Imperial College London

MEng Individual Project

Imperial College London

Department of Electrical and Electronic Engineering

State Estimation for Position Tracking in a Novel Bipedal Robot

Author:
Joseph Gardner

Supervisor:
Dr. Petar Kormushev

Second Marker:
Prof. Paul Mitcheson

August 12, 2020

Abstract

Position tracking or localisation is a necessary component of modern freely moving robotics systems. Despite the widespread need for this key functionality, the problem remains a difficult one with no universal solution available. As a single sensor is always insufficient, multiple sources of information from different sensors must be fused in order to provide an accurate location estimate at a useful speed. In this report, a state estimator making use of an inertial measurement unit (IMU) and an RGB-D (depth) camera is presented as a solution to this problem on a robot called SLIDER, a bipedal robot that uniquely uses prismatic (sliding) joints instead of knee joints. The estimator is able to run at 1kHz and diverges by less than 3cm over 50 seconds of walking in simulation and serves as a foundation for a more complex estimator.

Acknowledgements

I would first and foremost like to thank my supervisor Dr. Petar Kormushev for offering me this project, providing support to me throughout and giving much needed direction when the way forward with the project was unclear. I would also like to thank the members of the Robotic Intelligence Lab for consistently being available to talk to regarding any issues I had throughout, particularly Ke Wang. I thank the staff of the Department of Electrical and Electronic Engineering for providing the opportunity and organisation to do this project and for accommodating the changes that needed to be made due to the current global situation.

Contents

Chapter 1

Introduction

1.1 Motivation

Robotics is a fast evolving industry with applications in numerous fields, making the development of state-of-the-art algorithms to enable them ever more important. It is common to use bipedal designs to attain both a degree of human-likeness so that robot is more socially acceptable [1] and a very large range of physical capabilities. A robot has a significantly bigger range of use cases if it can navigate difficult terrain just like a human. SLIDER, detailed later, is a bipedal design that is intended to be used in this way.

Important variables or *states* of a robotic system are the location of the robot's torso or centre of mass and its velocity. Accurate tracking of these states can enable both a more reactive and stable control scheme (provided the states are high frequency) as well as tracking of the robot's position for navigation (i.e. path planning, mapping etc.). This is not a universally solved problem, and different researchers and companies use different algorithms and sensors to solve it in their specific case. This itself is part of a wider general problem known as state estimation.

State estimation is a field that aims to accurately estimate the current state (i.e. values of variables describing the state) of the robot in real-time using noisy sensor measurements and a system model. In order for advanced robot control systems to be able to successfully compensate for environmental, mechanical or other such disturbances, they must have access to vital information about the current state [2]. Information from just a single sensor alone is not enough due to noise, and the model of the system will be unable to account for unexpected issues that may not be captured by the sensor. It is therefore essential for the stable operation of SLIDER, that an estimator which fuses multiple complementary measurement sources be completed to enable both advanced control schemes and location tracking. There is a variety of state estimation algorithms that make use of different mathematical concepts to try and solve this problem, and an appropriate one must be chosen, as well as the correct sensors that complement each other.

1.2 Objectives

As outlined, the important states that need to be tracked are the torso's location and velocity; they must be tracked however such that they are useful for controlling the walking of the robot. The aim of this project is therefore to create software that can provide estimates of these states in real-time at high frequency with minimal delay. SLIDER uses a common software framework in the field known as the robot operating system, or ROS. It provides useful abstraction tools for designing real-time systems, allowing one to create various "nodes" (essentially python or C++ scripts) that run simultaneously and communicate with each other via "topics" (a messaging framework). The software must be written to operate as a ROS node and easily connect to sensor data streams present within SLIDER.

As access to the lab was limited at the time of the project, the software may only be tested in simulation. This means error in the estimate can be much more easily measured (ground truth is provided by the simulation), however the testing environment and sensors will not be as complex as the real world, meaning error will most likely be underrepresented. This issue cannot be solved, but the estimator should be designed in such a way that it can easily take simulated or real sensors as input, so that it may be quickly transferred onto the real robot once the lab is available.

1.3 Report Outline

In the following chapter, background material in this field will be reviewed and discussed in their relevancy to this project. This will include a brief look into the theory for state estimation, common solutions to the problem in the literature and existing competitors to SLIDER itself. After this, in the main three chapters the theoretical design of the estimator will be discussed, testing procedure explained, and results from the testing presented. The report will finish with an evaluation and conclusion discussing what the results show.

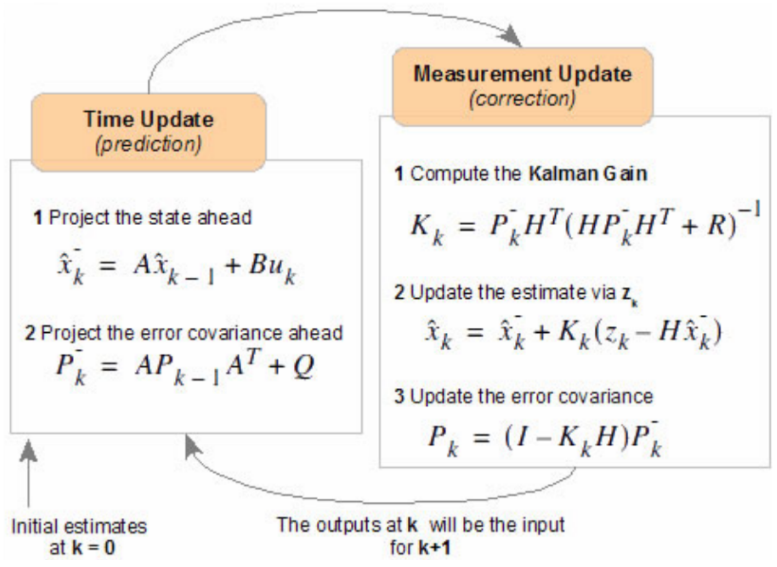
Chapter 2

Background

2.1 Brief Introduction to Existing Theory

2.1.1 Filtering

There has been a great degree of research into the field of state estimation. All current methods either use a batch smoothing method such as maximum a posteriori, or some form of Bayes filter, which utilise Bayesian statistics to model predictions and measurements as normally-distributed random variables with the conditional probability between them used to determine an estimate of the current state [3]. Several of these filters exist such as the Kalman filter, complementary filter, extended Kalman filter, unscented Kalman filter and particle filter. Their main benefit is speed and iterative structure, allowing them to be run in real-time.
The Kalman filter is an optimal estimator for a linear system, provided some noise values are known. It requires a state space model of the entire system in order to function, and computation speed grows at the rate $O(N^3)$. It is however, much more accurate than other simpler, non-probabilistic adaptive methods [4]. The algorithm consists of two key steps: predict and correct, shown below.



Time Update
(prediction)
1 Project the state ahead
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
2 Project the error covariance ahead
$P_k^- = AP_{k-1}A^T + Q$
Measurement Update
(correction)
1 Compute the Kalman Gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
2 Update the estimate via z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
3 Update the error covariance
$P_k = (I - K_k H)P_k^-$
Initial estimates at k = 0
The outputs at k will be the input for k+1

Figure 2.1: Steps for Kalman filtering [5]. Note: capital letters are matrices and lowercase letters are vectors, the standard bold convention is used in the text

Step 1 shows the general state space model for the system. **A** is the state transition matrix and **B** is the control input matrix, mapping the previous state and control inputs to the next state using a model chosen by the user. Varying complexities for these are used, from simpler linear inverted pendulum models (LIPM) to fully-body dynamics modelling [6].
$\mathbf{P}_k$ is the covariance of the estimate of state $\mathbf{x}_k$ (the hat above $\mathbf{x}_k$ in the picture means it is an

estimate) and $\mathbf{Q}$ is the covariance of the inherent noise in the prediction process. A projection of the new covariance is used to calculate a value known as the Kalman gain. Simply put in 1 dimension it is:

$$\frac{error\ in\ estimate}{error\ in\ estimate + error\ in\ measurement}$$

which will vary from 0 to 1, dependent on whether the measurement or prediction has less error. The equation to calculate Kalman gain $\mathbf{K}_k$ in figure 2.1, is an extension of this to multivariate statistics. The Kalman gain scales the difference between the the current sensor measurements $\mathbf{z}_k$ and the predicted sensor measurements based on the predicted state. Matrix $\mathbf{H}$ is the mapping from state $\mathbf{x}$ to measurement $\mathbf{z}$, again using models of the physical system. This scaled difference in measurement is added to the predicted state giving a result somewhere between the predicted and measured state. As this is an iterative process the Kalman gain is constantly updated and only ever requires the previous state, rather than the entire history of states. This recursion also makes it very easy to implement in code [6].

However, the Kalman filter alone only works for linear systems as it assumes a Gaussian distribution for the state estimate, and applying a Gaussian distribution to a non-linear function will lead to a non-Gaussian output distribution, making the covariance irrelevant [7]. Naturally in the real mechanical systems of real robots there is a lot of non-linearity, for example calculating orientation angles. This means in practice, extended Kalman filters are more often used. They contain the added step of linearising the state transition functions before use in projecting the covariance and computing the Kalman gain. Functions are linearised by calculating their Jacobian matrices [2][7].

2.1.2 Sensors

There exist several types of sensors commonly used to obtain useful measurements to fuse in a Kalman or extended Kalman filter. These can be divided into two categories: proprioceptive (those that measure values related to the internal state of the robot), and extroceptive (those that measure values related to the robot's environment). Examples of both are detailed below.

1. *Proprioceptive sensors*

    - **Inertial measurement units (IMUs)**: These generally contain a 3-axis gyroscope (measuring angular velocity), accelerometer (measuring linear acceleration) and sometimes magnetometer (measuring direction of magnetic field). These are widely used to obtain the absolute or relative orientation of robot torsos, dependant on whether there is a magnetometer or not to calibrate the yaw angle.
    - **Encoders**: Encoders output information about the movement of a joint. Rotary encoders can measure the angular position of motor, either relative to its starting point when powered on (incremental) or relative to a fixed point on the encoder (absolute). Linear encoders are similar but measure position along an axis, making them required for feedback on a prismatic (sliding) joint.
    - **Force sensors**: The most common form of force sensor is a strain gauge load cell, a passive component whose resistance changes based on the pressure force exerted on it. It is common for robots to have these in their feet to aid with detecting foot contact and the centre of pressure (CoP) [6].

