VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi – 590014

A PROJECT REPORT ON

SIMULTANEOUS LOCALIZATION AND MAPPING USING A KINECT IN AN INDOOR ENVIRONMENT

Submitted in partial fulfillment of the requirement for the award of degree of

BACHELOR OF ENGINEERING

in

INSTRUMENTATION TECHNOLOGY

Submitted by

SYED MISBAH AHMED	1BI13IT041
VISHWAS NAVADA B	1BI13IT048
ANKIT SINHA	1BI13IT003
SAURODEEP KAR	1BI13IT035

Under the guidance of

Dr. M B Meenavathi,

Professor and HOD,

Department of Electronics and Instrumentation Engineering

DEPARTMENT OF INSTRUMENTATION TECHNOLOGY

BANGALORE INSTITUTE OF TECHNOLOGY

BENGALURU-560004

2016-2017

BANGALORE INSTITUTE OF TECHNOLOGY

K. R. Road, V. V. Puram, Bengaluru-560004

DEPARTMENT OF INSTRUMENTATION TECHNOLOGY

CERTIFICATE

Certified that the project work entitled **"SIMULTANEOUS LOCALIZATION AND MAPPING USING A KINECT IN INDOOR ENVIRONMENT"** is a bonafide work carried out by **Syed Misbah Ahmed, USN: 1BI13IT041, Vishwas Navada B, USN: 1BI13IT048, Saurodeep Kar, USN: 1BI13IT035** and **Ankit Sinha, USN: 1BI13IT003,** in partial fulfillment for the award of **Bachelor of Engineering** in **Instrumentation Technology** of the **Visvesvaraya Technological University**, Belgaum, during the year 2016-17. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Signature of the Guide	Signature of the HOD	Signature of the Principal
Dr. M B Meenavathi	**Dr. M B Meenavathi**	**Dr. A G Natraj**

External Viva

Name of the Examiners: **Signature with Date:**

1.

2.

DECLARATION

We, **SYED MISBAH AHMED** (1BI13IT041), **VISHWAS NAVADA** (1BI13IT048), **SAURODEEP KAR** (1BI13IT035), **ANKIT SINHA** (1BI13IT003) students of 8th semester BE in Instrumentaion Technology, **Bangalore Institute of Technology**, Bengaluru, hereby declare that the project work entitled "**SIMULTANEOUS LOCALAIZATION AND MAPPING USING A KINECT IN AN INDOOR ENVIRONMENT"** submitted to the **Visvesvaraya Technological University** during the academic year 2016-17, is a record of an original work done by us under the guidance of **Dr. M B Meenavathi**, Professor and Head of Department of Electronics & Instrumentation Engineering, Bangalore Institute of Technology, Bengaluru. This project work is submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Instrumentation Technology. The results embodied in this thesis have not been submitted to any other university or institute for the award of any degree.

Date:
Place: Bengaluru

Syed Misbah Ahmed

Vishwas Navada

Saurodeep Kar

Ankit Sinha

ACKNOWLEDGEMENT

It would be our privilege to express our heart-felt gratitude and respect to all those who guided me in the completion of this project.

On the path of learning, the presence of a guide is indispensable. We would like to thank **Dr. M. B. Meenavathi**, Professor and HOD, Department of Electronics and Instrumentation Engineering, Bangalore Institute of Technology, for her constant encouragement, suggestions, support and guidance.

We would like to express our heart-felt gratitude to **Dr. M. B. Meenavathi**, Professor & Head of the Department of Electronics and Instrumentation Engineering, in Bangalore Institute of Technology for her constant support in building the report.

We express our deep sense of gratitude to our principal, **Dr.A.G.Nataraj** Bangalore Institute of Technology for providing an excellent academic environment in the college.

We would like to thank all the teaching and non-teaching staff, who have been an inspirational support.

Last but not the least, we would like to thank our parents and friends for their constant help and moral support throughout the completion of the seminar.

ABSTRACT

Localization and mapping are two of the most central tasks when it comes to autonomous robots. It has often been performed using expensive, accurate sensors but the fast development of consumer electronics has made similar sensors available at a more affordable price.

Building rich 3D maps of environments is an important task for mobile robotics, with applications in navigation, manipulation, semantic mapping, and telepresence. Most 3D mapping systems contain three main components: first, the spatial alignment of consecutive data frames; second, the detection of loop closures; and third, the globally consistent alignment of the complete data sequence. While 3D point clouds are well suited for frame-to-frame alignment and for dense 3D reconstruction, they ignore valuable information contained in images. Color cameras, on the other hand, capture rich visual information and are becoming more and more the sensor of choice for loop closure detection.

In this project a quadcopter and a Microsoft Kinect™ camera are used to perform Simultaneous Localization and Mapping, SLAM. This project aims to build rich 3D maps using RGB-D (Red Green Blue – Depth) Mapping. RGB-D mapping is a framework for using RGB-D cameras to generate dense 3D models of indoor environments. RGB-D Mapping exploits the integration of shape and appearance information provided by these systems. Alignment between frames is computed by jointly optimizing over both appearance and shape matches. Visual appearance is incorporated by extracting sparse feature points from the RGB images and matching them via a RANSAC (Random Sample Consensus) procedure. The resulting feature matches are then combined to determine the best alignment between the frames and build the final 3D map.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Simultaneous Localization and Mapping

SLAM or Simultaneous Localization and Mapping is an umbrella term for algorithms that build maps of an unknown environment and at the same time perform localization in that area. The method is often used by robots and other vehicles and is a first step towards making them autonomous. Once a map of the area is created and the vehicle knows its position with respect to the map, the next step, the navigation and route planning step, can take place. As a third step, smart decisions, dependent on the information available for the vehicle, can be made and the vehicle may then be regarded as truly autonomous.

1.2 Background

The research in the area of SLAM has been on the agenda for several universities and other institutions worldwide the last decades. The continuous improvement of the computational capacity in computers has made it possible for SLAM applications to be performed outside of pure test environments, and in the most latter years even outdoors [1][2] has often been performed using a laser scanner, which is an accurate but expensive sensor. In this project, the laser scanner is replaced by a Kinect™ camera, which is a much cheaper, but still qualified sensor unit.

The applications of SLAM are numerous. If mapping of a hazardous or poisonous environment is required a robot with SLAM-based navigation is an excellent option. Some concrete examples are indoor mapping of houses on fire or mapping of damaged nuclear reactors. Another interesting case is monitoring of electrical power lines. That kind of work might be expensive and time-consuming work for humans to do but probably cheaper when using an UAV1. The UAV will simply follow the power lines \ autonomously and at the same time check the power line for damage.

The research area of SLAM has matured during the past years and the 2D-SLAM problem using a laser scanner has been implemented in so many cases that it may now be considered

solved [3] The field of visual SLAM, i.e. using visual images rather than using measurements from sensors or IMU: s (Inertial Measurement Unit) has increased over the last years. Contributing to this development is a large variety of open source components, aiming at separate tasks in the SLAM machinery [3].

During the years' different types of sensors have been used to perform visual SLAM, e.g. 2D laser scanners, mono cameras and stereo cameras [3]. The development in the robotic community has lately shifted towards the use of sensors that are cheaper, lighter, smaller and thus often less accurate. This change allows for a use of UAVs to a bigger extent [4] and also increases the potential of getting SLAM based consumer products, e.g. autonomous vacuum cleaners, on the market.

The work load of implementing an embedded SLAM-system has decreased as the number of open source packages has increased. The Point Cloud Library, PCL, is a standalone, large scale, open project for processing of 3D point clouds, which might become handy when working with applications concerning 3D SLAM. Another neat package is the OctoMap package which is a library used to describe mapped environments in a way that is both probabilistic and scalable.

1.3 Problem Formulation

In this project SLAM will be performed using a Kinect™, instead of a laser scanner, as primary sensor. With a Kinect™ the sensor data cannot be expected to be as accurate as from a laser. Laser scanners are well known to have high accuracy in range. The Kinect™ has a maximum range of about eight meters while a laser scanner often has a maximum range of 50-80 meters.

Apart from being cheaper the Kinect™ has other advantages compared to a laser scanner. Its measurements are three dimensional (range and bearing in two directions) while the data from a laser scanner is two dimensional (range and bearing). This makes the data more information dense. In addition to that the Kinect™ also has a RGB-camera which captures the surroundings in video pace. This gives the possibility to get 3D maps with colored textures. The ability to recognize similarities between images is a key factor for a visual SLAM algorithm to be successful. Without that ability, the algorithm is lost. In this project, the performance of indoor visual SLAM using a quadcopter will be evaluated with the following questions as guidelines.

1.4 Limitations

In this project, the Kinect™ will always be mounted on a Quadcopter, and not be used in combination with any other robot platform. The different algorithms for e.g. visual feature detection, pose estimation or map building, will all come from open source libraries, but it is in the scope of this project to make adjustments to them to fit our purposes. Furthermore, the whole quadcopter platform and much of its functionality will be based on ROS2 and any possible limitations in ROS will consequently be reflected in the evaluations. It is not in the scope of this project to add any additional sensors to the quadcopter, neither will the existing hardware be modified in any way. Measurements will be performed indoors, in order not to expose the Kinect™ to too much IR radiation, in which case the measurements from the depth sensor is affected. When data is collected there will be no moving objects present in the environment, since such disturbances might affect the algorithm in an unpredictable way. There will not be any demands on real time performance on algorithms in this project. Real time performance here refers to the ability of mapping a room at the same time as the quadcopter moves around. It also means that all computations to build the map are completed in such a pace that the quadcopter can move around freely in the environment and still have the most recent estimate of the surroundings at most one second after that the corresponding image was captured.

1.5 Project Outline

The previous work already done on the SLAM algorithms are discussed in the literature review. project report continues with chapter 3 that gives an introduction to the quadcopter hardware and software. Chapter 4 explains the localization algorithms in more detail and presents the open source SLAM algorithm, RGBDSLAM, used in this project. Chapter 5 describes the modeling and filtering used to extend the original RGBDSLAM algorithm. Methods for implementation and evaluation of the different SLAM algorithms are found in Chapter 6. Finally, the results are presented in Chapter 7 and a discussion is found in Chapter 8.

CHAPTER 2

LITERATURE SURVEY

The field of robotics is still missing a general acknowledged solution to solving the SLAM problem. In this chapter, we review the previously published literature which lays the foundation and the basis for further work. SLAM represents the simultaneous localization and mapping by a quadcopter of the surrounding space. In the past decades, a tremendous effort has been put in finding a perfect solution to this problem due to the necessity of building autonomous vehicles used in fields like search and rescue, entertainment or transportation.

The autonomous vehicles should be able to determine their position relative to a map that they are able to generate. When the position of the quadcopter is known relative to the space that surrounds him, the problem is less complicated since it has limited computational complexity [9] and the only errors that are taken in consideration are the ones coming from the sensor that evaluates the environment. This problem expands when the initial position of the quadcopter is not known and besides the uncertainty of the map quality we have to take in consideration approximations and possible errors in determining the actual position of the quadcopter [12]. Another unknown that we are dealing with when talking about space recognition and mapping is the sensor employed for such a task. Laser-based, sonar based or vision based sensors are the ones that stand out and were most used by researchers [5]. All of them have advantages and disadvantages. The most recent focus is on vision based sensors since they offer the highest resolution and a good range but that comes with the disadvantage of complications in processing the output of the sensor. It appears that there is no definite solution to our problem even if the research has been ongoing for the past three decades in this field. Due to the numerous uncertainties and the computational complications that arise from these methods it becomes obvious that when dealing with large spaces and moving objects there isn’t an established solution to mapping and consequently the trajectory of the quadcopter is still variant upon the environment.

For mobile quadcopters mapping can be classified according to the underlying estimation techniques used in building that chart. The most popular approaches are described below; SLAM represents the simultaneous localization and mapping by a quadcopter of the surrounding space. The basic idea behind SLAM is to obtain a map of the environment and estimate where the quadcopter is located within that map concurrently. Furthermore, the map

and the quadcopter's pose both have to be estimated at the same time to accomplish the task well. SLAM problem involves finding appropriate representation for both observations of the environment and the motion of the quadcopter. In order to do so, the vehicle or quadcopter must be equipped with a sensorial system capable of taking measurements between landmarks and the vehicle itself while it is moving in the map. It is also known as a Concurrent Mapping and Localization challenge. The aim of this chapter is to introduce previous work on this issue, to list some advantages and disadvantages of current available techniques and also to identify gaps in research like possible new directions for improvement in future.

In 2001, a paper [15] was published by W.M Gamini Dissanayake which used MMW (millimeter waves) to build relative maps. EKF [16], EIF [17] and PF [18] methods might be sufficient for some applications to learn local maps only [19]. In addition, other papers focused in improving the way the information is received from the outside space and consequently the way that the map is estimated.

The improved usage of ultrasonic sensor for detecting obstacles and walls for -based quadcopters for immediate localization was presented by Sungyoung Jung [20]. This algorithm showed high degree of accuracy and also could solve the "kidnapping" problem in a fast operating time. There are methods used to correct the poses of the quadcopter based on the inverse of the covariance matrix [18]. The main advantage of SEIFs (Sparse Extended Information Filter) is that they make use of the approximate sparsity of the information matrix.

Another method was proposed in 2005, a smoothing method called Square root smoothing and mapping (SAM) [21]. Compared with EKF-based methods several advantages could be mentioned such as covering non-linearity and the speed in computing.

Differential Evolution Solution to the SLAM Problem [9] is the name of the other paper which presented a new solution for the SLAM problem and was published in 2007 by Luis Moreno and his colleagues. Chanki Kim provided a robust new algorithm based on the scaled unscented transformation called Unscented FastSALM(UFastSLAM) in 2008[10].

Due to the necessity of having useful details and also perfect trajectory the research extended on the existing Kalman filter (EKF) [14]. Besides EKF there is another method to solve SLAM which is called Information Filtering (IF) or Extended (EIF). EKF by approximating the quadcopter motion using linear functions accommodates the nonlinearities from the real world. One of the advantages of IF over EKF is that the data is filtered by simply summing the

information matrices and vector providing more accurate estimates [15]. Secondly, IF are more stable than KF. Finally, EKF is very slow when estimating high dimensional maps because every single vehicle measurement generally effects all parameters of the Gaussian, therefore the updates requires prohibitive times when dealing with environments with many landmarks [15]. For high dimensional state spaces, the need to compute all these inversions is generally believed to make the IF computationally poorer than Kalman filter. In fact, this is one of the reasons why the EKF has been vastly more popular than EIF. These limitations do not necessarily apply to problems in which the information matrix possesses structure [11].

