Liverpool Hope University

Application of Recurrent Neural Networks in Sensor Fusion of a nine-axis Inertial Measurement Unit

by Beniamin Kalmus

May 2020

School of Mathematics, Computer Science and Engineering

Abstract

MEMS IMUs are a popular choice in navigation applications such as quadcopter drones, vehicle control and automation due to their low cost, size and power requirements. Attaining accurate estimations from MEMS IMUs is a well-known challenge due to gyroscopic drift, accumulation of errors and non-linear scale factors. A sensor fusion solution is proposed using LSTM and GRU recurrent neural networks to estimate quaternion-based AHRS orientation of a low-cost 9-axis MEMS IMU. This paper compares sensor fusion approaches by Madgwick and Mahony to the LSTM and GRU neural networks. Input data is collected using MPU-9255 MEMS IMU mounted to a custom three axis turntable designed to output accurate true attitude of the IMU. The results show RMSE reduction in error by 12% (GRU) and 39% (LSTM) over the Madgwick algorithm, 80% (GRU) and 83% (LSTM) improvement over the Mahony algorithm. Additionally, LSTM and GRU predictions exhibit a quick response time to changes in attitude albeit the output requires filtering due to random fluctuations. It is concluded that the LSTM and GRU are a viable alternative to conventional sensor fusion algorithms.

Table of Contents

Introduction

Background

The development of the first gyrocompass in 1908 by Schuler for the purpose of ship navigation demonstrated the value of inertial navigation and an alternative to the inaccuracies of compass and star navigation at the time. Later, evolution of aircraft in military service stressed the advancement of inertial navigation technologies during encounters with poor visibility, weather and a requirement for quick decision making and reduced pilot workload led to the development of the first automatic pilot using a gyroscope by Elmer A Sperry (Iurato, 2015). The first inertial measurement units (IMUs) were utilised for guiding rockets during the World War II German V2 program which consisted of a simplistic gyroscope and lateral accelerometer with mechanical connections to stabilise flight (Wrigley, 1977). Development of inertial navigation systems (INS) began in the MIT Instrumentation Laboratory (to become Charles Stark Draper Laboratory) where the first navigation system was built for the US Air Force aboard a B-29 bomber aircraft (King, 1998). Further improvements to the system led to advanced precision and miniaturised IMUs for the purpose of INS and missile guidance in the late 1960s.

The invention of silicon semiconductors provided new techniques of fabricating micro-electromechanical systems (MEMS) leading to mass manufacturing of low-cost IMUs which are widely used in devices ranging from smartphones, unmanned aerial vehicle (UAV) navigation and robotics (Norhafizan Ahmad, 2013). IMUs are used to estimate orientation of a body in space without external measurements such as a radar or GPS. Unfortunately, MEMS type of IMUs are sensitive to vibrations and prone to noise which causes estimations of orientation to drift from the actual value (Jaw-Kuen Shiau, 2012). Normally, some sort of processing algorithm is implemented to filter out the noise and combine sensor reading for a more accurate estimate. A well-established method of processing sensor data is a Kalman filter which was developed for the Apollo program to estimate trajectories of manned spacecraft on their way to the Moon (Grewal & Andrews, 2010). Despite wide use in navigation, Kalman filters suffer from several drawbacks that many iterations presented over the years attempted to correct (P. & M., 1999), including alternate filtering methods specifically designed for MEMS IMUs. Recent successes and advancement of machine learning in data analysis tasks provides an opportunity to explore a machine learning approach to sensor fusion of MEMS IMU sensor data (Alpaydin, 2020). Virtual personal assistance, speech recognition and synthesis, image processing and classification dynamic pricing, prediction of weather and traffic have progressed tremendously as a result of novel machine learning methods to solve these problems. This research project's intent is to devise a machine learning solution to low cost MEMS IMU sensor fusion and compare the result to contemporary algorithms to assess effectiveness.

