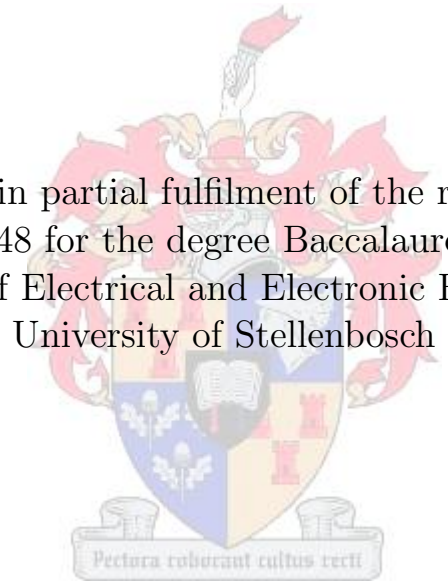
Monocular Vision Based SLAM Using Kinematic State Estimation

by

Aidan Russel Landsberg



Report submitted in partial fulfilment of the requirements of the module Project(E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Dr. C.E. (Corné) Van Daalen

May 2015

Summary

Opsomming

Acknowledgements

I would like to express my sincere gratitude toward the following individuals for their role in this project:

- My heavenly Father, for providing me with the intellectual capacity, guidance and support necessary during this project while remaining faithful and true in the toughest of times.

- My study leader, Dr. Corné Van Daalen, for his endless enthusiasm, guidance, patience, support, time and invaluable insight. As well as for personally setting aside the time to propose and supervise this project.

- My parents, for their endless support and motivation. As well as for making all the necessary sacrifices to provide me with the opportunity to complete this project.

- Mr. Arno Barnard, for his advice regarding the embedded design.

- My girlfriend Bianca La Gorcé, for supporting me endlessly.

- Benjamin Pannell, for being a great personal mentor and friend. As well as for providing me with insight regarding various software concepts.

- Luca Duesimi, for aiding in the design and construction of the stability platform.

- Warren Farmer, Kurt Coetzer and Lowku Leeuwenaar, for helping construct the stability platform.

Declaration

I, the undersigned, hereby declare that the work contained in this report is my own original work unless indicated otherwise.

Signature..... Date.....

Contents

List of Figures

List of Tables

Acronyms

2D	Two-dimensional
3D	Three-dimensional
CMOS	Complementary Metal-Oxide Semiconductor
EKF	Extended Kalman Filter
GPIO	General-purpose input/output
IMU	Inertial Measurement Unit
KF	Kalman Filter
I2C	Inter-Integrated Circuit Bus
ISR	Interrupt Service Routine
LIDAR	Light Detection and Ranging
MonoSLAM	Monocular Simultaneous Localisation and Mapping
PDF	Probability Density Function
PTAM	Parallel Tracking and Mapping
RV	Random Variable
SIS	Sequential Importance Sampling
SPI	Serial Peripheral Interface
SLAM	Simultaneous Localisation and Mapping
USB	Universal Serial Bus

List of Symbols

W	Inertial reference fame
C	Camera's free coordinate body frame
ΔT	Sampling instance
π	Constant denoting the ratio between a circles radius and its circumference
$\boldsymbol{\mu}$	Mean vector of a Gaussian random variable
$\boldsymbol{\Sigma}$	Covariance matrix of a Gaussian random variable
η	Normalisation contant
$\mathbf{I}$	Identity matrix
$\boldsymbol{\omega}$	angular velocity (expressed in radians per second)
$\mathbf{R}^{CW}$	Rotation matrix projecting an entity from the body frame to the inertial frame
$\mathbf{C}$	Camera calibration matrix
f	Focal length of camera
k_u	Focal length normalisation constant
k_v	Focal length normalisation constant
u_0	Principal Point x-coordinate
v_0	Principal Point y-coordinate

Notation

Notation	Entities
x	Lower case italic text represents a scalar
x^W	Superscripts represent the coordinate frame (e.g. W or C)
$\mathbf{X}^x$	Capital boldface text with a superscript variable represents a Jacobian
x^T	Superscript T however represents the transpose
x_t	Subscripts bind a values to a specific instance (e.g. time or feature)
$\bar{x}$	Overscore text represent an estimate
$\|x\|$	A modulus denotes the absolute value
$\|\|x\|\|$	A double modulus denotes the norm
$\dot{x}$	Dot symbols represent derivatives
$\mathbf{x}$	Lowercase boldface text represents a vector
$\mathbf{X}$	Capital boldface text represents a Matrix
	Processes
$quat(x)$	Function that converts a variable into a quaternion
$bel(x)$	Function that computes the Belief
$x \otimes x$	Represents a quaternion multiplication
$\frac{dx}{dt}$	Liebniz's notation to denote a standard derivative
$\frac{\partial x}{\partial t}$	Liebniz's notation to denote a partial derivative

1 Introduction

1.1 Robotic Localisation and Mapping

Robotics aims to equip machines with the capability of operating autonomously in the physical world to serve various practical purposes. These machines (robots) are designed to resemble human behaviour and action, so that they can substitute for humans in unknown and potentially hazardous environments ranging from planetary exploration to assembly lines [1, 2]. To achieve complete autonomous operation, it is essential that the robot is capable of observing its surrounding environment, subsequently building a map thereof in order to locate itself within this map. The aforementioned processes are more commonly referred to as *map building* and *localisation* respectively. Ultimately, the physical world presents many unforeseen factors and circumstances. These factors contribute to *uncertainty* and generally emerge due to a robot's lack of critical information. Factors such as environments, sensors, robots, models and computation all lead to an increase in uncertainty and can prohibit accurate map building and subsequently, localisation. *Probabilistic robotics* however, models these uncertainties mathematically in order to provide fundamental probabilistic algorithms that can be used to obtain reasonably accurate and efficient solutions to map building and localisation.

Occupancy grid mapping is map building algorithm that uses probabilistic algorithms. Initially presented in a study by Elfes and Moravec [3], occupancy grid mapping seeks to calculate the probability that a given position in the environment is occupied by an obstacle. Occupancy grid mapping typically uses range sensors, such as sonar or laser range finders to calculate these probabilities and represent them in a *dense* spatial map. An example of an occupancy grid map is depicted in Figure 1. The disadvantages of using occupancy grid mapping however, is that the robot's pose (a robot's position and orientation) is assumed to be known and that the environment is assumed to be static, limiting the practical applications thereof. A further overview of alternative map generating techniques is presented in a study by Thrun [4].

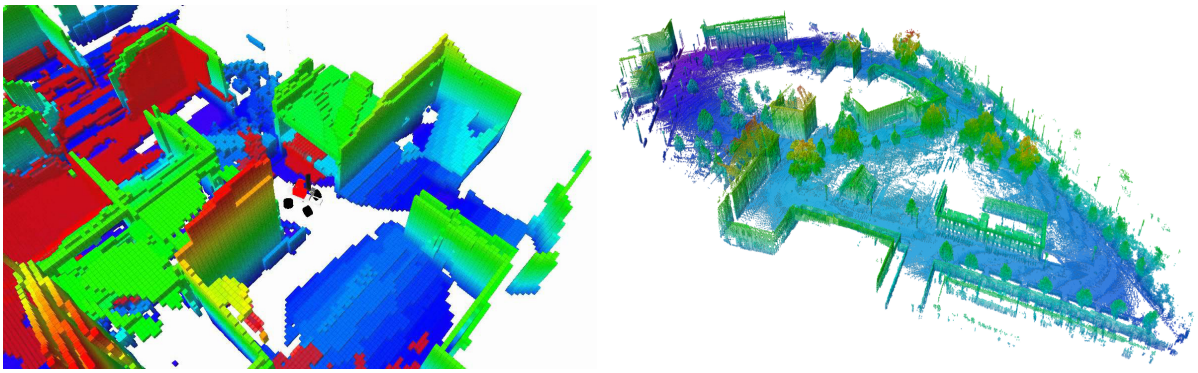


Figure 1: Examples of three-dimensional (3D) occupancy grid (dense) maps. Adapted from [5].

Solutions to the *Simultaneous localisation and mapping* (SLAM) problem however, presents methods that concurrently realises map building and localisation [6]. SLAM can be described as utilising both the sensor measurements and the control inputs of the robot to construct a continuously expanding map of features in the surrounding environment, while concurrently estimating its location with respect to these features. The sensor measurements provide information about the robot's surroundings and the control inputs

provide the information about how the robot is moving. A SLAM map typically represents the locations of its features as a *sparse* set (as opposed to a dense occupancy grid map). An example of a SLAM map is depicted in Figure 2.

The relationship between map building and localisation remains essential to the SLAM problem. If the localisation technique is incorrect, subsequently obtained sensor information will be incorrect, resulting in map estimates that differ from the actual state of the environment. An incorrectly modelled environment will render the sensor information useless, as this information will not correspond with those expected by the constructed map. Ultimately, the resulting localisation approximation will drift over time and eventually become extremely inaccurate. Most modern realisations of SLAM rely on certain probabilistic methods, namely *recursive state estimation* to provide suitable estimates that minimise the uncertainty regarding the mapping and localisation relationship.

The most commonly used recursive state estimation techniques include both *optimal filtering* techniques [**?**] as well as *sequential importance sampling* (SIS) techniques [**?**, **?**]. The differences regarding the aforementioned methods can be described as follows: an optimal filtering technique only considers a single hypothesis upon modelling, whereas SIS techniques consider multiple hypothesis. A further summary regarding the aforementioned probabilistic approaches is provided in a study by Chen [**?**].

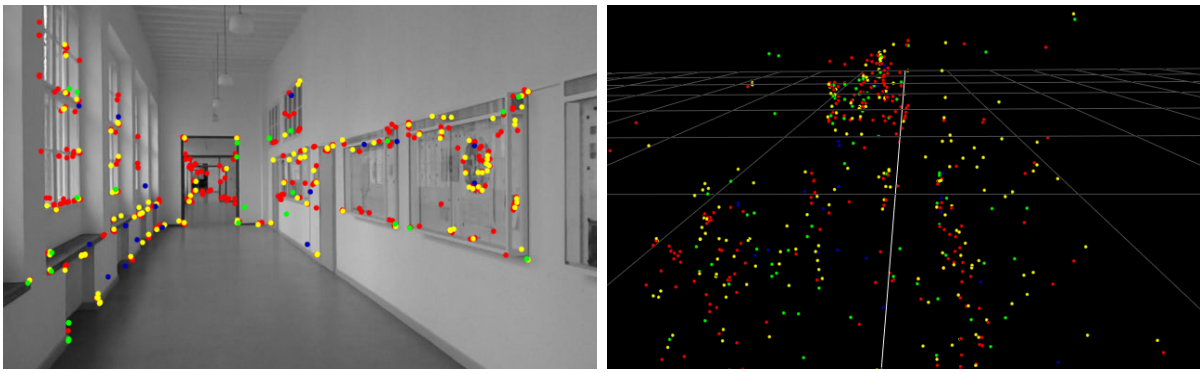


Figure 2: Left: Image frame indicating the feature points. Colours represent the size of a feature (warm colours being the smallest and cool colours being the largest). Right: Reconstructed SLAM three-dimensional (3D) map depicting the feature points with respect to a ground plane (shown as a grid). Adapted from [**?**].

The choice of implementation to solve the SLAM problem will primarily depend on the type of sensor(s) along with the time constraint imposed upon the application (e.g. online map generation vs. offline map generation). Furthermore parameters such as the resultant map's dimensionality as well as their sparsity representation (e.g. point clouds or sparse sets) are typically considered upon a suitable implementation.

Figure 3 depicts the SLAM procedure and is further explained in Table 1.1. The robot stores an internal representation (or estimate) of the positions of the features, its pose as well as the uncertainty associated with each of these entities. It should be noted that these uncertainties are not statistically independent of one another. At each frame, a *prediction* regarding the robot's pose, a *measurement* of the observed feature and an *update* of the internal representation is made.

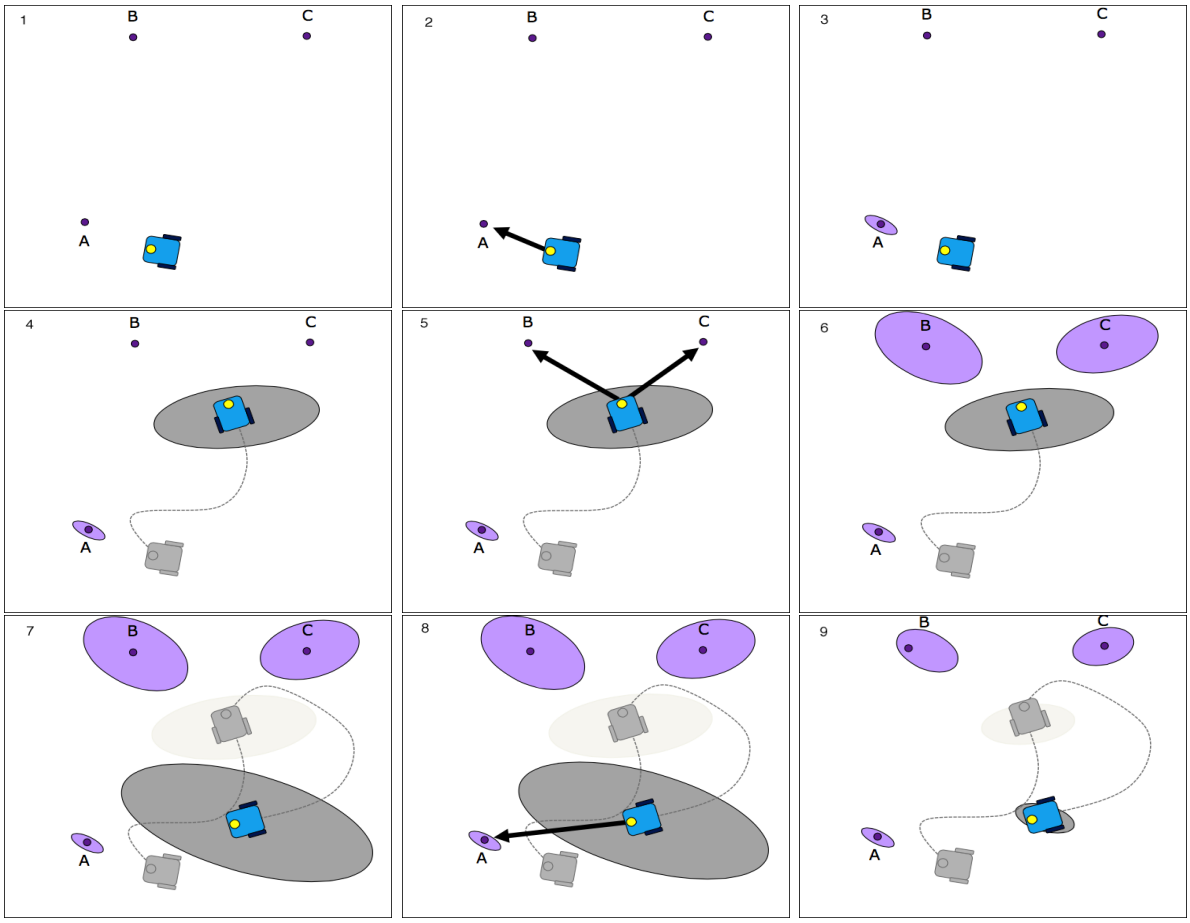


Figure 3: A basic representation of the SLAM procedure. Grey ellipses depict uncertainty regarding the robot's pose and purple ellipses depict uncertainty regarding the feature positions. Adapted from [**?**].

Table 1.1: Individual steps undertaken to solve the SLAM problem.

Image	Process
$1-3$	Robot begins exploration, observes feature A and updates the internal representation accordingly.
4	Robot moves, subsequently increasing uncertainty regarding its pose.
$5-6$	Robot observes features B and C and updates the internal representation. Note that the increase in pose uncertainty yields a greater uncertainty in feature position due to the relationship between the robot's pose and the feature estimates.
$7-8$	Robot moves, again increasing uncertainty regarding its pose before re-observing old feature A. This is referred to as *loop closure*.
9	Because the feature locations and robot's pose estimates are not statistically independent, the uncertainty regarding the pose as well as that of all feature positions decreases.

A feature can also be referred to as a landmark and the aforementioned terms will from hereon in be used synonymously.

1.2 Problem Description

A vision-based autonomous vehicle operating within an unknown and restricted environment requires a constant update regarding its *current location*. Many autonomous systems have limited knowledge regarding the surrounding environment but may possess a sensor capable of observing the environment - in this case a camera(s). It is therefore essential that a solution to this specific localisation problem incorporates the ability to use sensor measurements to build a map on the fly while concurrently locating itself within the map. In a restricted environment, it is likely that a robot will return to a previously observed region, making it essential to incorporate robust *repeatable* localisation where drift from ground truth can be corrected.

Vision-based SLAM implementations as depicted in Figure 3 provide the necessary functionality to achieve repeatable localisation. SLAM algorithms utilising single cameras [**?**, **?**, **?**] have provided elegant yet accurate solutions to the SLAM problem, reconstructing accurate SLAM maps and subsequently achieving localisation. One such algorithm presented by Davison et al. [**?**], termed *MonoSLAM*, allows for real-time repeatable localisation of a handheld camera moving within a restricted environment. MonoSLAM, along with the work succeeding it [**?**, **?**, **?**, **?**], has achieved successful results in retrieving the trajectory of a robot, forming a persistent SLAM map and ultimately maintaining repeatable localisation. Although a map of features is not the desired outcome, it remains essential to solving the localisation problem.

There are however, inherent disadvantages of a MonoSLAM system. Firstly, the utilisation of a single camera prohibits the system from immediately obtaining an accurate depth estimate. A feature is required to be observed from several different viewpoints before an accurate depth estimate can be made. Secondly, the motion model, namely a *constant linear and angular velocity* model, constrains the movement of the system to "smooth" trajectories - as depicted in Figure 4. If erratic forces or disturbances act upon the system, the pose of the robot is generally lost, and in most cases irrecoverable. Lastly, because there is no sufficient knowledge of the motion of the robot, the system is not well suited in certain practical scenarios.

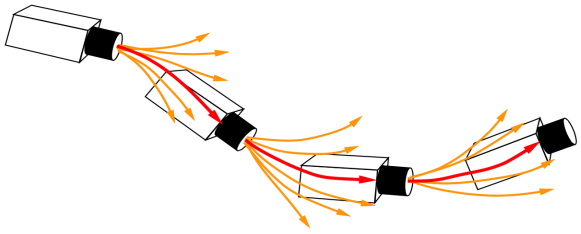


Figure 4: Visualisation of the smooth trajectories of the constant linear and angular velocity model. Adapted from [**?**]

Information obtained from additional sensors could potentially improve the disadvantages presented by MonoSLAM. Single camera SLAM implementations using information from an additional measurement sensor (e.g. a laser range finder), such as that presented in a study by Fu et al. [**?**], are not only too expensive but would be difficult to integrate with a single camera system from a modelling perspective. A more elegant solution that uses measurements to improve the motion estimates is required.

To extend the range of practical applications of MonoSLAM, a motion model capable of estimating the pose of a robot due to a variety of movements is essential. The current

constant linear and angular velocity model limits the robot to smooth movements and therefore needs to be adjusted or replaced. Using additional information can provide the current velocity motion model with more accurate state estimates, but cannot allow movements that aren't "smooth". Davison et. al [**?**], confirms that additional information such as angular velocities obtained from a *gyroscope* improve the state estimates, but cannot compensate for a change in motion dynamics.

A *kinematic estimator* directly measures the derivatives (first and second order) of the position and orientation of the robot as opposed to calculating them from the control inputs and the system's physical model. Inertial measurement units (IMU) can directly measure these derivates and typically comprise of an accelerometer and a gyroscope (and sometimes a magnetometer) that record the accelerations and angular velocities respectively. Incorporating such a subsystem could allow a wider variety of practical scenarios as the target system isn't constrained to a specific motion.

1.3 Project Objectives

This project seeks to utilise the aforementioned MonoSLAM system of Davison et al. and improve the localisation thereof by using *additional* sensor information. The improvement(s) of the system should allow the existing system to obtain better localisation and extend the range of applications upon which the system can be applied while maintaining the original performance standard: repeatable localisation at 30 Hz for approximately 100 features. The system should utilise a *single* camera as the measurement sensor and preferably support real-time operation. All processing regarding the SLAM algorithm can be done on a standard PC that communicates with the sensors via a serial port. The project budget is R 1500.00. The primary objectives of this project include:

- **Performing an overview on the current techniques used to realise SLAM**
  In order to understand how SLAM is implemented, an understanding of probability theory and state estimation is required. These concepts - specifically recursive state estimation and the Bayes filter - need to be researched and analysed before choosing a suitable technique to implement.

- **Analysis of the kinematic estimator as an alternative motion model**
  A kinematic estimator is suggested as a alternative motion model. The kinematic estimator needs to be researched, mathematically derived and simulated. The results from the simulation should correspond to the mathematical derivation. The advantages and disadvantages of the kinematic estimator also need to be investigated.

- **Hardware Design of the System**
  The kinematic estimator requires additional measurements obtained from an IMU. The IMU is required to interface with the PC via a micro-controller. The micro-controller allows precise synchronisation between the images sampled by the camera and the IMU measurements. A functional diagram of the system's hardware components are depicted in Figure 5.

- **Comparison between proposed and original MonoSLAM implementations**
  If time allows, a set of tests are required to be derived and implemented to establish whether the proposed improvement(s) indeed satisfy the desired requirements.