2. *Extroceptive sensors*

    - **LIDAR**: A rotating laser and detector that measure the distance to surrounding objects by measuring the reflections of laser light off their surfaces. They are widely used to build up an image of a robot's surroundings to aid simultaneous localisation and mapping (SLAM).
    - **RGB-D cameras**: Cameras that in addition to a standard image, output the distance to each point on the image from the camera. These are widely available in the form of Microsoft Xbox Kinect cameras.

2.1.3 Implementations in literature

In reality, numerous different sensors and techniques have been used to obtain the most accurate state estimate possible. Both proprio- and extroceptive sensors have been used to get very accurate position estimates for navigation [8]. It has been shown that without such extroceptive sensors, the yaw angle and global position are unobservable [9]. However all other internal states can be accurately observed, allowing for balance control with purely proprioception.

As bipedal robots (including SLIDER) use the concept of a zero-moment point (ZMP) to generate a stable trajectory [10], it is common for the centre of mass and pressure to be tracked. There has been some success using simple LIPMs with only an IMU in the robot's torso as sensory input to the filter, and direct encoder angles from every joint for use in forward kinematics [2]. In [11] however, noise on the encoders is modelled and incorporated into the Kalman filter to further improve accuracy. Full-body dynamics have also been utilised in force-controlled robots to improve accuracy further from a simple LIPM [12]. It has been shown that this can be decoupled into a torso and joint state estimator [13] to improve computational speed.

As well as simply tracking the robot's state, fall detection (and prevention) is an important aspect of bipedal robots. Earlier attempts utilise the torso pitch angle and its derivative, and check if they have crossed an empirically determined threshold [14]. Other methods include the use of the concept of a capture point [15], the point where a humanoid must step in order to avoid a fall, to successfully prevent falls, as demonstrated by team WPI-CMU at the DARPA robotics challenge in 2015 [13].

It has also been shown that multiple IMUs attached to every joint on a robot can be used to improve estimates of their velocities and position [6][16]. Approaches that do not use a Kalman filter also exist, where a customised non-linear optimisation algorithm is derived [17] [18]. These methods however involve significantly more complex mathematics that is outside the scope of a master's course.

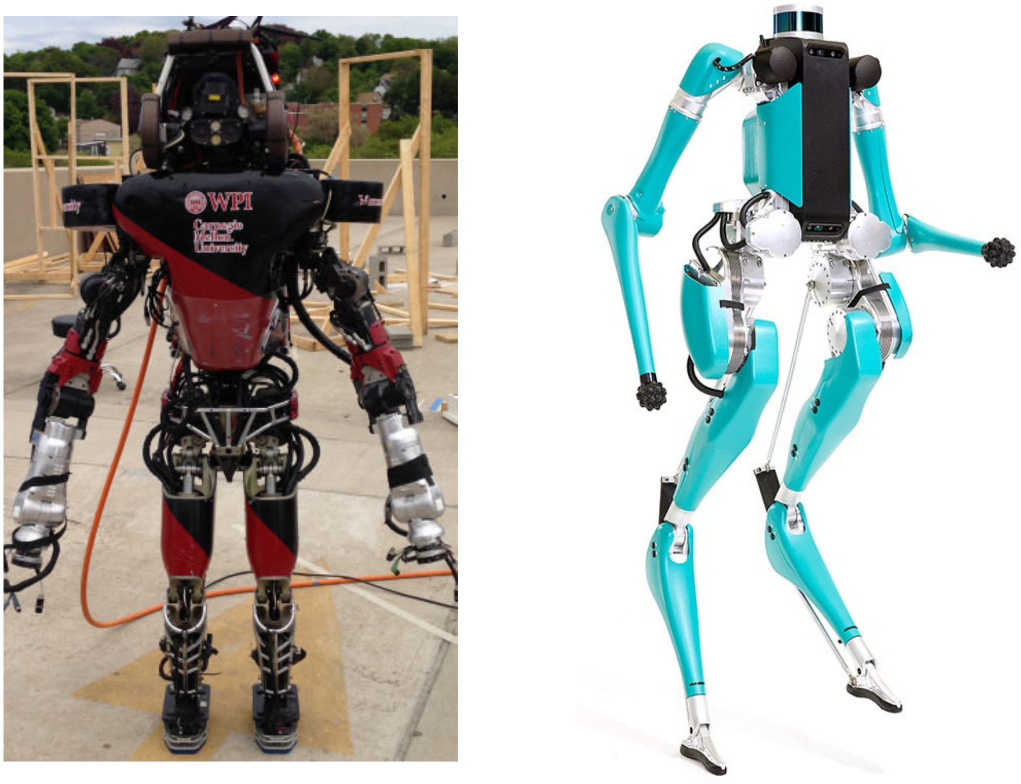


Figure 2.2: WPI-CMU's atlas robot and Agility Robotics' digit [6] [19]

2.2 Competitors

Currently there exist a number of competitors to a robot like SLIDER. An ubiquitous example of this would be Boston Dynamics' Atlas robot, already capable of running and performing parkour (and other intensive activities) [20]. Another example is Agility Robotics' Digit [21], which has already been taken up by Ford to pair with self driving cars for a fully automated logistics solution [22]. These examples show that bipedal robotics is moving forward fast, and SLIDER must be at least be able to walk stably (and perhaps even run) to gain a competitive edge over these other robots. Unfortunately no details are currently available on the state estimation of these systems, as expected, however a slightly older robot of Agility Robotics, Cassie, is widely used in research [23]. This allows for some gleaning of more advanced techniques to ideally improve estimation further.

2.3 Current Hardware and Software

SLIDER is a robot whose first prototype has already been built by the Robot Intelligence lab in the Department of Design Engineering of Imperial College London [24]. It has been designed specifically as a cheaper and more energy efficient bipedal robot by comparison to others on the market (such as those mentioned previously) which are often prohibitively expensive. The main factor in achieving this is the removal of traditional knee joints, replacing them with a prismatic joints based in the hip, as can be seen in figure 2.3. This makes the legs significantly lighter as they no longer contain the weight of the actuators. In addition to this, the robot is built in-house out of 3D printed materials, further reducing both cost and weight. It is expected that it will be easier for SLIDER to perform more agile movements due to these factors, giving it an advantage relative to competitors.
SLIDER is currently fitted with an inertial measurement unit (IMU) in its torso, as well as rotary encoders in every joint except the prismatic ones. Currently, force sensors are also being installed in the feet to allow for calculation of the CoP for balance control. These sensors offer a solid foundation to begin work on, however more sensors may be added (such as an IMU network and linear encoders for the prismatic joints) as improvements are progressively made to the estimator. In addition an RGB-D camera is available for use on SLIDER. As the estimator will be tested in simulation, the chosen sensors must also be available on the simulation platform.

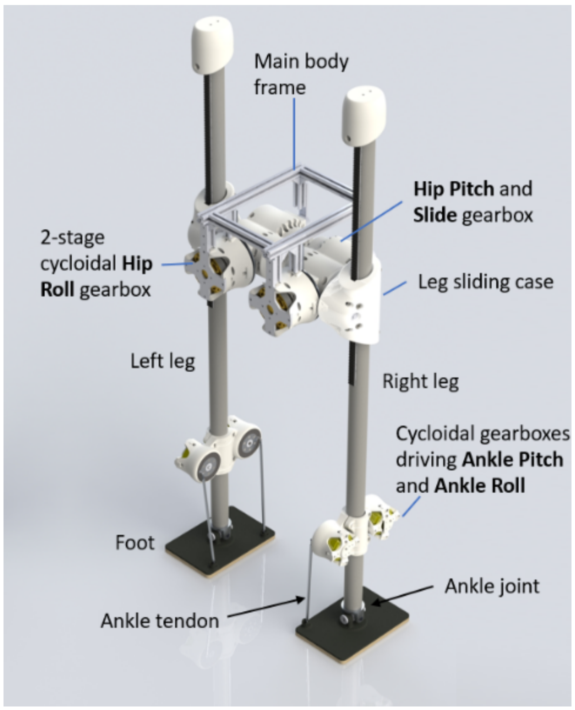

Main body frame
Hip Pitch and Slide gearbox
2-stage cycloidal Hip Roll gearbox
Leg sliding case
Left leg
Right leg
Cycloidal gearboxes driving Ankle Pitch and Ankle Roll
Foot
Ankle joint
Ankle tendon

Figure 2.3: The robot SLIDER

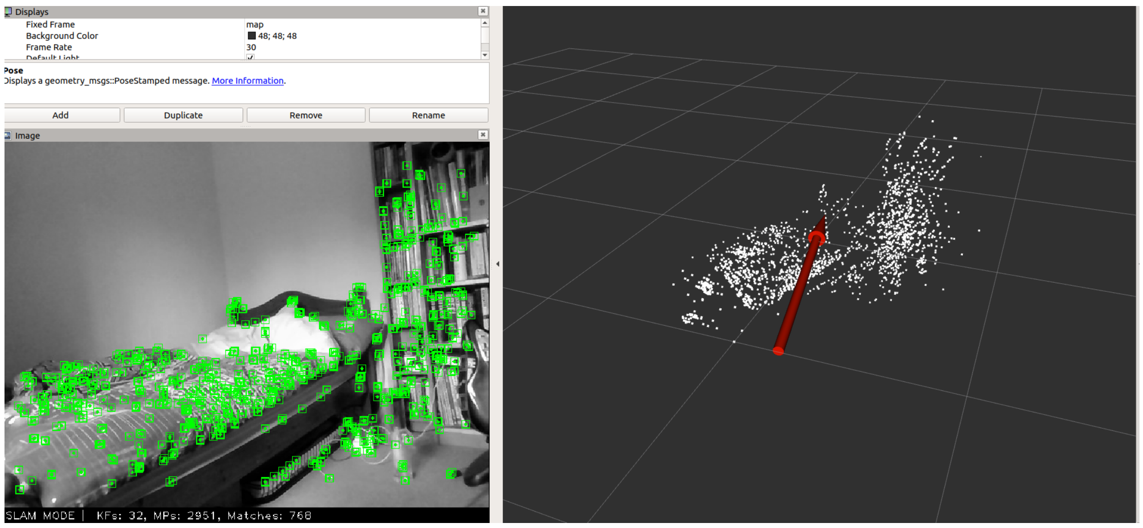
Chapter 3

Analysis and Design

3.1 Selection of Sensors

On the robot SLIDER there is an inertial measurement unit (IMU) available, specifically the LORD Microstrain 3DM-GX4-25 (figure 3.1). This type of sensor contains a 3-axis gyroscope, accelerometer and magnetometer. Use of IMUs for pose tracking is widely documented . This is because global orientation (orientation with respect to the Earth gravity and magnetic field) can be accurately calculated with these three sensors and used to rotate the acceleration measurements into the global frame of reference. It can then be integrated twice over time to obtain the relative position of the robot from the starting point.
However there are numerous issues caused by doing this. Noise present in the accelerometer will be integrated over time as well as any sensor bias (i.e. a constant offset), causing significant drift over long timescales [25]. The IMU runs very fast at 1kHz, meaning that despite this downside it is still very accurate over short timescales. Therefore it would be ideal to fuse the data with another sensor that can correct for errors accumulated from the IMU over time. The yaw angle estimate can also drift over time in the presence of magnetic interference.
As basic RGB-D cameras are readily available and cheap, the ORB-SLAM2 algorithm [26] is a viable choice. It is a visual feature-based simultaneous localisation and mapping (SLAM) algorithm and therefore also generates a map of the surrounding environment as the pose of the robot is tracked. It runs only at 20Hz, much too slow for being utilised in a control system, but is very accurate. It is a good fit for being fused with the IMU, as the two sensors' characteristics complement each other. Figure 3.2 shows the visualisations generated by the algorithm.