Motivation

With the rise in popularity and demand of self-driving vehicles, virtual reality, smartphone sat-nav and quadcopter drones in recent years, improvements to INS using MEMS Inertial Measurement Units is an important area of research in consideration of reliability and accuracy of these systems. Amazon's ongoing development of autonomous drone delivery (Jung & Kim, 2017) will require accurate MEMS IMUs in inertial navigation when GPS data is unavailable or erroneous. Other types of IMUs such as Ring Laser Gyroscopes are too large, heavy and expensive to be installed in every single drone and are typically reserved for commercial aircraft. Growth of smartphone sales has produced a demand for integrated satellite navigation services such as Google's Maps in lieu of dedicated satellite navigation devices (He Li, 2010). Global Positioning System (GPS) modules used in smartphones have limitations due to cost and size that require use of MEMS IMUs for adequate position and orientation; the standard data refresh rate for GPS modules is 1 HZ with high end modules (Sparkfun NEO-M9N) up to 25Hz rate as opposed to MEMS IMUs refresh rates of over 1000Hz such as the popular Invensense MPU6050. GPS often reports false position when the number of detected satellites is low in dense urban areas, during this period of time the sat-nav relies on the

IMU to estimate position and orientation thus it is crucial that the estimate is as accurate as possible (Aboelmagd Noureldin, 2008).

Machine learning has been applied extensively in the field of navigation especially in the development of self-driving cars. Neural networks (are a type of machine learning method) are well-suited for these tasks because of their abilities to approximate arbitrary functions, learn and generalise from data as well as recognise patterns (Zou, et al., 2006). Image classification problems such as identifying a traffic light from an image is a difficult task to program algorithmically, alternatively a neural network can be trained on images of traffic lights and optimised to generalise on different types. Neural networks have been previously used to compensate for MEMS gyroscope drift due to temperature (Chong, et al., 2016) resulting in reduced errors compared to traditional methods. Furthermore, experiments in dead-reckoning using a Kalman filter assisted by deep neural networks have proven effective at reducing positional errors with exceptional performance during GPS outages (Brossard, et al., 2020). Sensor fusion of MEMS IMUs have been conventionally performed by fusion algorithms, but machine learning solutions have not yet been thoroughly researched. Further examination of machine learning sensor fusion of MEMS IMUs is implemented in this project and compared to current algorithms.

Aims and Objectives

The purpose of this paper is to explore machine learning solutions to sensor fusion of low cost MEMS IMU devices and compare their effectiveness to sensor fusion algorithms. Key indicators in assessing success criteria will be a reduction in error and improved estimation accuracy compared with sensor fusion algorithms.

Project objectives:

1. Review relevant scientific literature in the area of MEMS IMU sensor fusion and use of neural networks in aiding this task.
2. Prepare a data collection method for the purpose of training and testing neural networks and implement suitable sensor fusion algorithms for comparison.
3. Compile suitable neural network models and train on collected data.
4. Calculate relevant metrics and graphs for the neural networks and sensor fusion algorithms.
5. Analyse and discuss results of neural networks in comparison to sensor fusion algorithms.

Preliminaries

Micro-electromechanical system (MEMS) is an umbrella term for miniature mechanical and electro-mechanical elements typically in the range of a few micrometres to millimetres in size. These micro devices have the ability to actuate, sense, control and convert mechanical energy into electrical signals (Judy, 2001). Their small form factor, low cost and high manufacturing volume makes them widespread in commercial and industrial products with applications ranging from lifesaving blood pressure sensors to fibre optic switches in networking and communications, break force and fuel sensors in the automotive sector (Takeda, 2001) (Barbic, 2002). In the context of Inertial Measurement Units (IMU) this paper is primarily concerned with the sensor branch of MEMS.