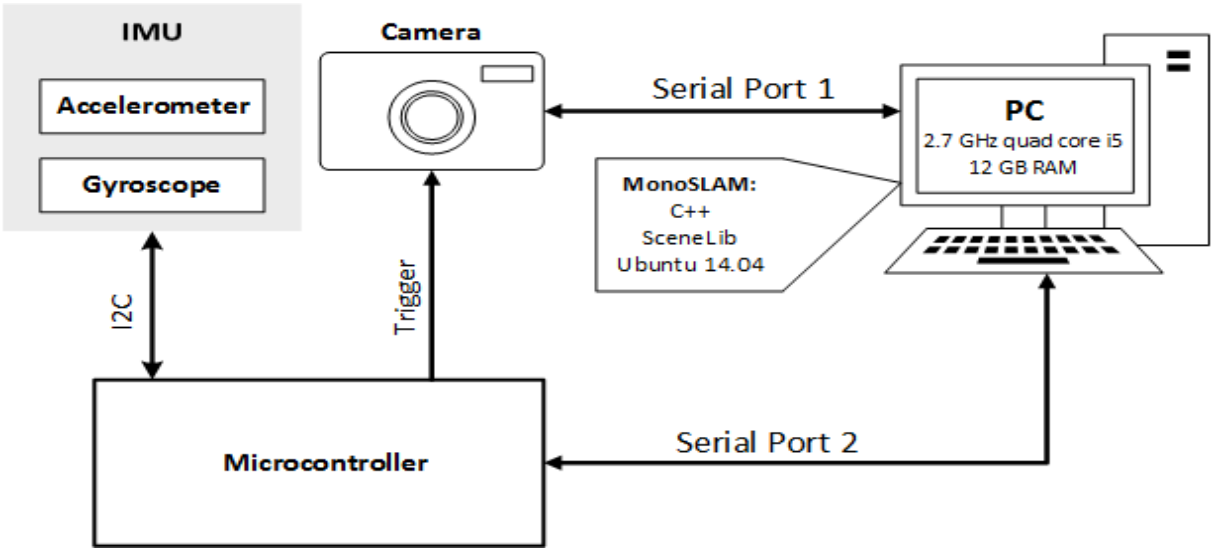


Figure 5: Diagram representing the interaction between the system's hardware and software components

1.4 Project Outline

The remainder of this report is presented as follows:

Chapter 2: Probabilistic State Estimation Techniques
This chapter presents all of the necessary research to effectively design and implement a solution to the SLAM problem. This chapter focusses on introducing the reader to the basic theory pertaining to probability theory and recursive state estimation. The chapter further explains how recursive state estimation can be implemented to provide a solution(s) to the SLAM problem.

Chapter 3: Design
This chapter provides a detailed overview of MonoSLAM. The relevant variable representation and functional components are initially discussed. The chapter then seeks to derive the state transition probability model, now comprised of a kinematic estimator. An overview of the extended Kalman filter's measurement update is also given.

Chapter 4: Implementation
This chapter sets out to predict the behaviour of the various subsystems through simulation. The results from the analysis of the kinematic estimator, extended Kalman filter and inertial measurement unit are all analysed accordingly.

Chapter 5: Testing and Results
This chapter presents all the relevant tests carried out to determine the proposals validity. The results of these tests are subsequently analysed.

Chapter 6: Conclusions and Recommendations
This chapter provides a summary recommendations by the author that is applicable to future work. The final conclusion is formulated analysing the state of the final system compared to the objectives of the project.

2 Theory: Probabilistic State Estimation Techniques

2.1 Introduction

The following chapter will provide a brief, yet concise introduction to the fundamental algorithms that are necessary to implement SLAM. The fundamental concepts, particularly the various techniques associated with the implementation thereof will be addressed. The goal of this section is to introduce the fundamental concepts as well as the mathematical and probabilistic principles that form the basis of state estimation in the robotics field of study.

Initially, the Bayes filter; that is the algorithm that forms the basis of all state estimation techniques presented in this report, will be introduced and formally discussed. Thereafter, the Gaussian Filter family - particularly the *Kalman Filter* (KF) - as well as its variants - are to be introduced, discussed and defined in terms of the context of this project. It is worthwhile to note that theory in this chapter resembles material from the book Probabilistic Robotics by Thrun et al. [**?**] and is adapted therefrom for convenience.

2.2 Recursive State Estimation

State variables define the mathematical state of a system's dynamics and describe the impact of the system's future behaviour in the absence of external factors. Although a robot's dynamics can be mathematically modelled, they have quantities that are not directly observable but can be obtained through sensor measurements. Sensors though, obtain limited data regarding certain quantities and most importantly, are corrupted by *noise*.

State estimation then, seeks to recover state variables using measured sensor data, control inputs to the system as well as the system model. The value obtained is referred to as a *state estimate*. In the case of SLAM, the state estimates of a robot incorporate the robot's pose as well the position of each landmark in the environment.

Recursive state estimation however, doesn't require the system to keep a complete history of *all* measurements and control inputs, using only the current control inputs and measurements to update the previous state estimate. Probabilistic state estimation algorithms - to be investigated in this section - compute *belief* distributions regarding state variables where the belief reflects a robot's internal knowledge of its state.

In the case of SLAM, the robot is required to know its location at each time instance t. The belief must thus be calculated at each time step. In order to achieve recursive state estimation, the state estimates of the robot should only incorporate the latest measurements. Additionally, the calculation of the belief should incorporate the previous estimates. Provided that the the system also obeys the Markov assumption - that previous data and future data are independent provided that the current state $\mathbf{x}_t$ is known - the *Bayes filter* provides such recursive state estimation.

2.3 Bayes filter

The previously mentioned belief of a robot can be represented by a probability distribution that assigns a probability to each state outcome. The belief distribution is a posterior probability over the state variable that is conditioned over the measurements and control data [**?**]. Mathematically the belief with regard to a state variable $\mathbf{x}_t$ is, as shown in the book Probabilistic Robotics by Thrun et al. [**?**]:

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t \mid \mathbf{z}_{1,..,t},\ \mathbf{u}_{1,..,t}). \tag{2.1}$$

This describes, for a given time t, a joint density of the robot state as well as the landmark locations given all of the previously recorded observations $z_{1,..,t}$ and control inputs $u_{1,..,t}$. Table 2.2 presents a pseudo-algorithm of the Bayes filter algorithm [**?**]:

Table 2.1: The Bayes filter Algorithm

Input:	previous belief $bel(\mathbf{x}_{t-1})$, control input(s) $\mathbf{u}_t$, measurement(s) $\mathbf{z}_t$
Output:	current belief $bel(\mathbf{x}_t)$
for all $\mathbf{x}_t$:	
1.	$\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t \mid \mathbf{u}_t,\ \mathbf{x}_{t-1}) bel(\mathbf{x}_{t-1}) d\mathbf{x}_{t-1}$
2.	$bel(\mathbf{x}_t) = \eta p(\mathbf{z}_t \mid \mathbf{x}_t) \overline{bel}(\mathbf{x}_t)$
3.	end for.

The recursive nature of the algorithm can thus be seen from Table 2.1; whereby the belief $\overline{bel}(\mathbf{x}_t)$ at the current time t is obtained through initially calculating the belief at the previous time step $t-1$. The Bayes filter contains two essential steps: *prediction* (line 1) and *measurement update* (line 2). The prediction step initially processes the control inputs before subsequently predicting the current belief based on the prior belief and the probability that a transition from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$ occurs. Thereafter, the measurement update improves the belief by adding information about the states, observed from new measurements.

The Bayes filter can be implemented in many different ways, two of which are discussed later in the report. Upon choosing a suitable implementation, a trade off between the following properties needs to be made:

- Computational efficiency
- Accuracy of the approximation
- Ease of Implementation

The Bayes filter provides a suitable solution through recursive state estimation. Additionally, the only approximation that is made upon deriving the Bayes filter is the Markov assumption. Along with its relatively simple implementation, the fact that no other approximations are made is why the Bayes filter is chosen as the solution to the SLAM problem in this project (and many others).

The mathematical derivation of the Bayes filter contains further technicalities. The techniques presented in this project require only a basic understanding of the Bayes filter. A detailed analysis of the Bayes filter can be obtained from the book Probabilistic Robotics by Thrun et al. [**?**].

2.4 Gaussian Filters

Amongst the many different implementations of the Bayes filter are the *Gaussian filter* family. The basic idea behind a Gaussian filter is that a belief can be represented as a multivariate Gaussian distribution, represented mathematically as follows [**?**]:

$$p(\mathbf{x}_t) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} \exp\left\{ -\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \boldsymbol{\mu}) \right\}, \tag{2.2}$$

where the density across the state variable $\mathbf{x}_t$ is characterised through two fundamental parameters: the mean $\boldsymbol{\mu}$ and the covariance $\boldsymbol{\Sigma}$. Such a parameterisation is known as the *moments parameterisation* (as the mean and covariance represent the first and second order moments respectively). This parameterisation allows a number of recursive filter algorithms to be derived, two of which are examined in this project: the *Kalman filter* (KF) and its non-linear counterpart, the *extended Kalman filter* (EKF). It is important to realise that both of the aforementioned filters belong to the same sub-class of filters - namely the KF Family - and therefore most of the fundamental concepts and functionality between them are identical. Each filter is discussed in further detail in the subsections that follow.

2.4.1 Kalman Filter

Probably the most fundamental of all Gaussian filter algorithms, is the *Kalman filter*. Introduced in a study presented by Kalman [**?**], the KF can be briefly described as an optimal estimator. It remains a popular technique for filtering and prediction of *linear* systems that contains Gaussian uncertainty. The KF seeks to describe a belief distribution of a state variable $\mathbf{x}_t$ as described in Equation 2.2. Subsequently, the state vector $\mathbf{x}_t$ is modelled by a single multivariate Gaussian distribution with a mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$, at each time instance t (while previous time steps are denoted as $t-1$, $t-2$, etc.). The general implementation as described above though, is only valid provided that the following three properties hold true - as listed in [**?**]:

1. The state transition model probability **must** be a *linear* function with additive Gaussian (process) noise.

$$g(\mathbf{u}_t, \mathbf{x}_{t-1}) : \mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{x}_t + \mathbf{w}_t. \tag{2.3}$$

2. The observation model probability **must** be a *linear* function with additive Gaussian (sensor) noise.

$$h(\overline{\boldsymbol{\mu}}_t) : \mathbf{z}_t = \mathbf{C}_t\mathbf{x}_t + \mathbf{v}_t, \tag{2.4}$$

   where $\mathbf{w}_t$ and $\mathbf{v}_t$ represent process and sensor noise respectively.

3. The initial belief $bel(\mathbf{x}_0)$ must be normally distributed.

The input to the KF is the belief $\overline{bel}(\mathbf{x}_t)$ at time $t-1$, represented by $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$. The KF requires a control input $\mathbf{u}_t$ and a measurement $\mathbf{z}_t$ at time t to update the belief. Like the Bayes filter, the KF too is executed in two (sequential) steps: the *control update step* and the *measurement update step*.

At time t, the prediction step aims to calculate a predicted belief $\overline{bel}(\mathbf{x}_t)$ represented by $\overline{\boldsymbol{\mu}}_t$ and $\overline{\boldsymbol{\Sigma}}_t$. The predicted belief is modelled by a deterministic version of the state

transition probability (Equation 2.3) that incorporates the control input $\mathbf{u}_t$ to the mean and covariance estimates. The update step aims to obtain the belief $bel(\mathbf{x}_t)$ from the predicted belief $\overline{bel}(\mathbf{x}_t)$ by incorporating the measurements $\mathbf{z}_t$. The KF computes a Kalman gain, that determines the influence of a measurement in the new state estimate. The update step of the incorporates a product of two Gaussian probabilities. This product results is a new Gaussian with a weighted *optimal* mean. The Kalman gain is thus optimal and is used to update the mean and covariance estimates such that the resulting belief distribution is optimal. An illustration of the weighted mean is depicted in Figure 6:

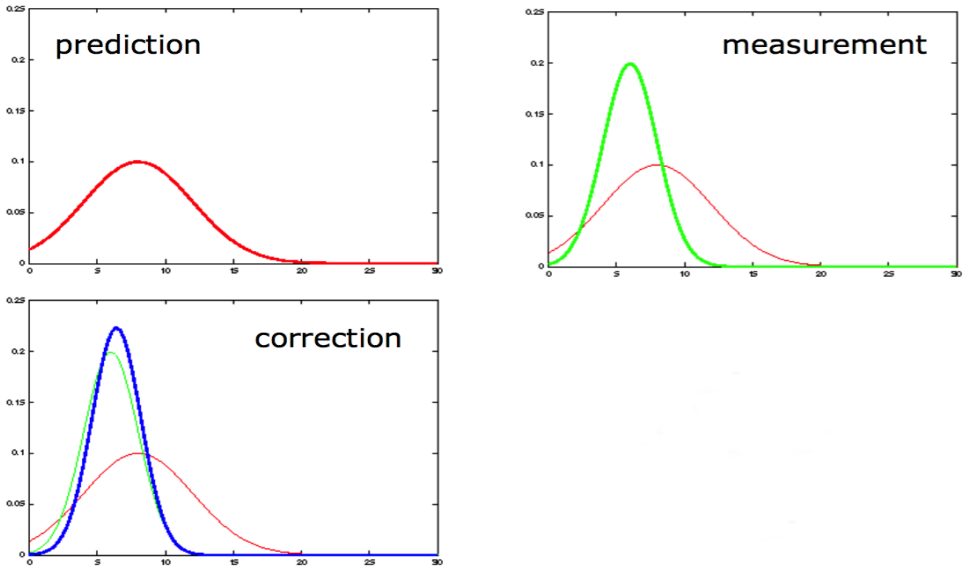


Figure 6: A graphical illustration of how the weighted mean is obtained. The correction (blue) is the product of the prediction (red) and measurement (green) Gaussian PDF's. Adapted from [**?**].

Each of the aforementioned steps are later discussed in more detail - with reference to implementations specific to this paper. Table 2.2 below, presents a pseudo-algorithm of the KF [**?**]:

Table 2.2: The KF Algorithm

Input:	previous mean $\boldsymbol{\mu}_{t-1}$ and covariance $\boldsymbol{\Sigma}_{t-1}$, control inputs $\mathbf{u}_t$, measurements $\mathbf{z}_t$
Output:	mean $\boldsymbol{\mu}_t$, covariance $\boldsymbol{\Sigma}_t$
	Control update step
1.	$\bar{\boldsymbol{\mu}}_t = g(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) = \mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t \boldsymbol{\mu}_t + \mathbf{w}_t$
2.	$\bar{\boldsymbol{\Sigma}}_t = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^T + \mathbf{R}_{w,t}$
	Measurement update step
3.	$\mathbf{K}_t = \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^T (\mathbf{C}_t \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^T + \mathbf{R}_{v,t})^{-1}$
4.	$\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t [\mathbf{z}_t - \mathbf{C}_t \bar{\boldsymbol{\mu}}_t]$
5.	$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \bar{\boldsymbol{\Sigma}}_t$

The KF is generally considered an efficient algorithm for sparse sets. The *computational complexity* of a matrix inversion is bounded by an order of $O(d^{2.8} + n^2)$ [**?**], where d represents the dimensions of the measurements vector $\mathbf{z}_t$ and n the state space.

2.4.2 Extended Kalman Filter

Considering that most practical systems of interest are non-linear, the KF in its purest form cannot be successfully implemented upon most modern day systems - including SLAM. Non-linear transformations of Gaussian random variables (RV) result in a different RV while any linear transformation of a Gaussian random variable yields another **different** Gaussian variable. This violates the important condition of the KF algorithm. This phenomenon can be illustrated through Figure 7:

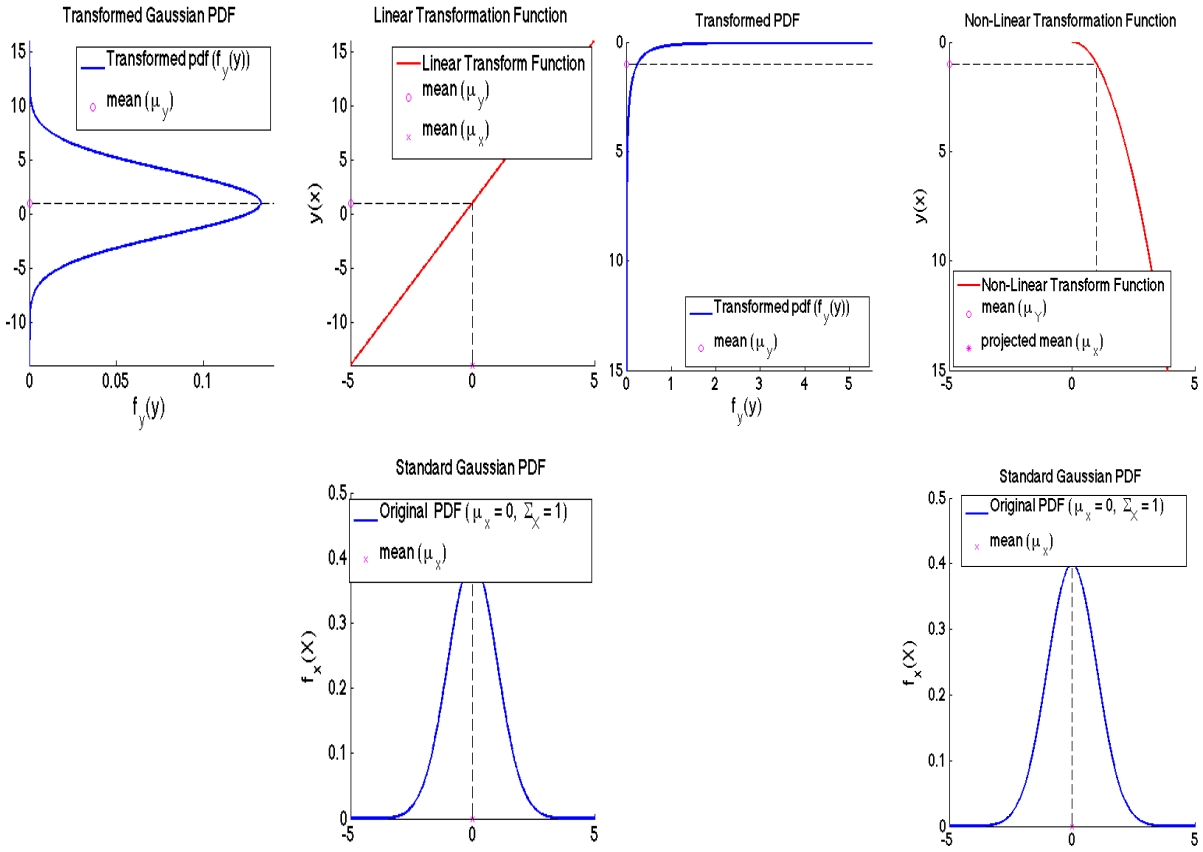


Figure 7: Left: Linear transformation of a Gaussian random variable. Right: Non-linear transformation of a Gaussian random variable.

The *extended Kalman filter (EKF)*, an extension of the general KF, aims to enable the modelling of non-linear systems through local linearisation. As previously mentioned, the state transition function $g(\mathbf{u}_t, \mathbf{x}_{t-1})$, as well as the observation model $h(\mathbf{x}_{t-1})$ of most practical systems are typically both non-linear in nature. These are typically only moderate non-linearities. Considering the aforementioned statement; it is necessary to determine a method for approximating a non-linear function as a linear function, more commonly referred to as linearisation. The linearisation process of EKF aims to linearise these functions so that the fundamental operations of the KF algorithm remain valid.

The linearisation process approximates an arbitrary non-linear function k by a linear function that is *tangent* to k at the mean value of the Gaussian, $\boldsymbol{\mu}$. If the Gaussian is then projected through this new linear approximation, the resultant transformation would yield a random variable that is Gaussian in nature (as in Figure 7). This technique is applied to both the state transition and observation functions. Many methods exist for linearisation of non-linear functions, but the EKF utilises the method of (first order)

Taylor expansion. The Taylor expansion creates a linear approximation of a non linear function, say k, by its own value as well as that of its gradient k'. The tangent of k can be depicted by its partial derivative with respect to the state vector $\mathbf{x}_{t-1}$:

$$k'(\mathbf{x}_{t-1}, \mathbf{u}_t) := \frac{\partial k(\mathbf{x}_{t-1}, \mathbf{u}_t)}{\partial \mathbf{x}_{t-1}}. \tag{2.5}$$

The argument of the function f is chosen as the most likely point at the linearisation instance. For Gaussians, the most likely point is the mean $\boldsymbol{\mu}_{t-1}$. The linear approximation of the function k can then be achieved through the linear extrapolation evaluated at its most likely point $\boldsymbol{\mu}_{t-1}$:

$$\begin{aligned} k(\mathbf{x}_{t-1}, \mathbf{u}_t) &\approx f(\mathbf{x}_{t-1}, \mathbf{u}_t) + k'(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \\ &= k(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{K}_t^{\mathbf{x}_{t-1}}(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \end{aligned}, \tag{2.6}$$

where $\mathbf{K}_t^{\mathbf{x}_{t-1}} = k'(\mathbf{u}_t, \mathbf{x}_{t-1})$ is the *Jacobian* matrix.

It is important to note that Jacobian matrix is determined at each linearisation instance (each individual time step) as its parameters differ from one linearisation instance to the next.

It is very important to note that because only a first order Taylor expansion is used to *approximate* the linearisation, severe non-linearities will prohibit acceptable approximations of the Gaussian distribution upon transformations. If the linearisation point is chosen at a point close to the mean, the EKF will yield an acceptable approximation from the linearisation process.

There are other variants of the KF that aren't discussed in this project, but have been carefully considered. It is assumed that the first order Taylor approximation provides a suitable approximation of the non-linearities that are expected in the system, namely the uncertainty regarding angle orientation errors. The *Unscented KF* is assumed to be a less appropriate choice of filter to the EKF considering the the lack of severe system non-linearities. The objective of this project seeks to provide localisation for a set of approximately 100 landmarks. The *Information Filter* will only provide a better computational complexity than the EKF if the number of landmarks are much larger. This analysis suggests that the EKF provides a suitable yet simple implementation of the Bayes filter.

Table 2.3 below, systematically and mathematically represents the steps associated with the EKF [**?**]:

Table 2.3: The EKF Algorithm

Input:	previous mean $\boldsymbol{\mu}_{t-1}$ and covariance $\boldsymbol{\Sigma}_{t-1}$, control inputs $\mathbf{u}_t$, measurements $\mathbf{z}_t$
Output:	mean $\boldsymbol{\mu}_t$, covariance $\boldsymbol{\Sigma}_t$
	Control update step
1.	$\bar{\boldsymbol{\mu}}_t = g(\mathbf{u}_t, \bar{\boldsymbol{\mu}}_{t-1})$
2.	$\bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}} = \mathbf{G}_t^{x_{t-1}} \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t-1}} \mathbf{G}_t^{x_{t-1}T} + \mathbf{R}_{w,t}$
	Measurement update step
3.	$\mathbf{K}_t = \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}} \mathbf{H}_t^{x_{t-1}T} (\mathbf{H}_t^{x_{t-1}} \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}} \mathbf{H}_t^{x_{t-1}T} + \mathbf{R}_{v,t})^{-1}$
4.	$\boldsymbol{\mu}_{\boldsymbol{t}} = \bar{\boldsymbol{\mu}} + \mathbf{K}_t[\mathbf{z}_t - h(\bar{\boldsymbol{\mu}})]$
5.	$\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t^{x_{t-1}T}) \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}}$

3 System Overview

3.1 Introduction

This chapter sets out to provide an overview of the MonoSLAM algorithm. Initially, a general overview of MonoSLAM is given, stating the problem, previous work done regarding this problem and the limitations imposed on this work. Thereafter, the adequate state representation will be shown along with the required modelling of the measurement sensors. Finally, a complete analysis of the *control update* and an overview of the *measurement update* will be given.