There are by now a few examples of successful EKF SLAM implementations. Another popular approach is the FastSLAM method [10] which uses particle filters. Inconsistencies due to poor data association, linearization errors and particle depletion are the main disadvantages of both methods. Fortunately, digital cameras are today ubiquitous and since their measurements are so accurate at very low cost in comparison with other sensors they have become very popular and commonly used. This reason has spawned a field in which the SLAM problem is solved only with cameras [13]. Without any other sensors measuring the platform dynamics, the image frame rate and the visual information contents in the environment are the only limiting factors for the ego motion estimation and hence the map quality.

Different cameras have been used in experiments. An omnidirectional camera can be used to estimate the distance of the closest color transition in the environment, mimicking the performance of laser rangefinders. These measurements are introduced into particle filter to determine the position of the quadcopter within a previously constructed map. Eventually only one camera was used (called MonoSLAM). The proposed method, which takes place in real-time, extracts a reduced but sufficient number of salient image features through which are identified by their associated image, patches.

The Expectation Maximization based method (EM) is a statistical algorithm that was developed in the context of maximum likelihood (ML) estimation and it offers an optimal solution, being an ideal option for map-building but not for localization [9]. This way is useful and can build a map when the quadcopter’s pose is known, for instance, by means of expectation. The main advantage of EM over KF is that it can tackle the correspondence problem surprisingly well.

We reviewed traditional methods of solving SLAM such as Kalman Filtering Particle Filtering, EM and IF methods and we also analyzed some new methods which use cameras to build maps

for mobile quadcopters in unknown environments. Obviously, more research is needed since there is no perfect solution to the SLAM problem. When the environment is changing the system should adapt the mapping and the trajectory accordingly. The maps need to contain more details especially when dealing with noisy environments like water.

CHAPTER 3

HARDWARE SETUP

The setup consists of two parts, a quadcopter which is an aerial vehicle that uses four rotors for lift, steering, and stabilization. The Kinect camera is mounted onto the quadcopter and connected to the computer using a thin USB cable.

3.1 UAV Quadcopter

A UAV quadcopter is an unmanned aerial vehicle with four rotating rotors used for lift and movement. It uses an electronic control system and electronic sensors to help stabilize itself. Quadcopter parts have been decreasing in price over the past couple of years due to technological advances. As a result, more hobbyists, universities, and industries are taking advantage of this opportunity to design and develop applications for the quadcopter.

3.1.1 Flight Control

A quadcopter consists of four motors evenly distributed along the quadcopter frame as can be seen in figure 3.1. The circles represent the spinning rotors of the quadcopter and the arrows represent the rotation direction. Motors one and three rotate in a clockwise direction using pusher rotors. Motor two and four rotate in a counter-clockwise direction using puller rotors. Each motor produces a thrust and torque about the center of the quadcopter. Due to the opposite spinning directions of the motors, the net torque about the center of the quadcopter is ideally zero, producing zero angular acceleration. This eliminates the need for yaw stabilization. A vertical force is created by increasing the speed of all the motors by the same amount of throttle. As the vertical forces overcome the gravitational forces of the earth, the quadcopter begins to rise in altitude. Figure 3.2 shows the vertical movement of the quadcopter. As above, the circles represent the spinning rotors, the larger arrows represent the direction the rotors are spinning, and the black arrows represent the forces caused by the spinning rotors. Pitch is provided by increasing (or decreasing) the speed of the front or rear motors. This causes the quadcopter to turn along the x axis. The overall vertical thrust is the same as hovering due to the left and right motors; hence only pitch angle acceleration is changed. Figure 3.3 shows an example of pitch movement of a quadcopter. As the front motor slows down, the forces created by the corresponding rotor are less than the forces created by the back rotor. These forces are

represented by the blue arrows. These forces cause the quadcopter to tip forward and this movement is represented by the red arrow [21].

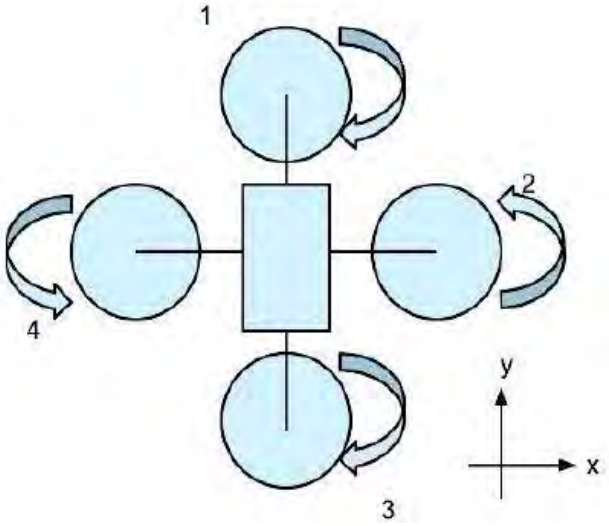


Fig 3.1: Quadcopter: Motor rotation directions.

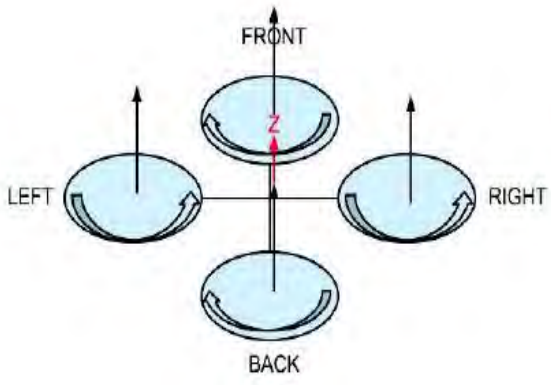


Fig 3.2: Quadcopter: Vertical thrust movement.

Roll is provided by increasing (or decreasing) the speed of the left rotor speed and right motors. This causes the quadcopter to turn along the y axis. The overall vertical thrust is the same as hovering due to the front and back motors; hence only roll angle acceleration is changed [22].

Figure 3.4 shows an example of roll movement of a quadcopter. As the right motor slows down, the forces created by the corresponding rotor are less than the forces created by the left rotor. These forces are represented by the blue arrows. This causes the quadcopter to tip to the right

and this movement is represented by the red arrow. Yaw is provided by increasing (or decreasing) the speed of the front and rear motors or by increasing (or decreasing) the speed of the left and right motors. This causes the quadcopter to turn along its vertical axis in the direction of the stronger spinning rotors. Figure 3.5 shows an example of yaw movement of a quadcopter. As the front and back motor slows down, the forces created by the corresponding rotors are less than the forces created by the left and right rotors. The quadcopter will begin to rotate in the same direction as the faster spinning rotors due to the difference in torque forces. This movement is represented by the red arrow.

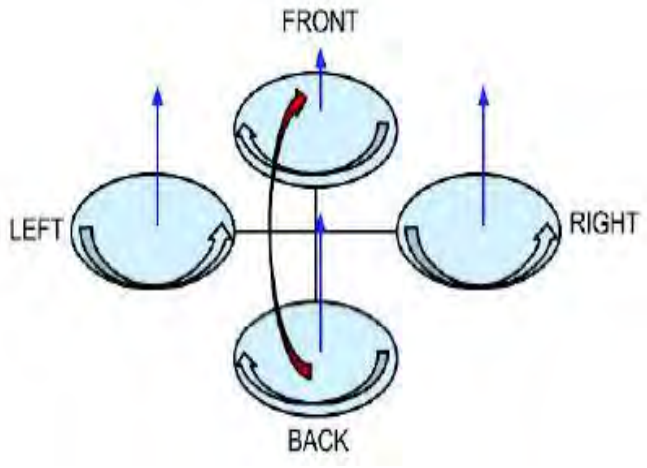


Fig 3.3: Quadcopter: Pitch movement.

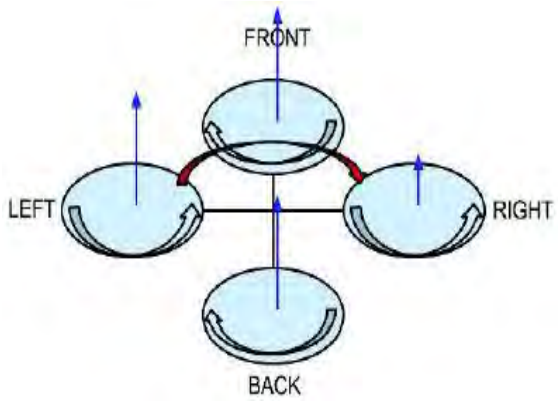


Fig. 3.4: Quadcopter Roll Movement

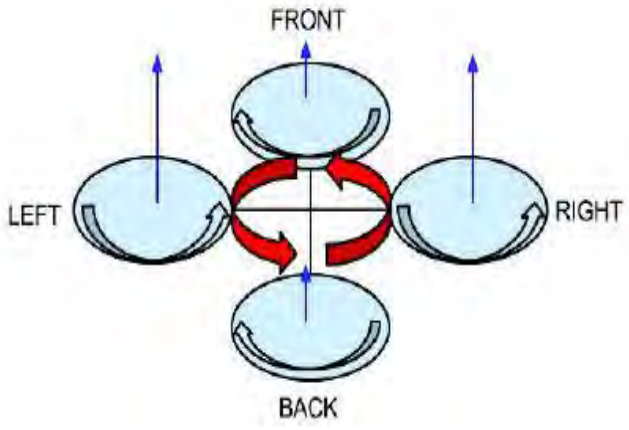


Fig 3.5: Quadcopter: Yaw movement

3.2 Microsoft Kinect

The Kinect™ camera is a low-cost sensor built specially for the Xbox console. It can deliver a RGB image and a depth image in parallel in video rate. Microsoft has not released any official hardware specifications for the Kinect™ sensor and therefore unofficial specifications from reversed engineering is the most accessible way to get insight in the machinery. The OpenKinect community states that the Kinect™ has two cameras, one RGBcamera and one range camera. The latter is based on structured light. According to PrimeSense, the manufactures of the micro controller in the Kinect™, the projected IR-points are processed by a PS1080A micro controller to produce a depth image. From design reference for PrimeSense PS1080A it can be inferred that:

Field of View (Horizontal, Vertical, Diagonal) = 58° H, 45° V, 70° D
Spatial x/y resolution (@ 2 m distance from sensor) = 3 mm
Depth z resolution (@ 2 m distance from sensor) = 1 cm
Operation range = 0.8 m - 3.5 m

Fig. 3.6: The Microsoft Kinect Sensor

3.2 Setup

The final hardware setup for the project looks like this:

Fig. 3.7: Shows the Kinect mounted on top of the quadcopter and connected to the workstation

CHAPTER 4

MAPPING ALGORITHMS

In this chapter, the functionality of a generic SLAM algorithm is explained. A presentation of one of the most central open source packages in this project, RGBD- SLAM, is also made.

4.1 Under the Hood of a SLAM Algorithm

A SLAM algorithm essentially consists of the following steps:

- Data acquisition; In this step measurements from the sensors, e.g. Laser scanner or video camera, are gathered.
- Feature extraction; a number of characteristic, and thereby easily recognizable, landmarks[1] are selected from the data set.
- Feature association; landmarks from previous measurements are associated with landmarks from the most recent measurement.
- Pose estimation; the relative change between the landmarks and the position of the vehicle is used to estimate the new pose of the vehicle.
- Map adjustment; the map is updated according to the new pose and the corresponding measurements.

The five tasks are continuously repeated and a trajectory of position estimates and a map is built up. Figure 4.1 illustrates the process. In Figure 4.1 the quadcopter has received input data. Landmarks are extracted from the data. In the case of visual SLAM, a landmark can be anything that is easily recognizable by a visual sensor, e.g.

- A corner
- An edge
- A dot in a protruding color

The quadcopter in Figure 4.2 has extracted three landmarks and these are stored in a database containing all landmarks that have been observed so far. In Figure 4.2 the quadcopter has moved forward and received new data. The quadcopter extracts landmarks from the data and searches through its database to see if there are any

matches with old landmarks. Extracted landmarks that are not found in the database are added and landmarks giving a match in the database are used to estimate the change of the quadcopter's pose. This is done by measuring the change in distance and angle to the old landmarks. When the new pose is estimated the quadcopter uses this estimate and the measurements to adjust the positions of the landmarks. SLAM can be regarded as a hen and egg problem. A proper map is needed to get a proper pose estimate and a proper pose estimate is needed to get a proper map.

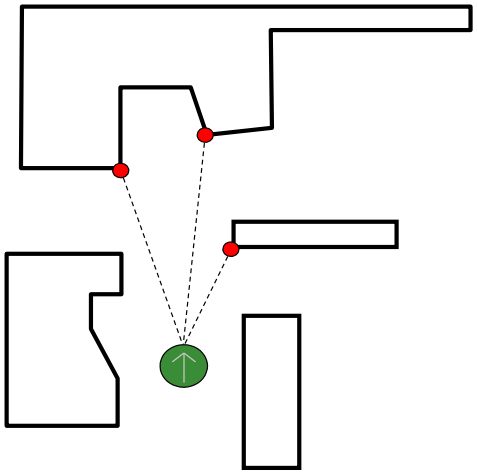


Fig 4.1: Quadcopter position and landmarks at time t_0.

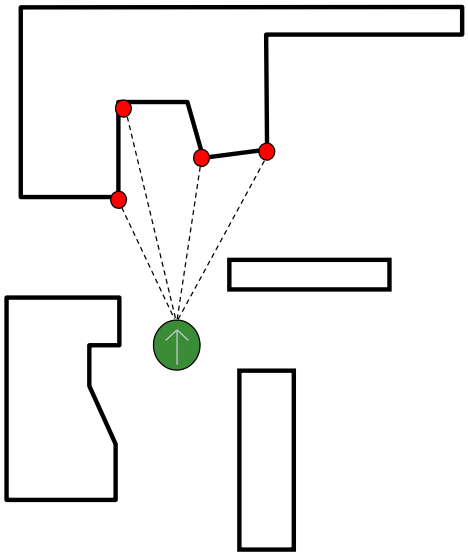


Fig 4.2: Quadcopter position and landmarks at time t1 > t0.

4.2 RGBDSLAM

This project is based on an algorithm called RGBDSLAM, which is six degrees of freedom (x, *y*, *z*, roll, pitch and yaw) algorithm that performs visual SLAM (VSLAM). The algorithm uses both the color (RGB) and the depth (D) in- formation and is generic due to the fact that it contains no motion model. The algorithm is developed by researchers from the University of Freiburg in Germany and is described in Engelhard et al. [29] and Enders et al. [28]. A schematic overview of the algorithm can be studied in Figure 4.3.