Inertial measurement unit (IMU) is a self-contained electronic device that measures force and angular rate using accelerometers and gyroscopes (Nawrat, et al., 2012). Some IMUs are paired with a magnetometer capable of measuring orientation with respect to the surrounding magnetic field of the body. An IMU typically contains three accelerometers perpendicular to each other to measure acceleration of a body in three axes (denoted *x, y, z*). Accelerometers detect all forces acting on the body including both static forces such as gravity and dynamic forces such as movement of the body or vibrations. Orientation is estimated by measuring the gravity vector of acceleration if the device is static and assumes no other forces are acting on the body (Mizell, 2003). IMUs also contains three gyroscopes that measure angular velocity which is integrated with respect to time to estimate orientation given a previous or known initial configuration. A gyroscope and the accelerometer are combined inside of an IMU because drawbacks of each sensor counteract and using an appropriate method, a more accurate orientation can be extracted (Soo, 2003); the gyroscope is less prone to short term noise but drifts in the long term while the accelerometer is unreliable in the short term. A sufficiently accurate gyroscope can detect the rotation of the Earth at about 15 degrees per hour (depending on the latitude) meanwhile high accuracy ring laser gyroscopes can even detect annual and Chandler wobble of the Earth's axis (Beverini, et al., 2015). Magnetometers (sometimes called magnetic sensors) are devices capable of measuring magnetic field intensity (Lenz & Edelstein, 2006). Magnetometers can measure the magnetic field intensity in three axes and therefore produce a field vector. Although magnetometers have many diverse uses such as magnetic surveys for mineral exploration (Budker, et al., 2013), in AHRS estimation magnetometers ideally only sense the Earth's magnetic field direction as any other magnetic sources increase distortions and errors.

Inertial Navigation Systems (INS) consist of IMUs to produce an estimate of the attitude, heading and position of the system given a known initial position (Kong, 2004). An IMU outputs raw data from its onboard sensors to the INS where some processing ensues to fuse data from accelerometers, gyroscopes and magnetometers to extract the orientation of a device. An attitude and heading reference system (AHRS) is a common output reference system used in aircraft navigation (Gebre-Egziabher, et al., 2002). An AHRS is usually represented by the pitch, roll and yaw of a system with respect to the Earth reference frame. The process of combining data from multiple sources is called sensor fusion and is performed by a suitable algorithm, for example a Kalman Filter is widely used in this task (Qi & Moore, 2002).

Literature Review

This section discusses literature in the field of MEMS IMUs and neural networks. Subsections are dedicated to individual topics.

Kalman Filter

Kalman filter (KF) is an iterative algorithm published by Rudolph Kalman in 1960 that provides estimates of variables given measurements collected over time (Kalman, 1960) using a form of feedback control. Kalman filters see use in a wide range of processes due to their generality and flexibility, most famously in the navigation of the Apollo Project (Grewal & Andrews, 2010). KF works by assuming measurement noise follows zero-mean Gaussian distribution which is an effective tool for estimation of linear systems, however most real world problems are nonlinear including IMU attitude estimation (Gui, et al., 2015). A common solution to this is a variant of KF called Extended Kalman filter (EKF) which linearises the state of measurement equations (Hellmers, et al., 2013) using Taylor series expansion, but other drawbacks arise as careful parameter tuning is required and computation requirements are considerably increased.

Nonlinear Complementary Filter

A paper titled *Nonlinear Complementary Filters on the Special Orthogonal Group* by Robert Mahony published in 2008 aimed to provide reliable attitude estimation of low cost IMUs (Mahony, et al., 2008). In the paper, Mahony proposes a nonlinear filter termed *explicit complementary filter* which exhibits low sensitivity to noise. The filter uses a proportional and integral controller to correct gyroscope bias using accelerometer and magnetometer measurements in a closed feedback loop. The filter has two tweakable parameters for the controller and returns a system's attitude expressed as a unit quaternion. The authors recommend use of quaternions for hardware implementations due to computational efficiency. Equations and derivations are listed in the cited paper and an implementation of the algorithm in *C++* programming language by (Winer, 2018) can be found in the appendix.

