi++) {
    smearing_pixel_t dst = {.xoff = (float)xi, .yoff = (float)yi, .dist = 0};
    float pd1 = point_point_dst (&pend, &dst);
    float pd2 = point_point_dst (&nend, &dst);
    float pd = pd1 < pd2 ? pd1 : pd2;
    float ld = (*point_line_dist_func) (&dst, A, B);
    float ad = point_point_dst (&src, &dst);

    if (ad > vd) {
      dst.dist = cutoff - pd;
      if (dst.dist > 0) {
        return_buffer [rc++] = dst;
        wa += dst.dist;
      }
    } else {
      dst.dist = cutoff - (pd < ld ? pd : ld);
      if (dst.dist > 0) {
        return_buffer [rc++] = dst;
        wa += dst.dist;
      }
    }
  }

  for (int i = 0; i < rc; i++) return_buffer [i].dist /= wa;

  return rc;
}

/**
 * @brief Helper function for computing a safe index.
 *
 *
 * @param index (I) the index to be evaluated
 * @param max (I) the maximum allowed
 */
int si (int index, int max) {
  if (index < 0) return 0;
  return index > max ? max : index;
}

/**
 * @brief Smears a frame along the provided vector
 *
 *
 * @param in (I) the input image
 * @param out (O) the smeared output image
 * @param xvec (I) the x component of the smearing vector
```