Figure 3.1: IMU available on SLIDER's torso



Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓

Pose
Displays a geometry_msgs::PoseStamped message. More Information.

```
SLAM MODE |  KFs: 32, MPs: 2951, Matches: 768
```

Figure 3.2: ORB-SLAM2 visualisation, camera view with feature markers on the left, pose (red arrow) on the right within a visualisation of the room created with white markers

3.2 Basic Fusion in ROS

3.2.1 Theory

At a most basic level, if ORB-SLAM2 is assumed to be completely accurate (at least by comparison to the IMU), the two sensors can be fused by calculating position based on the IMU data and then setting the position and velocity to that of the ORB-SLAM2 every time a new sample arrives. By doing this, essentially the gaps between the ORB-SLAM2 estimates are interpolated using the IMU. In order to do this, first a set of state equations must be established for predicting the position with the IMU. A state space representation is used here as it will be useful for extending the algorithm later to use a Kalman filter. Here $\mathbf{p}^W(t)$ is used to represent position $\mathbf{p}$ in the world frame W at time t. In addition the velocity $\mathbf{v}^W(t)$ and acceleration $\mathbf{a}^W(t)$ must be included as states for the motion model. Δt is the time period between samples and $\mathbf{g}$ is acceleration due to gravity, i.e. $[0 \quad 0 \quad -9.81]^T$.

$$\mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{3.1}$$

$$where : \mathbf{x} = \begin{bmatrix} \mathbf{p}^W \\ \mathbf{v}^W \\ \mathbf{a}^W \end{bmatrix}$$

$$\mathbf{p}^W(t+1) = \mathbf{p}^W(t) + \Delta t \mathbf{v}^W(t) \tag{3.2}$$

$$\mathbf{v}^W(t+1) = \mathbf{v}^W(t) + \Delta t \mathbf{a}^W(t) \tag{3.3}$$

$$\mathbf{a}^W(t+1) = \mathbf{R}^{WB}\mathbf{a}^B(t) \; - \; \mathbf{g} \tag{3.4}$$

The term $\mathbf{R}^{WB}$ represents the rotation matrix from the body frame to the world frame and is obtained directly from the IMU from on board fusion. The term $\mathbf{a}^B(t)$ is the accelerometer data from the sensor. As can be seen in the equations 3.2 - 3.4, the sensor data is used to directly calculate the instantaneous acceleration (through rotation and subtraction of gravity) which is then essentially integrated twice using basic equations of motion.
With the ORB-SLAM2 algorithm, pose estimates are received directly, allowing $\mathbf{p}^W$ to be set like so:

$$\mathbf{p}^W(t) = \mathbf{p}^W_{ORB}(t) \tag{3.5}$$

Error clearly also accumulates in the velocity estimate, so the velocity must also be set by the ORB-SLAM2 measurement. As it is not directly available it must be calculated crudely using the difference between the current and previous measurement.

$$\mathbf{v}^W(t) = (\mathbf{p}^W_{ORB}(t) - \mathbf{p}^W_{ORB}(t-1))/\Delta t \tag{3.6}$$

3.2.2 ROS Implementation

This initial algorithm was implemented in C++ as a single ROS node. The simulated IMU data is available on the topic */slider_gazebo/imu*, generated by a pre-existing simulation of SLIDER created in Gazebo [27]. A simulated RGB-D camera was added to the simulation to provide an input for the ORB-SLAM2 algorithm, which runs on its own node, receiving camera data from a specified topic and broadcasting pose estimates on the */orb_slam2_rgbd/pose* topic. For visualisation and error tracking purposes, an additional node was created which handles the ground truth and ORB-SLAM2 data, generating visualisations for rviz (the visualisation tool provided in ROS) and calculating the error between the estimate and ground truth (as well as the ORB-SLAM2 and ground truth). A diagram of the layout of the nodes is shown in figure 3.3. The simulated sensors and ground truth data can be seen to be broadcast by Gazebo and used by each node involved in the estimator. */robot_state_publisher* can be ignored.

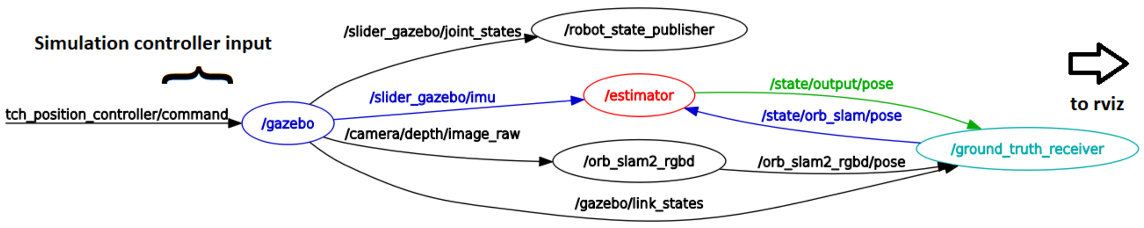


Figure 3.3: Visualisation of ROS nodes using the *rqt_graph* tool; connections to the estimator node are highlighted with inputs in blue, outputs in green

The estimator node is subscribed to the IMU data topic and the ORB-SLAM2 data topic sent from the ground truth receiver. The callback function for the IMU data implements equations 3.2 - 3.4 to project ahead the position with each sample. The callback function handling the ORB-SLAM2 topic implements equations 3.5 and 3.6, correcting the drift that may have occurred in the meantime. The Δt terms are measured dynamically to ensure the highest accuracy possible in the equations. The visualisations produced by *ground_truth_receiver* are the path of the position estimate, the path of the ground truth and a marker array plotting the individual ORB-SLAM2 position measurements.
The handling of the measurement data as callback functions to ROS topics allows the two sensor sources to operate asynchronously, making it very easy to set up. The algorithm manages to comfortably run at 1kHz, as there is only a small amount of mathematics involved. Results will be presented and discussed in the following chapter.

3.3 Kalman Filtering

The previous algorithm can be theoretically improved further by taking into account the error in the two measurement sources and dynamically adjusting the the level of each that is used. The Kalman filtering technique outlined earlier uses conditional probability and the assumption of Gaussian white noise on sensors to iteratively calculate a gain factor for fusing the two sources. By using this framework, the fusion should be more accurate as there no longer is the assumption that the ORB-SLAM2 measurements are perfect, creating a smoother fusion between the two sources (as a linear combination of the two is now used, rather than only one or the other), especially in situations where the uncertainty of the IMU is close to that of ORB-SLAM2. The covariance matrix of the state estimate is also estimated by the filter, which allows the reliability of the estimate to be known. For example, if the ORB-SLAM2 measurements are unavailable for a period of time, covariance will appropriately grow. In addition to this, the Kalman filter setup allows for further states and measurements to easily be added for future improvements, due to the matrix form of the algorithm.
The general Kalman filter equations discussed earlier are now adapted to work for this system. The IMU measurements are used as input to the "prediction" step of the algorithm, as shown below. The IMU error is expressed separately to the state as the vector $\mathbf{e}$ so that process noise matrix $\mathbf{Q}$ is simply the covariance matrix of the accelerometer. $\mathbf{e}_a$ is the accelerometer noise and $\mathbf{y}_a$ is the accelerometer data ($\mathbf{y}_a \neq \mathbf{a}^B$ anymore due to the presence of noise).

$$\mathbf{x}(t+1) = \mathbf{F}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{G}\mathbf{e}(t) \tag{3.7}$$

$$where: \mathbf{x} = \begin{bmatrix} \mathbf{p}^W \\ \mathbf{v}^W \\ \mathbf{a}^W \end{bmatrix}, \quad \mathbf{e}(t) = \mathbf{e}_a(t), \quad \mathbf{u}(t) = \mathbf{y}_a(t)$$

State equations:

$$\mathbf{p}^W(t+1) = \mathbf{p}^W(t) + \Delta t \mathbf{v}^W(t) \tag{3.8}$$

$$\mathbf{v}^W(t+1) = \mathbf{v}^W(t) + \Delta t \mathbf{a}^W(t) \tag{3.9}$$

$$\mathbf{a}^W(t+1) = \mathbf{R}^{WB}\mathbf{y}_a(t) + \mathbf{R}^{WB}\mathbf{e}_a(t) \; - \; \mathbf{g} \tag{3.10}$$

this gives the following matrices, where $\mathbf{I}_3$ is the 3x3 identity matrix and $\mathbf{0}_3$ is the 3x3 zero matrix:

$$\mathbf{F} = \begin{bmatrix} \mathbf{I}_3 & \Delta t \mathbf{I}_3 & \frac{\Delta t^2}{2}\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \Delta t \mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{0}_3 \\ \mathbf{0}_3 \\ \mathbf{R}^{WB} \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{R}^{WB} & \mathbf{0}_3 \end{bmatrix}$$

for use in equation 3.7 and:

$$\mathbf{P}(t+1) = \mathbf{F}\mathbf{P}(t)\mathbf{F}^T + \mathbf{G}\mathbf{Q}\mathbf{G}^T \tag{3.11}$$

$$where: \mathbf{Q} = \Sigma_a \quad (covariance\ matrix\ of\ \mathbf{y}_a)$$

Equations 3.7 and 3.11 are performed every time there is an IMU measurement sample (in the IMU topic callback function). As the IMU is much higher frequency than the ORB-SLAM2, this step runs much more frequently, about 50 times before the following "measurement step" is performed. The measurement step setup is the following:

$$\mathbf{z} = \begin{bmatrix} \mathbf{p}^W_{ORB} \\ \mathbf{v}^W_{ORB} \end{bmatrix} \tag{3.12}$$

$$\mathbf{y} = \mathbf{H}\mathbf{x}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \tag{3.13}$$

and the equations to implement are:

$$\mathbf{K}(t) = \mathbf{P}(t)\mathbf{H}^T(\mathbf{H}\mathbf{P}(t)\mathbf{H}^T + \mathbf{R})^{-1} \tag{3.14}$$

$$\mathbf{x}(t) = \mathbf{x}(t) + \mathbf{K}(t)(\mathbf{z} - \mathbf{H}\mathbf{x}) \tag{3.15}$$

$$\mathbf{P}(t) = (\mathbf{I} - \mathbf{K}(t)\mathbf{H})\mathbf{P}(t) \tag{3.16}$$

Equations 3.14 - 3.16 are carried out by the ORB-SLAM2 callback function in the estimator node. They update the state vector with the measurements scaled by the Kalman gain $\mathbf{K}(t)$ and update the covariance $\mathbf{P}$(t) to its appropriate value based on these changes.