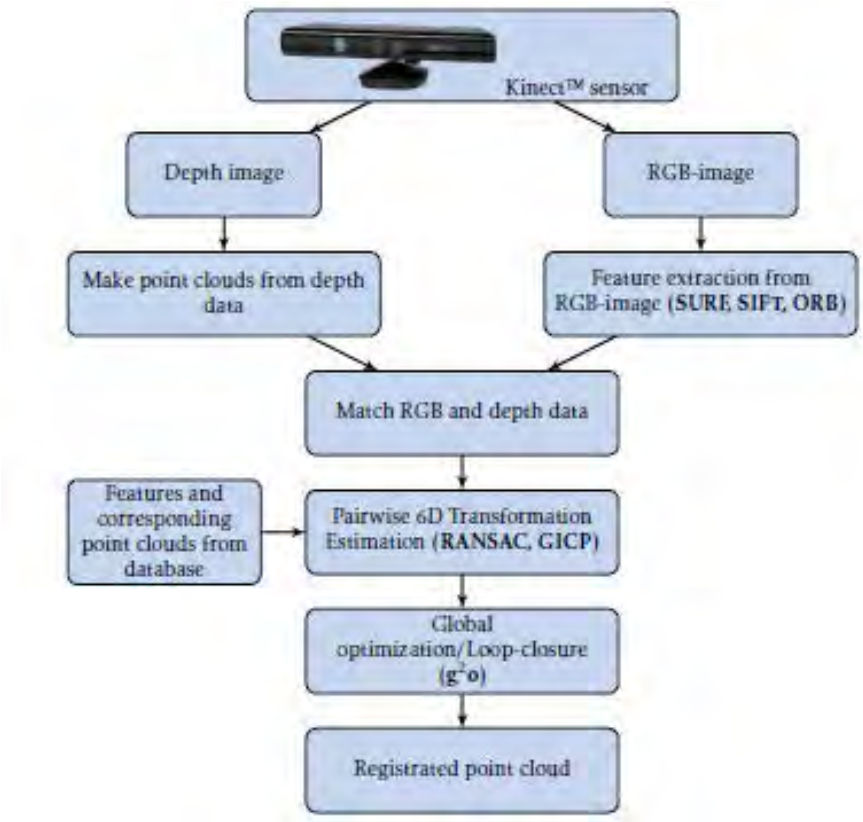


Fig 4.3: Schematic overview of the RGBDSLAM algorithm provided as a ROS package.

As a first step the depth and RGB-images are collected with synchronized time- stamps. Then features are extracted from the RGB-image by a feature extraction algorithm. RGBDSLAM has multiple feature extraction algorithms implemented. The implementations have different pros and cons in different environments and they differ in computation time [31]. The algorithms implemented are SURF, SIFT and ORB and the first two are described in section 4.4.

In the next step of the algorithm extracted features are projected to the depth image. This step introduces some uncertainty into the chain of operations. Mainly due to the synchronization mismatch between depth and RGB-images, but also because of interpolation between points with large differences in depth. The fact that a minor miss projection of a feature lying on an object border on to the depth image can result in a big depth error makes features picked at object borders unreliable.

To find a 6*D* transform for the camera position in this noise the RANSAC algorithm is used. Features are matched with earlier extracted features from a set of 20 images in the standard configuration. The set consists of a subset including some of the most recent captured images and another subset including images randomly selected from the set of all formerly captured images. Three matched feature pairs are randomly selected and are used to calculate a 6*D* transform. All feature pairs are then evaluated by their Euclidian distance to each other. Pairs whose Euclidian distance is below a certain threshold are counted as inliers. From these inliers, a refined 6*D* transform is calculated using GICP.

4.3 Random Sample Consensus

Random Sample Consensus, or RANSAC for short, is an iterative algorithm used to adapt the parameters of a mathematical model to experimental data. RANSAC is a suitable method when a data set contains a high percentage of outliers, i.e. measurements that suffer from measurement errors so large that the validity of the measurements is low. The method was first presented in the beginning of the eighties in [30] and was suggested to be a suitable method for automated image analysis.

Assume a mathematical model that has n free parameters which can be estimated given a set of measurements, P. The number of measurements in P has to be greater than n, $\#P > n$. Let S and T be two different varying subsets of P.Given the assumptions the RANSAC algorithm works as follows:

- Randomly select a subset of the measurements in P and call it S. Use S to make a first estimate of the n free parameters of the model.

- Use the current estimate of the model to select a new subset of points, *T from* the measurements that are within some error tolerance from the model.
- If *T* contains more measurements than some given limit then re-estimate the free parameters of the model according to this new subset. Calculate a measure of how well *T* and the model coincide, store that value and select a new subset S.
- If *It* does not contain more measurements than the given limit, randomly select a new subset S from *P* and start all over again.
- If none of the selected subsets hold more measurements than the limit, exit in failure, or if the maximum number of iterations has been reached, exit.

The method is characterized by three parameters:

1. The error tolerance used to determine when data is not part of the model.
2. The maximum number of iterations in the algorithm.
3. The minimum value on the number of measurements in a subset to be used for parameter estimation.

The big advantage with RANSAC is the robust way it handles outliers in the data set. The drawbacks with RANSAC is that it does not guarantee any solution, nor that a given solution is optimal. Furthermore, the three parameters mentioned above are to a large extent problem specific, which means that experimental adjustment to the specific case treated is required.

Fig 4.4: Example of Measurement

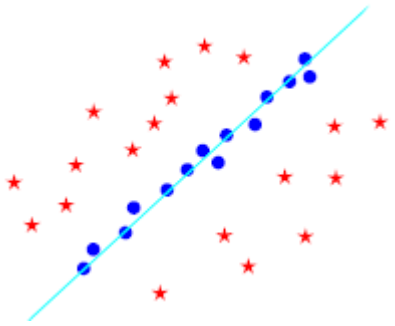


Fig 4.5: Measurements as seen by RANSAC with the model being a line. Stars are considered as outliers and dots are considered as inliers.

4.4 Feature Extraction

The two commonly used feature extractors Scale Invariant Feature Transform, SIFT, and Speeded Up Robust Features, SURF is briefly described below.

4.4.1 Scale Invariant Feature Transform(SIFT)

SIFT is a method for extracting and detecting landmarks in an image. The method was invented in 1999 [15] and has since been widely used. The algorithm is, as the name suggests, invariant to scale transformations in images and also rotational transformations. SIFT is much appreciated for its reliability and repeatability which are important properties of a feature extractor/detector [13]. The features are selected at maxima and minima of the color in an image that has first been smoothed and then filtered through a difference of Gaussian filter (DoG). The continuous two-dimensional difference of Gaussian convolution kernel is

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} - \frac{1}{2\pi\sigma^2 K^2} e^{-(x^2+y^2)/(2\sigma^2)}, \tag{4.1}$$

where K is a constant bigger than one which scales the standard deviation. The DoG filter creates two versions of the original image; one that is somewhat more blurred than the other. The blurring is carried out using approximate Gaussian filtering kernels. Then the most blurred image is subtracted from the other one and maxima and minima in the resulting image are detected. Detected points with low contrast or points lying along edges are discarded. Each detected key- point is characterized by the magnitude and rotation of the pixel wise image

gradient. Here $A_{i\,j}$ represents the value of each pixel, M_{ij} is the magnitude of the gradient and R_{ij} is the orientation of the gradient.

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2}$$
$$R_{ij} = arctan2(A_{ij} - A_{i+1,j}, A_{i,j+1} - A_{ij}) \quad (4.2)$$

The above calculated values are post processed to increase the robustness to illumination changes.

4.4.2 Speeded Up Robust Features(SURF)

The feature detection algorithm SURF is partly inspired by SIFT. The method aims at being a fast, but yet reliable, algorithm. The method is both scale- and rotation invariant and handles blur and changes in illumination in a satisfying manner [31]. The filters used in SURF are discretized and cropped which results in rectangular filters with integral values. In this way, the computational load is reduced and the extraction or detection of features is speeded up.

4.5 Registration

In order to build a map of several point clouds registration is necessary. Registration is the process of finding the best possible transformation to stitch two point clouds together. The point clouds are called destination, D, and source, S. This section presents some of the most used and known registration methods. In Figure 4.4 an example of registration between two point clouds is showed.

Fig 4.6: Example of registration between two point clouds.

The overlap of the point clouds is seen most easily in the bottom of the figure 4.6. There is approximately 80% overlap.

4.5.1 Iterative Closest Point - Point to Point

Iterative Closest Point (ICP) is one of the most intuitive registration methods. The algorithm has been widely used and was developed in the early 90's. A frequently cited analysis of the algorithm [34]. The algorithm can be summarized in two key steps.

1. Find matching feature points between the two point clouds.
2. Compute a transform, T that is a rotation and translation such that it minimizes. the Euclidian distance between the matching points.

An iteration of these two steps typically gives convergence to the desired trans- form. A key parameter to tune in the algorithm is the maximum matching distance threshold, e_{max}. With a complete overlap of the two point clouds a high value of $e_{max\ can}$ be used and the algorithm will still converge. The parameter $e_{max\ has}$ to be low if matching points only come from a small overlap, otherwise mismatch probably makes the algorithm diverge. The parameter is a tradeoff between convergence and accuracy [35] presents the algorithm as in Algorithm 1.

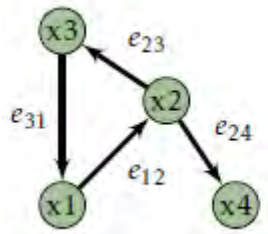
Algorithm 1 Standard Iterate Closest Point algorithm

Input: Two point clouds: $\mathbf{D} = \{a_i\}$ $\mathbf{S} = \{s_i\}$
1: Initial guess: Transform $\mathbf{T}_0$
Output: Correct Transform $\mathbf{T}$
2: $\mathbf{T} = \mathbf{T}_0$
3: **while** not converged **do**
4: **for** $i = 1$ to N **do**
5: $d_i = \texttt{FindClosestPointInD}(\mathbf{T} \cdot s_i)$
6: **if** $\|d_i - T \cdot s_i\| < e_{max}$ **then**
7: $w_i = 1$
8: **else**
9: $w_i = 0$
10: **end if**
11: **end for**
12: $\mathbf{T} = \underset{\mathbf{T}}{\operatorname{argmin}} \left\{ \sum_i w_i \|\mathbf{T} \cdot s_i - d_i\|^2 \right\}$
13: **end while**

A far more advanced algorithm to handle registration is Generalized ICP, GICP, first presented in [35]. This can be thought of as a plane to plane matching algorithm. By assigning probabilistic properties to extracted features a most likelihood estimate of the transform can be achieved. Furthermore, the probabilistic framework can make use of all general research in probabilistic techniques for increased robustness, such as outlier rejection.

4.6 General Graph Optimization

The General Graph Optimization package (abbreviated g^2o) is an open source library which implements a general and efficient graph optimizer. A graph optimizer achieves a minimization of a nonlinear error function represented as a graph, see Figure 4.5. The nodes represent vectors of parameters and the edges how well two parameter vectors match to external constraint, relating the two parameter vectors. In the case where g^2O is used by RGBDSLAM each node in the graph represents a state variable of the quadcopter and each edge represents a pairwise observation between the nodes that are connected by that edge. The meaning of a pairwise observation is that node B observes its pose with respect to node A, and vice versa. The algorithm is fast due to, among other things, its use of the sparse connectivity property of the graph and advanced solvers of sparse linear systems. The interested reader may refer [36] for a more thorough presentation.



$$F(x) = e_{12}{}^T I_{12} e_{12} + e_{23}{}^T I_{23} e_{23} + e_{24}{}^T I_{24} e_{24} + e_{31}{}^T I_{31} e_{31}$$

Fig 4.7 To the left a graph with nodes and edges and to the right them corresponding nonlinear error function where Ii j is the information matrix

CHAPTER 5

MODELS AND FILTERING

So far, this project has given an overview of the system and an introduction to the open source algorithm RGBDSLAM. In an attempt to improve the performance of the system, according to the guidelines, three steps are taken:

1. Odometry and gyro data is measured.
2. An appropriate motion model for the quadcopter is implemented.
3. Already existing measurements from visual odometry are fused with odometry, gyro measurements and the motion model using an Extended Kalman Filter.

These three modifications result in an algorithm that in the following will be called the modified algorithm as opposed to the original algorithm.

This chapter introduces the mathematical models used to formulate the SLAM problem in a rigorous manner. The models consist of both motion models that describe the motion of the quadcopter and measurement models that demonstrate how the sensor data is inserted into the framework. The chapter continues with an explanation of key properties of the filter that fuses the motion model and the measurement models.

5.1 Notation

The state vector of the quadcopter at time instance k is called x_k and holds a three-dimensional pose, i.e. a three-dimensional position and a three-dimensional orientation.

$$x_k = \begin{pmatrix} X_k & Y_k & Z_k & \phi_k & \theta_k & \psi_k \end{pmatrix}^T. \tag{5.1}$$

Here X_k, Y_k and Z_k represent the position of the quadcopter in a world fixed coordinate system and φ_k, θ_k and ψ_k, the roll, pitch and yaw angles, are defined as rotations around the world fixed X, Y and Z axes, see Figure 5.1. The quadcopter speed is denoted v_k and is the speed in the

case of 2D movement in X - and Y -direction. The angular velocity around the world fixed Z axis, i.e. the derivative with respect to time of the yaw angle is denoted v_k.

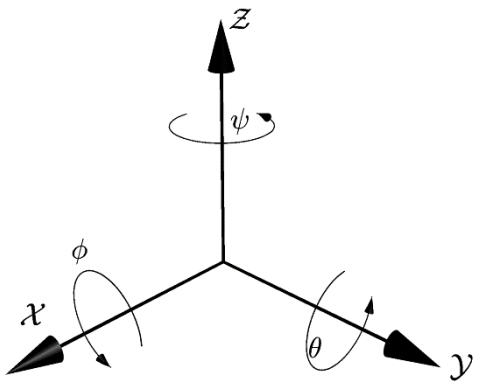


Fig. 5.1: Definitions of roll, pitch and yaw.

By using v_k and ω_k an extended state vector on the form can be formed. This state vector is used everywhere except in the original quadcopter node package.

$$x_k = \begin{pmatrix} X_k & Y_k & Z_k & \phi_k & \theta_k & \psi_k & v_k & \omega_k \end{pmatrix}^T \tag{5.2}$$

The sample time cannot be considered uniform and thus the sample time from event $k-1$ to k is denoted T_k.

The system can be modeled as a state space model, which on a generic form is

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) + w_k, \\ y_k &= h(x_k, u_k) + e_k, \end{aligned} \tag{5.3}$$

where w_k and e_k represent Gaussian noise with zero mean and covariance's Q and R respectively

$$\begin{aligned} w_k &\sim \mathcal{N}(0, Q), \\ e_k &\sim \mathcal{N}(0, R), \end{aligned} \tag{5.4}$$

and u_k *is* the direct input to the system. Inputs to the system are supplied using the state

vector and there will not be any direct inputs to the system. Therefore, $u_{k\ will}$ onwards be disregarded.