```
175  * @param yvec (I) the y component of the smearing vector
176  * @param lim (I) the distance of pixels along the vector to be incorporated (between 0.5 and 1 should be
         exected)
177  */
178 void smear_frame (IplImage * in, IplImage * out, float xvec, float yvec, float lim) {
179   int c;
180   smearing_pixel_t buff [500];
181 
182   c = collect_points (buff, xvec, yvec, lim);
183 
184   if (c){
185     for (int yi = 0; yi < in->height; yi++) for (int xi = 0; xi < in->width; xi++) for (int pi = 0; pi < in
          ->nChannels; pi++) {
186       float t = 0;
187       for (int ai = 0; ai < c; ai++) t += (uint8_t)in->imageData [si (yi + buff [ai].yoff, in->height) * in
          ->widthStep + si (xi + buff [ai].xoff, in->width) * in->nChannels + pi] * buff [ai].dist;
188       out->imageData [yi * out->widthStep + xi * out->nChannels + pi] = (char) t;
189     }
190   } else {
191     cvCopy(in, out, NULL);
192   }
193 }
```

Appendix E

Ultra Wide Band Simulation Data

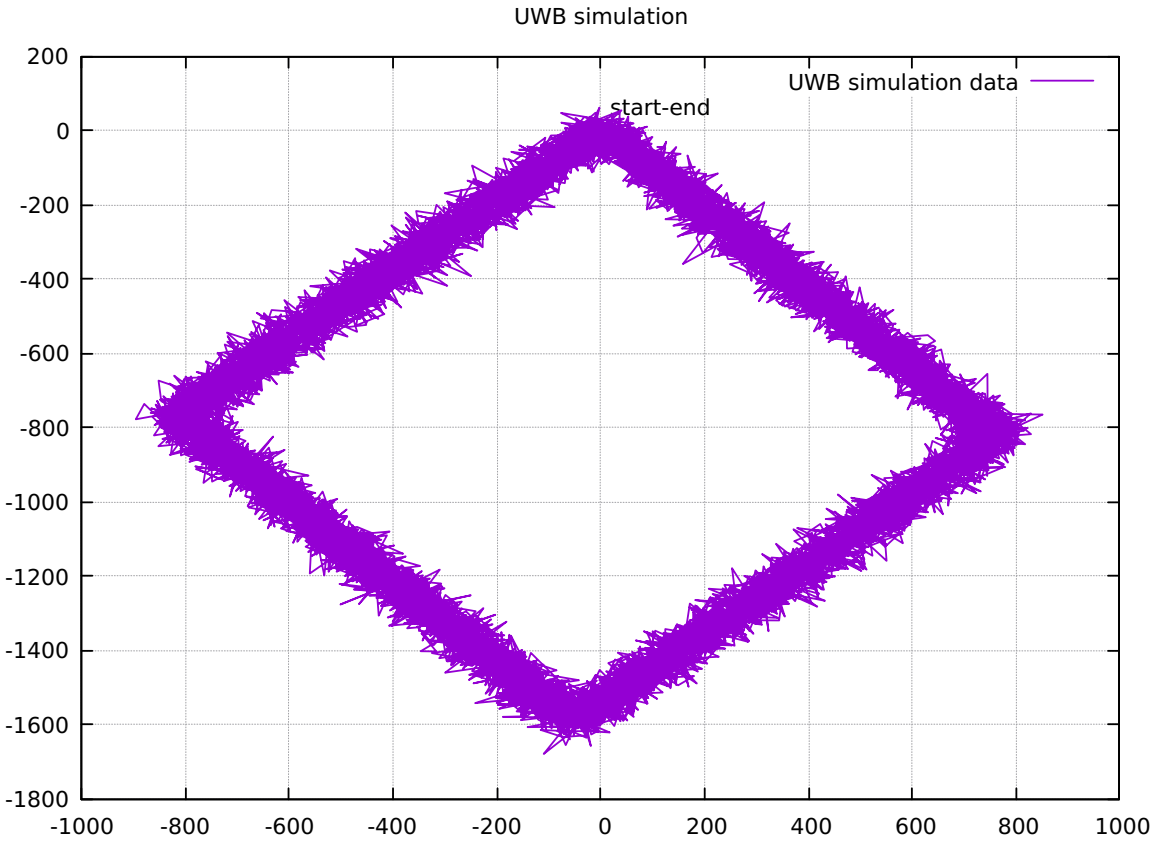