Madgwick Fusion Algorithm

A popular sensor fusion algorithm optimised for embedded systems and widely used in quadcopters, was published by Sebastian Madgwick (Madgwick, 2010). The filter is based on an optimized gradient-descent algorithm which computes attitude as a quaternion to improve computation efficiency. Compensation of gyroscope bias is corrected by the accelerometer and magnetometer although the paper discusses that in most applications these sensors report erroneous data and thus the filter provides *beta* and *zeta* gains to tune divergence rate and response time. The author recommends that the parameters are tweaked to find optimal values for individual system and suggests a dynamically tuned parameters for systems that experience quick variations. Paper presents a reduced error and improved accuracy with exceptional performance at low sample rates (10*Hz*) over a classic Kalman filter without the burden of high computational load.

Comparison of Sensor Fusion Techniques

An article intended to study performance of three sensor fusion algorithms using a KUKA robot with a mounted ST L3GD20 MEMS 9-dof IMU (Cavallo, et al., 2014). The paper tested an Extended Kalman Filter (EKF), Madgwick and Mahony's nonlinear filter solutions on static, slow moving and fast moving datasets with reference to robot's platform angles given by digital encoders. The results indicate EKF fusion algorithm had the lowest Root Mean Squared Error (RMSE) with Madgwick

following second and Mahony filter last, but at the expense of an increased computation time (on an embedded system) of 2.7*ms* for EKF versus 0.15*ms* and 0.11*ms* for Madgwick and Mahony filter respectively. According to the result, this is an order of magnitude higher for only marginal performance improvement. However, a limitation of the study is the rather small dataset, as the samples were collected for one minute at a rate of 500*Hz*. Additionally, the range of tested maneuvers were limited by the slow moving robot arm with the fastest trajectory rotating at an angular velocity of 45 $degs^{-1}$.

Researchers in 2018 analysed EKF, Madgwick and Mahony sensor fusion algorithms performance using quadcopter flight data in an indoor environment (Ludwig & Burnham, 2018). The ground truth Euler angles of the quadcopter were collected using a Vicon motion capture system while IMU sensor data was fed to the fusion algorithms. The Mean Absolute Error (MAE) metric indicates similar performance for each algorithm with largest difference of 1.0 MAE (sum of *x, y, z*) but EKF scored greatest in the pitch and roll although worst in yaw estimation. In this implementation, EKF had the longest execution time of 0.2895*s* versus 0.2080*s* and 0.1782*s* for Madgwick and Mahony filters respectively, which agrees with the results of paper (Cavallo, et al., 2014). The research was limited by a lacking dataset which contained 6,400 samples and the ranges of motion for pitch and roll were under ±20*deg* and ± *200deg* for yaw. This is a similar lack of diversity in the data alike to the previously work by Cavallo et al. in 2014. Graph plots of filter estimations show a reasonable amount of lag behind the ground truth Euler angles (a value for the lag was not given) particularly in the yaw axis which could be an indication of filter parameters tuned for smoothness over response time. The authors conclude that error measurements were similar for three sensor fusion algorithms, with Mahony outperforming the others in the RMSE norm metric and also the fastest code execution time.

Mahony filter and Kalman filter were compared in human motion analysis by attaching seven IMUs to ten male subjects (Kim, et al., 2015). Using optical markers and 12 Hawk digital cameras, the researchers were able to compare orientations of body segments with IMU estimations from Mahony filter and KF. Root mean square difference (RMSD) was calculated for four different stances for seven IMU sensors attached to the body. Both the KF and Mahony filter estimated joint angles consistently when compared to the optical system. RMSD presented in tables shows almost identical results for the sensor fusion filters with Mahony marginally outperforming the KF, in one example the mean RMSD for walking was 2.0 and 2.1 for Mahony and KF respectively. The paper concludes that the Mahony filter can be used for human joint kinematic analysis and notes its fast execution speed which confirms the results of prior research.