The measurement equation will in the forthcoming cases be linear and can consequently be simplified as while the motion model remains nonlinear.

$$\begin{aligned} y_k &= h(x_k) + e_k \\ &= Cx_k + e_k, \end{aligned} \tag{5.5}$$

5.2 Motion Models

RGBDSLAM has no motion model and hence there are no constraints to the estimated trajectory of the quadcopter. The absence of a motion model also makes the algorithm heavily dependent on a continuous stream of images that contains sufficiently many and re-detectable landmarks. To decrease the dependency of land- marks well-trimmed motion models are introduced.

5.2.1 Original Motion Model

The native navigation packages for the Quadcopter contain an EKF with a motion model. The filter uses a 6D model and estimates the quadcopter pose with a 3D position and 3D orientation estimate. The filter fuses data from the odometry, the gyro and the visual odometry. Given the state vector (5.6) the general motion model is

$$x_{k+1} = \begin{pmatrix} X_{k+1} \\ Y_{k+1} \\ Z_{k+1} \\ \phi_{k+1} \\ \theta_{k+1} \\ \psi_{k+1} \end{pmatrix} = f(x_k) + w_k = \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ \phi_k \\ \theta_k \\ \psi_k \end{pmatrix} + w_k \tag{5.6}$$

The model models a constant position and fixed angles around the axes. It is a very simple and perhaps not an appropriate model for a moving Quadcopter.

5.2.2 2D Constant Velocity Model

By using the extended state vector in (5.7), a more dynamic but still relatively simple model can be formed as

$$x_{k+1} = \begin{pmatrix} X_{k+1} \\ Y_{k+1} \\ Z_{k+1} \\ \phi_{k+1} \\ \theta_{k+1} \\ \psi_{k+1} \\ v_{k+1} \\ \omega_{k+1} \end{pmatrix} = f(x_k) + w_k = \begin{pmatrix} X_k + T_k \cos(\psi_k) v_k \\ Y_k + T_k \sin(\psi_k) v_k \\ 0 \\ 0 \\ 0 \\ \psi_k + T_k \omega_k \\ v_k \\ \omega_k \end{pmatrix} + w_k. \tag{5.7}$$

This model describes a 2D motion where the quadcopter is always in an upright pose-ton and travels with constant velocity. This model might fit the experimental set up better than the model in (5.7).

5.2.3 2D Coordinated Turn Model

The motion modelling can be extended further by using the current yaw, ψk *and* the angular velocity ω_k in a more sophisticated way. This is conveniently done by a 2D discretized coordinated turn model using polar velocity [37], which is given in (5.8)

$$x_{k+1} = \begin{pmatrix} X_{k+1} \\ Y_{k+1} \\ Z_{k+1} \\ \phi_{k+1} \\ \theta_{k+1} \\ \psi_{k+1} \\ v_{k+1} \\ \omega_{k+1} \end{pmatrix} = f(x_k) + w_k \tag{5.8}$$

$$= \begin{pmatrix} X_k + \frac{2v}{\omega_k}\sin(\frac{\omega_k T_k}{2})\cos(\psi + \frac{\omega_k T_k}{2}) \\ Y_k + \frac{2v}{\omega_k}\sin(\frac{\omega_k T_k}{2})\sin(\psi + \frac{\omega_k T_k}{2}) \\ 0 \\ 0 \\ 0 \\ \psi_k + T_k\omega_k \\ v_k \\ \omega_k \end{pmatrix} + w_k. \quad (5.9)$$

5.3 Measurement Models

Measurement models are used to define how the sensors' measurements and the states are related. The system consists of the three different sensor units

- O dometer,
- Gyro
- Visual odometer.

The three units contribute to the measurement equation in different ways. Note that the extended state vector (5.7) is used throughout this section.

5.3.1 Odometry

Tachometers are used to measure the speed of the quadcopter. The Quadcopter platform uses specific hardware details and integration of speed to provide measurements of X_k, Y_k, ψ_k and the speed vk. The measurement model of the odometry sensor is given as

$$y_{odom,k} = C_{odom}x_k + e_{odom,k}(d) \quad (5.10)$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ \phi_k \\ \theta_k \\ \psi_k \\ v_k \\ \omega_k \end{pmatrix} + e_{odom,k}(d)$$

$$= \begin{pmatrix} X_k \\ Y_k \\ \psi_k \\ v_k \end{pmatrix} + e_{odom,k}(d). \tag{5.11}$$

Here eod om,k (d) is the sensor noise. The measured states from the odometry sensor are incrementally updated from an initial position and therefore the noise is also increasing. A simplified model for the noise is a Gaussian noise model according to

$$e_{odom,k}(d) \sim \mathcal{N}(0, R_{odom}(d)) \tag{5.12}$$

where the covariance, Rod om (d), depends on the traveled distance, d , as :

$$R_{odom}(d) = dS, \tag{5.13}$$

where S is a design parameter in form of a constant covariance matrix for the specific sensor.

5.3.2 Gyro

The quadcopter has a single axis gyro which measures the yaw angle and the angular velocity around the z-axis i.e.

$$
\begin{aligned}
y_{gyro,k} &= C_{gyro}x_k + e_{gyro,k} \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ \phi_k \\ \theta_k \\ \psi_k \\ v_k \\ \omega_k \end{pmatrix} + e_{gyro,k} \\
&= \begin{pmatrix} \psi_k \\ \omega_k \end{pmatrix} + e_{gyro,k}.
\end{aligned}
\tag{5.14}
$$

The noise $e_{gyro;\,k}$ *is* modeled as Gaussian noise with zero mean:

$$
e_{gyro,k} \sim \mathcal{N}(0, R_{gyro}).
\tag{5.15}
$$

5.3.3 Visual Odometry

The visual odometry provides information from the RGB camera and the depth camera of the Kinect™. Collection and post processing of data is described in Section 5.2. The concerned algorithms that calculate useful information from the post processed data are presented in Chapter 5. The result of the algorithms is an estimate of the full 3D pose of the quadcopter, i.e.

$$
\begin{aligned}
y_{vo,k} &= C_{vo}x_k + e_{vo,k}(vo_{trust}) \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ \phi_k \\ \theta_k \\ \psi_k \\ v_k \\ \omega_k \end{pmatrix} + e_{vo,k}(vo_{trust}) \\
&= \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ \phi_k \\ \theta_k \\ \psi_k \end{pmatrix} + e_{vo,k}(vo_{trust})
\end{aligned}
\tag{5.16}
$$

Here e_{voke} is modeled as Gaussian noise according to

$$e_{vo,k} \sim \mathcal{N}(0, R_{vo}(vo_{trust})). \tag{5.17}$$

The measurement noise, $R_{vo}(vo_{truset})$, depends on the parameter $vo_{trust\,which}$ varies in a nonlinear way as described in Algorithm 3.

5.3.4 Total Model

Concatenation of the three measurement models above gives a total measurement equation on the form

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{5.18}$$

The measurement noise e_k becomes

$$e_k(d, vo_{trust}) = \begin{pmatrix} e_{odom,k}(d) \\ e_{gyro,k} \\ e_{vo,k}(vo_{trust}) \end{pmatrix} \tag{5.19}$$

With a corresponding covariance matrix R as

$$R(d, vo_{trust}) = \begin{pmatrix} R_{odom}(d) & 0 & 0 \\ 0 & R_{gyro} & 0 \\ 0 & 0 & R_{vo}(vo_{trust}) \end{pmatrix} \tag{5.20}$$

Due to the assumption that the different measurement noises are independent from each other.

5.4 Extended Kalman Filtering

In this section, the filtering process is described. All the sensors provide relative measurements, meaning that the measurements depend on earlier states. The input to the filter is given as the difference of the new measurement and an earlier estimated state $\hat{x}_k$. odometry and gyro measurements are processed at full rate and the visual measurements are processed at a sub rate. The filtering process can be studied in Figure 5.2. The filter is an Extended Kalman Filter using first order Taylor expansion. It is implemented by the open source project Orocos Bayesian Filtering Library [38]. The filter used in this project is described by Algorithm 2.

Algorithm 2 Extended Kalman Filter

Require: $\hat{x}_{0|0}$ and $\hat{P}_{0|0}$

Time update

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k})$$
$$P_{k+1|k} = f'_x(\hat{x}_{k|k}) P_{k|k} (f'_x(\hat{x}_{k|k}))^T + Q_k$$

Measurement update

$$S_k = h'_x(\hat{x}_{k|k-1}) P_{k|k-1} (h'_x(\hat{x}_{k|k-1}))^T$$
$$K_k = P_{k|k-1} (h'_x(\hat{x}_{k|k-1}))^T S_k^{-1}$$
$$\varepsilon_k = y_k - h(\hat{x}_{k|k-1})$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \varepsilon_k$$
$$P_{k|k} = P_{k|k-1} - P_{k|k-1} (h'_x(\hat{x}_{k|k-1}))^T S_k^{-1} h'_x(\hat{x}_{k|k-1}) P_{k|k-1}$$

where Q_k is the process noise and $h(x)$ defines the measurement equation according to the general notation in Equation.

5.4.1 Filter and Model Evaluation

An overview of the filter and its information flow can be studied in Figure 5.2. The loop updating the filter with visual odometry data is run in a sub rate com- pared to the loop updating the filter with odometry and gyro measurements, which is run at full filter rate.

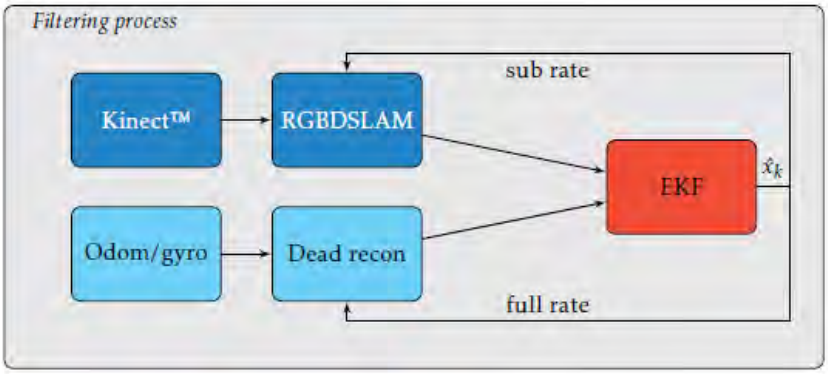


Fig 5.2: Filtering process. The filter has two loops with different rates.

A sample of how the input information to the filter may look is showed in Figure5.3. The covariance of visual odometry depends on the $vo_{trust\ parameter}$ which in turn depends on the number of reliable landmarks. The covariance of the odometry sensor depends on the traveled distance since last visual odometry update. These models the increased uncertainty related to the odometry sensor as the distance to the last visual measurement increases. Figure 5.4 shows the prior distribution of the filter, i.e. the distribution after system update but before measurement update. The posterior distribution is updated by the filter during every measurement update. The dots every third update symbolizes that visual odometry has provided information to the filter. These dotted estimates are the ones sent back to the RGBDSLAM algorithm. It can also be seen that the filter covariance increases in the same pattern as the odometry covariance.

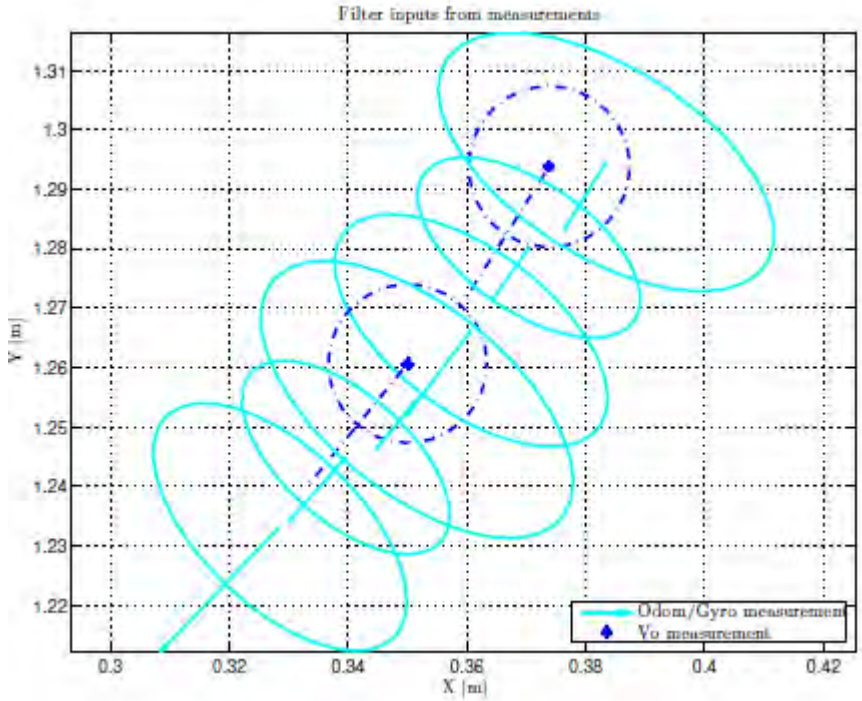


Fig 5.3 : Measurements from odometry, gyro and visual odometry sensors.

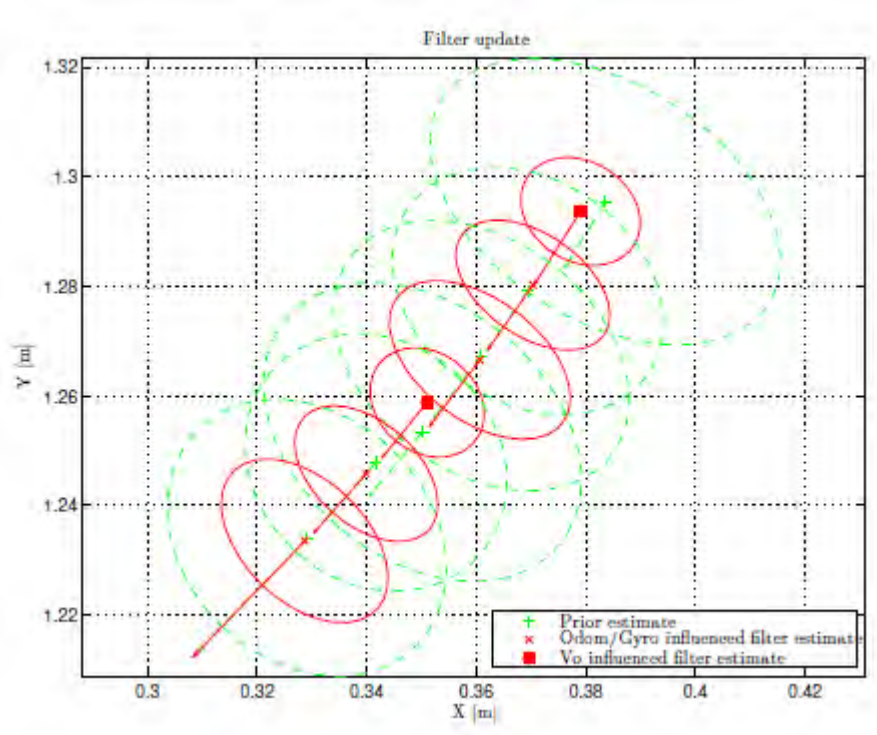


Fig 5.4 : Filter update event.