In terms of node interconnections, this algorithm is implemented in the same way. The contents of the callback functions however needed to be heavily modified to accommodate the filter. The Eigen 3 library [28] was made use of heavily to store the matrices and perform operations on them. The matrices $\mathbf{Q}$ and $\mathbf{R}$ needed to also be determined. As there are no values for variance available for either ORB-SLAM2 or the simulated IMU, they were calculated while the estimator is running. While SLIDER is on, the variance of the IMU accelerometer is directly measured from the data every 0.1 seconds. The variances of $\mathbf{p}^W_{ORB}$ and $\mathbf{v}^W_{ORB}$ are also measured, however only when the robot is at rest. This is because the error of the ORB-SLAM2 becomes overrepresented when the robot is moving. The windows over which these variances are measured was determined by trial and error, arriving at a window size of 0.5 seconds (500 IMU samples or 10 ORB-SLAM2 samples). Further investigation could be made into these parameters, however this was not the main focus of the project.

$$\mathbf{R} = \begin{bmatrix} \Sigma_{p^W_{ORB}} & \mathbf{0}_3 \\ \mathbf{0}_3 & \Sigma_{v^W_{ORB}} \end{bmatrix} \tag{3.17}$$

3.4 Linear Interpolation

In addition to these two fusion algorithms, linear interpolation of the ORB-SLAM2 will be tested. Linear interpolation does not involve any IMU measurements and simply just extends the gradient created by the previous two ORB-SLAM2 samples until a new sample arrives. Another way of looking at it is that it takes the velocity $\mathbf{v}^W_{ORB}$ and uses it to predict the next position instead of using IMU measurements. The position estimates are spaced at 1kHz to match the IMU frequency.

Measurement step:

$$\mathbf{p}^W(t) = \mathbf{p}^W_{ORB}(t) \tag{3.18}$$

$$\mathbf{v}^W(t) = (\mathbf{p}^W_{ORB}(t) - \mathbf{p}^W_{ORB}(t-1))/\Delta t \tag{3.19}$$

Prediction step:

$$\mathbf{p}^W(t+1) = \mathbf{p}^W(t) + \Delta t \mathbf{v}^W(t) \tag{3.20}$$

Chapter 4

Testing

4.1 Environment

As mentioned previously, a Gazebo simulation of the robot will be used to carry out tests. The walking motion provided by the Robot Intelligence Lab consists of a short walk forward a few metres followed by standing still for a few seconds. This motion repeats indefinitely. The model of the robot is placed inside a model of a lab available in Gazebo to provide enough features for the ORB-SLAM2 to function correctly (figure 4.1). As this environment is relatively simple, there may be implications on what can exactly be gleaned from this testing, which will be discussed later in the evaluation.

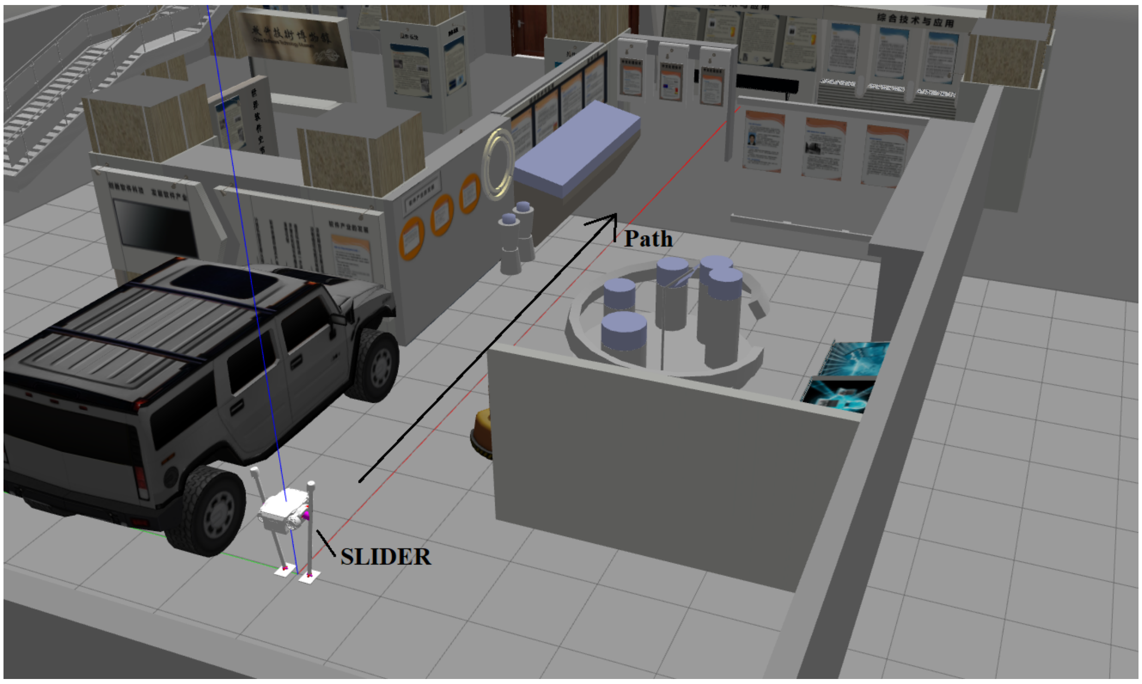


Figure 4.1: Testing environment

4.2 Visualisations

A number of visualisations will be used to provide a sense of an algorithm's performance before plotting the error from the ground truth. ROS itself contains a powerful visualisation tool known as "RVIZ" which allows particular message types to be displayed in a 3D simulation environment. For this testing, the following message types are required:

- *geometry_msgs::PoseStamped*: contains a "pose" which is a 3-dimensional position coordinate and a quaternion representing orientation (a 4-D number). It also contains a timestamp. It is used to show the current pose of the robot, visualised as an arrow with the base at the current position and the arrow head pointing at the current orientation.

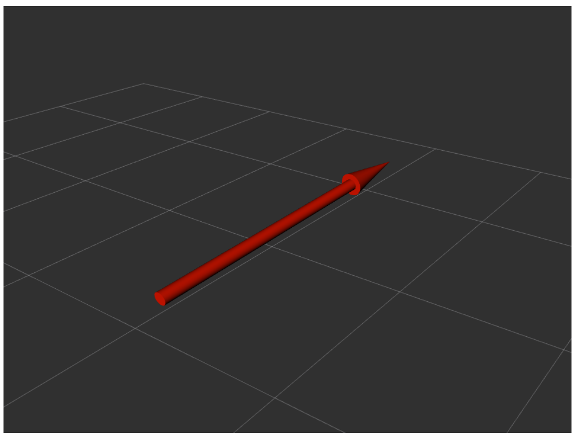


Figure 4.2: Pose arrow visualisation

- *nav_msgs::Path*: a path is an array of poses, all the past poses plus the current one. It visualises the path taken by the robot as a line (that is straight between adjacent poses), which is very useful for quickly evaluating the performance of an algorithm, especially at the 1kHz of the estimator as the line will appear smooth. Path messages are transmitted by the code less frequently than the pose in order to save processing power.

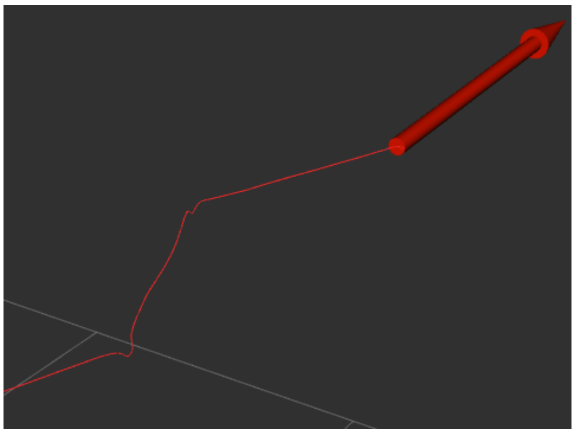


Figure 4.3: Ground truth path visualisation with arrow at current pose

- *visualization_ msgs::MarkerArray*: an array of 3-D points, similar to a path however does not draw a continuous line and instead draws an individual marker for each data point. Useful for visualising ORB-SLAM2 estimates as they are low frequency.

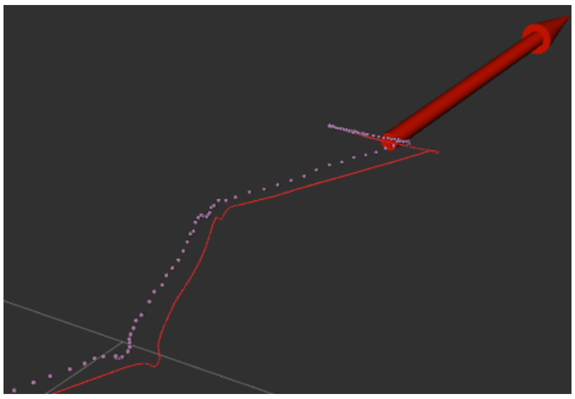


Figure 4.4: ORB-SLAM2 as a marker array alongside ground truth path and pose

4.3 Method of Comparison

In order to quantify the performance of each algorithm, the error must be measured and compared. The mean square error per sample was used for this purpose, which is the same as the mean of the squared "Euclidean distance" between each ground truth position sample and the current estimated position sample. The ground truth data runs at 1kHz in order to match the frequency of the estimates. The mean is used to so that comparisons across different frequencies can be made, for example the ORB-SLAM2 alone is only 20Hz so the error would be underrepresented without this. $\mathbf{p}_{gt}$ below is the ground truth position.

$$e_{MSE}(t) = \frac{1}{t}\sum_{i=0}^{t} ||\mathbf{p}(i) - \mathbf{p}_{gt}(i)||^2 \tag{4.1}$$

In addition to this, the current error from the ground truth can also be used. This does not quantify the performance in a single number but allows the variance of the algorithm over time to be observed graphically. The expression for this is simpler as:

$$e(t) = ||\mathbf{p}(i) - \mathbf{p}_{gt}(i)|| \tag{4.2}$$

Graphs comparing these metrics across the three designed algorithms and the ORB-SLAM2 on its own will be produced and explained in the following chapter.