Brief Introduction to Machine Learning

Most machine learning problems can be grouped into two categories, classification and regression (Ayodele, 2010). The former's goal is to predict discrete values, for instance detecting if an image contains a car or not would be a binary classification problem. Regression problems concern prediction of continuous values such as temperature of a particular place at some point in the future. Types of machine learning algorithms are separated into four groups: supervised, unsupervised, semi-supervised and reinforcement learning. Supervised learning can be thought of as function approximation where features *X* attached to labels *Y* are used for training of the algorithm to best fit the data (Reed & Marks, 2014). Neural networks and support vector machines (SVM) are examples of supervised learning algorithms (Tao, et al., 2006). Unsupervised learning is used when the data doesn't contain or cannot have labels such that the algorithm attempts to detect patterns and categorize data. IMU sensor fusion is a time series regression problem as data is collected from sensors at some sample rate and using an external method, labels can be collected for the data. Therefore, the type of machine learning algorithms appropriate to perform MEMS IMU sensor fusion task is supervised multivariate time series forecasting.

In 2018, researchers developed a recurrent neural network model to reduce gyroscope noise of a MT MSI3200 MEMS IMU (Jiang, et al., 2018). The IMU was placed on a table and samples were collected at 400*Hz* for 10 minutes. The network model employed a variant of recurrent neural network (RNN) called Simple Recurrent Unit (SRU); this type of recurrent neural network has fewer parameters to optimise training time compared to LSTM RNNs and GRU RNNs. Training was performed for 100 epochs with a learning rate of 0.01 and batch size of 128. The results for gyroscope noise were compared with raw gyroscope readings as well as attitude estimation algorithm prior and after network model denoising. The *x, y* and *z* axes noise were reduced by 0.6%, 60.5% and 11.3% respectively. For roll, pitch and yaw estimations, the neural network reduced errors by 19.2%, 82.1% and 69.4% respectively. Unfortunately, the MEMS IMU was static for the entire duration of data collection, the IMU did not include accelerometer/magnetometer sensors and an AHRS algorithm was required to compute Euler angles. Furthermore, despite promising results the dataset was collected for a short period of time of 10 minutes, even though the collection method permitted a much longer sampling time (static placement on a table). Authors recommend further experimentation with different types of RNNs and propose a feasibility study in GPS/INS integration during 60 second GPS signal dropouts.

Another approach of denoising MEMS IMUs using supervised learning algorithms was analysed and compared by authors of journal article (Gonzaleza & Catania, 2019). The setup involved four MEMS IMUs (gyroscope and accelerometer) mounted to a vehicle driven around a car park in figure eight and loops several times. Ground truth was established using GNSS and a non-MEMS Honeywell H764G-1 high accuracy navigation-grade IMU. Three denoising techniques carried out include moving-average, multi-layer perceptron (MLP) and time-delayed multiple linear regression (TD-MLR). Models were validated on a separate trajectory denoted *S2* which closely resembled real world movement of a road vehicle as opposed to *S1* training trajectory. Data was evaluated using RMSE which demonstrated an improvement in noise reduction of MLP and TD-MLR models over moving average and raw data consistently. The study was extensive with 20 minutes of data sampled at 250*Hz* using four separate models of IMUs and included motion in contrast with previous research. The data was a good representation of the real world and combined several sensors and ground truth methods. The paper builds a strong case for use of neural networks in MEMS IMUs however a comparison to conventional methods was not included as moving average is rarely used without other filters. Therefore it is difficult to interpret the result and effectiveness of neural networks against conventional methods such as an Extended Kalman filter.