CHAPTER 6

IMPLEMENTATION AND EVALUATION METHODS

In this chapter, the cooperation of RGBDSLAM and the filter is described. Methods for data collection and post processing are also presented. The chapter is ended by a section that explains the experimental setup and evaluation criteria that are used when evaluating the implemented algorithm.

6.1 Filter Integration in RGBDSLAM

The models in Chapter 5 are well defined and strict in a mathematical sense but are not well suited for direct implementation in C++ and ROS. The theoretical models need some adjustments to fit the framework and additional functionality

e.g. synchronization of data is required.

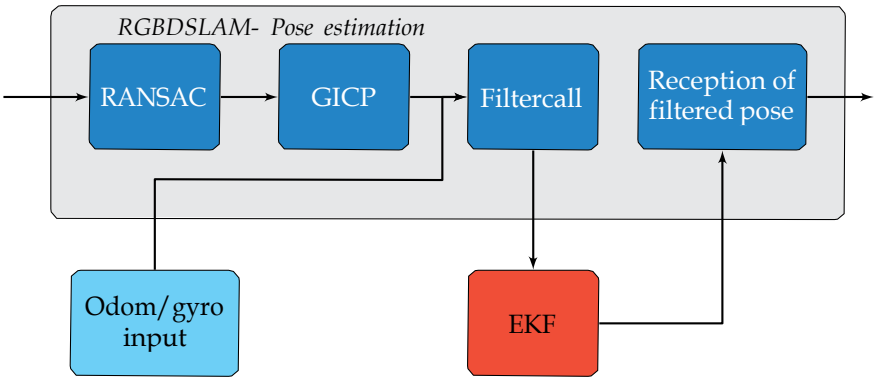


Fig. 6.1: The tasks of pose estimation.

Figure 6.1 gives an overview of how the pose estimation is performed in the algorithm. When new RGB and depth images, odometry data and gyro data are captured, three things can happen:

1. All of the information is processed.
2. Only the information from the odometry and gyro sensors is processed.
3. None of the information is processed.

In the *first case*, where all data is used, the visual data is joined by the odometry and gyro data in the box named “Filter call “. Here the measurements from the three sensors are inserted into a ROS service call and sent, using a blocking call- response method to the filter. During the service call the input data to the filter is published on the input topics of the filter node. After reception of new data, the filter checks the time stamps of the measurements and, if the three measurements are sufficiently close in time, the distribution of the internal Gaussian probability distribution is updated accordingly. A new estimate of the mean value of the distribution is computed and returned as the current pose estimate.

In the *second case*, where the visual data is disregarded, a multirate filter call is carried out. In this case a filter update with only odometry and gyro measurements (and information from the motion model) are used to create a new estimate of the quadcopter position. This estimate is not sent back to RGBDSLAM, but used internally in the filter.

The *third case*, where none of the data is used, is when the parameters for data skipping are used to lower the rate of input data to the algorithm.

If data is scattered in time the distribution is not updated. As a consequence, the time between updates, in Chapter 5 referred to as T_o, is varying.

6.1.1 Covariance of Visual Odometry

The implemented software has a parameter, v^0_{trust} (as in visual odometry trust) which adapts the trust in the visual data. Its value reflects how well the visual SLAM algorithm has succeeded in finding and matching landmarks in the processed images. The parameter is set according to the following algorithm.3D orientation estimate. The filter fuses data from the odometry, the gyro and the visual odometry. Given the state vector (5.7) the general motion model is

Algorithm 3 Set Trust in Visual Odometry

Input: Number of total matches between images, N_m, number of inliers (found by RANSAC), N_i, Boolean use_loop_closure, sequential ID of nodes that are matched, $ID1$ and $ID2$

Output: Noise parameter for visual measurement vo_{trust}

```
if use_loop_closure then
    if ID2 – ID1 < minimum distance for loop closure then
        vo_trust is set to adjustable value less than one
        loop closure = true
    end if
end if
if !loop closure then
    if vo_trust > 10 & Nm +. 0 & Ni >= 10 then
        if N_i/N_m > 0.7 then
            vo_trust = vo_trust/10
        end if
    else
        vo_trust = 1
        if N_i == 0 then
            vo_trust = 10000
        else if N_i < 10 then
            vo_trust = 1000
        else if N_m +. 0 then
            if N_i/N_m < 0.75 then
                vo_trust = 5
            else if N_i/N_m < 0.8 then
                vo_trust = 3
            else if N_i/N_m < 0.85 then
                vo_trust = 2
            end if
        else
            vo_trust = 1000000
        end if
    end if
end if
```

As Algorithm 3 shows the value of the parameter is decreased if there is a possible loop closure and the value is increased if there are few matches of landmarks between images.

6.2 Data Collection and Post Processing

In order to test parameter adjustments or new coding strategies in a repeatable way, it is not feasible to every now and then start up the Quadcopter and run it along some predefined track. The reasons for this are various, but amongst them are:

- The time consumption,
- The variation of the flown route and
- Changes in the environment.

The solution is to record the data generated by a flight in such a way that it can be replayed time after time. This is conveniently done by the *rosbag* command.

The visual data is presented in different ways on different topics. The data from the gyro and odometry sensors are given in one single form on one single topic each and there are hence no alternatives on how to record them. In addition to the concrete measurement data, the */ft.* topic is recorded. This topic contains the transformations between different coordinate systems fixed in both the Turtle- Bot and its surroundings. In the post processing, when RGBDSLAM is running, the */ft.* topic is supplemented with other transforms published by RGBDSLAM. Spelled out, all the recorded topics are:

- */camera/depth/camera_info*
- */camera/depth/image*
- */camera/rgb/camera_info*
- */camera/rgb/image_color*
- */imu/data*
- */odom*
- */tf*

Recording all these topics gives data in a rate of about 70 [Mb/s]. The data cannot be replayed at full speed, since that would make the input buffers of RGBDSLAM to overflow. Instead data is played in a speed of 1/2 to 1/20 of full speed depending on the parameter values used in the algorithm. The data flow when replaying a bag file can be seen in Figure 6.2.

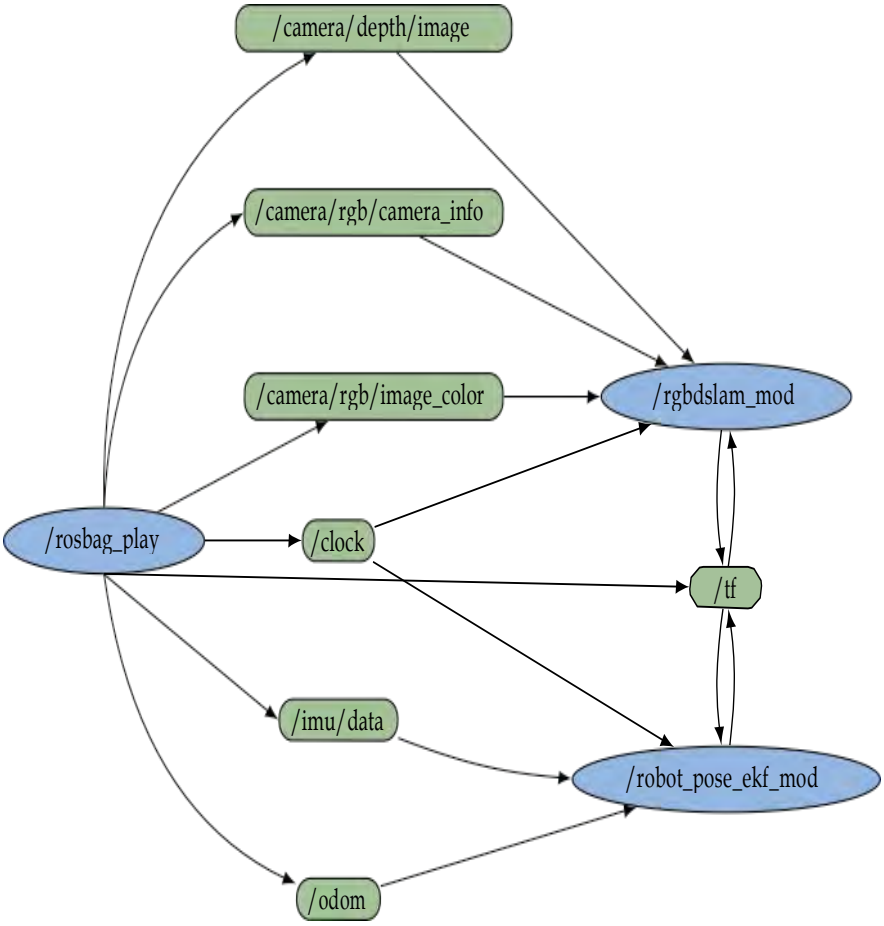


Fig. 6.2: Data flow when playing a bag file. The ellipses are ROS nodes.

6.3 Evaluation Criteria

Evaluation methods can be very different depending on the experimental setup and the resources available. A motion tracking system would for example give the opportunity to determine the position of the quadcopter with a high precision. A sensitive laser scan equipment could, by scanning the environment where tests are carried out, give a sort of ground truth for the estimated 3D map. In this project however, such high precision evaluation instruments have not been used. Instead the results have been evaluated using more ordinary techniques: 3D maps have been evaluated using ocular inspection and the estimated trajectories have been evaluated using the ground truth matcher.

There are three key factors which are altered in order to evaluate the performance of the modified algorithm. Trajectories and maps can be computed

1. with or without filter
2. with the original algorithm parameters set to give a robust or a less computationally heavy performance
3. using different data sets.

The parameter alternatives mentioned above are specified in Table 6.1. The different choices essentially consist of deciding how often to call the filter, how many key points to extract and how many matches to require between images.

The filter frequency determines how often to send both visual and odometric in- formation to the filter whereas the multirate filter frequency sets how often to send odometric data only. The maximum and minimum number of key points simply specify how many key points to extract from the images using the chosen image extractor (SURF is used in this project). The sufficient matches and min. matches parameters decide how many matches between the sets of key points from different images that are to be seen as a sufficient number and a minimum number, respectively. If the sufficient number of key points is reached, the algorithm will not try to get further matches and if there are fewer than the minimum value of matches, the algorithm will not try to calculate any transform between the corresponding images.

Two data sets have been used in the evaluation. These are referred to as the “figure- eight “data set and the “few landmarks “data set. In the figure-eight set the quadcopter is run two laps in a figure-eight pattern, starting with a left turn at the base of the eight. The data set is quite ordinary in the sense that the room where it is recorded is not so big and the environment gives the quadcopter varying depth and RGB information. The second data set, the few landmarks set, unsurprisingly give the algorithm rather few landmarks to work with. It is recorded in a partly more sterile environment with a flat, white wall. Both data sets are recorded by flying the quadcopter manually with a PlayStation 3 control on a track marked with tape on the floor. The different environments can be studied in Figure 6.3 and Figure 6.4.

	Robust setup	Slimmed setup	Comment
Filter frequency	2.5 hz	1.5 hz	Vo, odom and gyro measurements
Multirate filter frequency	7.5 hz	6 hz	Only odom and gyro measurements
Max. key points	400	75	Maximum number of key points extracted from image
Min. key points	100	50	Minimum number of key points extracted from image
Sufficient matches	100	60	Sufficient number of matches between key- points from different images
Min. matches	75	20	Minimum number of matches between key- points from different images

Table 6.1: Parameter specifications.

6.3(a) Overview of the office room.

6.3 (b) Another view from the office room.

Fig. 6.3: Views from the office in which the figure-eight data set was recorded.

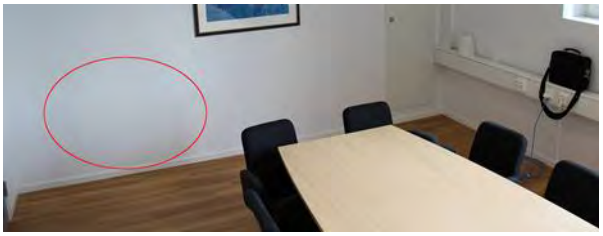


Fig.6.4: Overview of the meeting room in which the few landmarks data set was recorded. The ellipse indicates the area with few landmarks.

Setup	Filter	Data set	Parameter setting
1	No	Figure-eight	Robust
2	No	Figure-eight	Slimmed
3	No	Few landmarks	Robust
4	No	Few landmarks	Slimmed
5	Yes	Figure-eight	Robust
6	Yes	Figure-eight	Slimmed
7	Yes	Few landmarks	Robust
8	Yes	Few landmarks	Slimmed

Table 6.2: Eight different combinations of evaluation setups.

6.4 Ground Truth Matcher

The ground truth matcher is written in MATLAB. It takes data of where the ground truth trajectory (represented by a tape on the floor) is located as well as measurements of the

estimated trajectory. The origins of the two data sets have to be a common spot, fixed in the world. The estimated trajectory is rotated an angle θ_k, around its origin and the degree of match between the two patterns is calculated. The method is visualized below in Figure 6.5. The generated match value in the example is 0.058, which means that the nodes have an average deviation of 5.8 cm compared to ground truth.