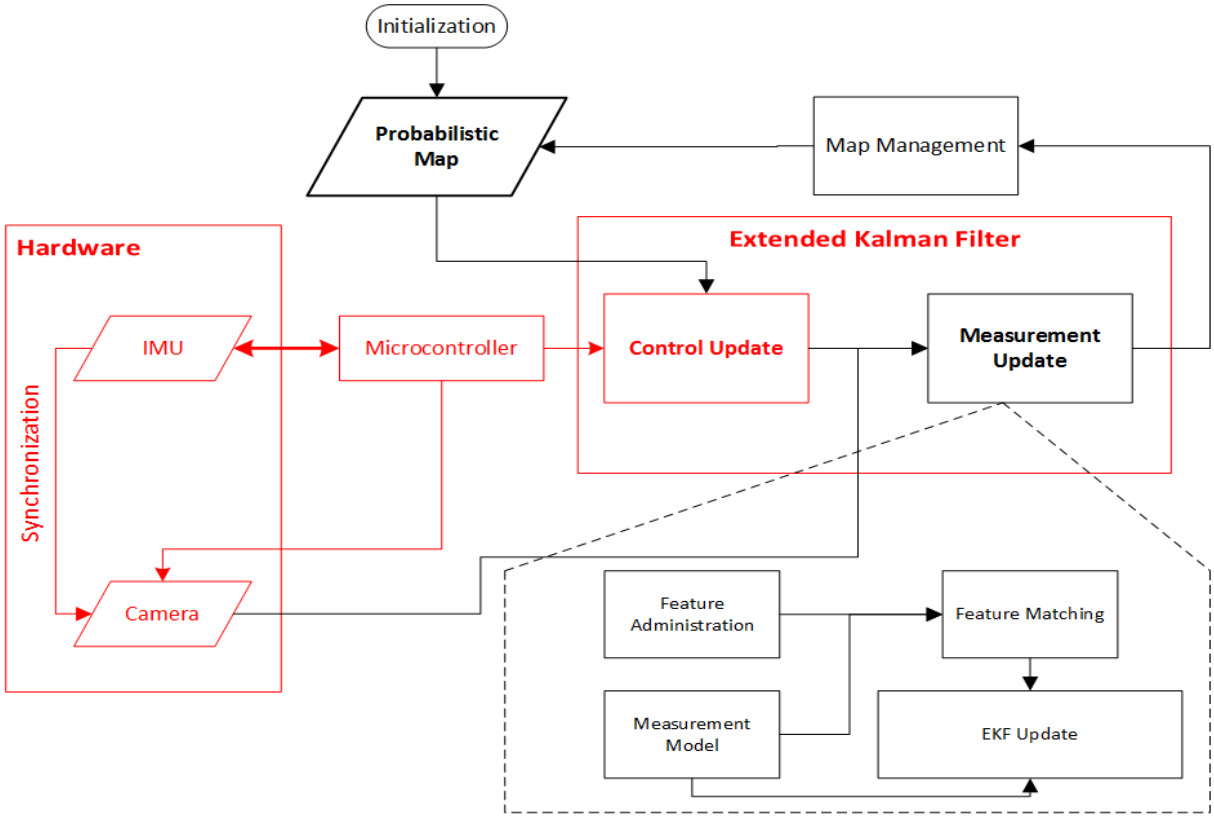


Figure 8: Functional diagram of the complete SLAM system. The sections in red depict the subsystems that have been accordingly altered or replaced in the implementation proposed in this project. The dashed lines show the internal processes of the measurement update - initially this is not trivial.

3.2 MonoSLAM

Originally presented in a study by Davison et al. [**?**], MonoSLAM provides suitable real-time and repeatable localisation using a single camera SLAM implementation. In MonoSLAM, the robot (typically a handheld camera) assumes a constant linear and angular velocity motion model, using the images from the camera to correct the state estimates. MonoSLAM is not the only successful implementation of single camera SLAM. Separate studies presented by Sola [**?**] and Klein [**?**] respectively show alternative solutions to the single camera SLAM problem. Davison et al. [**?**] however, provides the source code of MonoSLAM, a *modular* system, under the GNU version 3 license so that possible improvements, such as those presented in this project, can be practised. This modularity and

apparent ease of implementation as opposed to the particle filter based implementations presented by the alternative methods suggest MonoSLAM as an appropriate choice.

MonoSLAM differs from other SLAM implementations in that it has no observable control inputs or other odometry. The system itself (a handheld camera) has no prior information of the users movements. Instead, the users movements are statically modelled through a constant linear and angular velocity model that expects linear and angular accelerations of a Gaussian profile. Incorporating additional measurements of the motion of the robot doesn't violate the methodology of MonoSLAM. The system still contains no prior information regarding its movement, but rather incorporates a control input equivalent that measures the movement.

Figure 8 depicts a functional diagram of the proposed MonoSLAM system. The sections in red have been added/or altered from the original MonoSLAM implementation. This project seeks to utilise a kinematic estimator as an alternative to the current velocity and angular velocity motion model incorporated within the control update. The kinematic estimator will measure the angular velocities and linear accelerations of the robot from an inertial measurement unit as opposed to calculating them from physical models.

The remaining components of the proposed approach presented in this study are identical to that of the MonoSLAM implementation: that is, using an EKF where the measurement update of the EKF depends on the measurements from a single camera.

This particular vision based approach aims to use salient image *patches* as long term landmarks as presented in [**?**, **?**]. These aforementioned patches are typically 11 $\times$ 11 pixels and are obtained through the image detection operator of Shi and Tomasi [**?**] from raw monochrome (greyscale) images. The goal is to repeatedly re-identify these image template patches over time after considerable camera movements. Invariably, basic 2D template matching algorithms are of little use, considering that any particular movements of the camera (even minimal) can severely alter the shape of a saved template patch. As a result, MonoSLAM assumes that each patch lies on a locally planar surface and that the surface normal is parallel to the vector from the feature to the camera at the instance that it is initialised. Once the depth of a patch has been determined - this is done through a small particle filter - the patch is stored to be used as a long term landmark. MonoSLAM stores these patches to provide a template for matching against a newly obtained 2D image at a later stage. Because the patches are never updated and remain in memory, long term localisation is possible. The *innovation search* in Figure 8 uses the measurement model to obtain a search area around which the system expects an initialised feature should be.

With regard to the management of the probabilistic map, it remains essential to the SLAM algorithm that decisions regarding the identification and deletion of landmarks be accurate and efficient. MonoSLAM's map-maintenance criterion demands that 12 reliable "good" features be visible within the camera's field of view in order to maintain accurate localisation. A good feature implies that a feature remains visible within the camera's field of view. A new feature is initialised using the image operator of Shi and Tomasi [**?**] upon a box of pixels (80 $\times$ 60 pixels) placed within an image. This box position is chosen at random with the constraints that it shouldn't overlap with any existing features and that it remains in the camera's field of view. If a visible feature is unsuccessfully matched more than 50% of the time, the landmark is deleted. It must be stressed that the aforementioned methods regarding vision based MonoSLAM measurements and map-management are described and implemented in this project exactly as they are in [**?**].

3.3 State Representation

With reference to Chapter 2.2, state variables represent the mathematical "state" of system. In order to calculate a belief distribution, the system must possess a model to predict future states. This model previously introduced as the state transition probability is commonly referred to as the *motion model*. All relevant states are embedded within the state vector $\mathbf{x}_t$. The state vector is defined at each timestep and consists of the robot's *actual* pose and the *actual* landmark positions within the map.
Mathematically, the probabilistic map is typically represented as a state *estimate* that consists of a mean state vector $\boldsymbol{\mu}_t$ and a covariance matrix $\boldsymbol{\Sigma}_t$, represented at each time instance t. The mean state vector is a single column vector containing the means of the estimates of the robot as well as the landmark positions, and the covariance matrix is a square matrix containing the covariances of each state estimate with respect to every other state estimate. These quantities can be mathematically shown according to the book Probabilistic Robotics by Thrun et al. [**?**]:

$$\boldsymbol{\mu}_t = \begin{pmatrix} \hat{\mathbf{x}}_{v,t} \\ \hat{\mathbf{y}}_{1,t} \\ \hat{\mathbf{y}}_{2,t} \\ \vdots \\ \hat{\mathbf{y}}_{n,t} \end{pmatrix}, \quad \boldsymbol{\Sigma}_t = \begin{pmatrix} \Sigma_{x,x} & \Sigma_{x,y_1} & \Sigma_{x,y_2} & \cdots & \Sigma_{x,y_N} \\ \Sigma_{y_1,x} & \Sigma_{y_1,y_1} & \Sigma_{y_1,y_2} & \cdots & \Sigma_{y_1,y_N} \\ \Sigma_{y_2,x} & \Sigma_{y_2,y_1} & \Sigma_{y_2,y_2} & \cdots & \Sigma_{y_2,y_N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Sigma_{y_n,x} & \Sigma_{y_n,y_1} & \Sigma_{y_n,y_2} & \cdots & \Sigma_{y_n,y_N} \end{pmatrix}. \tag{3.1}$$

These quantities then allow us to approximate the uncertainty regarding the generated feature map as a N-dimensional single multi-variate Gaussian distribution, where N, as stated above, is the total number of state estimates within the state vector and n is the total number of landmarks within the map.

3.3.1 Position State Representation

The camera position state $\mathbf{x}_v$ represents all relevant information regarding the camera's position and orientation in a 3D space. The position state vector is comprised of the 3D position vector $\mathbf{r}^W$, the unit orientation *quaternion* $\mathbf{q}^{WC}$ and the linear velocity vector, $\mathbf{v}^W$ representing the first derivatives of the position vector. The state camera vector - comprising 10 individual states - is described as presented in a study by Davison et al. [**?**]:

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WC} \\ \mathbf{v}^W \end{pmatrix}, \tag{3.2}$$

where $\mathbf{r}^W = (x \ y \ z)^T$ indicates the 3D cartesian position of the camera, $\mathbf{q}^{WC}$ the unit orientation *quaternion* indicating the camera orientation (represented in the body frame C) relative to the inertial reference frame W while $\mathbf{v}^W$ indicates the *linear* velocities of the camera relative to the inertial reference frame W. The reference frames are depicted in Figure 9. A quaternion, as previously mentioned, represents the camera's orientation. Quaternions are chosen as opposed to Euler angles to prevent the scenario where one degree of freedom is lost due to two of axes aligning. This scenario is more commonly referred to as a *gimbal lock* and can cause a body to rotate unexpectedly between time steps. It should be noted that all quaternions represented in this paper are unit quaternions. This implies that the square root of the sum of all the squared elements is always

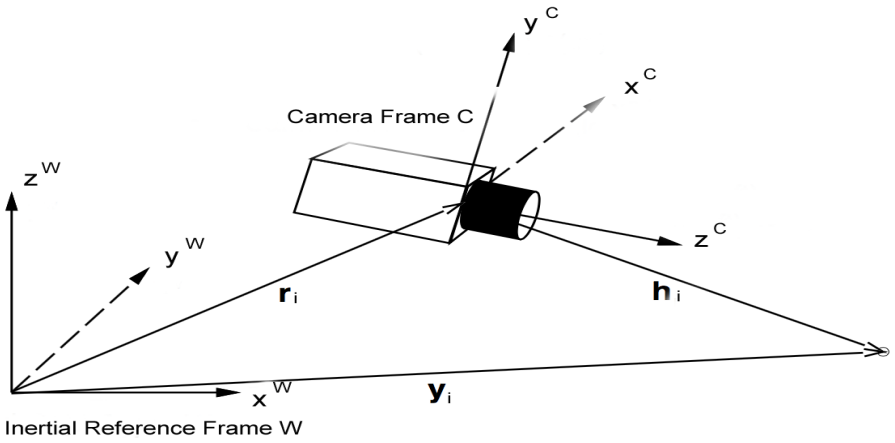


Figure 9: Graphical representation of the appropriate reference frames. Adapted from [**?**]

equal to 1:

$$q_{0,t}^2 + q_{1,t}^2 + q_{2,t}^2 + q_{3,t}^2 = 1. \tag{3.3}$$

When a robot's orientation changes, the process of computing the quaternions that causes this orientation change involves obtaining an angle-axis as well as a magnitude by which this axis is to be rotated. This process is described later in this chapter with reference to the state transition model probability.

Often, the modelling of dynamic systems require that additional parameters - apart from those describing the position and orientation of the robot - be included in the state vector. This is illustrated in Equation 3.2, where the linear velocity vector, $\mathbf{v}^W$, forms the additional information required for system modelling. This is due to the equivalent control input, that is of such a nature that an intermediary state (namely the linear velocity) is required to describe the control input's effect on the actual position.

3.3.2 Feature Representation

The probabilistic map contains a 3D position of *each* observed landmark as well as a corresponding uncertainty. The feature estimates $\mathbf{y}_i$ - consists of n landmarks - are described through three individual coordinates - x, y and z respectively:

$$\mathbf{y}_n = (x_n \;\; y_n \;\; z_n)^T, \tag{3.4}$$

where the subscript n denotes a specific landmark.

3.3.3 Sensor Model: Camera

The sensor of the the robot is a single CMOS digital camera. The *pinhole camera model* depicted in Figure 10 is used to provide a reasonable approximation for a projection of a 3D point in space. The pinhole model however, ignores the effects of distortions caused by lenses and finite apertures. The model can however be adapted to account for these assumptions and better approximate the distortion.

The pinhole camera model contains a two-dimensional plane, with the projections of the 3D point $(x_i \ y_i \ z_i)^T$. The process of representing a 3D coordinate in terms of a 2D coordinate $(u_i \ v_i)^T$ is known as *perspective projection*. The 2D plane is commonly referred to as the *pinhole plane* which contains a infinitesimal hole at its centre - the *pinhole*. The camera possess its own 3D coordinate system with coordinate axes X_C, Y_C and Z_C (also referred to as the camera's *optical axis*). The pinhole is situated at the origin of this 3D coordinate system - this is also referred to as the *optical centre*. The *image plane* is located at a positive distance f from the optical centre O, parallel to the pinhole plane. This distance f is referred to as the camera's *focal length*.

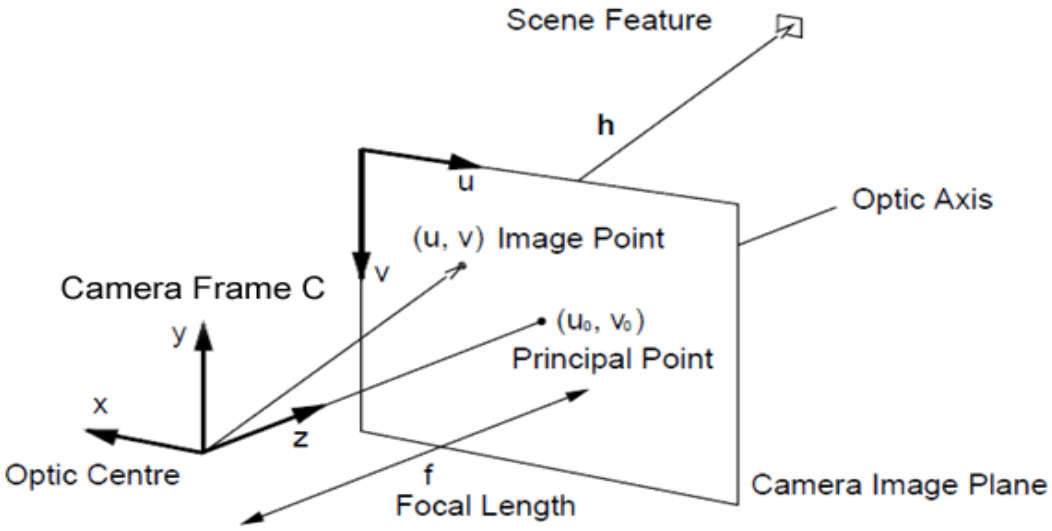


Figure 10: Graphical representation of the pinhole camera model. The Figure shows the projection of a scene feature onto the camera image plain with respect to the camera's optic centre. The camera reference frame C is defined at the optic centre of the camera. Adapted from [**?**].

The image of the camera represents a projection of a 3D point in space. Euclidean geometry requires complicated calculations to describe these projections. *Projective geometry* offers an alternative representations of the geometry that is both simpler to implement. Projective geometry uses a system of coordinates called *homogenous coordinates* and is able to represent transformations in a single matrix. The projection of an object r is defined as homogenous if the same object is represented by $r = \lambda r$. The relationship between homogenous and Euclidean coordinates is shown according to a lecture presented by Stachnis [**?**]:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} wx \\ wy \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.5}$$

where x and y are the Euclidean coordinates.

The undistorted projection of a point in the image plane (u_i,v_i) is be derived using similar triangles and homogenous coordinates as presented in a thesis by Albrecht [**?**]:

$$\begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{u_i}{f} \\ \frac{v_i}{f} \\ 1 \end{pmatrix} = \mathbf{C} \begin{pmatrix} \frac{u_i}{f} \\ \frac{v_i}{f} \\ 1 \end{pmatrix} \mathbf{C} = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.6}$$

where $\mathbf{C}$ represents the camera's *intrinsic matrix*, u_0 and v_0 represents the *principal point* located at the image centre.

The camera's intrinsic matrix describes the parameters that are unique to a camera containing the principal point as well as the focal lengths. Once obtained, this matrix can be used to obtain the projections of any 3D point in an image plane, as well as an 3D approximation of a projected point. As previously mentioned, the pinhole camera model ignores the effects of distortion that generally present in practical systems. To account for the errors that such an approximation may yield, it is proposed that the *projected* coordinates (u_i, v_i) be warped with a *radial distortion* as is done in [**?**], in order to obtain a coordinate (u_d, v_d), where the distortion has been accounted for. This radial distortion is mathematically shown as in a study presented by Swaminathan and Kayer [**?**]:

$$\begin{aligned} u_d - u_0 &= \frac{u - u_0}{\sqrt{1 + 2K_1 r^2}} \\ v_d - v_0 &= \frac{v - v_0}{\sqrt{1 + 2K_1 r^2}}, \\ r &= \sqrt{(u - u_0)^2 + (v - v_0)^2}, \end{aligned} \tag{3.7}$$

where K_1 and r represent the radial distortion parameters.

3.3.4 Control Input Equivalent

The following concept describes the dynamics to the system as a result of external influences. Upon considering the original MonoSLAM approach, it is evident that there are no *observable* control inputs. The approach presented in this paper though, aims to use measurements obtained through an IMU as an equivalent control input in order to derive a motion model that is potentially more accurate than the implementation of Davison et al. [**?**].

In most instances of robotics, it is essential to describe the dynamics involving a robot's movement. In the context of this paper, the robot (camera) is free to move as it pleases. In the approach presented by Davison et al., a constant velocity model is assumed and at each time step, unknown linear and angular acceleration zero-mean, Gaussian processes are introduced that cause linear and angular velocity impulses. The model contains very little, if any information about the movement of the camera. It can be assumed that utilising additional information regarding the camera's movement will provide greater accuracy upon state estimation.

The IMU is comprised of inertial sensors, namely the accelerometer and gyroscope, that are mounted onto the camera. This allows the camera to be modelled as a rigid body upon which a kinematic estimation can be applied. The IMU directly measures the total acceleration $\mathbf{f}_t$ as well as the total angular velocities $\boldsymbol{\psi}_t$ with respect to the camera's rigid body frame C.

The measured acceleration and gyroscope measurements form the *equivalent control vector* $\mathbf{u}_t$ that describes, at each time step, the external forces exerted on the system with noise. The control vector equivalent is adapted as follows:

$$\mathbf{u}_t = \begin{pmatrix} \mathbf{f}_t \\ \boldsymbol{\psi}_t \end{pmatrix} = \begin{pmatrix} f_{x,t} & f_{y,t} & f_{z,t} & \psi_{x,t} & \psi_{y,t} & \psi_{z,t} \end{pmatrix}^T. \tag{3.8}$$

Because the IMU measures the actual movements through sensors, namely an accelerometer and a gyroscope, the process noise is embedded within these measurements. Moreover, the uncertainty regarding the state transition model probability, namely the process noise, is all incorporated within the noise measurements of the IMU. This noise can be modelled as a zero mean, Gaussian RV's $\mathbf{w}_t$ with a corresponding covariance matrix $\mathbf{R}_w$. Because changes in orientation incorporates additional uncertainty to the system, this covariance matrix is required to be updated at each time step. The system noise can be then be mathematically described as follows:

$$\mathbf{w}_t = \begin{pmatrix} \mathbf{n}_{\mathbf{a},t} \\ \mathbf{n}_{\omega,t} \end{pmatrix} = \begin{pmatrix} n_{\ddot{x}_t} & n_{\ddot{y}_t} & n_{\ddot{z}_t} & n_{\omega_{x,t}} & n_{\omega_{y,t}} & n_{\omega_{z,t}} \end{pmatrix}^T, \tag{3.9}$$

with the aforementioned noise model is assumed to be a Gaussian random variable for each of the above elements.