Authors of another paper explored a novel (Liu, et al., 2018) UAV attitude estimation method using a long-short term memory (LSTM) neural network. The UAV was a quadcopter drone containing a 9dof MEMS IMU onboard a PIXHAWK flight controller. Data was sampled at 100Hz for 80 seconds during quadcopter flight and 10% of the data was used as validation of the network model. The input data for the LSTM model was the IMU sensor (contained within the flight controller) meanwhile labels were collected by a modified complementary filter which calculated pitch, roll and yaw at the end of a run. Unfortunately the researchers did not use an external system to measure ground truth attitude of the quadcopter. The limitation of this approach is that the LSTM model tries to best approximate the complementary filter instead of true quadcopter angles. Researchers state that careful parameters were picked to maximize accuracy and that the LSTM model replaces complex attitude calculations of the complementary filter. Using only 1 hidden layer with 15 cells, the model achieved mean squared error (MSE) of 0.000855, 0.005985 and 0.003255 for pitch, roll and yaw estimations. Graph plots of the predicted angles show fluctuations rather than a smooth curve yet the trend mirrors the calculated angles accurately. Although this is an optimistic result, further research should acquire more sample data with a better system of labelling reliable ground truth. The data in this report was collected while the quadcopter hovered for 80 seconds but lacked any challenging real world maneuvers which if applied could confuse the LSTM model during linear accelerations, magnetometer noise and extreme attitude angles.

Further work used LSTM neural networks with good results for calibration and bias prediction of MEMS IMU MUG2213M accelerometer and gyroscope (Wang, et al., 2019). The IMU was attached to a three axis turntable which allowed collection of data with reliable labels from the turntable's angular rate given by digital encoders. The authors justify the choice of the neural network due to LSTM's effectiveness in identifying time series data. The LSTM layer consisted of 32 cells, 16 linear cells and 3 cells for the output layer. The turntable rotated on all three axes in a sinusoidal manner of fixed frequency and amplitude. Results displayed by a graph show a significant reduction in noise and bias of the IMU closely following the real curve. The authors conclude that the LSTM model can achieve high precision calibration of the IMU and its non-linear factors while being insusceptible to time-correlated noise. However, this paper doesn't test other IMU sensors in the calibration. The paper fails to address the possibility of the LSTM network adapting to predictable sinusoidal motion rather than movements of real world devices such as the land vehicle research project in previously discussed paper (Gonzaleza & Catania, 2019). Additionally, the authors do not state how much data was collected and only show a sample of 8 seconds with a maximum angular rate of 60$degs^{-1}$.

Research into sensor fusion using artificial neural networks (ANN) was conducted (Kolanowski, et al., 2013) where training data was generated using quadcopter flight data, a setup similar to the paper (Liu, et al., 2018). An AHRS algorithm generated output labels pitch, roll and yaw using 9dof MEMS IMU sensor data. About 30 seconds of data was collected to train the neural network and the authors conclude that a larger data set is required for improved approximations. Unlike research of sensor fusion using LSTMs by Liu et al., graph plots of the output in this work are smooth and closely resemble the AHRS algorithm without random fluctuations encountered previously. Comparing the accelerometer, gyroscope and magnetometer inputs of these papers shows a distinct difference in noise and variations of the input data. Further work by researchers of (Kolanowski, et al., 2018) article used an Elman ANN (a type of RNN) with quadcopter flight data sampled at a rate of 50Hz. Training labels were created using EKF estimation for the dataset. A graph plot displays 3000ms of attitude Euler angles ranging from -100 to 100 degrees with quickly changing movements. Unfortunately, error metrics were not provided in this paper, however a graph plot of errors was shown which indicates large uncertainty during quick maneuvers. The output attitude also encountered fluctuations like other research papers which the authors attributed to high dynamics of the input data. The dataset size is lacking and authors acknowledge that more data is required in future research. The conclusion states that Elman ANN sensor fusion can be used as an alternative to EKF as it is less computationally expensive and does not require complex operations.

Summary of Findings

From prior work of other researchers, it is apparent that the popular choice is a Kalman filter due to its flexibility and low error estimations, even though many researchers note the drawback of implementation complexity and expensive computation (Gui, et al., 2015). Mahony's complementary filter algorithm provides an alternative to the Kalman filter for embedded hardware implementations without compromising accuracy. Madgwick's filter demonstrates good results as another alternative which sees popular use in quadcopters due to its fast convergence and quick parameter tuning.