Chapter 5

Results

5.1 Data

The testing methods described in the previous chapter were carried out, plotting the error and mean square error of the three algorithms. The ORB-SLAM2 on its own was also tested, to evaluate the improvements on it made by the fusion algorithms. In the following plots, ORB-SLAM2 is shown as purple, linear interpolation as light blue, Kalman filtering as dark blue and basic fusion as red. The x axis in all these figures is in seconds and the y axis is the magnitude of error.

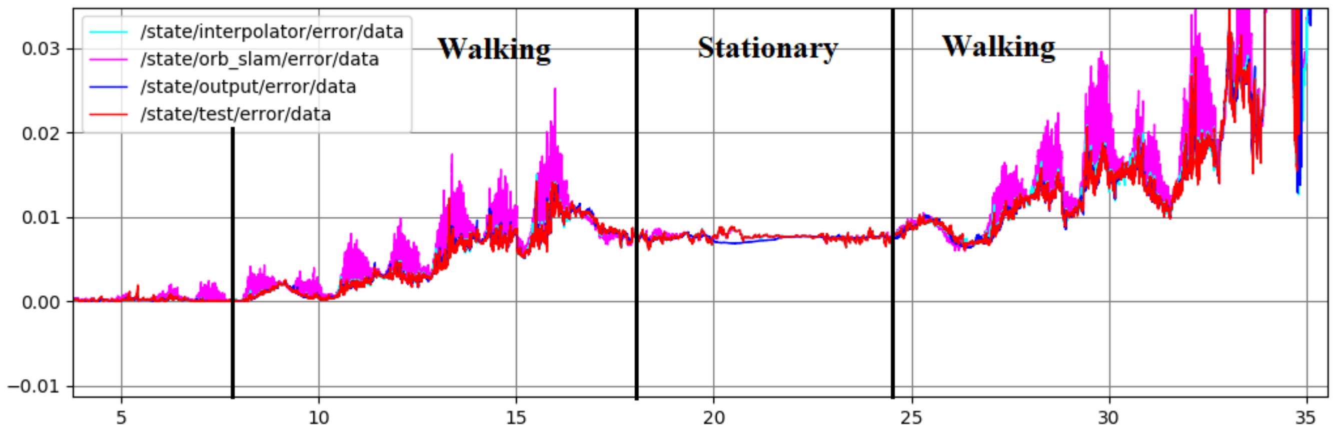


Figure 5.1: Realisation of the error against time for a straight walk

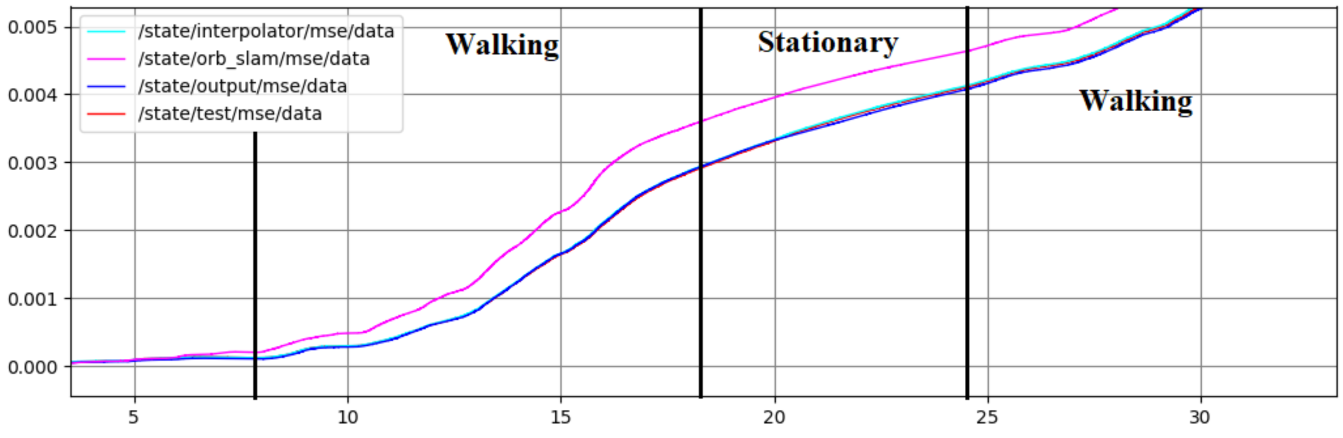


Figure 5.2: Corresponding mean square error against time

In figure 5.1 it can be seen that ORB-SLAM2 has much higher error than the fusion algorithms when SLIDER is walking which leads to a build up in mean square error observed in figure 5.2. As all three algorithms are comparatively much closer together, closer snapshots of the graphs must be investigated.

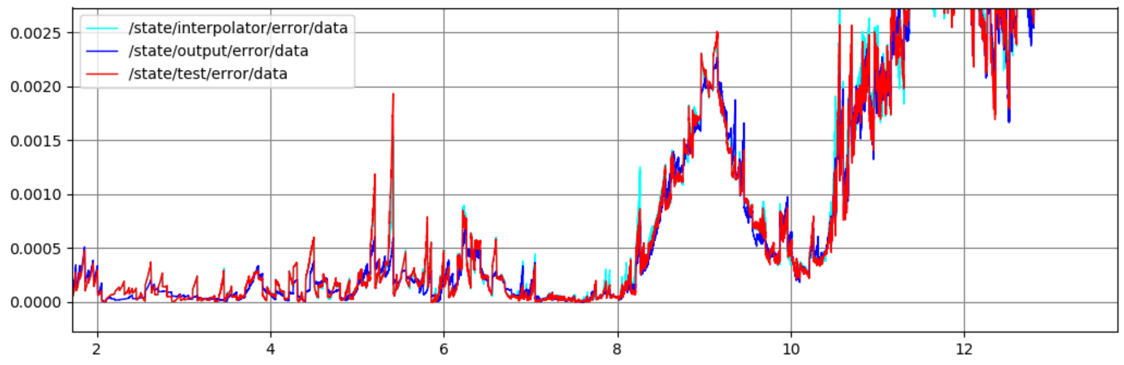


Figure 5.3: Closer zoomed graph of the error, without ORB-SLAM2 for clarity

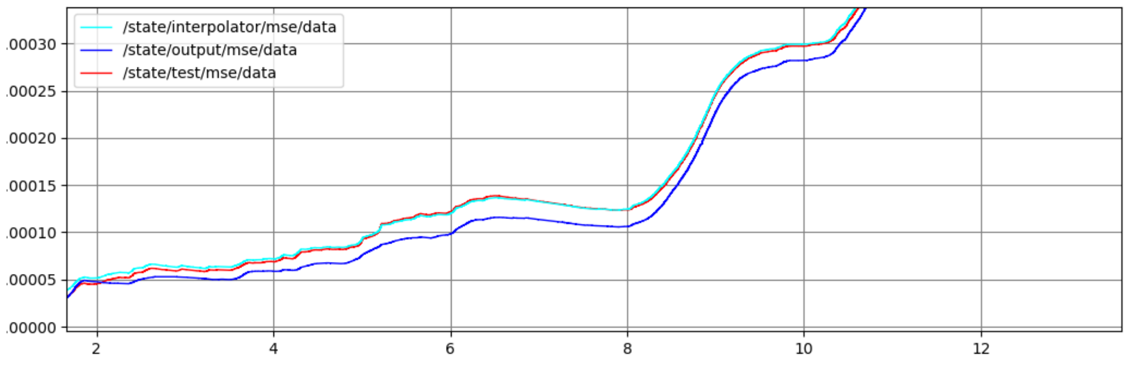


Figure 5.4: Corresponding mean square error

It can be seen that the Kalman filter performs slightly better than the others, maintaining a lower mean square error, which can also be observed in the error graph where there is less visible variance. However if a later segment of the graph is looked at, the improvement doesn't appear as strong.

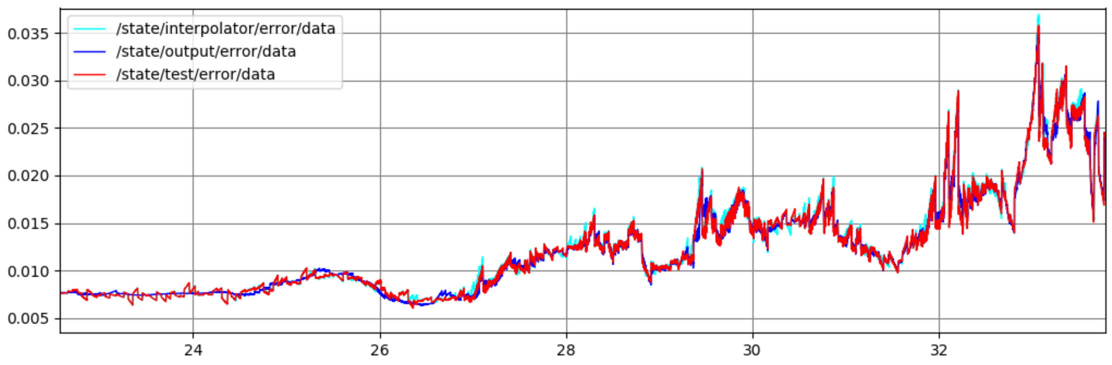


Figure 5.5: Later segment of the error

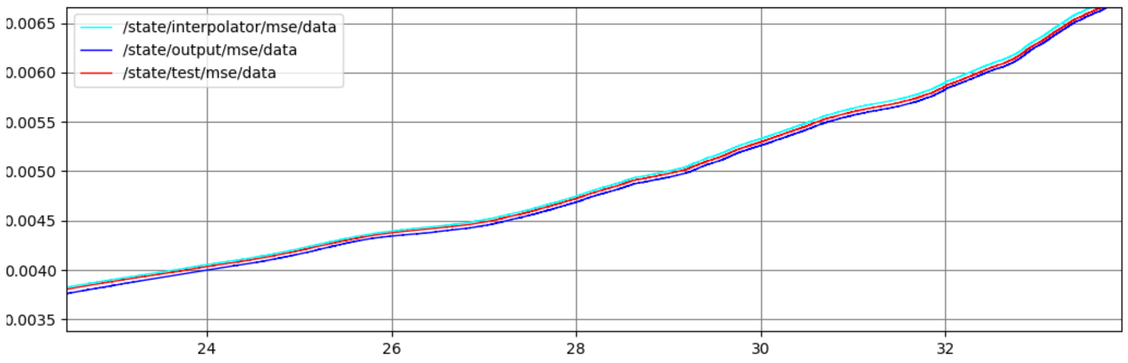


Figure 5.6: Corresponding mean square error

Upon zooming further into the stationary region, the difference between the algorithms becomes a little clearer. The Kalman filter is clearly much smoother with less variance, indicating it is more accurate (as the robot should not be moving). It can be seen however that its mean square error is not necessarily always lowest, as a crossover can be seen occurring at around 20 seconds.