Furthermore the covariance matrix corresponding to the process noise is defined as follows:

$$\mathbf{R}_w = \begin{pmatrix} n_{a_x,t} & 0 & 0 & 0 & 0 & 0 \\ 0 & n_{a_y,t} & 0 & 0 & 0 & 0 \\ 0 & 0 & n_{a_z,t} & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{\psi_x,t} & 0 & 0 \\ 0 & 0 & 0 & 0 & n_{\psi_y,t} & 0 \\ 0 & 0 & 0 & 0 & 0 & n_{\psi_z,t} \end{pmatrix}^2 . \tag{3.10}$$

The Kinematic estimator however, requires that the linear portion of the total acceleration $\mathbf{f}_t$ be obtained from the IMU measurement. It is known that the total acceleration measured by the IMU's accelerometer is adapted from a study presented by Servent et. al [**?**]:

$$\mathbf{f}_t = \mathbf{R}^{CW}(\mathbf{a}_t^C + \mathbf{g}), \tag{3.11}$$

with $\mathbf{a}_t$ is the linear acceleration vector with respect to the cameras reference frame C, $\mathbf{g}$ is the gravity vector and $\mathbf{R}^{CW}$ is the rotation matrix that transforms the camera's body coordinate frame C, into the inertial reference frame W.
The rotation matrix is defined according to a tutorial presented by Davison [**?**]:

$$\mathbf{R}^{CW} = \begin{pmatrix} q_{0,t}^2 + q_{1,t}^2 - q_{2,t}^2 - q_{3,t}^2 & 2(q_{1,t}q_{2,t} - q_{0,t}q_{3,t}) & 2(q_{1,t}q_{3,t} - q_{0,t}q_{2,t}) \\ 2(q_{1,t}q_{2,t} + q_{0,t}q_{3,t}) & q_{0,t}^2 - q_{1,t}^2 - q_{2,t}^2 - q_{3,t}^2 & 2(q_{2,t}q_{3,t} - q_{0,t}q_{1,t}) \\ 2(q_{1,t}q_{3,t} - q_{0,t}q_{2,t}) & 2(q_{2,t}q_{3,t} + q_{0,t}q_{1,t}) & q_{0,t}^2 - q_{1,t}^2 - q_{2,t}^2 + q_{3,t}^2 \end{pmatrix}. \tag{3.12}$$

The linear acceleration is a processed measurement. Rearranging Equation 3.11:

$$\mathbf{a}_t = \mathbf{R}^{WC}\mathbf{f}_t - \mathbf{g}, \tag{3.13}$$

where $\mathbf{R}^{WC}$ is the inverse of the rotation matrix $\mathbf{R}^{CW}$ and the inverse of a rotation matrix is its transpose:

$$\mathbf{R}^{WC} = (\mathbf{R}^{CW})^{-1} = (\mathbf{R}^{CW})^T. \tag{3.14}$$

The rotation in Equation 3.13 presents an uncertainty that cannot be modelled as the additive noise defined in Equation 3.9. Incorporating this uncertainty is discussed in Chapter 3.4.3.

3.4 Control Update Step

With reference to the probabilistic form of the solution to the SLAM problem, the control update step requires a description in terms of a belief distribution. The description of the aforementioned belief, in terms of the probability distribution on the state transitions, is described in step 1 of Table 2.1 and repeated for convenience :

$$\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t \mid \mathbf{u}_t,\ \mathbf{x}_{t-1}) bel(\mathbf{x}_{t-1})\ d\mathbf{x}_{t-1}, \tag{3.15}$$

The state transition model however, assumes the form of a Markov process, yielding that the current state $\mathbf{x}_t$ is only dependent upon the state immediately preceding it - $\mathbf{x}_{t-1}$ - as well as the input control $\mathbf{u}_t$. Additionally, the uncertainty regarding the state transition model is independent of the uncertainty regarding both the observation model. This allows an implementation of the Bayes filter to be used, more specifically the EKF. This section presents the necessary steps to incorporate the control update of the EKF.

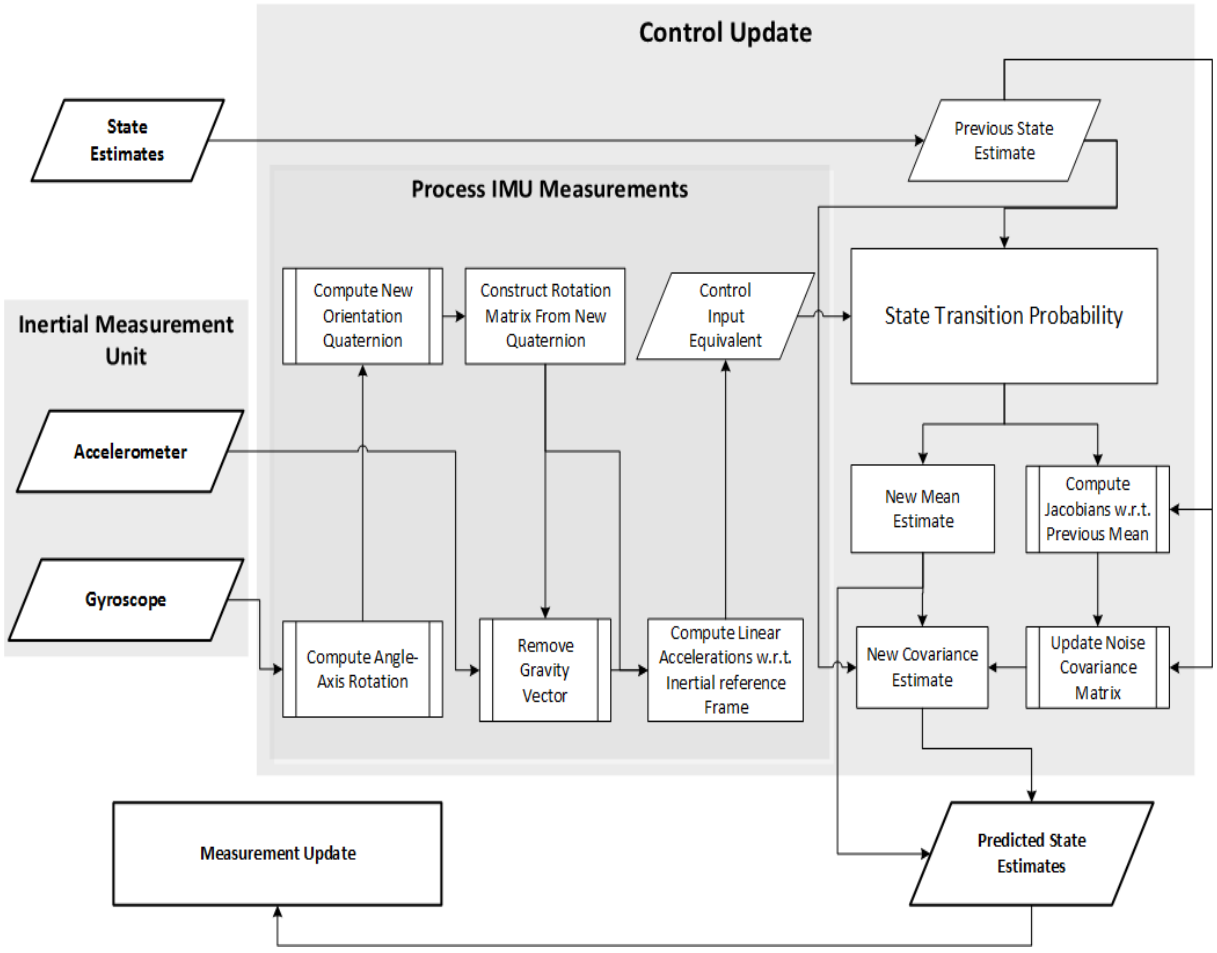


Figure 11: A detailed functional diagram of the control update step of the EKF. The greyed blocks depict functional subsystems

3.4.1 State Transition Probability Model

The description of the aforementioned state transition probability model can then, in terms of the probability distribution on the state transitions, take the following form:

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) = \frac{1}{\sqrt{|2\pi \mathbf{R}_w|}} \exp\left\{ \frac{1}{2} \left[\mathbf{x}_t - g(\mathbf{u}_t,\ \boldsymbol{\mu}_{t-1}) - \mathbf{G}_t^{\mathbf{x}_{t-1}} (\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \right]^T \right. \\ \left. \mathbf{R}_w^{-1} \left[\mathbf{x}_t - g(\mathbf{u}_t,\ \boldsymbol{\mu}_{t-1}) - \mathbf{G}_t^{\mathbf{x}_{t-1}} (\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \right] \right\} , \quad (3.16)$$

where $\mathbf{G}_t^{\mathbf{x}_{t-1}}$ represents the Jacobian of the state transition motion and $\mathbf{R}_w$ is the process noise.

With reference to Chapter 2.4.2, the EKF requires a state transition probability model in order to obtain the mean estimate $\bar{\boldsymbol{\mu}}_t$, of current state of the system. In short, the state transition probability model describes the transition from the previous state to the following state with regard to the robots kinematic motion as well as the control inputs. In order to derive the state transition model for the system at hand, it is vital that certain characteristics of the system be understood. Firstly, the robot system - from here on in to be referred to as the **camera** - is comprised of a monocular camera and an attached IMU package. Secondly, the camera is to be considered as a six degree of freedom (DOF) rigid body. Briefly the six DOF describe the camera's three *translational* and three *rotational* degrees of freedom.

We therefore set out to define a kinematic estimation based state transition probability model - using Newton's laws of motion - to describe the camera's movement through the environment as a result of initially unknown, external inputs to the system. Lastly, embedded within the motion model should be the impacts of uncertainty through both internal and external factors (noise).

A derivation for the suitable motion model will be obtained before a first order Taylor approximation is obtained as required by the EKF. recall the states and control inputs (equivalents):

$$\begin{aligned} \mathbf{x}_t &= \begin{pmatrix} \mathbf{r}_t^W & \mathbf{q}_t^{WC} & \mathbf{v}_t^W \end{pmatrix}^T \\ \mathbf{u}_t &= \begin{pmatrix} \mathbf{f}_t^C & \boldsymbol{\psi}_t^C \end{pmatrix}^T \end{aligned} . \quad (3.17)$$

Considering that the EKF is a recursive, numerical evaluation, it is necessary to convert the *continuous* linear differential equations that describe the state transition model into a discrete counterpart. Various methods of discretisation exist, though this specific implementation makes use of the forward difference (Eulers) method as the sampling period is assumed to be small enough. This method *approximates* the derivative for a state for a sampling period ΔT as follows:

$$\begin{aligned} \dot{\mathbf{x}}_{t-1} &= \lim_{\Delta T \to 0} \frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\Delta T} \\ \mathbf{x}_t &\approx \dot{\mathbf{x}}_{t-1} \Delta T + \mathbf{x}_{t-1} \end{aligned} . \quad (3.18)$$

Newton's second law of motion, describing the relationship between a body's mass and

its acceleration, is used to derive the linear motion model after which the aforementioned method of discretisation is applied to obtain the discrete motion model. A full derivation of the linear state transition model is shown in Appendix **??**.

The non-linear state transition function of the kinematic estimator $\mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1})$ is defined at the current time t, is dependent on both the current control input equivalents $\mathbf{u}_t$ as well as the actual states $\mathbf{x}_{t-1}$. The non-linear behaviour is a result of the change in orientation of the camera. The state transition function is defined and adapted from a tutorial presented by Davison [**?**]:

$$\begin{aligned}\mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1}) = \begin{pmatrix} \mathbf{r}_t^W \\ \mathbf{q}_t^{WC} \\ \mathbf{v}_t^W \end{pmatrix} &= \begin{pmatrix} \mathbf{r}_t^W + \mathbf{v}_{t-1}^W \Delta T \\ \mathbf{q}_{t-1}^{WC} \otimes \text{quat}\big(\boldsymbol{\omega}_t^C \Delta T\big) \\ \mathbf{v}_t^W + \mathbf{a}_{t-1}^W \Delta T \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{r}_{t-1}^W + \mathbf{v}_{t-1}^W \Delta T \\ \mathbf{q}_{t-1}^{WC} \otimes \text{quat}\big(\boldsymbol{\omega}_t^C \Delta T\big) \\ \mathbf{v}_t^W + \mathbf{R}_{t-1}^{CW}\big(\mathbf{a}_{t-1}^C \Delta T\big) \end{pmatrix}\end{aligned}, \tag{3.19}$$

where $\text{quat}\big(\boldsymbol{\omega}_t^C \Delta T\big)$ denotes the process of obtaining the quaternions of the rotation $\boldsymbol{\omega}_t^C \Delta T$ and ΔT is defined as the sample period between the previous time $t-1$ and t.

The angular velocities $\boldsymbol{\omega}_t^C$ and the linear accelerations $\mathbf{a}_{t-1}^W$ are measured from the control input equivalent $\mathbf{u}_t$ and the position $\mathbf{r}_{t-1}^W$, linear velocity $\mathbf{v}_{t-1}^W$ and the orientation quaternion $\mathbf{q}_{t-1}^{WC}$ - and subsequently the rotation matrix $\mathbf{R}_{t-1}^{CW}$ - are obtained from the previous state vector.

In order to compute the quaternions, the rate at which the camera's rotational degrees of freedom are changing is required. The control input equivalent, measure exactly this quantity from the gyroscope - the angular velocity. The angular velocity is subsequently numerically integrated in order to obtain the angular position $\boldsymbol{\theta}_t$, before the quaternion is taken thereof. As previously mentioned, an angle-axis as well as a magnitude by which this axis is to be rotated is required to compute a quaternion. The angle-axis $\boldsymbol{\gamma}_t$ is defined at a particular time instance t according to a tutorial presented by Davidson [**?**]:

$$\boldsymbol{\gamma}_t = \Big(\boldsymbol{\theta}_t, \|\boldsymbol{\theta}_t\|\Big) = \left(\begin{pmatrix} \theta_{x,t} \\ \theta_{y,t} \\ \theta_{z,t} \end{pmatrix}, \big\|\boldsymbol{\omega}_t^C \Delta T\big\|\right) = \left(\begin{pmatrix} \frac{\omega_{t,X}^C \Delta T}{\|\boldsymbol{\omega}_t^C \Delta T\|} \\ \frac{\omega_{t,Y}^C \Delta T}{\|\boldsymbol{\omega}_t^C \Delta T\|} \\ \frac{\omega_{t,Z}^C \Delta T}{\|\boldsymbol{\omega}_t^C \Delta T\|} \end{pmatrix}, \big\|\boldsymbol{\omega}_t^C \Delta T\big\|\right), \tag{3.20}$$

where $\omega_{t,\beta}$, $\beta \in \{X, Y, Z\}$ denotes the angular velocity about each respective coordinate axis. The result in Equation 3.20 is then represented as a unit quaternion denoting the same rotation [**?**]:

$$\mathbf{q} = \begin{pmatrix} \cos\frac{\alpha}{2} & \frac{\theta_x}{\|\boldsymbol{\theta}\|}\sin\frac{\alpha}{2} & \frac{\theta_y}{\|\boldsymbol{\theta}\|}\sin\frac{\alpha}{2} & \frac{\theta_z}{\|\boldsymbol{\theta}\|}\sin\frac{\alpha}{2} \end{pmatrix}^T. \tag{3.21}$$

A quaternion multiplication between the quaternion $\mathbf{q}_{t-1}^{WC}$ and the angle-axis rotation quaternion in Equation 3.21 represents the product of the two rotation matrices each quaternion represents. This quaternion multiplication is denoted by the $\otimes$ operator in

equation 3.19 is formally defined according to a tutorial presented by Davidson [**?**]:

$$\begin{aligned}\mathbf{q}_t^{WC} &= \mathbf{q}_{t-1}^{WC} \otimes \text{quat}(\boldsymbol{\omega}_t^C \Delta T) \\ &= \mathbf{q}_1 \times \mathbf{q}_2 \\ &= \begin{pmatrix} q_{1,0}q_{2,0} - (q_{1,2}q_{2,1} + q_{1,2}q_{2,2} + q_{1,3}q_{2,3}) \\ q_{1,0}\begin{pmatrix} q_{2,1} \\ q_{2,2} \\ q_{2,3}q_{2,0} \end{pmatrix} + \begin{pmatrix} q_{1,1} \\ q_{1,2} \\ q_{1,3} \end{pmatrix} + \begin{pmatrix} q_{1,2}q_{2,3} - q_{2,2}q_{1,3} \\ q_{1,3}q_{2,1} - q_{2,3}q_{1,1} \\ q_{1,1}q_{2,2} - q_{2,1}q_{1,2} \end{pmatrix} \end{pmatrix}. \end{aligned} \tag{3.22}$$

This completes the steps necessary to implement the state transition probability model.

3.4.2 Mean Estimate

The control update uses the previously defined state transition probability function to obtain a state estimate $\bar{\boldsymbol{\mu}}_t$ at the current time instance. This procedure is shown in Step 1 of Table 2.3. Recalling Equation 3.19, the state transition function is dependent on both the current equivalent control inputs as well as the actual states to obtain a suitable state estimate:

$$\mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) = \begin{pmatrix} \mathbf{r}_t^W + \mathbf{v}_{t-1}^W \Delta T \\ \mathbf{q}_{t-1}^{WC} \otimes \text{quat}\big(\boldsymbol{\omega}_t^C \Delta T\big) \\ \mathbf{v}_t^W + \mathbf{R}_{t-1}^{CW}\big(\mathbf{v}_{t-1}^C \Delta T\big) \end{pmatrix}. \tag{3.23}$$

It is worthwhile to note that the equivalent control inputs (which are the measurements from the IMU) incorporates zero-mean process noise but is modelled accordingly in the covariance update in Chapter 3.4.3.

3.4.3 Covariance Update

Before the control update can be concluded, the covariance matrix $\bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}}$, corresponding to the previously determined mean vector $\bar{\boldsymbol{\mu}}_{\boldsymbol{t}}$ is required to be updated as a result of the linearisation process undertaken by the EKF. This procedure is denoted in Step 2 of Table 2.3. It can be noticed that the previously described Jacobian matrix of $\mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1})$ evaluated at the predicted mean $\boldsymbol{\mu}_{t-1}$ is thus required to realise this procedure. The Jacobian matrix $\mathbf{G}_t^{\mathbf{x}_{t-1}}$ can be mathematically defined using Equation 2.5:

$$\mathbf{G}_t^{\mathbf{x}_{t-1}} = \frac{\partial \mathbf{g}(\mathbf{u}_t, \mathbf{x}_{t-1})}{\partial \mathbf{x}_{t-1}}\bigg|_{\mathbf{x}_{t-1}=\boldsymbol{\mu}_{t-1}} = \begin{pmatrix} \frac{\partial \mathbf{r}_t^W}{\partial \mathbf{r}_{t-1}^W} & \mathbf{0} & \frac{\partial \mathbf{r}_t^W}{\partial \mathbf{v}_{t-1}^W} \\ \mathbf{0} & \frac{\partial \mathbf{q}_t^{WC}}{\partial \mathbf{q}_{t-1}^{WC}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{v}_t^W}{\partial \mathbf{v}_{t-1}^W} \end{pmatrix}, \tag{3.24}$$

with the non-zero elements of the Jacobian further trivially defined as follows (and according to the model defined in Equation 3.19),

$$\frac{\partial \mathbf{r}_t^W}{\partial \mathbf{r}_{t-1}^W} = \frac{\partial \mathbf{v}_t^W}{\partial \mathbf{v}_{t-1}^W} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbf{I}, \tag{3.25}$$

and

$$\frac{\partial \mathbf{r}_t^W}{\partial \mathbf{v}_{t-1}^W} = \Delta T \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \Delta T \ \mathbf{I}. \tag{3.26}$$

The specially defined Jacobian with a partial derivative that is taken with respect to a quaternion however, requires a more intricate solution. The process of defining a new quaternion from the measured angular velocity $\boldsymbol{\omega}_t^C$ - is shown in Equation 3.21

$$\mathbf{q}_t^C = \text{quat}(\boldsymbol{\omega}_t^C \Delta T) \tag{3.27}$$

Hereafter, the final non zero element of the Jacobian can be defined in terms of the aforementioned new quaternion:

$$\frac{\partial \mathbf{q}_t^{WC}}{\partial \mathbf{q}_{t-1}^{WC}} = \begin{pmatrix} q_{1,t}^C & -q_{2,t}^C & -q_{3,t}^C & -q_{4,t}^C \\ q_{2,t}^C & q_{1,t}^C & -q_{4,t}^C & q_{3,t}^C \\ q_{3,t}^C & q_{4,t}^C & q_{1,t}^C & -q_{2,t}^C \\ q_{4,t}^C & -q_{3,t}^C & q_{2,t}^C & q_{1,t}^C \end{pmatrix}. \tag{3.28}$$

In order to complete the covariance update, the process noise covariance is still required to be updated. The additive process noise is captured by the IMU measurements and modelled as a zero-mean Gaussian process. The noise covariance $\mathbf{R}_w$ however, incorporates additional uncertainty regarding the processing of the linear acceleration $\mathbf{a}_t$ from the total acceleration $\mathbf{f}_t$:

$$\begin{aligned} \mathbf{R}_{w,t} &= \frac{\partial \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})}{\partial \mathbf{w}_{t-1}} \mathbf{R}_{w_{t-1}} \frac{\partial \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})}{\partial \mathbf{w}_{t-1}}^T \\ &= \begin{pmatrix} \mathbf{I}\Delta T & \mathbf{0} \\ \mathbf{0} & \frac{\partial \mathbf{q}_t^{WC}}{\partial \boldsymbol{\omega}_{t-1}^C} \end{pmatrix} \mathbf{R}_{w_{t-1}} \begin{pmatrix} \mathbf{I}\Delta T & \mathbf{0} \\ \mathbf{0} & \frac{\partial \mathbf{q}_t^{WC}}{\partial \boldsymbol{\omega}_{t-1}^C} \end{pmatrix}^T, \end{aligned} \tag{3.29}$$

where the Jacobian element $\frac{\partial \mathbf{q}_t^{WC}}{\partial \boldsymbol{\omega}_{t-1}^C}$ incorporates an uncertainty due to rotations.

This completes the deviations necessary to fully incorporate the control update step of the EKF.

3.5 Measurement Update

With reference again to the probabilistic form of the solution to the SLAM problem, the measurement step too, requires a description in terms of a belief distribution. This belief, in terms of the probability distribution on the measurements, is described in step 2 of Table 2.1 and repeated for convenience :

$$bel(\mathbf{x}_t) = \eta p(\mathbf{z}_t \mid \mathbf{x}_t)\overline{bel}(\mathbf{x}_t) \tag{3.30}$$

The correction step seeks to incorporate an actual measurement $\mathbf{z}_t$ into the state estimate through incorporating the previously mentioned weighted mean upon updating the mean prediction and covariance prediction estimates. The measurement update of the EKF in this project is identical to the measurement update of the existing MonoSLAM implementation [**?**]. A functional diagram of the measurement update is shown in Figure 12.