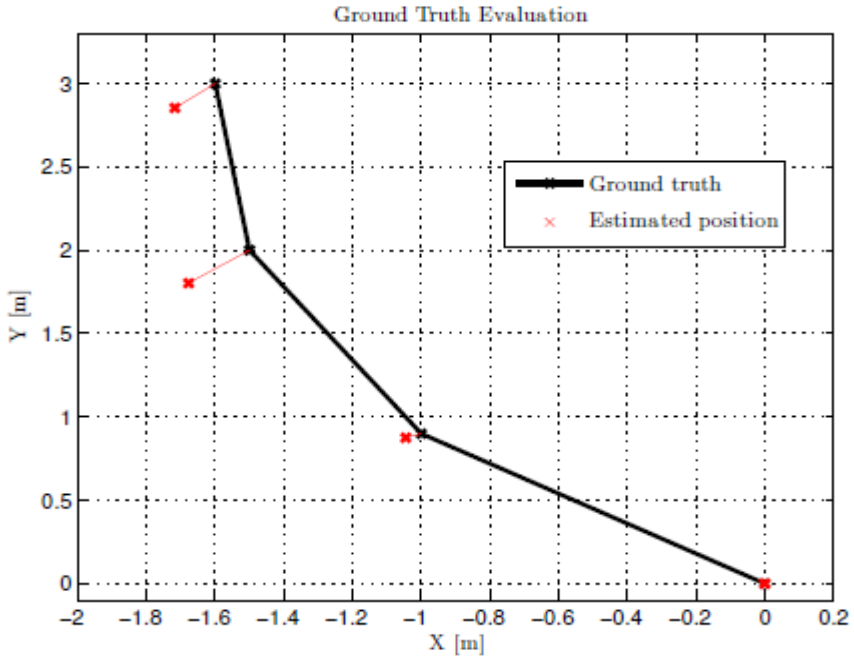
To calculate the match value, first let $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$, where θ_k is the angle of rotation of the trajectory in iteration k, k = 1 . . . K. The degree of match as function of θ, M(θ), is calculated according to

$$M(\theta) = \frac{1}{N}\sum_{i=0}^{N} a_i(\theta), \tag{6.1}$$

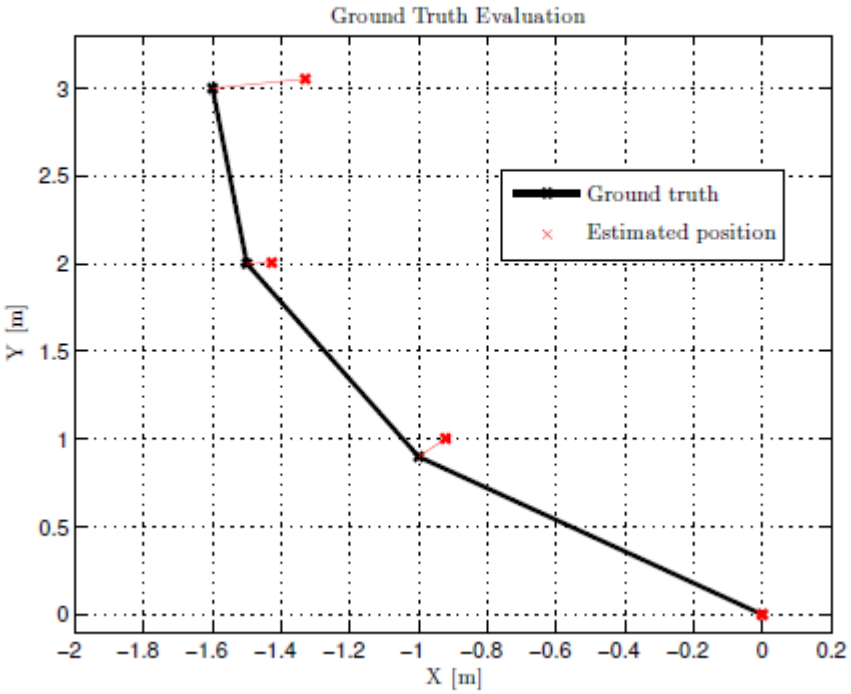
where a_i (θ) is the shortest Euclidian distance from node i in the estimated trajectory to any node in the ground truth given the angle of rotation θ. N is the number of positions in the estimated trajectory. The final match value is the minimum of all match values.

$$M = \min_{\theta \in \Theta} M(\theta). \tag{6.2}$$

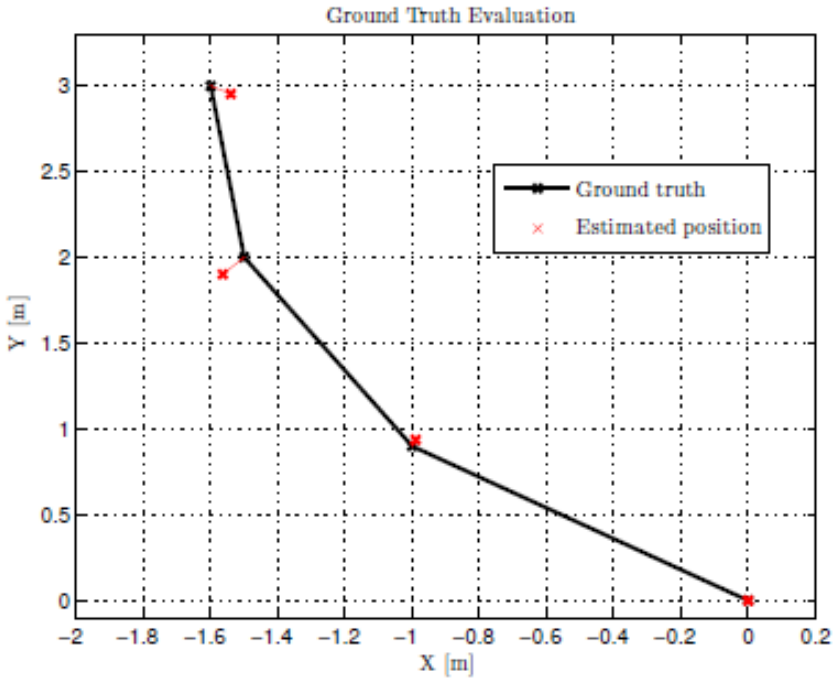
.



6.5(a) Estimated trajectory to the left of ground truth.



6.5 (b)Estimated trajectory to the right of ground truth.



6.5(c) Optimal match.

Fig 6.5: Illustration of how the match against ground truth works.

CHAPTER 7

RESULTS

This chapter starts with a section that demonstrates the accuracy of the sensors. This is followed by a review of the results in Section 7.2 and presentation of the 3D maps is found in 7.3.

7.1 Accuracy of Sensors

This section shows how the gyro and odometry sensors perform under different circumstances. This gives a hint of the accuracy of the sensors as well as an idea of how well these sensors can be trusted when fusing their measurements with the visual odometry.
Three different evaluations are made; two using a circular track and one using a straight track. The circular track is approximately one meter in diameter and its curvature is in that way comparable with the curvature in the figure-eight data set that will be used in the evaluations. Two runs are made; one clockwise and one counterclockwise. Each run consists of two laps and the quadcopter is flown in three different speeds. The results of these tests show whether the gyro and odometry sensors are calibrated or not.

The run along the straight line will show whether the gyro or the odometry suffers from drift and in the odometry case also how well it measures the traveled distance. The line is ten meters long and the runs are made in three different speeds.

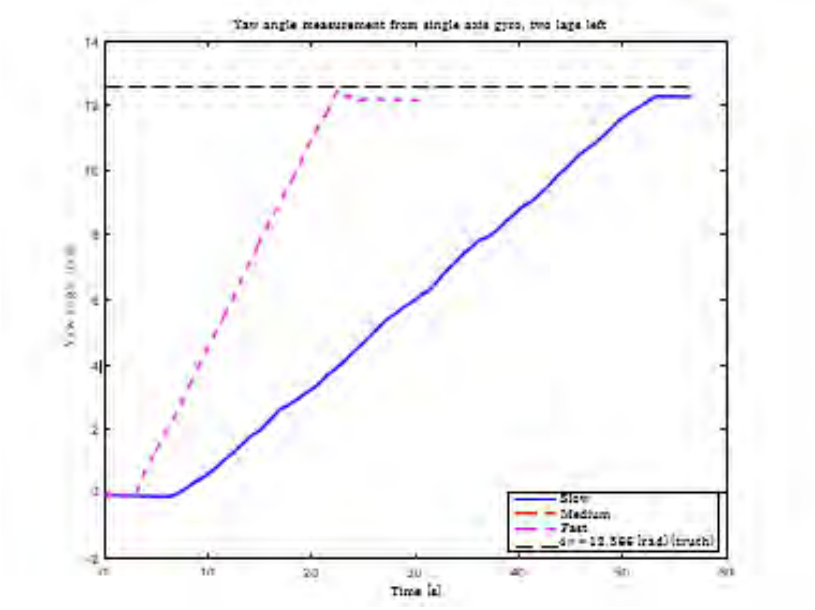
The speeds in which the quadcopter is flown in all three evaluations are as follows:

1. Low speed is approximately 15-20 [cm/s],
2. medium speed is approximately 20-25 [cm/s] and
3. fast speed is approximately 30-35 [cm/s].

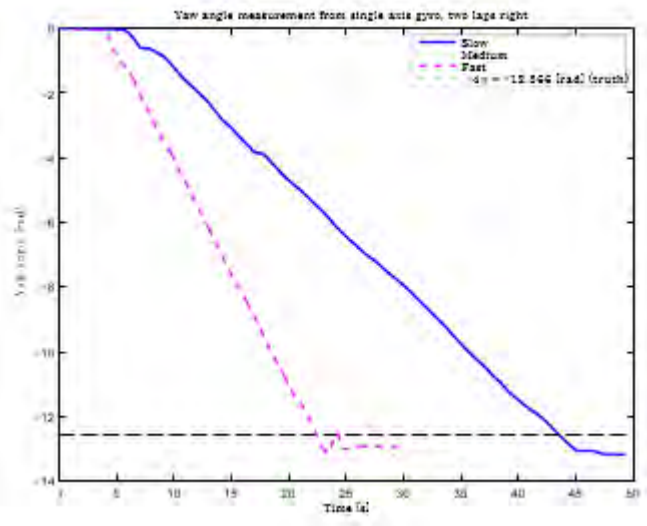
Figure 7.1 shows the measurements from the gyro when the quadcopter is flown a- round in a circular pattern. Measurements in Figure 7.1a are captured when the quadcopter moves counterclockwise and the plot indicates that the measurements have not quite reached the true level, except in the run with medium speed. Figure 7.1b, related to the clockwise movement of the quadcopter, shows that the measurements are all too small. This can be explained by a small

negative drift in the gyro. The odometry also measures the yaw angle and from the results in Figure 7.2 the conclusion is that the measurements vary widely.

Another test of the sensors has been performed flying on a straight line with a length of ten meters. Figure 7.3 shows gyro measurements along the line and Table 7.1 presents the measured length by odometry. The results indicate measurements near true values when the quadcopter is flown at low speed, whereas the deviations from true values are visible at higher speeds.

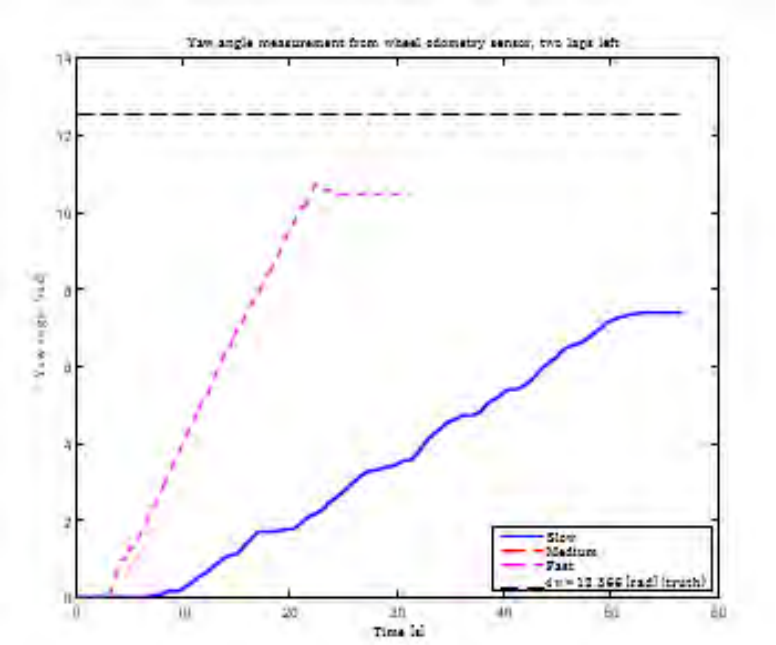


7.1 (a) Gyro measurements when flying two laps counter- clockwise. The measured values are a little too small except in the run with medium speed.

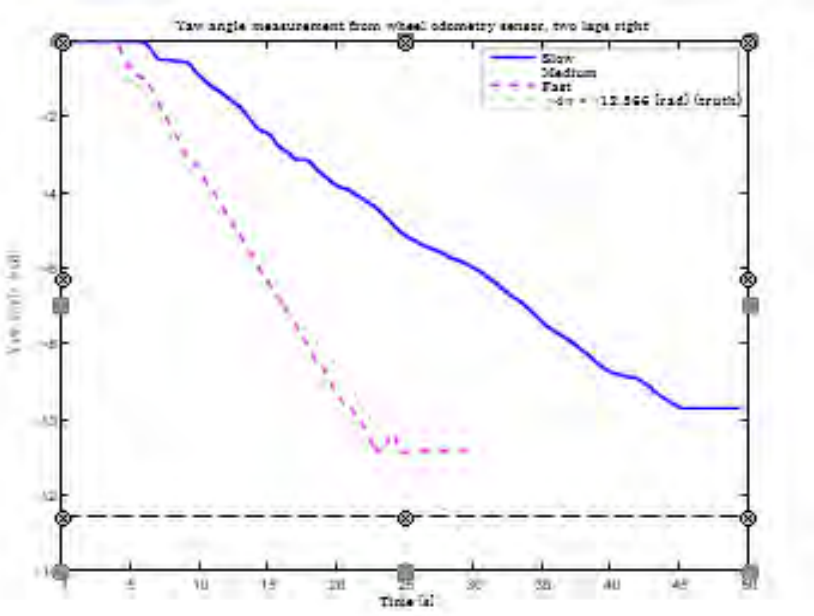


7.1 (b) Gyro measurements when flying two laps clockwise. The measured values are too big in all three cases.

Fig 7.1: Yaw angle measurements from the gyro when flying with three different speeds, two laps in a circle with a diameter of one meter. The measured values indicate a small negative drift for the gyro.



7.2 (a) Odometry measurements when flying two laps counterclockwise. The measured values are a little too small except in the run with medium speed.



7.2 (b) Odometry measurements when flying two laps clockwise. The measured values are too small in two of three cases.

Fig 7.2: Yaw angle measurements from the odometry sensor when flying with three different speeds, two laps in a circle with a diameter of one meter.

The measured values indicate that the odometry does not measure the yaw angle as accurately as the gyro.