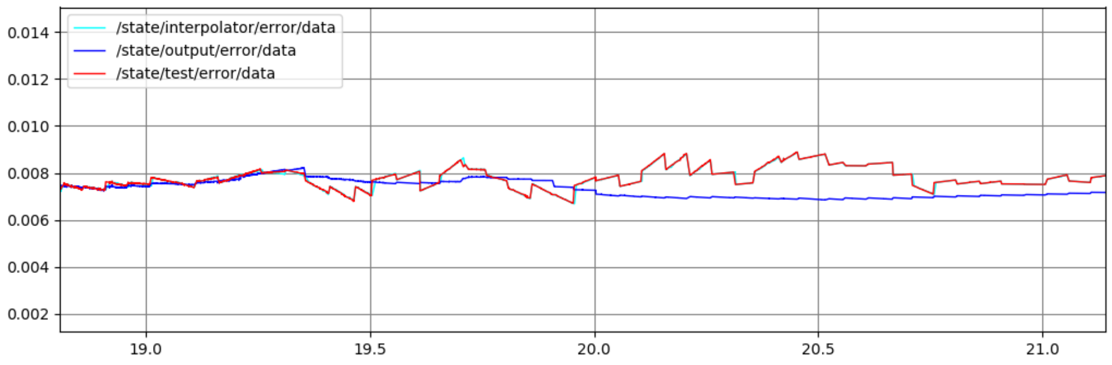


Figure 5.7: Close up of the error during the stationary phase

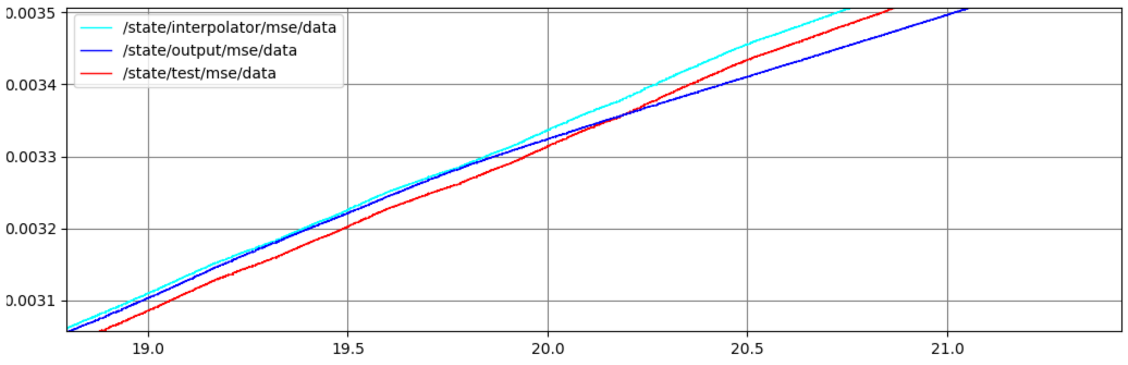


Figure 5.8: Corresponding mean square error

A second realisation of the simulation is shown below. It can be seen that in this realisation the Kalman filter does not have the lowest error. Note that the x axis labels for the error graphs are mislabelled, they are reading $t + 1370$ instead of just t.

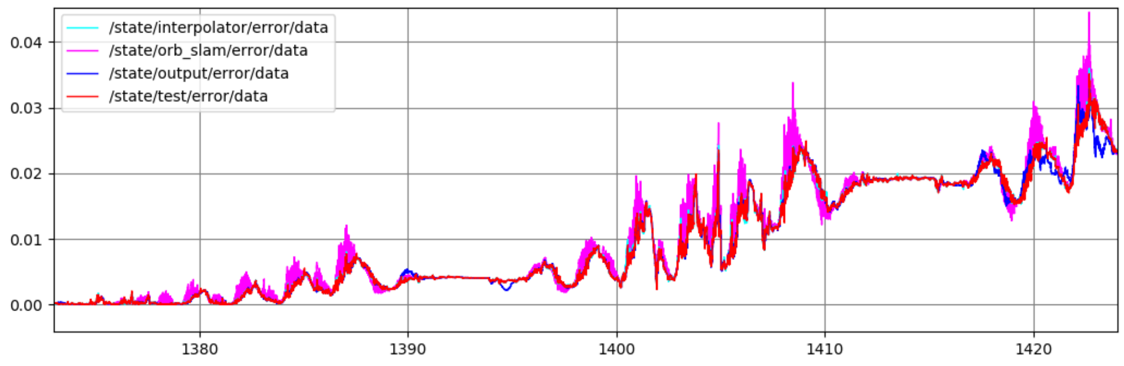


Figure 5.9: Second realisation of the error against time for a straight walk

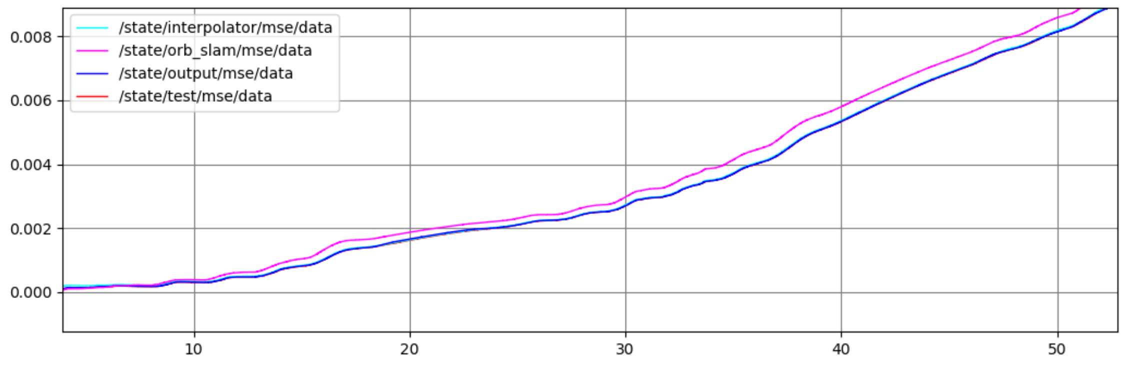


Figure 5.10: Corresponding mean square error

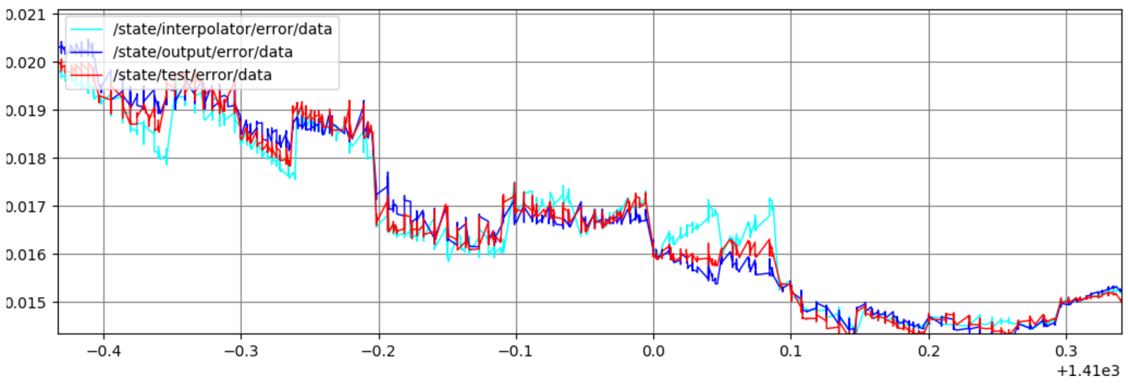


Figure 5.11: Close-up of the error during a walking phase

(further graph on next page)

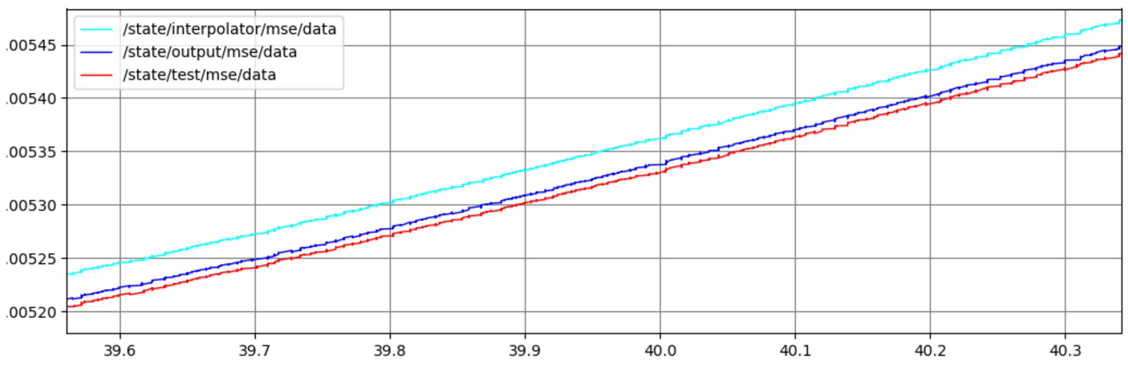


Figure 5.12: Corresponding mean square error

5.2 Analysis

The main point that can be observed is that the Kalman filter does not perform much better than the other simpler algorithms, contrary to what was expected. It can be seen from the smoothness of the position estimate during stationary periods that there is some degree of improvement. This particular finding can be justified from what was predicted earlier: during these rest periods the IMU variance is measured to be comparatively lower, resulting in fewer sudden jumps in position. Whilst the robot is moving however the ORB-SLAM2 has a significantly lower variance, causing the filter to behave almost identically to the basic fusion.
As well as this, the interpolation performs better than expected. This could be due to the movement of SLIDER being quite smooth; it walks in a straight line through a room with perfectly flat ground. One would expect the accuracy of interpolation to drastically fall if the segments between each ORB-SLAM2 sample are no longer close to straight lines. This analysis points towards further testing required in more complex environments. The tests that were carried out were done in this way due to the fast time to implement and compare multiple algorithms given the limited time and computing power available. An ideal step forward would be to transfer these algorithms to the real robot and continue similar tests out of simulation.
It can also be seen that the position error of the robot at the end of the walk is quite low, at roughly 2-3cm after 50 seconds in the second realisation of the simulation. This is a metric that will be useful for further refining of the algorithm and comparisons to other research as it is often mentioned.