The focus of this project is to incorporate additional information to improve the motion estimation of a robot. The literature study showed that the utilisation of additional measurement sensor is both expensive and difficult to integrate with the existing implementation (this is stated in Chapter 1.2). It is also expected that improvements to the existing measurement update of MonoSLAM will not improve the state estimates significantly. As a result, a decision is made to incorporate the existing measurement update directly as presented in the study by Davidson et al. [**?**]

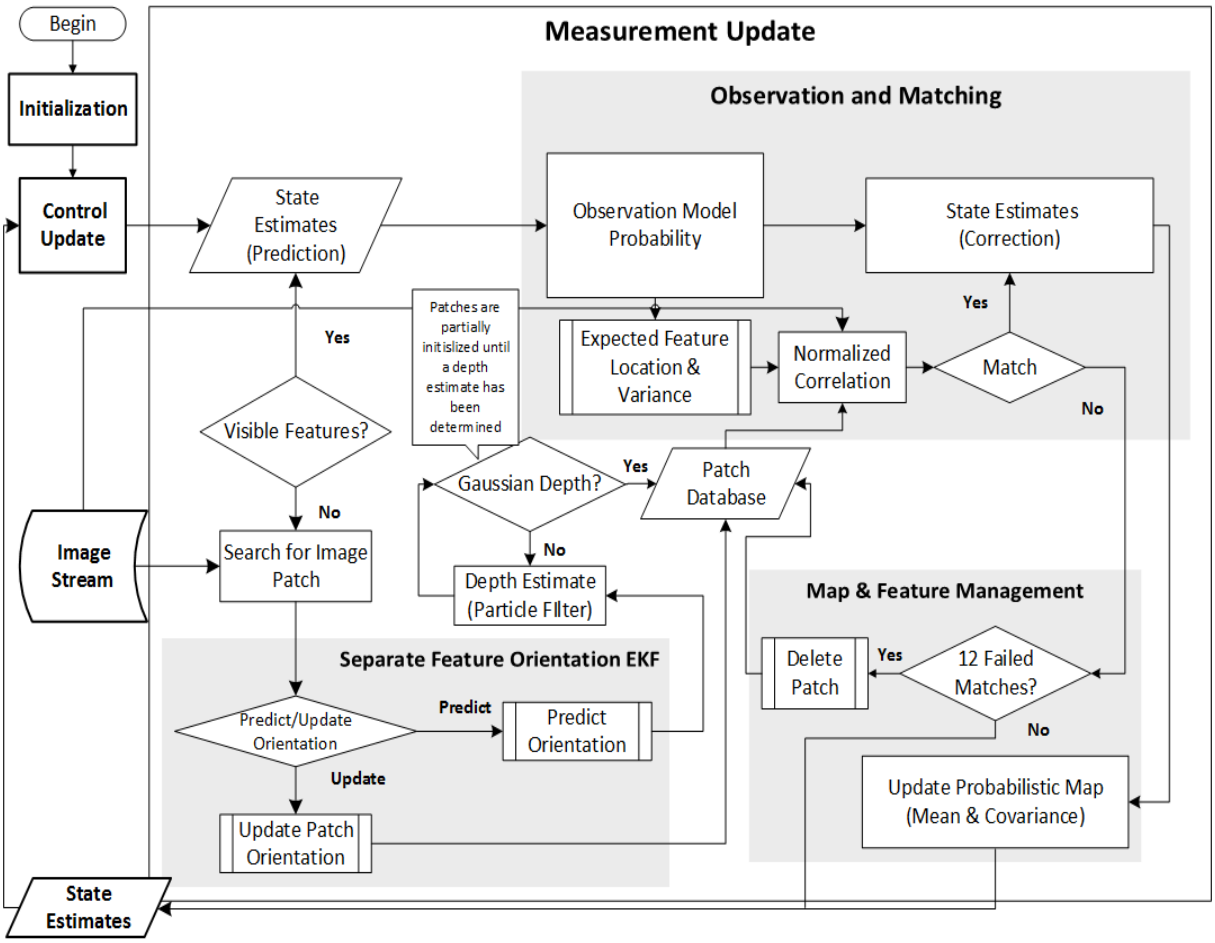


Figure 12: A detailed functional diagram of the measurement update step of the EKF. The greyed blocks depict functional subsystems

3.5.1 General Overview

This section seeks to provide an informative yet concise overview of the measurement update that is implemented in this project. Although no alterations were made to this subsection of the project, the reader requires an understanding of the basic operations that are fundamental to the realisation of this project.

This particular vision based approach aims to use salient image *patches* as long term landmarks as presented in [**?**, **?**]. These aforementioned patches are typically 11×11 pixels and are obtained through the image detection operator of Shi and Tomasi [**?**] from raw monochrome images. The goal is to repeatedly re-identify these image template patches over time after considerable camera movements. Invariably, basic 2D template matching algorithms are of little use, considering that any particular movements of the camera (even minimal) can severely alter the shape of a saved template patch. As a result, MonoSLAM assumes that each patch lies on a locally planar surface and that the surface normal is parallel to the vector from the feature to the camera. Once the depth of a patch has been determined - this is done through a small particle filter - the patch is stored to be used as a long term landmark. MonoSLAM stores these patches to provide a template for matching against a newly obtained 2D image at a later stage. A warped version of the patch is projected back into the image upon matching to account for the potential error provided by the orientation assumption. Because the appearance of the patches are never updated and remain in memory, long term localisation is possible. The orientation of the features however, are updated using a separate EKF - this is depicted in Figure 12.

The measurement model can provide a prediction regarding a feature's coordinate in the current image based on the prior movement estimates of the camera. The *innovation matrix* describes the 2D Gaussian uncertainty regarding this position, providing an elliptical region around the predicted coordinate. A normalised cross-correlation is conducted within this search region seeking to match an initialised feature within the current image.

The EKF measurement update is concluded through the steps shown in Table 2.3 after the probabilistic map is updated. An additional process outside of the operation of the EKF is required to achieve this process. This process is described in Chapter 3.5.2.

3.5.2 Probabilistic Map Management

In order to realise the solution to the SLAM problem, the measurement update of a system requires an additional process - namely *data association* or *map management*. The criteria regarding this map management typically differs according to the application, but the general idea is that the map continuously deletes and adds features accordingly.

In MonoSLAM, the criteria of this map management is determined via a trade off between two factors: number of features in a camera's field of view and the computation expense. MonoSLAM seeks to, at every given time instance, keep the number of features visible within the camera's field of view close to a predetermined value. This value is 12 by default, but can be adjusted according to the computing power available.

The detection operator of Shi and Tomasi [**?**] seeks to initialise new features if the number of features visible in the cameras field of view drops below the predetermined value. These features are initialised if a prediction based on the camera's movement suggests that the feature will remain in the camera's field of view in the subsequent frames. Additionally, a feature is deleted from the map if it fails 50% of successive detection and matching attempts.

4 Implementation

4.1 Introduction

This Chapter sets out to discuss the hardware and software decisions that are made with respect to this project. Initially, the system requirements are presented and possible solutions are discussed. Thereafter, the individual hardware components used in this project are presented and motivated. This chapter also provides the necessary configuration for each component to realise the system requirements.

4.2 System Requirements

The project objectives in Chapter 1.3 allow processing regarding the MonoSLAM implementation to be conducted on a personal computer (PC). In the original implementation of MonoSLAM, a 1.6 GHz Pentium M processor is used. Standard PC's however, provide no simple interface with external components that don't possess the peripheral ports present on the PC e.g. USB, Ethernet, Firewire and VGA. If indeed components can be connected via one of the aforementioned ports, a complicated procedure is often required if certain external components are required to communicate with one another through the PC.

The MonoSLAM implementation described in this project requires sensor measurements from a camera and motion measurements from an IMU. These components are also required to be synchronised. While cameras typically possess USB/Firewire/Ethernet ports allowing an interface with a PC, IMU's are typically IC's mounted on a PCB and contain no ports whatsoever.

Wireless connectivity such as Bluetooth and Wifi provide an alternative for data communication. This wireless connectivity requires as both the PC and each sensor to possess an expensive integrated chip to enable this functionality and can also be difficult to implement. Micro-controllers however are cheap devices that allow a virtual interface to the PC for external hardware components. Micro-controllers are programmable devices and typically contain a range of digital and analogue input and output (I/O) ports. The programmable nature of these devices allow a certain degree of control regarding the design and functionality of the system.

As a result, this project utilises a micro controller to provide the synchronisation between the camera and IMU as well as obtaining the data from the IMU and relaying it to the PC.

An overview of the components required to realise the overall system as well as their interaction between one another is depicted in Figure 5. It is evident that the system contains three hardware subsystems:

- Micro-controller.
- Inertial measurement unit.
- CMOS camera.

Each of of the subsystems require a unique setup configuration in order to represent a *functional* unit of the whole system.

4.2.1 Inertial Measurement Unit

Chapter 3.4.2 shows that inertial measurements are required in order to realise the implementation proposed in this project. The inertial measurements consists of 3 accelerometer and 3 gyroscope measurements. The combination of these sensors form a 6 *degrees of freedom* (DOF) IMU. Accelerometers and gyroscopes can be analogue or digital devices that represent their measurements as either a voltage or a digital value respectively. This project seeks digital values that can be used by the PC to incorporate the measurements into the MonoSLAM algorithm.

A digital IMU with an accelerometer and gyroscope mounted on the same IC should ideally be purchased as there is time limit imposed on this project. Table 4.1 compares the necessary characteristics with a suitable IMU option, the SparkFun 6DOF IMU:

Table 4.1: IMU necessary characteristics

Characteristic	SparkFun 6DOF IMU	SparkFun 9DOF IMU	Required
Serial communication	I2C/SPI (100 kHz & 400 kHz)	I2C/SPI (100 kHz & 400 kHz)	I2C (100 kHz)
Components	Accelerometer, Gyroscope	Accelerometer, Gyroscope, Magnetometer	Accelerometer, Groscope
Low pass filter (anti-aliasing)	yes	yes	yes
Maximum data rate	3200 Hz	3200 Hz	30 Hz
Price	R 489.99	R620	$\leq$ R 500

The IMU cannot precisely sample measurements at the given system sampling period (30 Hz). The closest possible output data rate that exceeds the required system sampling period is 50 Hz. This however shouldn't limit the system functionality as the output data rate is greater than what is required. It should be noted that the IMU internal sampling rate is set to 3200 Hz for the accelerometer and 1000 Hz for the gyroscope - far more than the Nyqvist frequency. Additionally, the low pass filter is set to a bandwidth of 100 Hz to prevent aliasing.

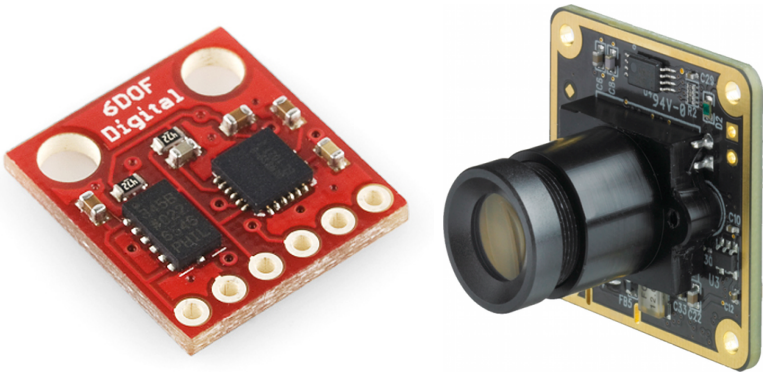


Figure 13: Left: SparkFun 6DOF IMU. Photo by SparkFun. Adapted from [**?**]. Right: The Imaging Source 22BUC03-ML CMOS board camera. Adapted from [**?**]

The SparkFun 6 DOF IMU as depicted in Figure 13 provides the necessary characteristics for this project. This board is comprised of a ITG3200 MEMS 3-axis gyroscope [**?**] and a ADXL345 3-axis linear accelerometer [**?**]. This component was chosen as it contains both the necessary sensors on the same chip while supporting I2C communication, an anti-alias filter, suitable range, sample code.

Alternative products such as the 9DOF SparkFun IMU shown in Table 4.1 generally contain an additional 3 DOF while the required project specifications are identical or marginally better than that of the SparkFun 6DOF IMU. The additional 3 DOF provided by the magnetometer are assumed to be redundant and overlooked considering that these products are considerably more expensive than the chosen component.

Finally, the IMU is configured in measurement mode with the following properties:

Table 4.2: IMU configuration properties

Parameter	Value
Serial communication	I2C
Accelerometer range	$\pm$4g per second
Resolution	10 bits
Gyroscope range	$\pm$2000° per second
Output data rate	50 Hz (100Hz low pass filter bandwidth)

4.2.2 CMOS Machine Vision Camera

The measurement sensor is required to be a single camera. The problem description only requires the camera to have a 30 Hz frame rate. The camera chosen for this project is a CMOS camera by The Imaging Source as depicted in Figure 13. The DFM 22BUC03-ML model is a machine vision board that contains 4 general purpose input/output (GPIO) pins that allow the shutter to be triggered by a digital pulse. The pulse is to be triggered externally and the data is sent to the PC via a USB port. The camera was used for a previous project but still provides the necessary functionality required for this project and is thus incorporated due to a limited budget. The *intrinsic* matrix of the camera in Equation 3.6 and the distortion coefficients are determined through camera calibration made available by OpenCV [**?**]. The results of this procedure are given as follows:

Table 4.3: Intrinsic and distortion camera parameters

Parameter	Values
focal length $f_u = f_v$	1019 pixels
Principal point u_0	319 pixels
Principal point v_0	241 pixels
Radial distortion coefficient k_1	0.504
Radial distortion coefficient k_2	0.493

4.2.3 Micro-controller

This project requires a micro-controller for the following two reasons:

- Digital interface with the IMU.
- Synchronisation between the IMU and camera measurements.

The requirements state that only a small micro-controller is required. In order to realise these requirements, the micro-controller only requires 1 GPIO pin that can be triggered by a timer interrupt as well as a I2C interface that can operate at the IMU's specified 100 Khz or 400 Khz. A board based micro-controller (like an Arduino) shouldn't only be considered, micro controllers seated in dual in-line packages (DIP) are considerably cheaper and will also be considered.

Three micro-controllers are considered in project. These include the board based Arduino Uno SMD Rev3 and PIC32-Pinguino and the DIP MSP430 Launchpad. Table 4.4 provides a general overview of some important aspects regarding theses micro-controllers.

Table 4.4: General overview of the considered micro-controllers

Characteristic	Arduino Uno	Pinguino	MSP430 Launchpad	Required
Clock Frequency	16 MHz	80 MHz	32 KHz	400 kHz
I2C (400 kHz)	yes	yes	yes	yes
IDE	yes	yes	yes	no
Interruptible GPIO	yes	yes	yes	1 pin
Deliver Community	Large	Moderate	Small	No
Price	R 267	R 210	R 120	$\leq$ R350

Although very cheap, the MSP430 Launchpad provides the necessary specifications (one interruptible GPIO pin, I2C and minimum clock frequency) for this project, the developer community is very small, with limited the support compared to the Arduino and the Pinguino.

The Pinguino also provides the necessary specifications but with a much larger developer community than the Launchpad. The powerful MIPS processor however will not influence this project as the micro-controller is only used for data communication.

Although more expensive than the alternatives, the Arduino UNO SMD Rev3 depicted in Figure 14 micro-controller still provide a suitable, low cost solution and it the controller chosen to be implemented in this project. The Arduino Uno provides the functionality necessary for this project. The user friendly integrated development environment, very helpful developer community along with source code for existing IMU implementations all contributed to the decision.

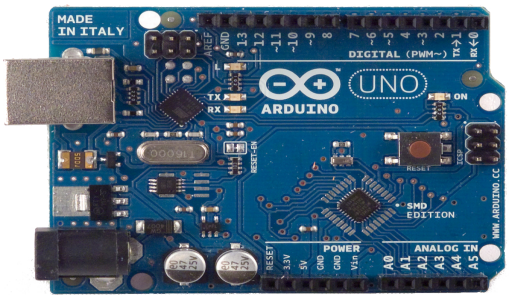


Figure 14: Arduino UNO SMD by Arduino. Adapted from [**?**].

4.3 Hardware Configuration

This Chapter seeks to discuss the necessary practical procedures undertaken to realise the system requirements defined in Chapter 4.2. The system configuration as well as the data communication is done by the Arduino. The necessary code is uploaded onto the Arduino micro-controller over a USB port using the Arduino IDE. All of the code on the Arduino is written using C++ and compiled using g++.

The hardware components need to be designed in order to meet the following specifications:

- 30 Hz data rate.
- Camera and IMU synchronisation.
- Reliable data transfer from hardware to PC.

4.3.1 Inertial Measurement Unit

Before any IMU measurements can be obtained, the IMU needs to be correctly configured. Communication between the micro-controller and the IMU is occurs via the I2C interface. The appropriate registers of the accelerometer and the gyroscope need to be set to resemble the configuration shown in Table 4.2.

Table 4.5: Accelerometer and gyroscope register values

Register	Value	Name	Function
Accelerometer			
0x31	0x09	DATA_FORMAT	Set the accelerometer to $\pm 4g$ and 10 bit resolution.
0x2D	0x08	POWER_CTL	Set the accelerometer to measure mode with minimum power consumption.
Gyroscope			
0x16	0x1B	Full Scale	Set the gyroscope to $\pm 2000°$, 1 kHz sample rate and 100 Hz low pass filter bandwidth.

The IMU sensors can now be measured and subsequently converted from analog to digital values and transferred to the PC. Both devices (accelerometer and gyroscope) are set a 10 bit resolution. This means that the analog value will be represented as 10 bits and each measurement will be stored in two separate bytes (16 bits). There are a total of 6 IMU measurements - 3 from the gyroscope and 3 from the accelerometer. A suitable baud rate must also be determined to ensure these measurement are sent to the PC within the sample period.

The baud rate represents the number of bits per second that are transferred on a bus. The system sample period contains $\frac{1}{30} = 0.0\overline{333}$ seconds, meaning that the data sampled from the IMU needs to be transferred over the USB in-between the instances that a trigger to sample the camera is set and that 33 ms elapses.

The I2C burst-read protocol [**?**, **?**] for reading from a slave device at 100 kHz results in a 38 bits $\times$ 6 measurements $\approx$ 228 bits. At 100 kHz this data transfer consumes only 2.28 ms, well within the system sampling period of 30 Hz. The baud rate for the data transfer between the Arduino and the PC is thus essential. This project uses a baud rate

of 19200 to allow all of the data to be sent though to the PC before the following system sampling instance.

4.3.2 Sensor Bias and Variance

Measurement sensors often have offsets that cause a measurement *bias*. These biases can however be measured and subsequently subtracted from each measurement. If each measurement is modelled with additive Gaussian RV, the bias is the mean of this noise. The procedure for measuring such a bias is quite simple: if the gyroscope is stationary, the angular velocities are expected to be zero. The procedure for measuring the accelerometer bias is slightly more difficult, as an accelerometer measures gravitational acceleration. If the accelerometer is rotated about each axis, the accelerations of the x- and y-accelerations should resemble a sinusoid due with a peak equal to the gravitational acceleration. The offset of this sinusoid is typically the measurement bias. If the accelerometer can be placed completely level, the noise on each axis can be measured. The expected value on each axis should be zero (except the z-acceleration that expected the gravitational acceleration) but the measurements will show a varying non-zero value as depicted in Figure 18. The average of this varying value is the bias.

Table 4.6: Statistical quantities of the measurement noise.

Parameter	Mean	Variance
Acceleration $x-$axis	0.3405	6.7923e-04
Acceleration $y-$axis	0.6026	7.2212e-04
Acceleration $z-$axis	0.0671	8.2147e-04
Gyroscope $x-$axis	0.3587	0.0283
Gyroscope $y-$axis	0.0538	0.0188
Gyroscope $z-$axis	-0.1028	0.0638

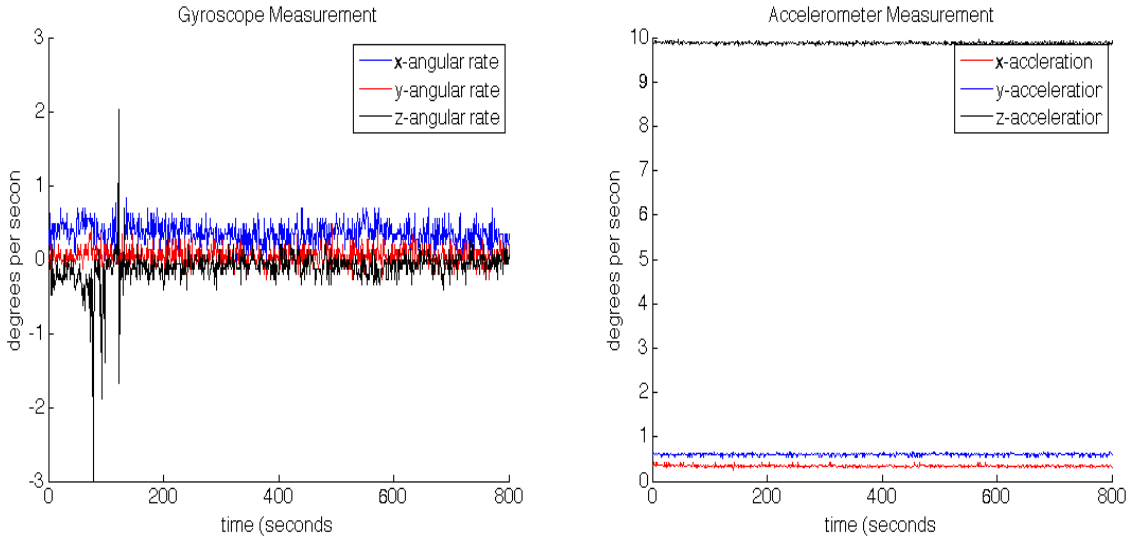


Figure 15: Left: Measurements of the stationary gyroscope. Right: Averaged measurements from the accelerometer after 3-axis individual rotations.

The modelling of the additive noise in Chapter 3.3.4 is based statistical quantities provided in Table 4.6.

4.3.3 Micro-controller

The IMU and camera measurements need to be *synchronised*. A suitable method to achieve such synchronisation is to run a timer interrupt every sample period of the camera (i.e 30 Hz). The interrupt subsequently calls an interrupt service routine (ISR) that triggers a pulse to the camera to capture a frame while simultaneously sampling the IMU measurements. According to the data sheet of the camera [**?**], a digital pulse must be set high for at least 10μ seconds in order to set the moment of exposure. The exposure itself is set through external software. This procedure is shown in Figure 16.