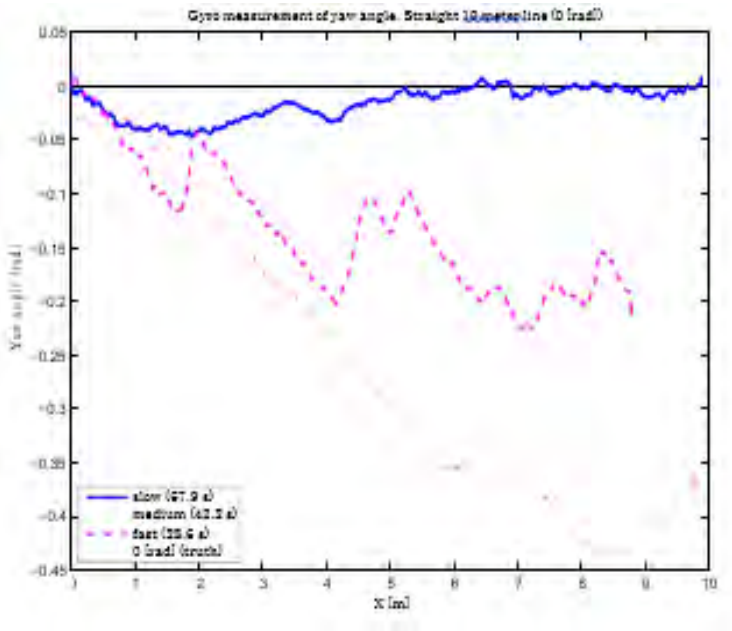


Fig 7.3: Yaw angle measurements from the gyro when flying on a straight ten-meter line.

The slow run gives a result close to the true value, while the drift in the two other cases is noticeable.

Speed	Measured lenght [m]	Time [s]
Slow	9.91	67.9
Medium	9.92	42.2
Fast	8.81	28.6

Table 7.1: Length measurements by odometry of a ten meters' straight line.

7.2 Evaluation of the SLAM Algorithms

In this section, the performance of both the original and the modified algorithm is evaluated. Original algorithm here refers to the non-altered open source algorithm RGBDSLAM, which uses visual information only, while modified algorithm refers to the one using visual information as well as odometry and gyro data.

7.2.1 Original Algorithm. Figure-Eight Data Set

The evaluation is started by running the original algorithm on the figure-eight data set.

Robust Parameters

The estimated trajectory, using the robust parameter alternative looks as in Figure 7.4. The joint of the two circles is approximately 10 cm too far away in the Y -direction compared to ground truth. The first and the second lap overlaps to a very large extent. The trajectory is somewhat jagged and uneven in a way that is not plausible compared to how the quadcopter was flown.

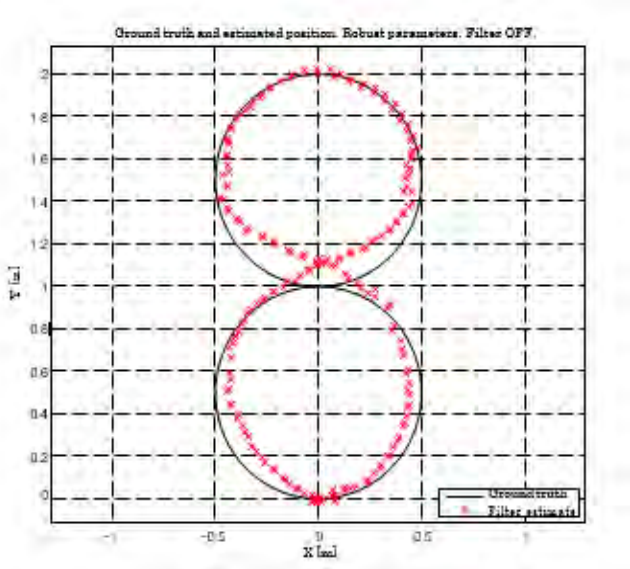


Fig 7.4: The estimated trajectory compared to ground truth.

Slimmed Parameters

Using the same data set, but setting the parameters of the algorithm to the more slimmed alternative, the result is as according to Figure 7.5. The attempt to lessen the computational burden results in that the estimated trajectory does not resemble the ground truth. The first estimates lie on track but once the environment is such that the matching between the extracted landmarks is too bad, the algorithm is lost and has very little chance to find its way back. The associated 3D map can be studied below.

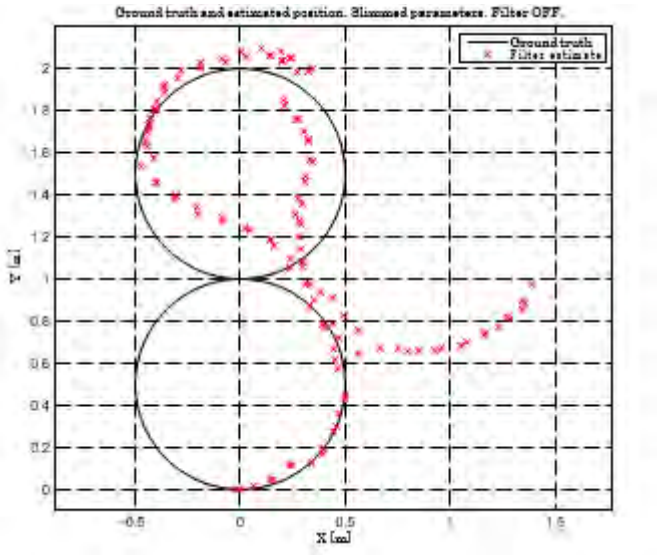


Fig 7.5: The estimated trajectory compared to ground truth. Slimmed parameters and no filter. The estimated trajectory does not resemble the ground truth.

7.2.2 Modified Algorithm. Figure-Eight Data Set

The concluding evaluation of the modified algorithm with the figure-eight data set is presented below.

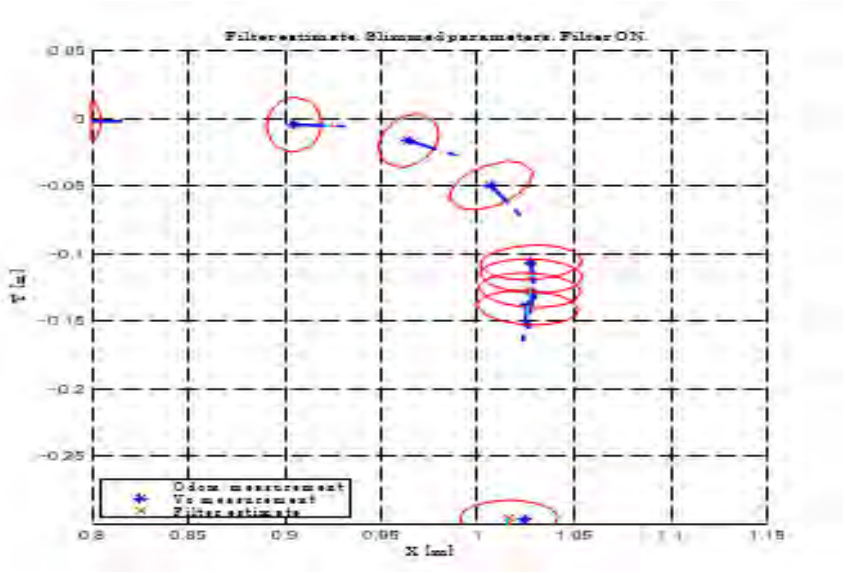


Fig 7.6: Pose and direction of travel estimates by the filter and direction of travel measurements by odometry and visual odometry.

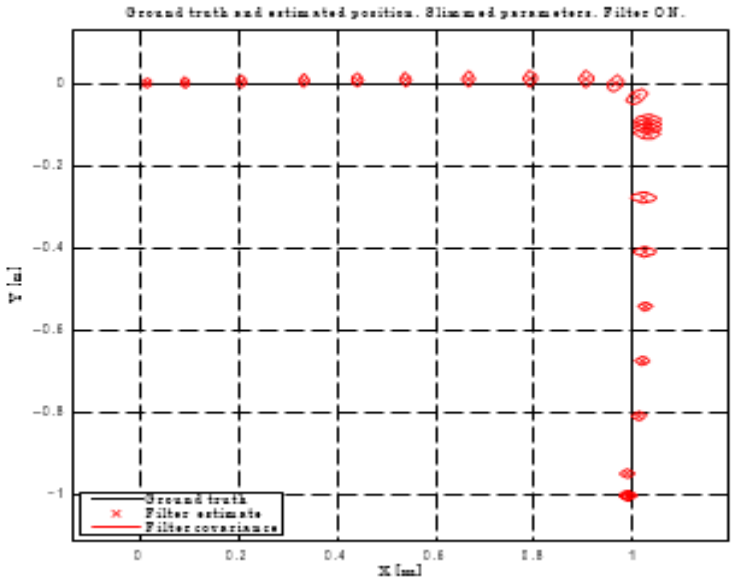


Fig 7.7: Match between the estimated trajectory and ground truth. Slimmed parameters and filter on.

Robust Parameters

Using the robust parameter setup and the figure-eight data set, the input looks as in Figure 7.14. A sample of estimated positions and measured directions are showed in Figure 7.15. The match against ground truth can be seen below.

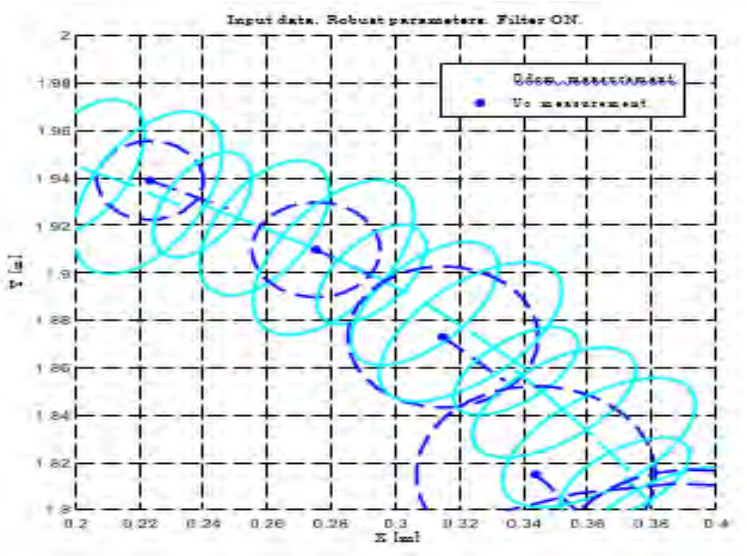


Fig 7.8: Visual and odometry inputs to the filter. Robust parameter settings in the figure-eight data set gives high confidence in visual odometry.

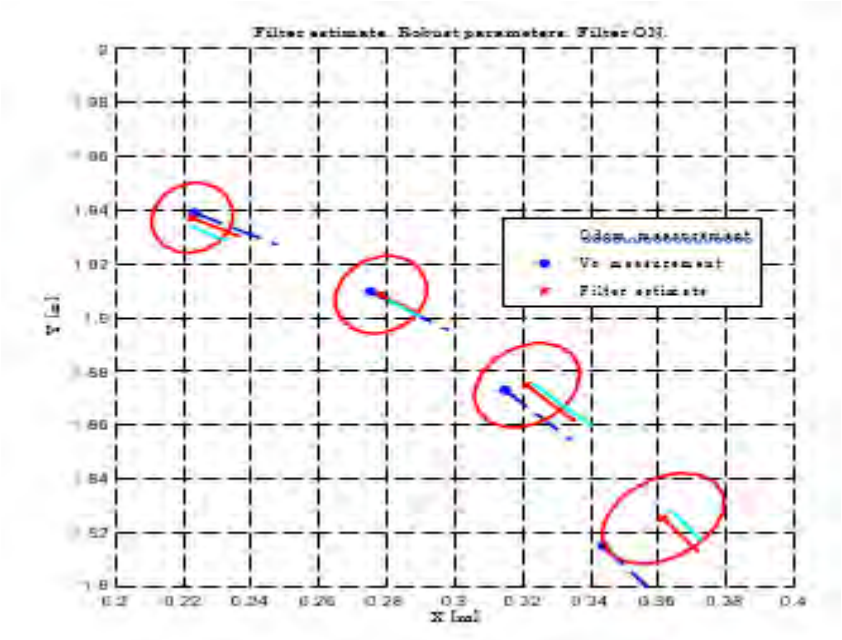


Fig 7.9: Filter estimates and measurements in the top right of the figure- eight route when the visual odometry have high reliability.

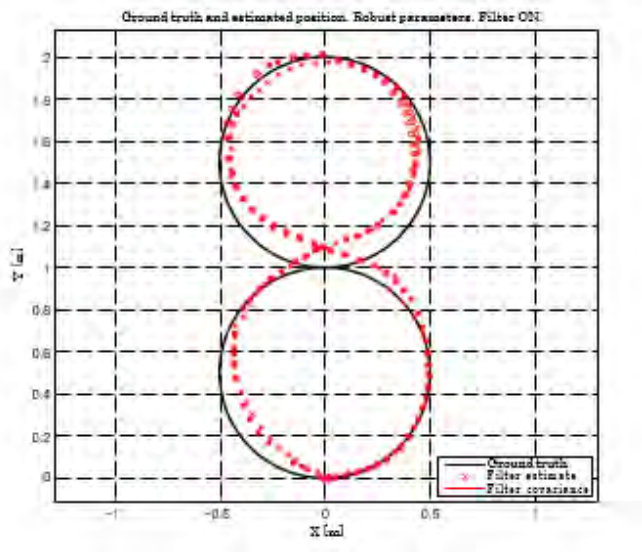


Fig 7.10: Match between the estimated trajectory and ground truth. Robust parameters and filter on.

Slimmed Parameters

The change to slimmed parameters results in less trust in the visual odometry, which is clear from Figure 7.11. Figure 7.12 shows that the uncertainty in the estimates from the filter is bigger, which follows from the more uncertain visual odometry measurements. The match against ground truth is presented in Figure 7.13 and 3D maps are found in Figures 7.17.

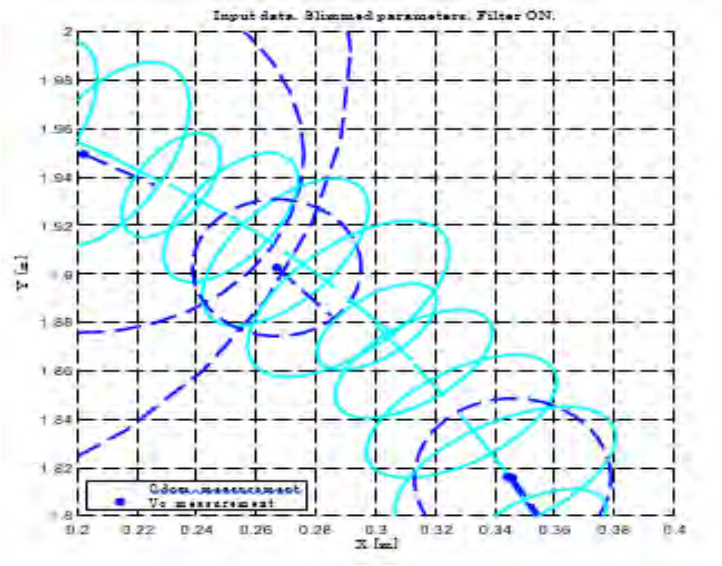


Fig 7.11: Visual and odometry inputs to the filter. Covariances of the visual odometry larger than in the robust parameter case.