Researchers employ recurrent neural networks in MEMS IMUs due to their ability to recall previous states in their output which is important in time-series problems as gyroscopes report angular velocity which requires time integral to calculate orientation. Prior work focuses on labelling training data using other sensor fusion algorithms, however this has several limitations because a neural network approximates the algorithms that also exhibit errors with respect to the actual AHRS. This is seen in the paper (Kolanowski, et al., 2013) where graph plots of attitude between ANN and AHRS show that neural network underestimates the output at extreme attitudes. Prior work reveals several methods of collecting data to produce a dataset for training neural networks. Methods that use a vehicle or a quadcopter drone to require expensive equipment to track ground truth data using optical tracking or ring laser gyroscopes. Alternatively, a three axis turntable can be used to reproduce any type of

motion without the use of external equipment for ground truth data labels (which are provided from the turntable's internal circuitry).

Methodology and Implementation

The aim of this research project is to compare sensor fusion using neural networks to current methods of sensor fusion. This section describes the methods used to solve the problem.

Introduction

A three-axis turntable for data collection is expensive therefore a modified solution is presented in this paper. A custom built turntable is produced using stepper motors and 3D printing technology to assemble the final design. Tracking rotations of three stepper motors is used to establish ground truth attitude of the turntable. Stepper motors do not contain a sensor to keep track of their position like servo motors but provide precise control over movement and can rotate fully in any direction (servo motors are restricted to a range of angles) (Acarnley, 2002). The solution to record stepper motor positions is done by counting the number of steps a motor takes from its known initial position.

The MEMS IMU chosen for this project is 9dof (9 degree-of-freedom) MPU-9255 containing an accelerometer, gyroscope and magnetometer (InvenSense Inc., 2014). The MPU-9255 is a popular choice among enthusiast quadcopter pilots due to its versatile configuration of sensors: accelerometer axis range from $\pm 16G$(G is the acceleration due to gravity of value $9.81 ms^{-2}$) to high precision mode $\pm 2G$, gyroscope maximum range $\pm 2000\ degs^{-1}$ to high precision $\pm 250 degs^{-1}$. The MPU-9255 requires an I^2C bus compatible device to retrieve data therefore a STM32F103C8 microcontroller (STMicroelectronics, 2015) is used for its fast $72MHz$ processor clock and $128KB$ memory. Moreover, due to the popularity of this model there are a wide range of resources, availability and documentation to ease development.

The neural networks selected for sensor fusion are the LSTM and GRU RNNs as prior research recommended their use in MEMS IMU processing and many results supported this advice. The implementation of the LSTM and GRU is done in the Tensorflow library using Keras high level API for python programming language. Keras and Tensorflow provide a wide range of tools for neural network training such as regularisation, variety of optimisers, metrics, model checkpoint saving and tensorboard tracking. The neural network solution is compared to Mahony and Madgwick sensor fusion algorithms. Research from literature review compares these algorithms to established methods like Extended Kalman filter and conclude that Mahony and Madgwick implementations provide similar results while being computationally efficient.

Theoretical Context

Recurrent Neural Networks (RNN)

Recurrent neural networks are characterised by their ability to remember previous states of the network (Mandic & Chambers, 2001) in contrast with traditional feedforward neural networks such as a multi-layer perceptron (MLP). The limitation of a classic perceptron neuron is that if a sequence of information is inputted into the network, the neuron will process the data at each time step individually without influence of prior states. RNNs overcome this limitation by allowing past information to be weighted in the output of a neuron's current state. This can be thought of as a process where the state at each time step is added to the current state in a cycle. A diagram in Figure 1 illustrates unrolling of a single RNN neuron into a layer of RNNs at various time steps.