The timer interrupt is required to be initiated by enabling the interrupt during the setup after the timer pre-scaler has been set to realise an interrupt ever sampling period of 33.33 ms.

Communication between the IMU and the Arduino is done via I2C. The accelerometer and the gyroscope are slave devices on the bus while the Arduino is the master. The IMU is configured according to the configuration explained in Chapter 4.3.1 in the once-off setup function before entering the infinite loop() function that waits for a command from the PC in order to send the IMU data.

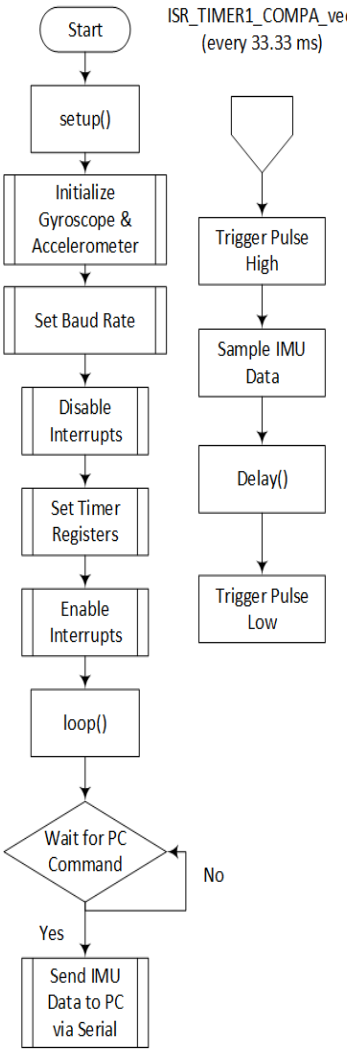


Figure 16: A flow diagram for the micro-controller program required to synchronise the sampling of the IMU and camera measurements.

4.4 System Configuration

4.4.1 MonoSLAM

The implementation discussed in this paper seeks to provide an improvement to the original implementation of MonoSLAM by Davison et al. [**?**]. A determining factor in choosing the MonoSLAM implementation proposed by Davison et al. is the availability of the C++ based application called *Scenelib* that is GNU Lesser General Public License (version 2). This allows the code to be used and updated provided that the updated version of the code is released under the same version. The original Scenelib implementation is adapted to incorporate the aspects proposed in this project as follows:

- The constant linear and angular velocity motion model is removed and replaced by the kinematic estimator.
- The system is required to incorporate additional data from the IMU.

All processing of the data is done on a PC using Ubuntu 14.04 and C++ (cmake for compiling). MonoSLAM provides an interactive graphical use interface (GUI) that shows the image tracking as well as the constucted probabilistic map in real time.

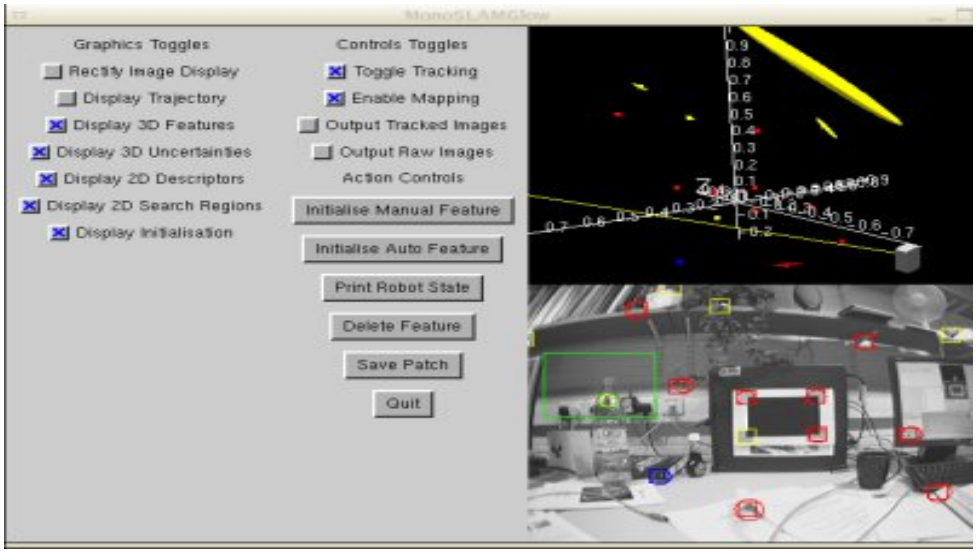


Figure 17: The GUI of the MonoSLAM application. Adapted from [**?**].

MonoSLAM requires a simple setup procedure, where a known initial pose and a known feature is required to be initialised in an initialisation file prior to system execution to provide a scale. The typical known feature is a black A5 rectangular block placed on a black background - the four corners are typically initialised as the known patch features. This image is place in a precisely known location in the environment and the location of the initialed patches are place into a initialisation file that the system reads upon execution.

In the implementation proposed in this project, the feature is placed on a (approximately) perfect vertical surface so that the IMU can be orientated with respect to the gravitational acceleration initially (level on a surface) - allowing a known initial pose.

5 Analysis: Testing & Results

5.1 Introduction

Due to the time constraints imposed on this project, a detailed testing analysis of the entire system is not possible. Instead, tests that seek to validate the functionality of each subsystem were implemented accordingly. The validity of the raw gyroscope and accelerometer measurements are initially inspected. Thereafter, the system sampling period as well as the data rate transfer are inspected. Additionally the validity of the kinematic state estimator is tested by providing it with simulated measurements. Finally, the issues regarding the implementation of the adapted MonoSLAM system are discussed.

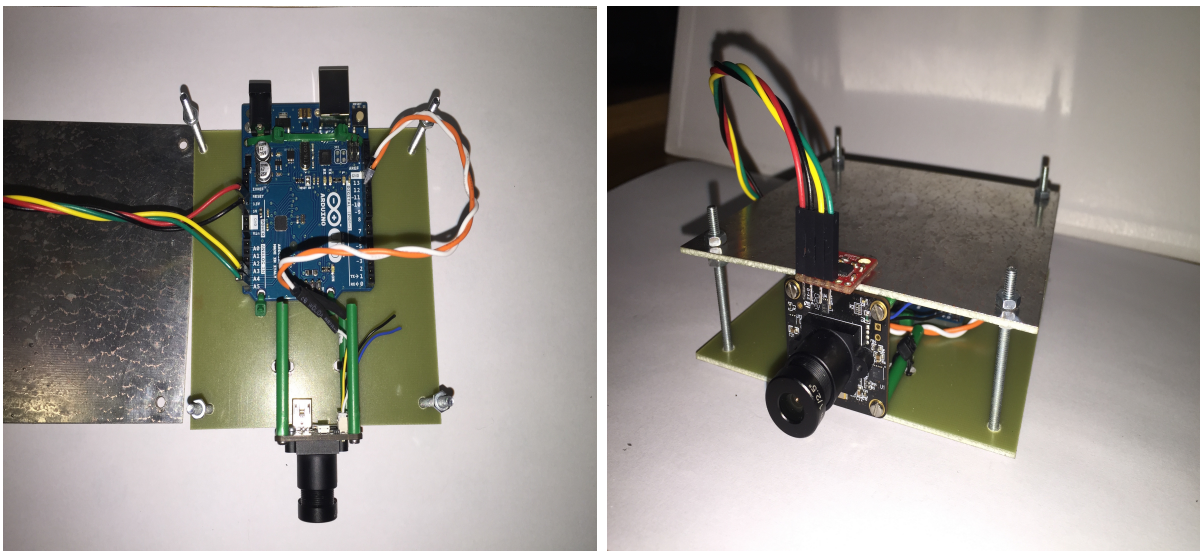


Figure 18: Left: Top view of the hardware components of the system system. Right: Fully constructed platform to house the hardware components of the system.

5.2 Subsystem testing

The performance of the hardware components of this project requires analysis through physical measurements and simulation before being integrated into the final system. In order to successfully implement a project of this magnitude that incorporates several subsystems, it is essential that the design as well as analysis thereof is accurate. The hardware components are initially designed to meet the relevant specifications and functionality stated in Chapter 4 and Chapter 3. Thereafter, simulation and measurement equipment can be used to verify that the subsystems do indeed behave as defined in Chapter 3. This process will greatly simplify the final integration process.

The validity of the accelerometer and gyroscope measurements are initially tested. Thereafter, the function of the micro-controller that generates a pulse and samples an IMU measurement within a system sample period is tested. Finally, the kinematic estimator is evaluated as a state estimator using simulated accelerometer and gyroscope measurements to reconstruct a trajectory.

5.2.1 Accuracy of Gyroscope Raw Measurements

In order to verify the functionality of the gyroscope, the physical measurements need to be measured and analysed. It is very difficult to intuitively analyse an angular velocity. The angular displacement however, is a much better test case upon verifying that the gyroscope measurements are correct. The gyroscope can be approximately rotated at known angles and the resulting angular velocities can be numerically integrated to obtain the angular displacement. This is a suitable method of analysing whether the information obtained form the gyroscope measurements resemble the actual movements .

The following set of test conditions will utilise actual gyroscope measurements. An approximately known angular displacements will be exerted upon the IMU e.g. rotate 90° about the positive x-axis. The resulting gyroscope measurements will be plotted and analysed to verify its functionality.

Considering that the sample period is relatively small, the angular displacement Θ_t can be approximated in terms of the measured angular velocities ψ_t as follows:

$$\begin{aligned} \psi_t &= \frac{d\Theta_t}{dt} \\ \Theta_t &= \Delta T \psi_t \end{aligned} \tag{5.1}$$

It can also be assumed that given the small sample period, the relationship between the angular displacement and the angular velocity (velocity) can be approximated as linear.

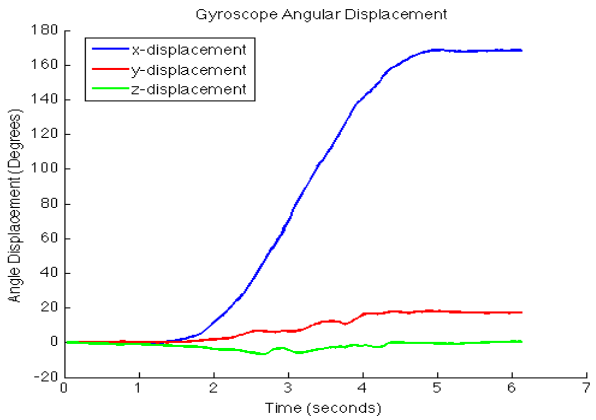
1. **Positive** 180° **rotation about the x-axis:**
   The measurements depicted in Figure 19a show an angular position x that begins at zero and ends at approximately 180° due to the rotation described by the test case.

2. **Positive** 90° **rotation about the y-axis:**
   The measurements depicted in Figure 19b show an angular position y that begins at zero and ends at approximately 90° due to the rotation described by the test case.

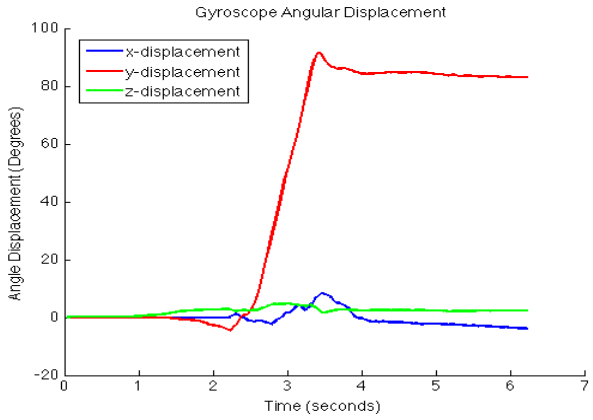
3. **Positive** 90° **followed by a negative** 90° **rotation about the y-axis:**
   The measurements depicted in Figure 19c show an angular position y that begins at zero and goes to approximately 90° before falling back to approximately zero due to the rotation described by the test case..

4. **Positive** 360° **rotation about the z-axis:**
   The measurements depicted in Figure 19d show an angular position z that begins at zero and ends at approximately 360° due to the rotation described by the test case.

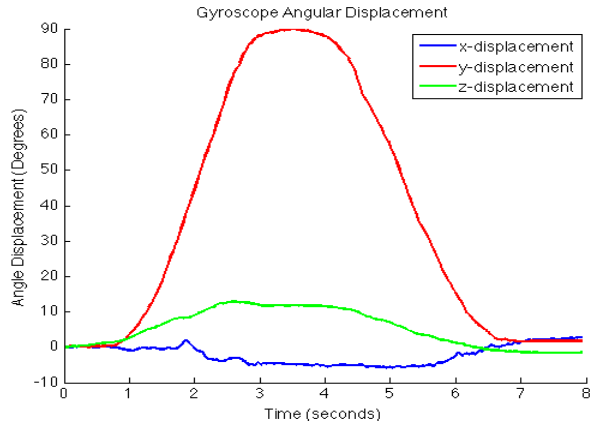
Additionally, it can be observed in Figure 19 that the rate at which the angular displacement is changing is approximately linear in all the cases. The gyroscope measurements are thus validated given that the mathematics and measurement simulation are in agreement.



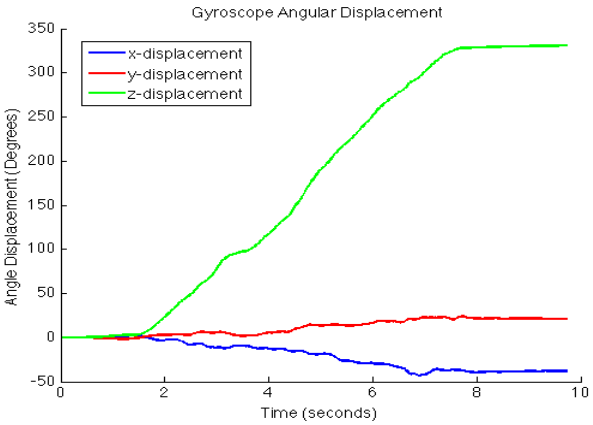
(a) Angular displacement after positive 180° rotation about the x-axis.



(b) Angular displacement after positive 90° rotation about the y-axis.



(c) Angular displacement after positive 90° negative then 90° rotation about the y-axis.



(d) Angular displacement after positive 360° rotation about the z-axis.

Figure 19: Results of the test conditions 1 depicting angular displacements from gyroscope measurements.

It should be noted that the gyroscope gradually drifts over time. It is a known that the gyroscope produces a constant error that is subsequently integrated over time (upon seeking to obtain an angular position) producing a linear angular drift. It is expected though that the measurement update in the SLAM filter will account for these errors.

5.2.2 Accelerometer Raw Measurements

In order to verify the validity of the accelerometer, the physical measurements need to be measured and analysed. Although is is a lot easier to intuitively analyse an acceleration than an angular rate, the process of obtaining the linear acceleration is very complex.

The accelerometer measurements may not drift over time as in the case of the gyroscope, but all accelerometer measurements incorporate gravitational acceleration. Additionally, an accelerometer cannot itself distinguish between the gravitational and linear accelerations that it measures. This typically results in the components of the gravity vector being assumed upon other axes of the accelerometer resulting in incorrect measurements. An additional sensor is typically required - in this instance the gyroscope. The idea is to measure the degree by which the orientation changes (using the gyroscope measurements), and at each instance subtract the gravity vector.

As previously mentioned however, the gyroscope drifts over time. The orientation at each time instance needs to be precisely known in order to successfully subtract the gravitational acceleration. The error in orientation then, causes linear acceleration measurements that cannot accurately reconstruct the position of a robot.

This issue was realised late in this project and couldn't be rectified due to the hard deadline of the project. An alternative suggestion was to account for this problem in the context of this project, by substantially increasing the variance of the accelerometer noise. This procedure tells the EKF that you are very uncertain regarding this measurement and the EKF will take this into consideration upon obtaining the optimal weighting of the state estimate.

5.2.3 System Sampling Period

In order to verify that the system sampling period is according to the desired requirements (30 Hz), a set of measurements need to be obtained measuring the timer-interrupt realised by the micro-controller in Chapter 4.3.3. An oscilloscope can be used to measure the port that triggers the camera's GPIO pin. This pin expects a pulse with a width of at least 10μs repeated at least every 30 Hz.

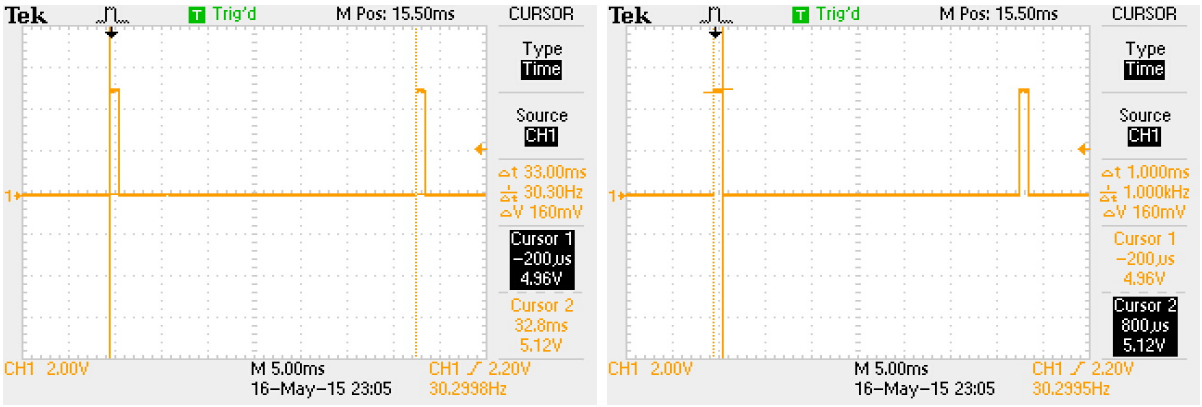


Figure 20: Left: Measurement depicting the frequency between the pulses - 30.3 Hz Right: Measurement depicting the width of the pulses - 10μs.

5.2.4 Sufficient Data Transfer Speed

In order to verify that the data transfer of the IMU measurements onto the PC system are within sampling period, a set of measurements need to be obtained measuring the data transfer by the micro-controller in Chapter 4.3.3. An oscilloscope can be used to measure the transmit port of the UART. A full set of IMU measurements (3 gyroscope and 3 accelerometer) are sampled and subsequently transmitted on this pin. The data should be transferred to the PC within the system sampling period.

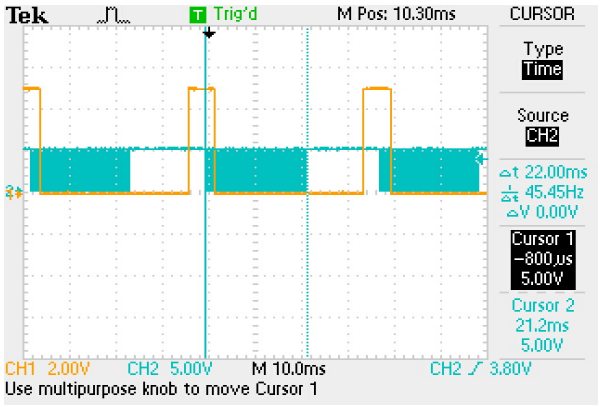


Figure 21: Measurement depicting the time taken for a single, full IMU data measurement to be sampled and sent to the PC.

Figure 21 shows that the full data transfer is well within the system sampling period and the speed of transmission is thus sufficient.

5.3 Simulation

This study seeks to investigate the performance of the kinematic estimator as a state transition model. An EKF-based kinematic estimator *simulation* is designed to meet the following specifications:

- Simulated measurements are calculated by adding noise to the states.
- The kinematic estimator should precisely track a trajectory of a robot from simulated IMU measurements, given that there is no uncertainty regarding the system.

In order to effectively analyse its performance, a simulation of an EKF that uses a kinematic estimator as the state transition model is run under a set of test conditions. These conditions will accurately analyse the performance of the kinematic estimator with regard to the aforementioned design specifications. A functional diagram of the EKF-based kinematic estimator (based on Equation 3.19) is shown in Figure 22. The kinematic estimator is incorporated within the control update of the EKF.

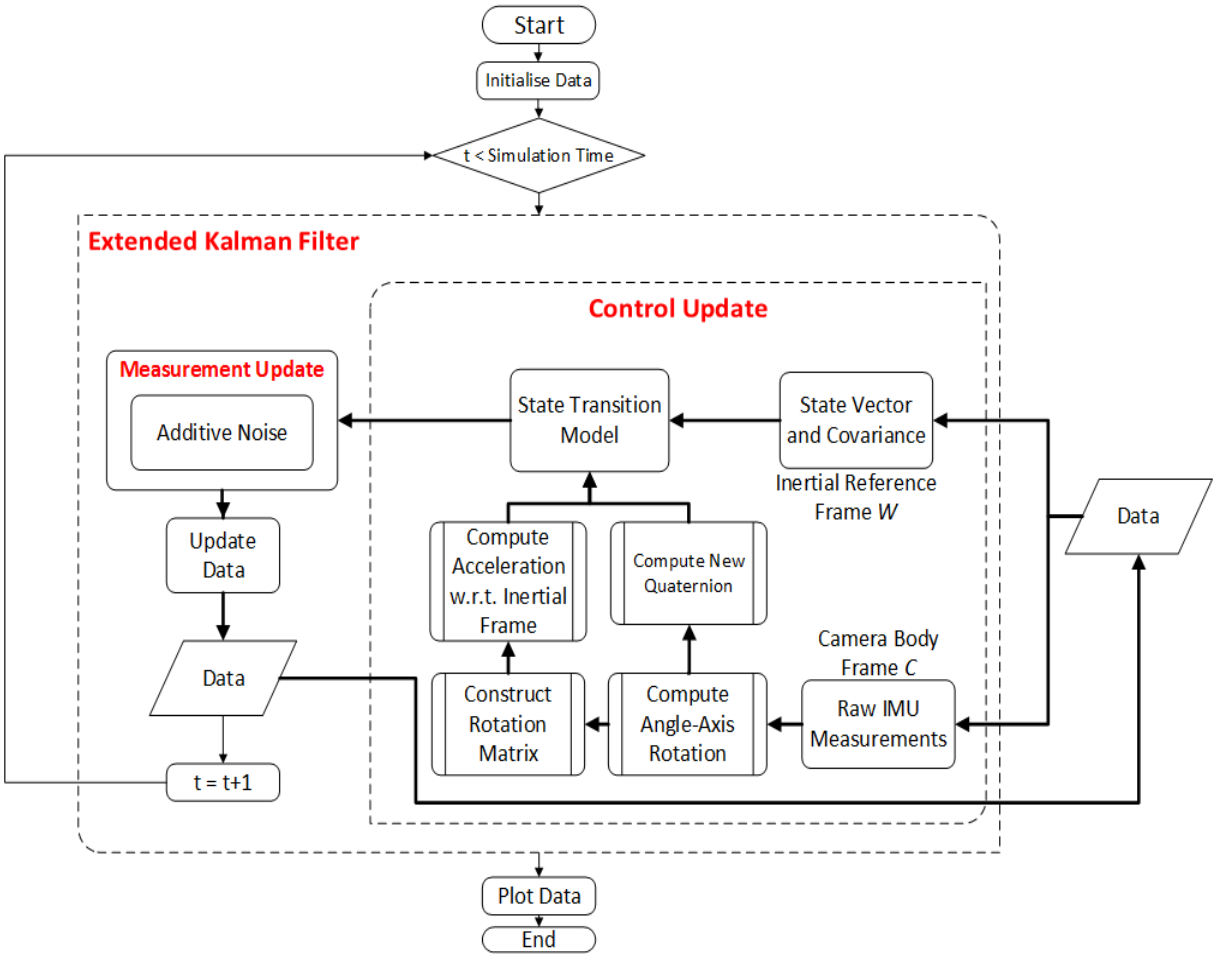


Figure 22: EKF using a kinematic estimator to provide the control update.