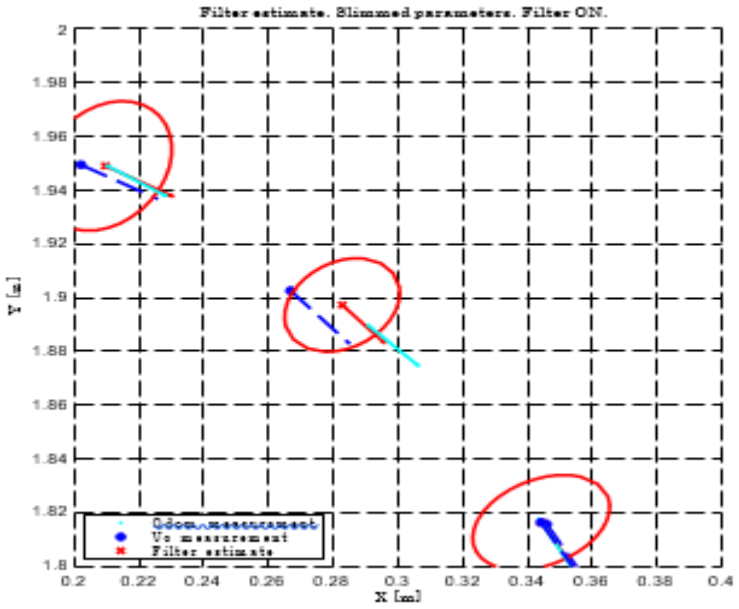


Fig. 7.12: Filter estimates and measurements in the top right of the figure eight route when the visual odometry has low reliability.

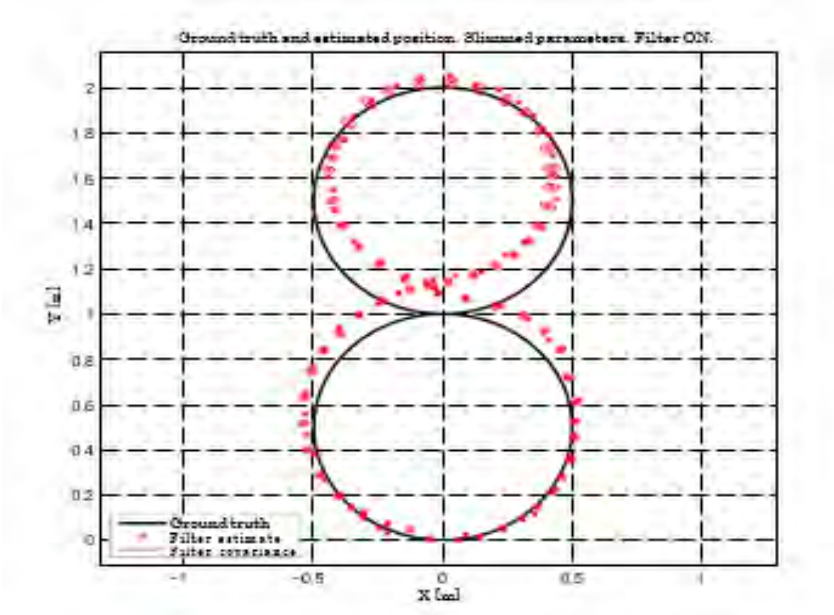


Fig 7.13: Match between the estimated trajectory and ground truth. Slimmed parameters and filter on. The resulting trajectory is not quite as smooth as with robust parameters.

7.3 3D Maps

In this section, the 3D maps generated by RGBDSLAM are presented. There is no quantitative quality measurement applied to them. Instead their strengths and weaknesses are discussed from what is visible in the figures. It is worth mentioning that the coordinate systems that can be seen in the maps represent the Kinect's position and not the one of the quadcopter base in contrast to the figures above.

7.3.1 Original Algorithm. Figure-Eight Data Set

Figures 7.14 and 7.15 show the results from the original algorithm run on the figure-eight data set. The first figure shows a rectangular room with a little mismatch on the right-hand side walls. Figure 7.16 shows no mismatch on the legs of the tables, while Figure 7.15 shows a non-rectangular room.

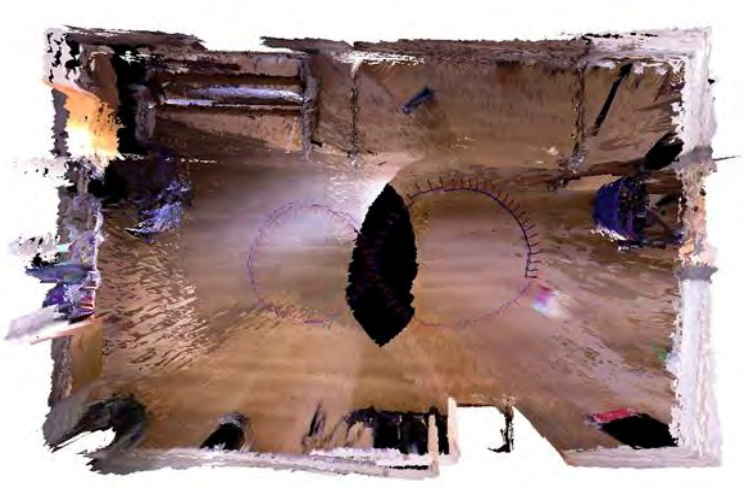


Fig 7.14: 3D map from the figure-eight data set. Robust parameters, no filter. Overview.

Fig 7.15: 3D map from the figure-eight data set. Slimmed parameters, no filter.

7.3.2 Modified Algorithm. Figure-eight Data Set

Fig 7.16 to 7.17 show maps estimated using the modified algorithm and the figure-eight data set. Both overviews show a rectangular room, although the map originating from the slim parameter setup has thicker walls. From Figures 7.16 and 7.17 it is also clear that table legs are more spread using slim parameters.

Fig 7.16: 3D map from the figure-eight data set. Robust parameters and filter. Overview of the rectangular shaped room. The right wall is thick

Fig 7.17: 3D map from the form figure-eight data set. Slimmed parameters and filter. An overview of the room. The rectangular shape of the true room is kept and the map has details that seem to be in one piece.

CHAPTER 8

CONCLUSION

This chapter sums up the results and discusses factors that affect the result and matters that could have been treated differently. The chapter also gives a list of future work with relevant tasks that could improve the performance and usability of the system.

8.1 Results and discussion

The overall results of the project indicate that the SLAM algorithms can run successfully even when the camera moves at a high speed in a 3D environment. They can also handle an environment with few landmarks and low lighting conditions.

The 3D maps and associated building models are enabling the convergence of several established disciplines, including engineering computer-aided drafting (CAD), architectural building information management (BIM), and geographic information systems (GIS). Our project, 3D reconstruction using Kinect and RGB-D SLAM, presents an efficient and stable way of reconstructing an 3D environment. 3D mapping can be used for large scale topographic surveys; city planning and management; academic research; utilities and energy management such as powerline inspections; scoping, planning and management of mines; forestry design, management and operation. The preliminary result shows us that our system, although suffer a little from portability issue, can create 3D representation of a 20-25-bulding in 20 minutes. Our immediate future goal is to improve portability and usability, so that the system is usable even at very high altitudes and can process and compute the data faster.

8.2 Future Work

The list of future work can be made long. Here follows an abstract of tasks that are interesting to implement as a follow up to this project.

1. The whole project can be made on a portable small computer like the Raspberry Pi 3 which will eliminate the need for any wired connection. The 3D map depth data from the Kinect will be streamed to the main workstation using Samba. The processing will be done on a powerful workstation which will be based on an x86 processor instead of arm or arm64 architecture.

2. Investigate possible advantages with other feature detectors - Early on it was in the scope of this project to investigate whether different feature detectors would give any difference when extracting landmarks from sparse featured images. That task has been given low priority and has not yet been done. Even so it still remains an interesting task.

3. Exporting 3D maps to OctoMap - Exporting the produced 3D maps to OctoMap is a way to get a map that is suitable for route planning and, in the end, autonomous navigation. It is also a very efficient way of saving maps of the environment [39].

4. Evaluate and shorten the computation time - RGBDSLAM came with an implementation to measure the computation times for different steps. The extension with the filtering procedure is interspersed in such a way that it invalidates these measurements and it would be interesting to restore the order and be able to accurately measure computation times again. This is a step towards optimizing the code and in the long term, perhaps, approach real time performance.

REFERENCES

1. T. Bailey. Mobile robot localization and mapping in extensive outdoor environments. 2002.
2. P. Newman, D. Cole, and K. Ho. Outdoor slam using visual appearance and laser ranging. In Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pages 1180–1187. IEEE, 2006.
3. K. Granström and J. Callmer. Large scale slam in an urban environment. 2008.
4. C. Hertzberg, R. Wagner, O. Birbach, T. Hammer, and U. Frese. Experiences in building a visual slam system from open source components. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 2644–2651. IEEE, 2011.
5. Joseph Aulinas, Yvan Petillot, Joaquim Salvi, Xavier Llado, "The SLAM problem: a survey", Proceeding of the 2008 conference on Artificial Intelligence Research and Development Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence (2008)
6. Zoran Sjanic, Martin Skoglund, Thomas B. Schon, Fredrik Gustafsson,” Solving the SLAM Problem for Unmanned Aerial Vehicles Using Smothed Estimates”, Proceedings of the Reglermöte Swedish Control Conference ,2010
7. Mondragón Iván, Olivares-Méndez F., Campoy Miguel, Martínez A., Mejias Pascual, Mejias Carol, Mejias Luís,” Unmanned aerial vehicles UAVs attitude, height, motion estimation and control using visual systems”, Autonomous Robots, Volume: 29(1), pp.17-34, 2010
8. Jung Sungyoung, Kim Jungmin, Kim Sungshin, “Simultaneous localization and mapping of a wheel-based autonomous vehicle with ultrasonic sensors”, Artificial Life and Robotics, Vol.14(2), p.186-190 ,2009
9. Moreno Luis, Garrido Santiago, Blanco Dolores, Muñoz M. Luisa, “Differential evolution solution to the SLAM problem”, Robotics and Autonomous Systems, Vol.57(4), p.441-450, 2009
10. Kim C, Sakthivel R, Chung WK, “Unscented FastSLAM: A Robust and Efficient Solution to the SLAM Problem”, IEEE transactions on robotics, Vol.24 (4), p.808-820, 2008
11. Kümmerle Rainer, Steder Bastian, Dornhege Christian, Ruhnke Michael, Grisetti Giorgio, Stachniss Cyrill, Kleiner Alexander,” On measuring the accuracy of SLAM algorithms”, Autonomous Robots, Vol.27(4), p.387-407, 2009

12. Georgios Lidoris, Kolja Kuhnlenz, Dirk Wollherr, Martin Buss," Combined Trajectory Planning and Gaze Direction Control for Robotic Exploration", Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007
13. F. A Moreno, J. L Blanco. Gonzalez, "Stereo vision specific models for particle filter-based SLAM", Robotics and Autonomous Systems, Volume: 57 Issue: 9, Publisher: Elsevier B.V., Pages: 955-970 ISSN: 09218890 ,2009
14. P. Moutarlier and R. Chatila, "Stochastic multisensory data fusion for mobile robot location and environment modeling, "in 5th international symposium on robotics research, MIT Press 1989, pp. 85-94
15. W.M Gamini Dissanayake, "Solution for SLAM and map building SLAM problem", IEEE Transactions on Robotics and Automation Volume: 17, Issue: 3, Publisher: IEEE, Pages: 229- 241, 2001
16. Kalman filters (Leonard and Durrant-Whyte 1991) Smith et al. (EKFs) Mobile robot Localization by Tracking geometric beacons IEEE Transactions on Robotics and Automation 7(4), 376-382 1990.
17. Sparse extended information filters 2005 IEEERSJ International Conference on Intelligent Robots and Systems (2005) Publisher: IEEE Pages: 3281-3288 (Eustice et al 2005. Thrun et al 2004)
18. Particle filter (Monternelmo et al 2005) Gutmann and Konolige 1999. The International Journal of Robotics Research (August 2004), 23 (7-8), pg. 693-716 Oslon et al 2006.)
19. [On measuring the accuracy of SLAM algorithms Autonomous Robots (November 2009), 27 (4), pg. 387-407 (Hermosillo et al 2003; Thrun et al 2006 Yguel et al 2007).
20. Simultaneous Localization and Mapping (SLAM) of a wheel-based autonomous vehicle with ultrasonic sensors Artificial life and robotics 2009, vol. 14, no 2, pp. 186-190; (Sungyoung Jung et al 2006).
21. Atul Kumar Gupta, Vivek Jha, Vijay Kumar Gupta, "Design and Development of Remote Controlled Autonomous Synchronic Hexaroter Aerial (ASHA) Robot", Procedia Technology, vol. 14, pp. 51, 2014, ISSN 22120173.
22. Endrowednes Kuantama, Dan Craciun, Ioan Tarca, Radu Tarca, Mechanisms and Machine Science, vol. 46, pp. 269, 2017, ISSN 2211-0984, ISBN 978-3-319-45449-8.
23. Square root smoothing and mapping (SAM) Dellaert The International Journal of Robotics Research (2006) Volume: 25, Issue: 12, Publisher: Citeseer, Pages: 1181-1203 2005.B.

Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual slam for flying vehicles. Robotics, IEEE Transactions on, 24(5):1088–1093, 2008.

24. J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart. Towards a benchmark for rgb-d slam evaluation. In Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS), Los Angeles, USA, volume 2, page 3, 2011.
25. R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 1–4. IEEE, 2011.
26. Project OpenKinect http://openkinect.org, 2012.
27. Robot Operating System http://www.ros.org, 2012.
28. F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. 2011.
29. N. Engelharda, F. Endresa, J. Hessa, J. Sturmb, and W. Burgarda. Real-time 3d visual slam with a hand-held rgb-d camera. In Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Västerås, Sweden, volume 2011, 2011.
30. M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381–395, 1981.
31. L. Juan and O. Gwun. A comparison of sift, pacifist and surf. International Journal of Image Processing, 3(4):143–152, 2009.
32. H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. Computer Vision ECCV 2006, pages 404–417, 2006.
33. D. G. Lowe. Object recognition from local scale-invariant features. In Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, volume 2, pages 1150–1157 vol. 2. Ieee, 1999.
34. P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2):239–256, 1992.
35. A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In Proc. of Robotics: Science and Systems (RSS), 2009. Cited on page 20.
36. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In Robotics an Automation (ICRA), 2011 IEEE International Conference on, pages 3607–3613. IEEE, 2011a.
37. F. Gustafsson. Statistical Sensor Fusion. Studentlitteratur, 2010.

38. Orocos Bayesian Filter Library http://www.orocos.org/bfl, 2012. Cited on page 30.
39. K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation, volume 2, 2010.