Chapter 6

Evaluation

So far three different methods of position estimation have been designed, implemented and tested. From the analysis of the results, it seems clear that the use of an IMU is necessary if the error is to be minimised. However the theoretical Kalman filter advantages appear to not be completely realised, in the sense that it performed similar to without one. If other research is compared, often it seems that the kinematics of the robot are taken into account (with joint angles measured by rotary encoders) and used as a further constraint to the system, which is tested in [18]. Specifically here, the visual SLAM was integrated into the optimisation framework, which was not done in this project; instead only the output position was used. This tight integration into a "factor graph" optimisation allows their algorithm to better determine the relative accuracies of the sensors and achieve a more optimal fusion, although not perfect. Ideally at least the kinematics of the robot should be taken into account for situations where terrain is known. It is expected that this would reduce the variance of the estimate by a lot. Moving away from the Kalman filter to non-linear optimisation such as using factor graphs is possible, however it is difficult task due to the higher mathematical complexity involved.
In its current form however, the system is quite general. No assumptions are made about the geometry of the robot or environment. It could be argued that its current state is already quite useful as a baseline for any freely moving robotic system requiring a position estimate and a local map. The code design also allows easy modification based on the individual system. For SLIDER in particular however, it is most likely necessary to improve the system further as bipedal robots require precise control to be stable.
In addition, it is likely possible to obtain a more accurate position estimate in the prediction step of the filter by using a different method of numerical integration. The Runge-Kutta (RK4) method is known to better approximate integrals than a simple difference equation, like those used. As with incorporating kinematics, this could improve the periods between each ORB-SLAM2 estimate, therefore reducing the variance in the error overall.
The original aim of the project was to estimate the location of the centre of mass (CoM) of SLIDER. Since then, the available sensors to use has changed, causing the estimate to become focused more on global position. With addition of encoder sensors (allowing the relative location of limbs to be calculated) this position could be extended to also provide the CoM by considering the geometry of the robot. In terms of achieving a global position estimate however, the results are promising. Due to the low error reached per unit walked, the estimator appears to have a lot of potential to become very robust and accurate once these aforementioned improvements are made. The limitations of the testing methods discussed earlier are very important to consider however, and the most important factor is that further testing be performed on the real robot. In reality, noise will be greater, the ground will not be perfectly flat and ORB-SLAM2 may not always be available if there are not enough features, depending on what can be seen by the robot.

Chapter 7

Conclusion

In this report, several methods of varying complexity for fusing visual and inertial data for position tracking were evaluated. They were implemented in C++ using ROS, and powerful visual tools used to observe the estimates. Sensors used were chosen to complement one another and the experiments were performed in simulation, which changed the expected outcomes and testing methods from what was initially decided.
The largest hurdle in development of the project was the determination of ORB-SLAM2 error, which is not provided by the algorithm. The solution of measuring variance during rest periods allowed the algorithm to still perform to an acceptable level, however possible improvements can be made through a deeper understanding of the algorithm.
It was clear that the Kalman filter and basic fusion performed the best, however the differences between these two was minimal. Due to the design of the Kalman filter, it is relatively simple to expand and should be by including either further sensor data (encoders) or improving the prediction step with more accurate models (e.g. RK4). In addition, further tests in real life must be carried out to determine the reliability in various conditions. The project has succeeded in providing a state estimation baseline, which should be refined through further development.

Appendix A

Code GitHub Link

https://github.com/joeggg/FYP-State-Estimation-SLIDER

Appendix B

RVIZ Visualisations

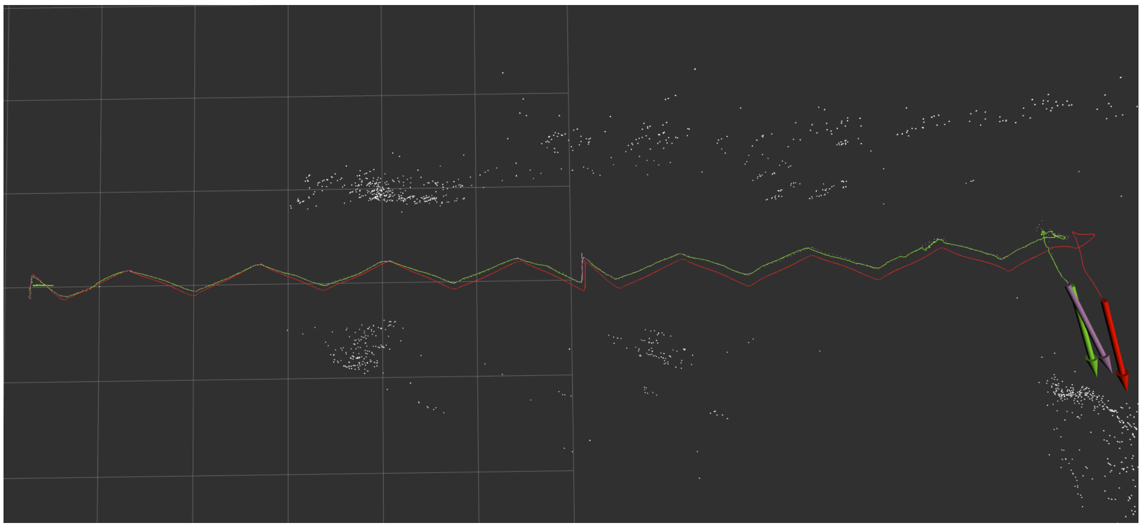


Figure B.1: Visualisation of the state estimates and ground truth, ground truth = red, estimate = green, ORB-SLAM2 = purple