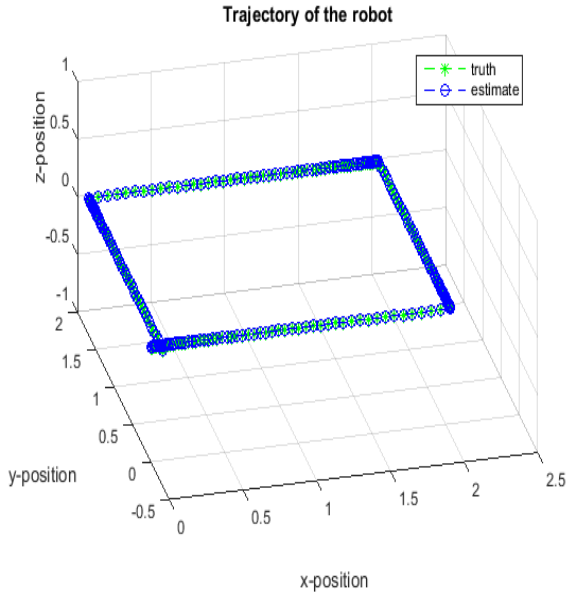
5.3.1 Recovering a Trajectory from Simulated IMU Measurements

will provide simulated gyroscope and accelerometer measurements in a noiseless environment. The EKF simulation measurement model is defined as a mapping of the state vector with additive noise. With no uncertainty regarding the measurement or process noise then, the simulated state estimates should exactly resemble the actual state vector.

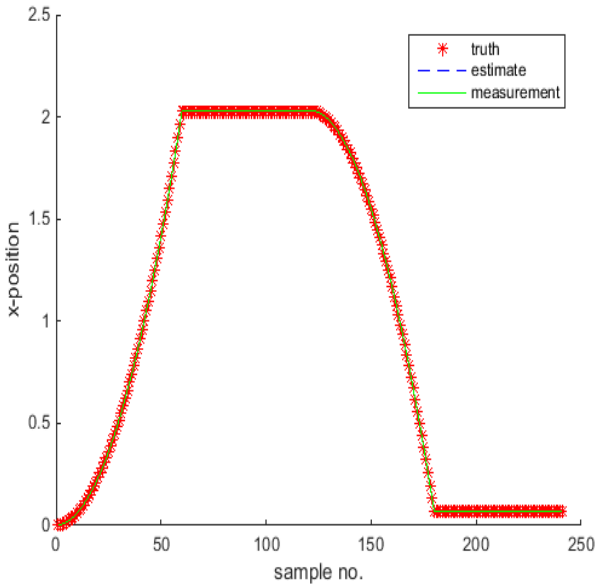
The expected outcome then, is that the kinematic estimator will precisely track a given robot trajectory. This trajectory will be simply defined so that the results can be accurately analysed.

The length of a simulation is 10 seconds at 30 Hz or 300 samples. The initial position is always $\mathbf{r}^W = \{0\ 0\ 0\}$ with an orientation quaternion of $\mathbf{q}^{WC} = \{1\ 0\ 0\ 0\}$):

1. **1 metre per second squared x-acceleration with three consecutive 90° rotations about the z-axis:**
   The simulation depicted in Figure 23a shows a square in the z-plane. This suggests that the trajectory incorporates three 90° rotations about the z-axis as the robot is moving at a constant acceleration with respect to its own reference frame. The simulation depicted in Figure 23b confirms that there is an initial acceleration in the x direction, before a change in direction. The estimated trajectory directly tracks the true trajectory.



(a) Trajectory: 1 m/s^2 x-acceleration with three consecutive 90° rotations about the z-axis.



(b) x-position: 1 m/s^2 x-acceleration with three consecutive 90° rotations about the z-axis.

Figure 23: Results of the test conditions 1 depicting zero-uncertainty.

5.4 MonoSLAM

This implementation of section in practice proved to be a very challenging part of this project. As a result of the time constraints imposed on this project, a functional system upon which fundamental tests could be undertaken was not possible. Initially, the system imposed a number of challenges that were not related to the theoretical concepts of this project. These challenges are listed as follows:

- Ubuntu 14.04 no longer supports ffmpeg as default and has replace this with libav which doesn't support the USB camera feeds used by Scenlib. A previous version of Scenelib was required to be used that only uses saved image sequences.
- The numerical libraries that are used by Davison et al. (Oxford universities vision workshop (VW)) is outdated and has no meaningful documentation. The existing code had to be studied in order to understand the library well enough to incorporate it within the proposed alterations to the project.
- It was initially very difficult to understand the Scenelib library due to its complexity, lack of a comprehensive documentation and my lack of experience using C++.

The kinematic state estimator was however implemented successfully within the control update of Scenelib with the results from the simulation depicting the results in Scenelib given the same simulated variables. Figure 24 depicts the simulations of the simulation in Matlab against the simulation of Davision given precisely the simulated variables:

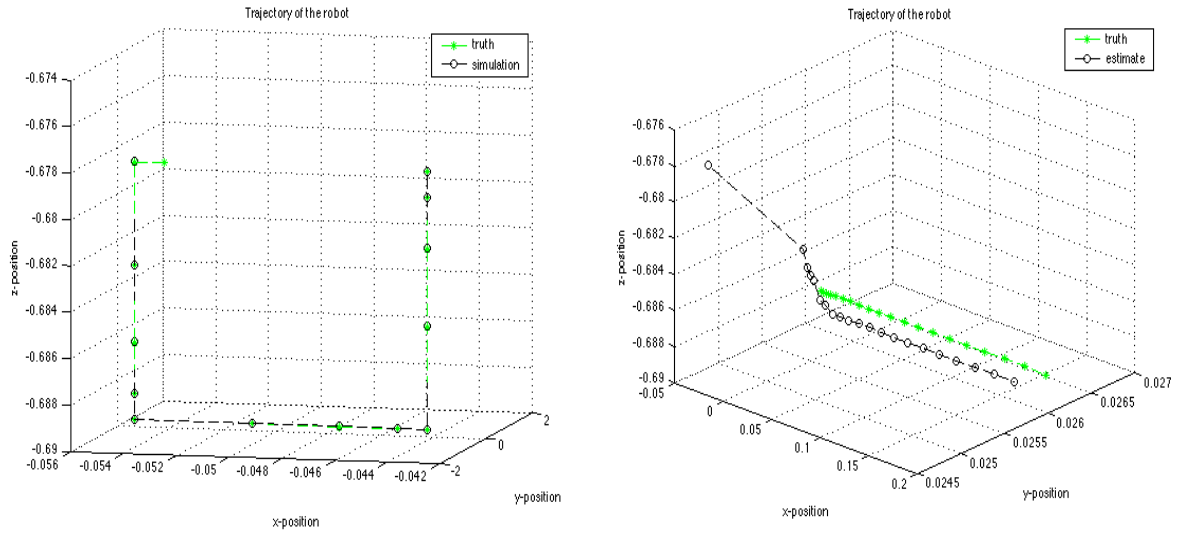


Figure 24: Left: Trajectory of the simulation against the results of the Scenelib control update. Right: Trajectory of the simulation against the results of the Scenelib control update.

The minor differences in the value are possibly due to rounded values. The direness are minimal though and the two simulations can be approximated as identical.

The measurement update of the MonoSLAM system in Scenelib contained errors in the re-projection of the image patches for matching. As a result, accurate matching never occurs and the system terminates. It is concluded that the distortion model approximated dorm Davision et al. in Equation 3.7 incorporates too much radial distortion and as a

result projection of images are incorrect. This problem was however identified too late in the project to correct given the hard deadline.

6 Conclusions and Recommendations

6.1 Introduction

This chapter concludes all the work conducted throughout this project and provides recommendations on further improvements to the project.

6.2 Conclusion

The core of the project, namely the kinematic state estimator along with the individual hardware subsystems was proven successful as far as possible. The final implementation of the MonoSLAM system however was not successful. The failure to successfully implement the proposed system is possibly be due to various individual factors, both practical, theoretical and time, as well as a combination of these factors. The discussion that follows provides the possible causes of the inability to realise a successful SLAM implementation.

The accelerometer used in this project aims to measure the linear accelerations of a body. This process proved to be a lot more difficult than expected as there is now possible way to determine the linear acceleration unless the orientation is precisely known. The method used in this project proposed to use the measurements from the gyroscope to obtain the changes in orientation. If the initial orientation is known (this is a requirement of the SLAM algorithm proposed in this project) the measurements form the gyroscope can be used to obtain the change in orientation. The gravity vector can subsequently be subtracted at each time step as its orientation with respect to the reference frame is always the negative z-direction. The gyroscope measurements however provide a constant error that is integrated with time resulting in drift and subsequently, inaccurate orientation estimates upon which the system depends in order to remove the gravity vector. This problem was initially accounted for by vastly increasing the uncertainty of the measurements but could possibly be overcome by additionally incorporating a magnetic sensor (magnetometer). A magnetometer can obtain additional information regarding the orientation of gravity by measuring the magnetic fields surrounding the earth. This process is common in inertial navigation and can be realised with an inexpensive sensor and minor additions to the equivalent control input calculations.

Furthermore the failure of the MonoSLAM Scenelib system to conduct a thorough SLAM solution could be attributed to the fact that the radial distortion approximation assumed from Davison et al. [**?**] causes an incorrect re-projection that is used to subsequently match the feature in the frames that follow. A distortion model suited to the specific camera used in this implementation was not researched accordingly. This problem can be corrected by obtaining a suitable distortion model to allow successful reproduction of the features for the best chance of correlation.

Finally it should be noted that even though a full implementation of the system was not achieved, the functional components of the system (besides the accelerometer) as well as the core kinematic estimator were implemented successfully as far as possible given the time constraints. These components include an IMU (besides the accelerometer) that can approximately measure an angular displacement, a Matlab simulation of the kinematic estimator that is also correctly implemented on Scenelib and the micro-controller that successfully achieves the system sampling period.

6.3 Recommendations

Although the kinematic estimator MonoSLAM was not fully realised, many of the systems were successfully designed and accordingly implemented. The factors influencing the failure to fully implement the system are given as follows:

- Inaccurate accelerometer measurements based on gravity vector. The gyroscope error over time prevents accurate orientation estimates to correctly subtract the gravity vector.
- The incorrectly assumed radial distortion.

The successfully designed and implemented subsystems however provide a good basis for the project to be completed with additional time with possible further development. The project objectives that were achieved are as follows:

- **Performing an overview on the current techniques used to realise SLAM**
  In order to understand how SLAM is implemented, an understanding of probability theory and state estimation is required. These concepts - specifically recursive state estimation and the Bayes filter - need to be researched and analysed before choosing a suitable technique to implement.
- **Analysis of the kinematic estimator as an alternative motion model**
  A kinematic estimator is suggested as a alternative motion model. The kinematic estimator needs to be researched, mathematically derived and simulated. The results from the simulation should correspond to the mathematical derivation. The advantages and disadvantages of the kinematic estimator also need to be investigated.
- **Hardware Design of the System**
  The kinematic estimator requires additional measurements obtained from an IMU. The IMU is required to interface with the PC via a micro-controller. The micro-controller allows precise synchronisation between the images sampled by the camera and the IMU measurements. A functional diagram of the system's hardware components are depicted in Figure 5.

References

[1] M. H. Hebert, *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[2] T. Brogårdh, "Present and future robot control development—an industrial perspective," *Annual Reviews in Control*, vol. 31, no. 1, pp. 69–79, 2007.

[3] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, pp. 116–121, 1985.

[4] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intellegence in the New Milenium* (G. Lakemeyer and B. Nebel, eds.), Morgan Kaufmann, 2002.

[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[6] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 229–241, 2001.

[7] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[8] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *AAAI/IAAI*, pp. 593–598, 2002.

[9] M. Montemerlo and S. Thrun, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, pp. 63–90, 2007.

[10] Z. Chen, "Bayesian filtering: From kalman filters to particle filters and beyond," *Statistics*, vol. 182, no. 1, pp. 1–69, 2003.

[11] S. Hilsenbeck, A. Möller, R. Huitl, G. Schroth, M. Kranz, and E. Steinbach, "Scale-preserving long-term visual odometry for indoor navigation," in *International Conference on Indoor Positioning and Indoor Navigation*, vol. 13, 2012.

[12] M. Chli, "Slam: Simultaneous localization and mapping." http://margaritachli.com, July 2011.

[13] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, pp. 1052–1067, June 2007.

[14] J. Sola, "Consistency of the monocular ekf-slam algorithm for three different landmark parametrizations," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 3513–3518, IEEE, 2010.

[15] G. K. D. Murray, "Parallel tracking and mapping on a camera phone," in *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Orlando), October 2009.

[16] P. Gemeiner, A. Davison, and V. Markus, "Improving localization robustness in monocular SLAM using a high-speed camera," in *Proceedings of Robotics: Science and Systems IV*, (Zurich, Switzerland), June 2008.

[17] H. Strasdat, J. M. M. Montiel, and A. Davison, "Scale drift-aware large scale monocular slam," in *Proceedings of Robotics: Science and Systems*, (Zaragoza, Spain), June 2010.

[18] J. Civera, A. J. Davison, and J. M. M. Montiel, "Unified inverse depth parametrization for monocular slam," in *In Proceedings of Robotics: Science and Systems*, 2006.

[19] S. Fu, H. ying Liu, L. fang Gao, and Y.-X. Gai, "Slam for mobile robots using laser range finder and monocular vision," in *Mechatronics and Machine Vision in Practice, 2007. M2VIP 2007. 14th International Conference on*, pp. 91–96, 2007.

[20] T. Sebastian, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[21] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[22] C. Stachniss, "Robot mapping: Extended kalman filter." University Lecture, 2013.

[23] A. J. Davison and D. W. Murray, "Mobile robot localisation using active vision," in *ECCV (2)* (H. Burkhardt and B. Neumann, eds.), vol. 1407, pp. 809–825, Springer, 1998.

[24] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 593–600, IEEE, 1994.

[25] G. Bruneau, S. Dubray, and A. Murguet, "Matlab implementation of monoslam." University Lecture, February 2012.

[26] S. Albrecht, "An analysis of visual mono-slam," Master's thesis, Universit´at Osnabr´uck, October 2009.

[27] R. Swaminathan and S. K. Nayar, "Nonmetric calibration of wide-angle lenses and polycameras," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 1172–1178, October 2000.

[28] F. Servant, P. Houlier, and E. Marchand, "Improving monocular plane-based slam with inertial measures," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (Taipei, Taiwan,), pp. 3810–3815, 2010.

[29] A. J. Davison, *Models and State Representation in Scene: Generalised Software for Real-Time SLAM*, 2007.

[30] C. Commons, "Attribution noncommercial sharealike 3.0." http://creativecommons.org/licenses/by-nc-sa/3.0/, 2015.

[31] T. I. S. E. GmbH, "The imaging source." http://www.theimagingsource.com/products/oem-cameras/usb-cmos-color/dfm22buc03ml/, May 2015.

[32] InvenSense Inc., 1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A., *Digital-output, 3-axis MEMS gyroscope*, 1.4 ed., March 2015.

[33] Analog Devices, One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., *Digital Acceleromter 3-axis*, 0 ed., 2009.

[34] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[35] T. I. S. Europe, *USB CMOS - 22, 42 and 72 Series - Trigger and I/O*, July 2014.

A Summary of Work done

Table A.1: The project planning schedule.

Time	Task
Feb. 2 − Mar. 2	Literature Study.
Mar. 3 − Mar. 30	Kinematic state estimator design.
Mar. 11 − Mar. 28	Hardware and hardware communication design.
Apr. 1 − Apr. 2	Simulation Suite for kinematic estimator.
Apr. 3 − Apr. 24	Kinematic state estimator implemented in Scenelib (C++).
Apr. 3 − Apr. 14	Inertial measurement unit and camera implementation.
Apr. 25 − May. 14	Integration of subsystems and main system.
May. 1 - May.15	Subsystem testing
May. 16 - May. 26	Reporting

B Project Specification

Project Title
Monocular vision based SLAM using kinematic state estimation.

Project Description
The following project aims to provide a real-time algorithm that is able to track and recover the trajectory and motion of a single (monocular) camera system - that is also aided by an inertial measurement unit - moving freely in 3D space while simultaneously constructing a sparse map of its (previously unknown) surrounding environment. This is one particular example of the simultaneous localisation and mapping (SLAM) problems. This particular version is known as the monocular SLAM (MonoSLAM) problem. Possible extensions of this project include utilising and/or comparing other existing implementations of SLAM algorithms and drawing a comparison between them.

Project Objectives
This project seeks to utilise the aforementioned MonoSLAM system of Davison et al. and improve the localisation thereof by using *additional* sensor information. The improvement(s) of the system should allow the existing system to obtain better localisation and extend the range of applications upon which the system can be applied while maintaining the original performance standard: repeatable localisation at 30 Hz for approximately 100 features. The system should utilise a *single* camera as the measurement sensor and preferably support real-time operation. All processing regarding the SLAM algorithm can be done on a standard PC that communicates with the sensors via a serial port. The project budget is R 1500.00. The primary objectives of this project include:

- **Performing an overview on the current techniques used to realise SLAM**
  In order to understand how SLAM is implemented, an understanding of probability theory and state estimation is required. These concepts - specifically recursive state estimation and the Bayes filter - need to be researched and analysed before choosing a suitable technique to implement.

- **Analysis of the kinematic estimator as an alternative motion model**
  A kinematic estimator is suggested as a alternative motion model. The kinematic estimator needs to be researched, mathematically derived and simulated. The results from the simulation should correspond to the mathematical derivation. The advantages and disadvantages of the kinematic estimator also need to be investigated.

- **Hardware Design of the System**
  The kinematic estimator requires additional measurements obtained from an IMU. The IMU is required to interface with the PC via a micro-controller. The micro-controller allows precise synchronisation between the images sampled by the camera and the IMU measurements. A functional diagram of the system's hardware components are depicted in Figure 5.

- **Comparison between proposed and original MonoSLAM implementations**
  If time allows, a set of tests are required to be derived and implemented to establish whether the proposed improvement(s) indeed satisfy the desired requirements.

C Achieved ECSA Exit Level Outcomes

Table C.1: The project planning schedule.

Time	Task
Feb. 2 − Mar. 2	Literature Study.
Mar. 3 − Mar. 30	Kinematic state estimator design.
Mar. 11 − Mar. 28	Hardware and hardware communication design.
Apr. 1 − Apr. 2	Simulation Suite for kinematic estimator.
Apr. 3 − Apr. 17	Kinematic state estimator implemented in Scenelib (C++).
Apr. 3 − Apr. 14	Inertial measurement unit and camera implementation.

D Theoretical Concepts

D.1 State Space Model

As previously discussed, the EKF requires a state transition model in order to estimate the current state of the system. In short, the motion model describes the transition from the previous state to the following state with regard to the robot's kinematic motion as well as the control inputs. The *ideal* motion model in this particular instance can be described through a **linear** differential equation of the following form:

$$\dot{\mathbf{x}}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t, \tag{D.1}$$

where the state matrix $\mathbf{A}$, describes the manner in which state evolves from the previous timestep to the current timestep without the influence of noise and controls, the input matrix $\mathbf{B}$, describes how the control vector $\mathbf{u}_t$ evolves from the previous timestep to the current timestep and $\mathbf{w}_t$ is a **zero-mean** Gaussian process representing the process noise with a covariance matrix $\mathbf{R}_w$.

Considering that the EKF is a recursive, numerical evaluation, it is necessary to convert the previously defined continuous model into its discrete counterpart. Various methods of discretisation exist, though this specific implementation makes use of the forward difference/Eulers method. This method *approximates* the derivative for a state for a sampling period ΔT as follows:

$$\begin{aligned} \dot{\mathbf{x}}_k &= \lim_{\Delta T \to 0} \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\Delta T} \\ \Delta T \dot{\mathbf{x}}_k &\approx \mathbf{x}_{k+1} - \mathbf{x}_k \end{aligned}. \tag{D.2}$$

The state estimate of the discrete counterpart at the following sampling instance, namely $k+1$, is then presented as follows (given a small enough sampling instance ΔT):

$$\mathbf{x}_{k+1} = \big(\mathbf{I} + \mathbf{A}\Delta T\big)\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \Delta T + \mathbf{w}_k \Delta, \tag{D.3}$$

where $\big(\mathbf{I} + \mathbf{A}\Delta T\big) = \mathbf{A}_d$ is the discrete state matrix, $\mathbf{B}\Delta T = \mathbf{B}_d$ is the discrete input matrix and $\mathbf{w}_k \Delta T = \mathbf{w}_{d,k}$ is the discrete input process noise.