Figure E.1: Ultra Wide Band Simulation data. Scale in centimeters. Generated by taking the reference and adding a noise having a standard deviation of 30cm to each measurement interval.

Appendix F

The Roll Rotation Induced Measurement Ripple

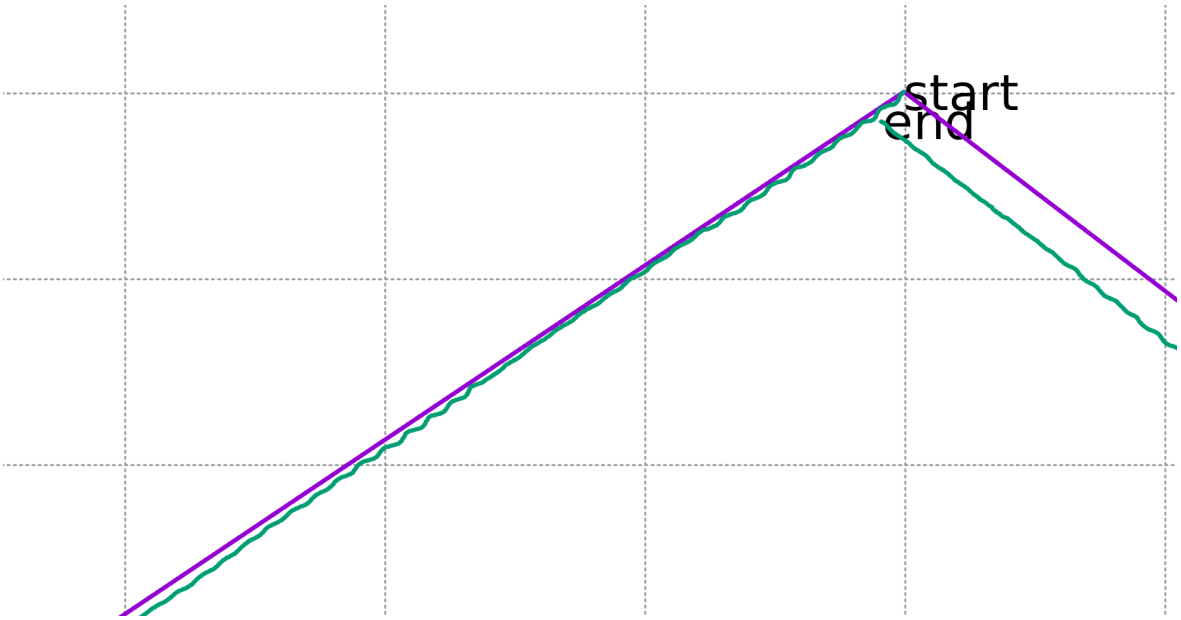

Figure F.1: The ripple effect in the integrated measurement of unit test 1 (Fig: 7.3), as produced by the rail system's sinusiodal rotations in the roll axis.

Appendix G

An Example of Optical Flow

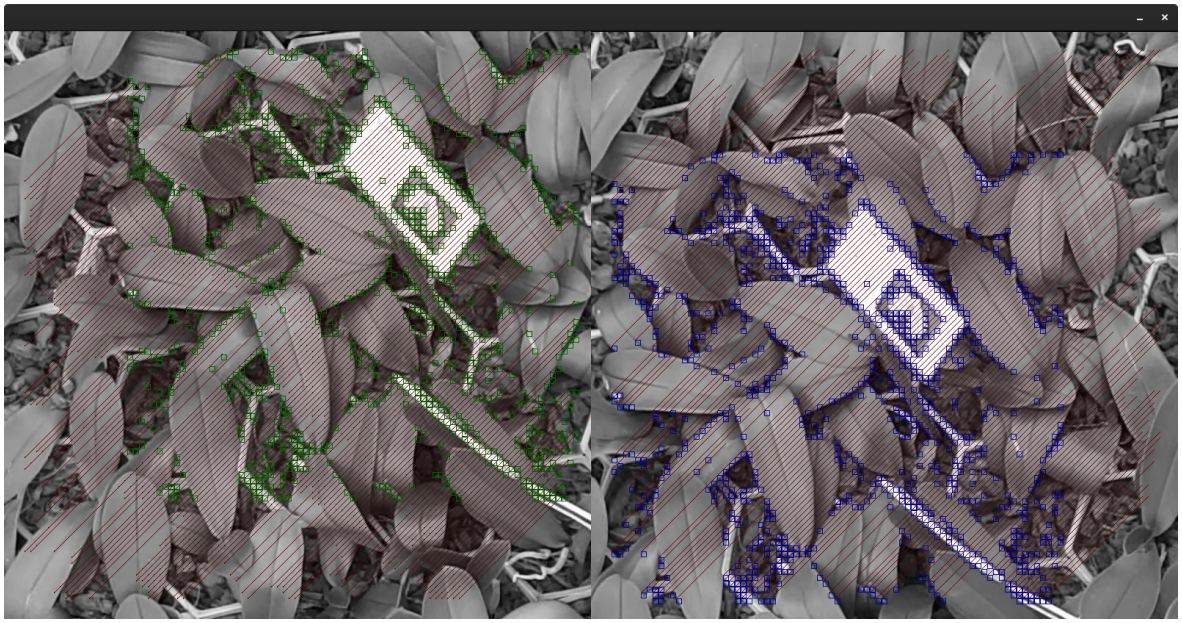

Figure G.1: An example of optical flow where the right image is translated 140 pixels down and to the left compared to the left image. The *green* blocks on the left can be seen as the reference features, and the *purple* blocks on the right the corresponding feature locations in the translated image.

Appendix H

Trajectory Simulation

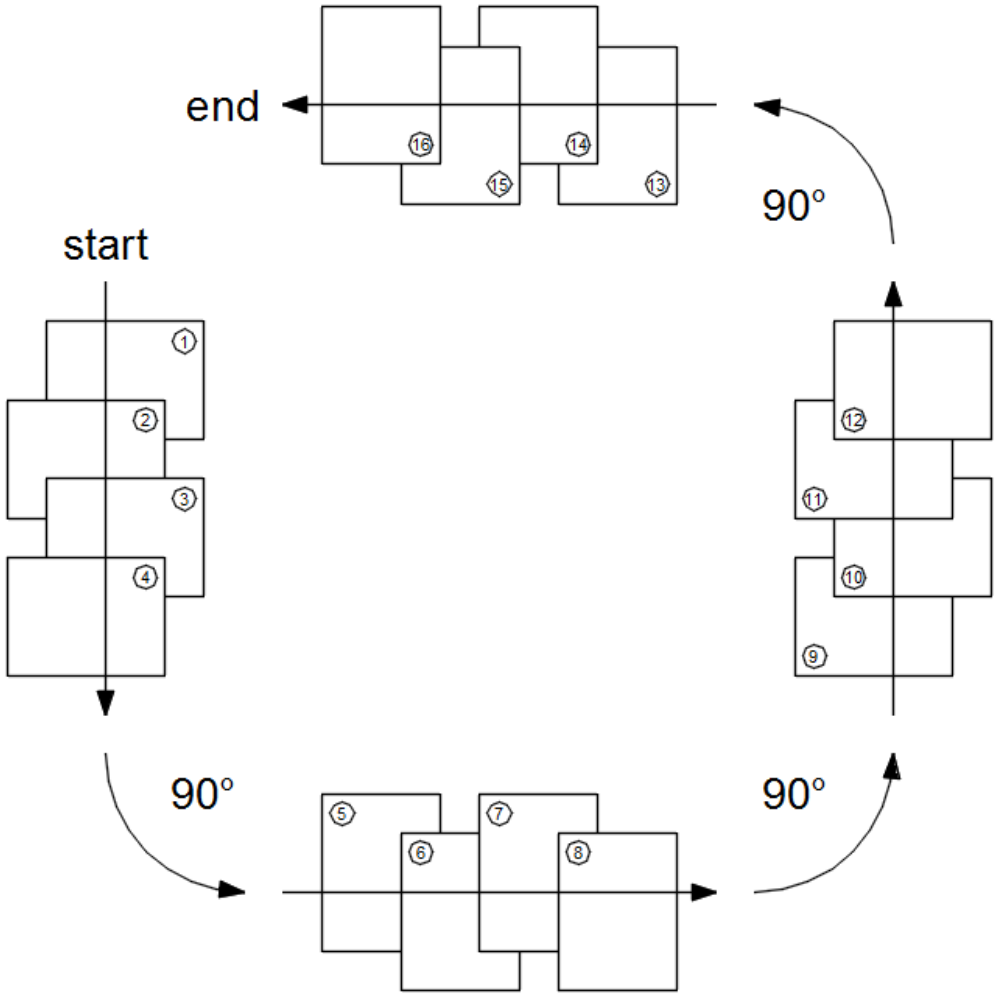

Figure H.1: The frames of the straight line trajectory are rotated 90° every quarter of the total trajectory thus forming a square trajectory.