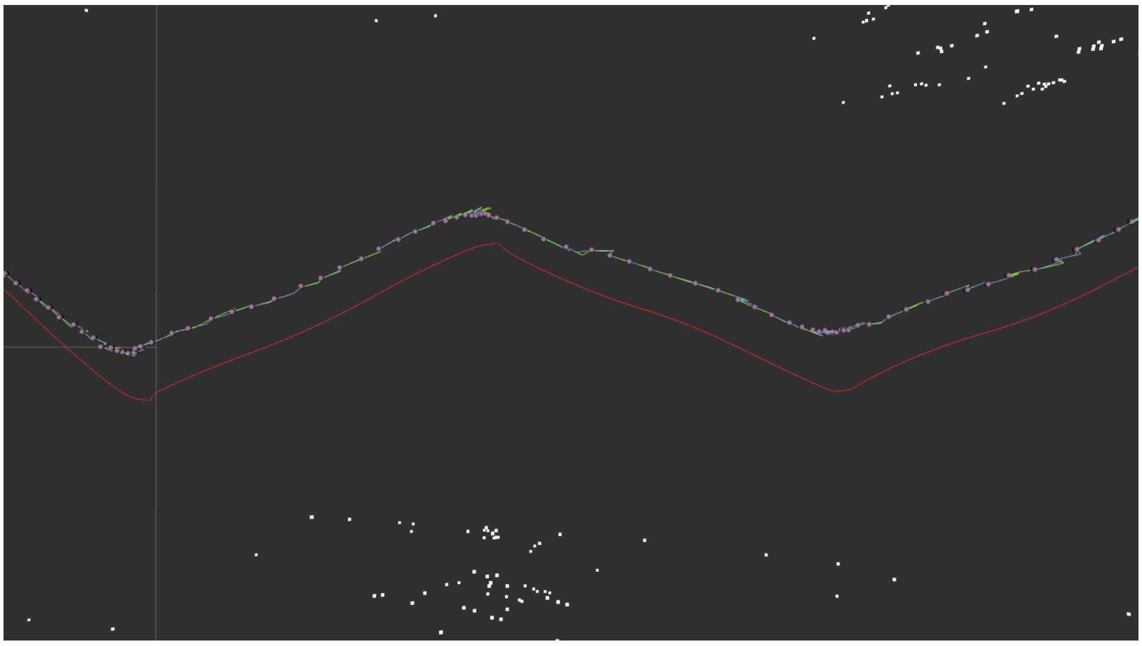


Figure B.2: Closer zoom

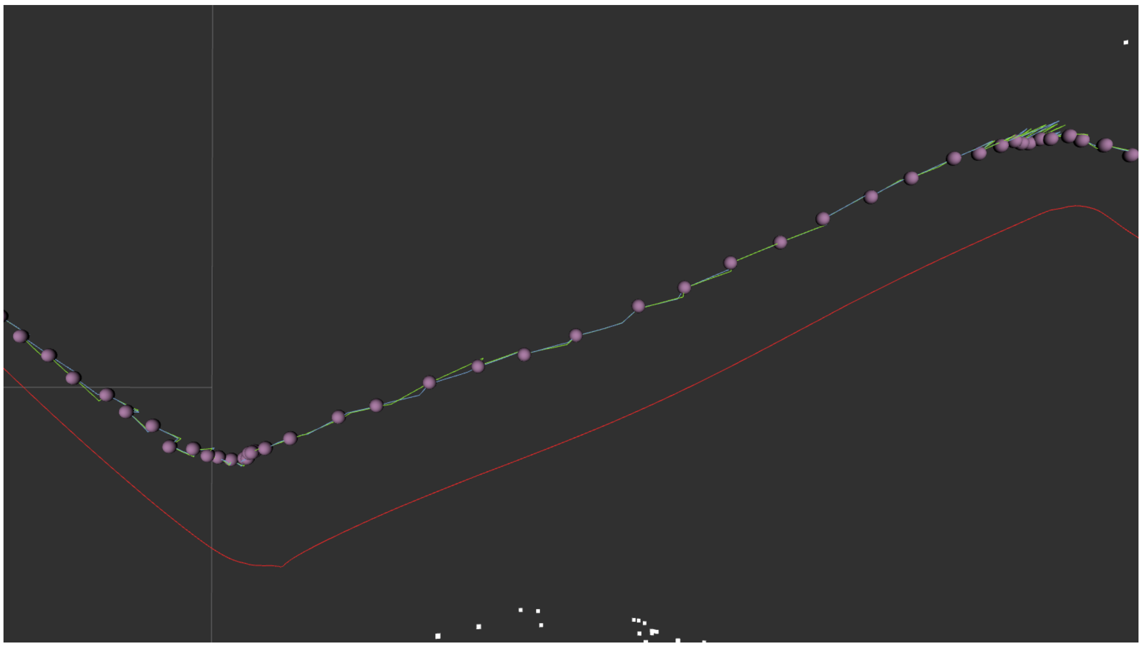


Figure B.3: Even closer zoom

Appendix C

Additional Graphs

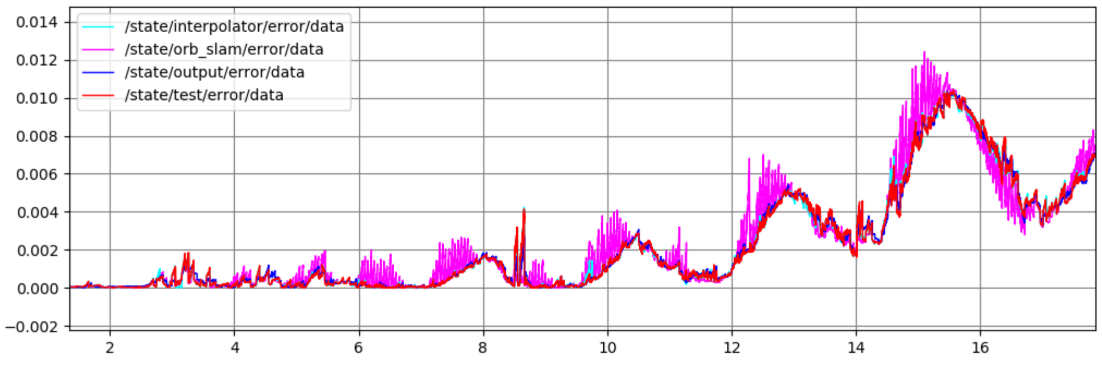


Figure C.1: Another iteration of the error graph

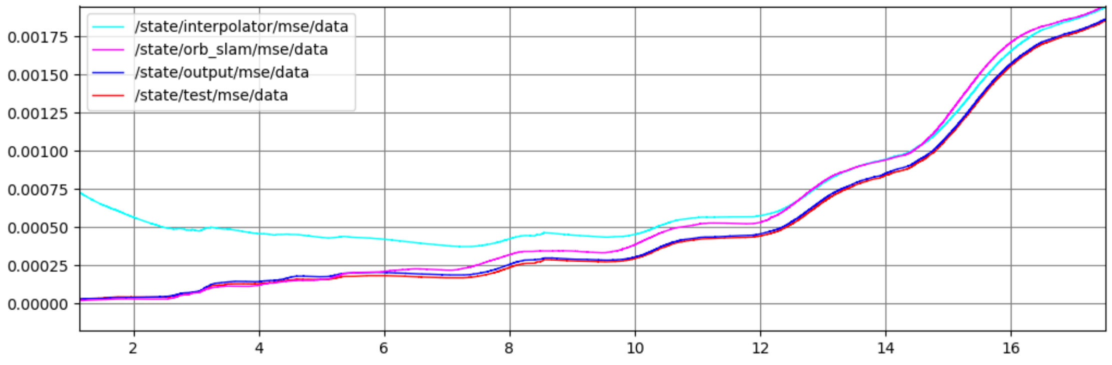


Figure C.2: Corresponding mean square error

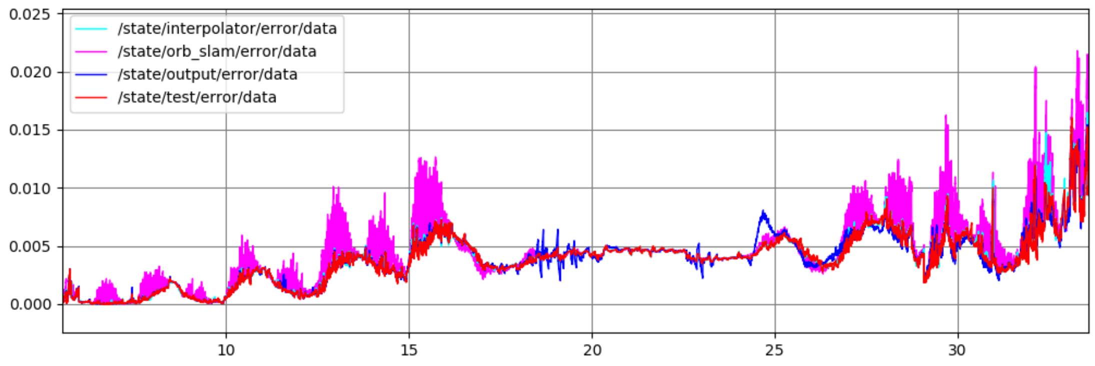


Figure C.3: Another iteration of the error graph

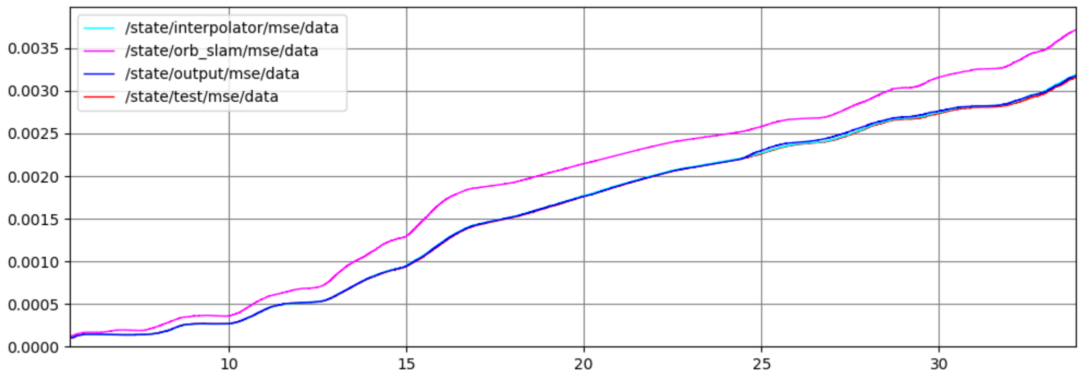


Figure C.4: Corresponding mean square error

Bibliography

[1] M. Mori, "The uncanny valley."

[2] C.-P. Chen, J.-Y. Chen, C.-K. Huang, J.-C. Lu, and P.-C. Lin, "Sensor data fusion for body state estimation in a bipedal robot and its feedback control application for stable walking," *Sensors (Basel, Switzerland)*, vol. 15, pp. 4925–46, 03 2015.

[3] N. Gupta, "State estimation for legged robots using proprioceptive sensors."

[4] S. Srini, "The kalman filter: An algorithm for making sense of fused sensor insight."

[5] A. S. S. Team, "Kalman filter." Available at http://www.ece.montana.edu/seniordesign/archive/SP14/UnderwaterNavigation/kalman_filter.html.

[6] X. Xinjilefu, *State Estimation for Humanoid Robots.* PhD thesis, Carnegie Mellon University, 2015.

[7] H. S. Chadha, "Extended kalman filter: Why do we need an extended version?."

[8] S. Nobili, M. Camurri, V. Barasuol, M. Focchi, D. Caldwell, C. Semini, and M. Fallon, "Heterogeneous sensor fusion for accurate state estimation of dynamic legged robots," Robotics: Science and Systems Foundation, 2017.

[9] M. Bloesch, C. Gehring, P. Fankhauser, M. Hutter, M. A. Hoepflinger, and R. Siegwart, "State estimation for legged robots on unstable and slippery terrain," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6058–6064, Nov 2013.

[10] M. Dekker, "Zero-moment point method for stable bipedal walking."

[11] M. Bloesch, M. Hutter, M. A. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy, and R. Siegwart, "State estimation for legged robots. consistent fusion of leg kinematics and imu," (Sidney), RSS 2012: Robotics: Science and Systems Conference, 2012. Robotics: Science and Systems Conference (RSS 2012); Conference Location: Sidney, Australia; Conference Date: July 9-13, 2012.

[12] B. J. Stephens and C. G. Atkeson, "Dynamic balance force control for compliant humanoid robots," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1248–1255, Oct 2010.

[13] X. Xinjilefu, S. Feng, W. Huang, and C. G. Atkeson, "Decoupled state estimation for humanoids using full-body dynamics," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 195–201, May 2014.

[14] J. Ruiz-del-Solar, J. Moya, and I. Parra-Tsunekawa, "Fall detection and management in biped humanoid robots," in *2010 IEEE International Conference on Robotics and Automation*, pp. 3323–3328, May 2010.

[15] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pp. 200–207, Dec 2006.

[16] N. Rotella, S. Mason, S. Schaal, and L. Righetti, "Inertial sensor-based humanoid joint state estimation," *CoRR*, vol. abs/1602.05134, 2016.

[17] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration theory for fast and accurate visual-inertial navigation," *CoRR*, vol. abs/1512.02363, 2015.

[18] D. Wisth, M. Camurri, and M. Fallon, "Robust legged robot state estimation using factor graph optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4507–4514, 2019.

[19] "Digit picture source." Available at `https://robots.ieee.org/robots/digit`.

[20] B. Dynamics, "More parkour atlas." Available at `https://www.youtube.com/watch?v=_sBBaNYex3E`.

[21] A. Robotics, "Digit v2: Special delivery." Available at `https://www.youtube.com/watch?v=LSk8uCHN5CY`.

[22] J. Vincent, "Ford's vision for package delivery is a robot that folds up into the back of a self-driving car: https://www.theverge.com/2019/5/22/18635439/robot-package-delivery-ford-agility-robotics-autonomous-digit."

[23] R. Hartley, M. G. Jadidi, J. W. Grizzle, and R. M. Eustice, "Contact-aided invariant extended kalman filtering for legged robot state estimation," *CoRR*, vol. abs/1805.10410, 2018.

[24] K. Wang, A. Shah, and P. Kormushev, "Slider: A bipedal robot with knee-less legs and vertical hip sliding motion," in *Proc. 21st International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines (CLAWAR 2018)*, (Panama), September 2018.

[25] M. Kok, J. D. Hol, and T. B. Schön, "Using inertial sensors for position and orientation estimation," *Foundations and Trends® in Signal Processing*, vol. 11, no. 1-2, pp. 1–153, 2017.

[26] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[27] "Gazebo simulator website." Available at `http://gazebosim.org/`.

[28] "Eigen 3 website." Available at `http://eigen.tuxfamily.org/index.php?title=Main_Page`.