Ultimately, the form of the final difference equation describing the system at each individual sampling instance is given as follows:

$$\mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d \mathbf{u}_k + \mathbf{w}_{d,k}. \tag{D.4}$$

D.2 State Transition: Linear Model

In order to derive the motion model for the system at hand, it is vital that the certain characteristics of the system be understood. Firstly, the robot system is comprised of a monocular camera and an attached IMU package. Secondly, the camera is to be considered as a 6 DOF rigid body. Briefly the six DOF describe the camera's three *translational* and three *rotational* degrees of freedom.

We therefore set out to define a kinematic motion model - using Newton's laws of motion - to describe the cameras movement through the environment as a result of initially unknown, external inputs to the system. Lastly, it should be stressed that embedded within the motion model, should be the impacts of uncertainty through both internal and external factors. It must also be stressed that initially, a stochastic, linear discrete-time model is adopted to approximate the motion model. Using the kinematic equations of linear and angular motion, it is aimed to ultimately and complete the previously defined state space model. We begin by describing all relevant states and control inputs:

$$\begin{aligned} \mathbf{x}[k] &= \begin{bmatrix} x_k & y_k & z_k & \dot{x}_k & \dot{y}_k & \dot{z}_k & q_{0,k} & q_{1,k} & q_{2,k} & q_{3,k} \end{bmatrix}^T \\ \mathbf{u}[k] &= \begin{bmatrix} \ddot{x}_k & \ddot{y}_k & \ddot{z}_k & \dot{q}_{0,k} & \dot{q}_{1,k} & \dot{q}_{2,k} & \dot{q}_{3,k} \end{bmatrix}^T \end{aligned}, \tag{D.5}$$

and extend the discrete-time difference equation describing the system to incorporate the motion model,

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k + \mathbf{w}_{d,k}. \\ \mathbf{A}_d &= \begin{bmatrix} 1 & 0 & 0 & \Delta T & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = (\mathbf{I} + \mathbf{A}\Delta T\big), \\ \mathbf{B}_d &= \begin{bmatrix} \Delta T & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \Delta T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \Delta T \end{bmatrix} = \mathbf{B}\Delta T, \end{aligned} \tag{D.6}$$

$$\mathbf{w}_{d,k} = \mathcal{N}(0, \mathbf{R}_w) = \begin{pmatrix} \mathbf{n}_{\mathbf{a}_t,k} \\ \mathbf{n}_{\omega_t,k} \end{pmatrix} = \mathbf{w}_{d,k}\Delta T.$$

It can be observed from the model above that the motion model adheres to the forward method of discretisation derived in equation **??**.

D.3 Measurement Update

D.3.1 Measurement Model

The measurement model models the uncertainty regarding a measurement taken at any given time instance $\mathbf{z}_t$, given that the locations of both the robot as well as the location of the landmarks are known. This uncertainty can be described in the following form:

$$p(\mathbf{z}_t \mid \mathbf{x}_t) = \frac{1}{\sqrt{|2\pi\mathbf{R}_v|}} \exp\left\{\frac{1}{2}\big[\mathbf{z}_t - \mathbf{h}(\bar{\boldsymbol{\mu}}_t) - \mathbf{H}_t^{x_t}(\mathbf{x}_t - \bar{\boldsymbol{\mu}}_t)\big]^T \mathbf{R}_v^{-1}\big[\mathbf{z}_t - \mathbf{h}(\bar{\boldsymbol{\mu}}_t) - \mathbf{H}_t^{x_t}(\mathbf{x}_t - \bar{\boldsymbol{\mu}}_t)\big]\right\}, \tag{D.7}$$

where $\mathbf{H}_t$ represents the Jacobian of the observation model and $\mathbf{R}_v$ is the sensor noise.

The correction step of the EKF aims to ultimately correct the previously estimated robot pose and landmark position through sensor measurements. With regard to the implementation proposed in this project, these measurements are obtained through the use of a camera. The measurement process generally involves a measurement estimate that incorporates uncertainty.

With reference to figure 9, a feature's cartesian position can be described through a cartesian vector $\mathbf{h}_i^W(\bar{\boldsymbol{\mu}})$, where the feature's cartesian **point** is shown in relation to the camera's centre:

$$\mathbf{h}_i^W(\bar{\boldsymbol{\mu}}) = \mathbf{R}^{CW}\big(\mathbf{y}_i^W - \mathbf{r}^W\big) = \left(\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \mathbf{r}^W\right) \tag{D.8}$$

the subscript i corresponds a directional vector $\mathbf{h}^C$ from its cartesian position $\mathbf{r}^W$ to the cartesian position of a given landmark $\mathbf{y}^W$.

A camera however, cannot directly measure a cartesian vector. Instead, a camera measurement (based on the model presented) obtains a vector $\mathbf{h}_i$ that is a function of $\mathbf{h}_i^W$. This vector describes a given feature's horizontal and vertical image positions (u, v). For an undistorted image, the vector $\mathbf{h}_i$, more commonly referred to as the measurement function is defined according to Equation 3.6:

$$\mathbf{h}_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} u_0 - fk_u \dfrac{h_{i,x}^R}{h_{i,z}^R} \\ v_0 - fk_v \dfrac{h_{i,y}^R}{h_{i,z}^R} \end{pmatrix} \tag{D.9}$$

where u_0 and v_0 represent the principal point and fk_u and fk_v are the camera calibration parameters described in Chapter 3.3.3.

It is evident from the model presented in equation **??** cannot be directly inverted to obtain a feature's position. The projection of a feature onto the camera's image plane removes any information required to directly obtain the depth of the feature.

D.3.2 Feature Matching

The following section discusses the measurement of a feature *fully* initialised within the SLAM map. The measurement process seeks to initially estimate the cartesian position of a given feature $\mathbf{y}_i$ within the SLAM map. Thereafter, the feature can be compared via a matching sequence. Generally, feature matching is conducted using a normalised cross-correlation search, where a 2D template of the 3D feature is scanned is across the entire image (at each pixel location) until a peak is obtained. MonoSLAM however, seeks to utilise an *active* approach for matching, minimising the the search field and improving efficiency.

The EKF inherently contains information that may be utilised in order to prohibit a full cross-correlation search. The measurement function $\mathbf{h}(\bar{\mu})$ for instance, provides an estimate for a given features location, namely $\mathbf{u}_d = (u_d, v_d)$. Knowledge of this location therefore allows an active search region to be described within the vicinity of this location. The location estimate of the feature is not the only information regarding the feature that is available as a result of the EKF. Additionally, the uncertainty regarding a given feature's location is stored within the state vector covariance matrix $\mathbf{\Sigma}_t$. This information can be used to determine the size of the active search region surrounding the location estimate; where the size of the search region is directly proportional to the uncertainty of its location. If the feature cannot be matched within the aforementioned search region, it cannot contribute to the correction of the robot's pose estimate and is therefore deleted form the SLAM map. The aforementioned process of defining the active search region can be mathematically defined through the *innovation covariance matrix* $\mathbf{S}_i$:

$$\mathbf{S}_i = \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{x}_v} \Sigma_{\mathbf{x}_v,\mathbf{x}_v} \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{x}_v}^T + \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{x}_v} \Sigma_{\mathbf{x}_v y_i} \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{y}_i}^T + \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{y}_i} \Sigma_{y_i \mathbf{x}_v} \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{x}_v}^T + \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{y}_i} \Sigma_{y_i y_i} \frac{\partial \mathbf{u}_{d,i}}{\partial \mathbf{y}_i}^T + \mathbf{R}_v \tag{D.10}$$

The symmetric 2×2 matrix $\mathbf{S}_i$ represents a 2D Gaussian PDF around the estimated image coordinate. The innovation covariance matrix can then be used to determine an active region the a given feature should lie within. Typically, the active search region is defined to confine within 3 standard deviations (3σ) of the mean.

Furthermore, the innovation matrix provides a measure of the amount of content expected within an eventual actual measurement $\mathbf{z}_i$. In the event that many potential measurements are available, features containing a higher $\mathbf{S}_i$ present the EKF with more information regarding the camera's position. Candidates for feature estimates are thus chosen according to those that present the most information regarding the position estimate. Feature searches per sampling instance are generally limited (usually about 12 features) due to computational constrains.

Finally, as described in [**?**], an active search will always reduce the area of the template matching search region at the potential *additional* cost of calculating the reduced search region.

D.3.3 Feature Initialisation

The inherent disadvantage of a monocular camera, as previously mentioned, is the inability to immediately provide an estimate for the depth of a feature. As a result, a given feature is required to be observed at various viewpoints before its depth can be approximated through a multiple view triangulation. Instead, Davison et al. presents an alternative approach whereby a feature is initialised to lie along an infinite 3D line. This line, originating from the position at which the camera is estimated, extends indefinitely in the direction of the feature. The depth of the feature lies somewhere along the aforementioned line. This depth can be modelled as a uniformly distributed set of discrete depth hypothesis. Briefly, the feature's depth can be interpreted as a 1D probability density, represented only by particle distribution instead. The feature is can then *partially* initialised in the SLAM map as follows:

$$\mathbf{y}_{pi} = \begin{pmatrix} \mathbf{r}_i^W \\ \hat{\mathbf{h}}_i^W \end{pmatrix} \tag{D.11}$$

where $\mathbf{r}_i^W$ represents the origin of the line and $\hat{\mathbf{h}}_i^W$ is a unit vector representing its direction. The uncertainty describing the aforementioned entities are Gaussian in nature.

After a feature has been partially initialised, it can be assumed that the feature is re-observed and that each additional observation improves the depth estimate. The particle filter based depth estimation process itself is to a large extent complex, and is explained in more detain in [**?**]. Intuitively, the depth estimation process can be explained as follows: each particle in the particle set is projected into the image and subsequently matched across each observation. The resulting observations transform the initially uniformly distributed depth probability into one that better resembles a Gaussian density. Once the depth covariance is below a certain threshold, the depth is approximated with a Guassian probability density. Thereafter a feature becomes *fully* initialised, assigned with a standard 3D Gaussian representation.

D.4 Performance of the EKF-based Kinematic Estimator

This set of test conditions will provide *randomly* simulated gyroscope and accelerometer measurements **with** simulated process and measurement noise. The EKF simulation measurement model is still defined as a mapping of the state vector with additive noise. The EKF should now behave as described in Chapter 2.4.2 while maintaining the performance of the kinematic state estimator.

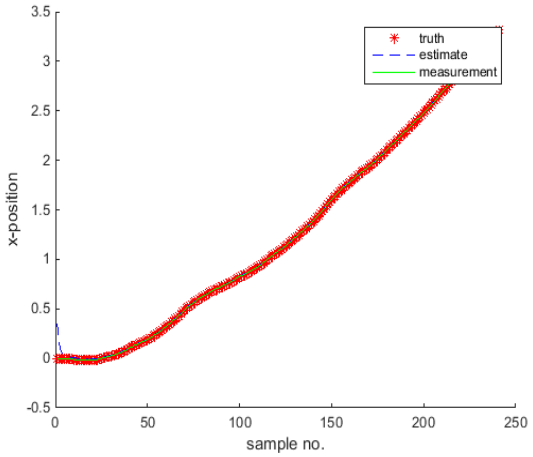
The expected outcome then, is that the kinematic estimator will track the given robot trajectory according to the uncertainty of both the measurement and process noise. Intuitively, the EKF should provide a state estimate that is dependent upon the process and measurement noise e.g. it the system has a high measurement noise uncertainty but a low process noise uncertainty, the EKF should provide a state estimate that depends less on the measurements.

D.5 Additional Analysis

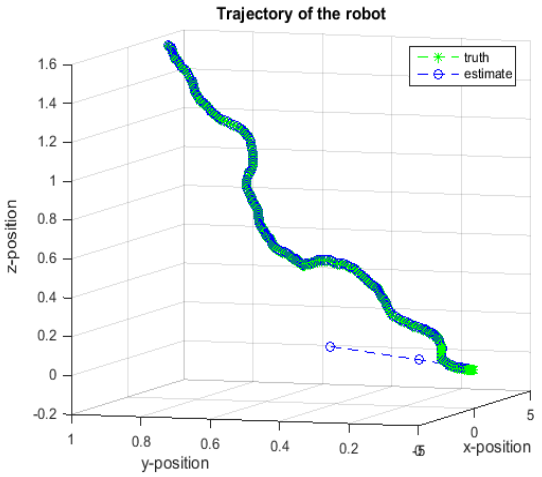
1. **No uncertainty: Process noise variance = 0 and measurement noise variance = 0**
   This simulation depicted in Figure **??** shows a trajectory estimate that directly resembles the true trajectory. This is due to the fact that there is no uncertainty regarding the process or the measurement noise and the EKF can accurately estimate the actual state of the system through modelling. Figure **??** confirms that both the measurements and estimates agree are in correspondence with the states. This particular case though is only possible in simulation and practically impossible as every practical system incorporates uncertainty.

2. **Equal yet small uncertainty regarding the measurement and process noise**
   This simulation depicted in Figure **??** shows a trajectory estimate that is noisy but generally tracks the actual trajectory. This behaviour corresponds to the aforementioned prediction that the EKF is not entirely certain regarding both the measurements and equivalent control inputs and subsequently provides an estimate that incorporates information from both factors. Figure **??** shows that the state estimate incorporates uncertainty regarding both the measurements and the equivalent control inputs. Because these uncertainties are small though, a relatively accurate reconstruction of the actual trajectory can be estimated.

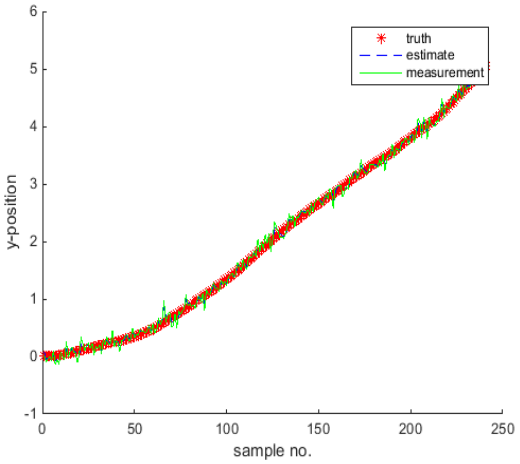
3. **Less certain process noise: Process noise variance = large and measurement noise variance = small**
   This simulation depicted in Figure **??** shows a trajectory estimate that is noisy and heavily incorporates the measurements. This behaviour corresponds to the aforementioned prediction that the EKF very uncertain regarding the equivalent control inputs and subsequently provides an estimate that incorporates more information from measurements. Figure **??** shows that the state estimate incorporates more information regarding the measurements. Because the measurement uncertainty isn't very small, the EKF cannot provide an accurate approximation of the true trajectory. This is due to the fact that the EKF "trusts" the measurements far more than the equivalent control inputs, but because the measurements themselves contain a substantial amount of uncertainty, the best estimate that the EKF can provide is itself uncertain.



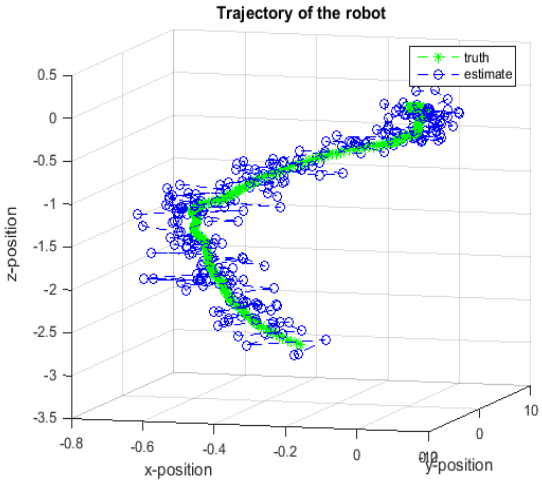
(a) X-position of random control inputs with zero-uncertainty.



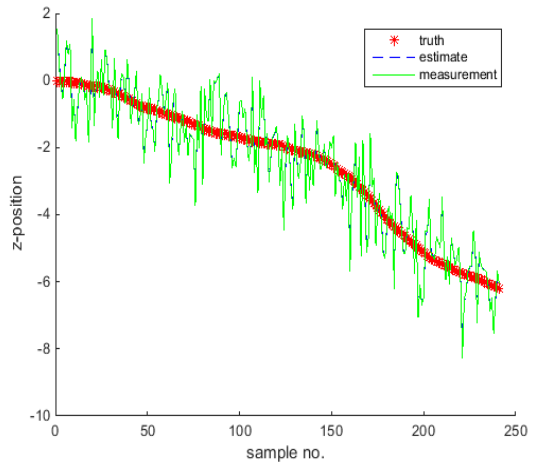
(b) Trajectory of random control inputs with zero-uncertainty.



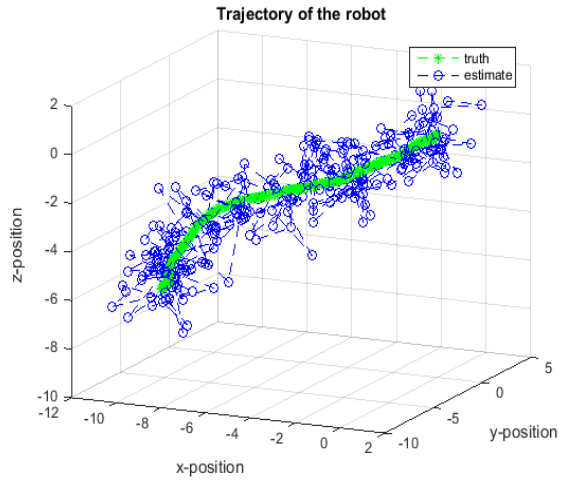
(c) Y-position of random control inputs with a small, equal measurement and process uncertainty.



(d) Trajectory of random control inputs with a small, equal measurement and process uncertainty.



(e) Z-position of random control inputs with greater process uncertainty.



(f) Trajectory of random control inputs with greater process uncertainty.

Figure 25: Results of the conditions depicting random equivalent control inputs with uncertainty.

E Figures & Diagrams

E.1 Schematics & Circuit Diagrams

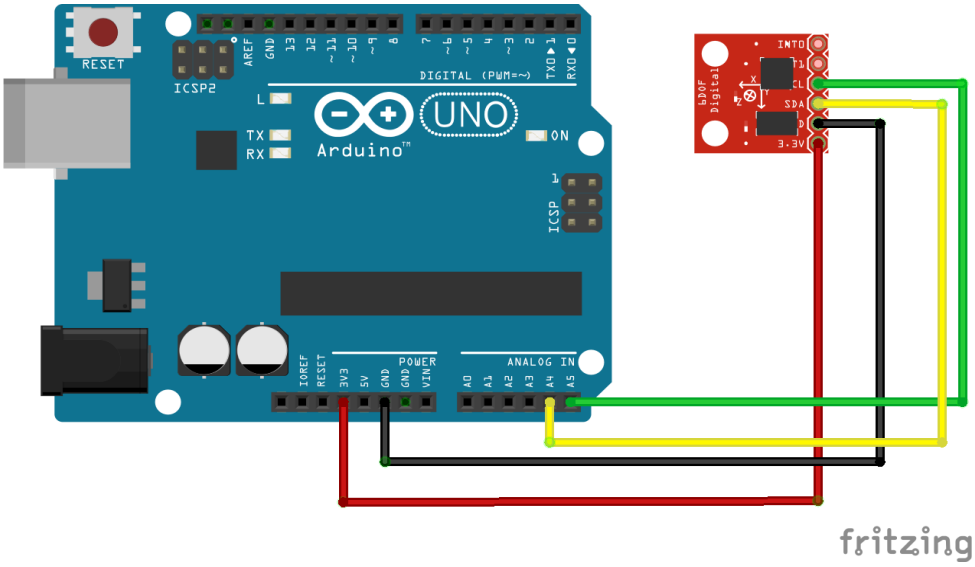


Figure 26: Circuit Diagram of the IMU.

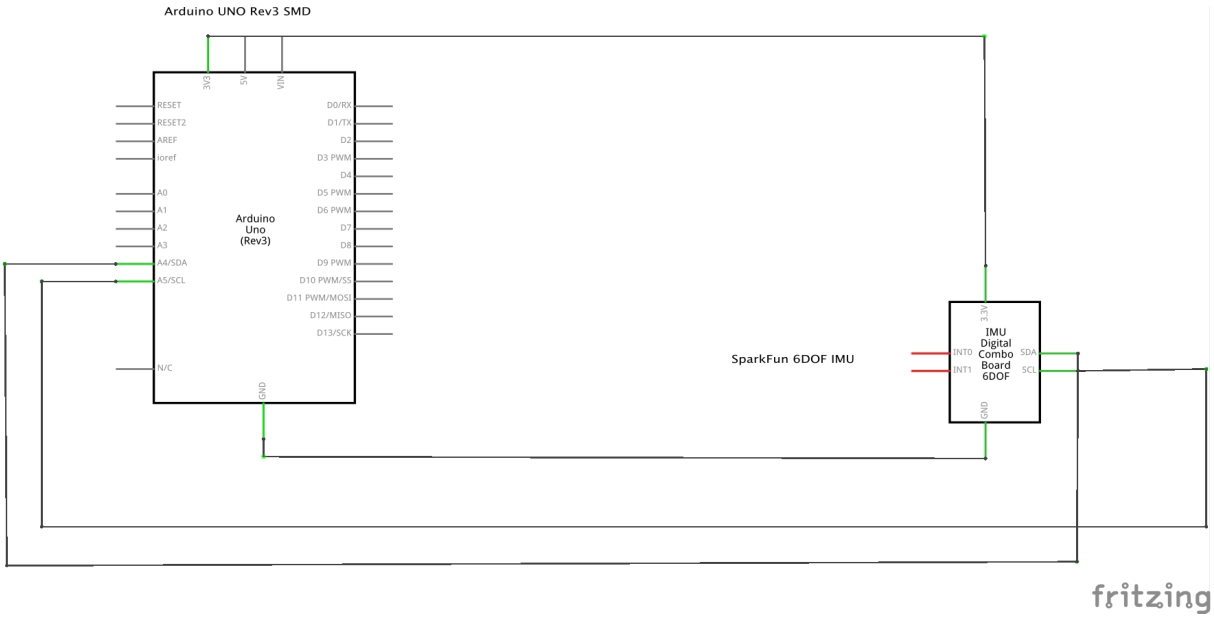


Figure 27: Schematic of the IMU.

These images were created by Fritzing (http://fritzing.org) [**?**].