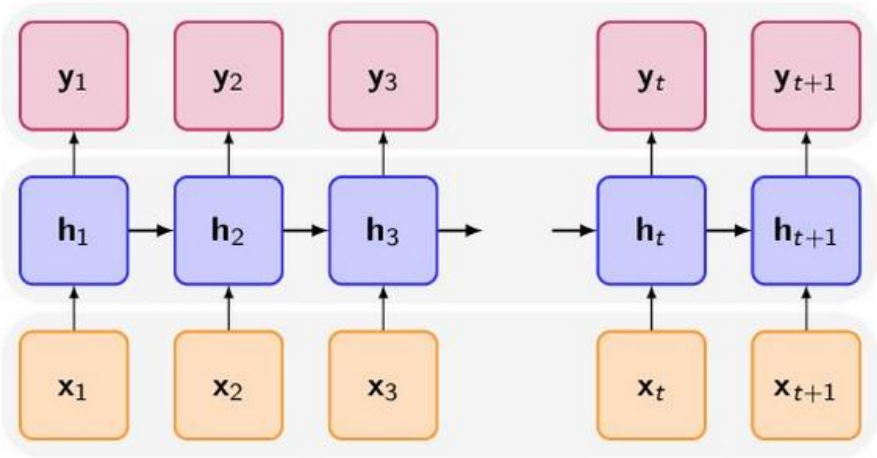

Figure 1: Diagram of RNN cell unrolled into t timesteps (Kelleher & Dobnik, 2017).

RNNs quickly rose in popularity due to their success in natural language processing and time series forecasting. However, a drawback of a standard RNNs is learning long-term time dependencies (Kostadinov, 2018), such as extracting context from a paragraph. This is difficult due to vanishing gradient problem which occurs during backpropagation where the derivative of the loss function (such as sigmoid) decays exponentially with time (Boden, 2001) (Bengio, et al., 1994). This causes states from sequences further in the past to have a decaying weight on the current state until eventually it becomes negligible.

Long Short Term Memory (LSTM)

A paper titled *Long Short Term Memory* presents an improved type of RNN with long-term dependencies specifically designed to circumvent the vanishing gradient problem (Hochreiter & Schmidhuber, 1997). The new LSTM cell encloses what the authors describe gate structure containing three gates: an input gate, forget gate and an output gate. These gates control the flow of information through the LSTM cell. A diagram of an LSTM cell is displayed in Figure 2.

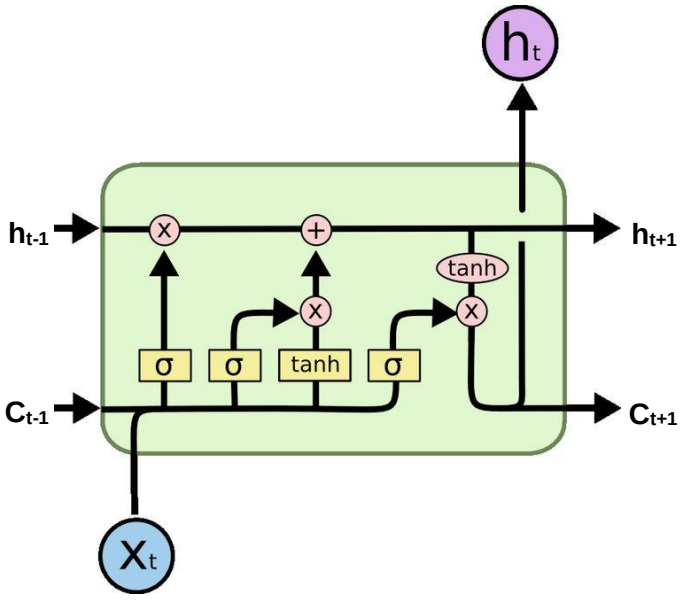

Figure 2- Illustration of an LSTM cell structure of three gates, hidden and cell states (O'Shea, et al., 2016).

The forget gate, denoted f_t, can be considered as its own layer inside of the neuron with a sigmoid activation function and inputs x_t and h_{t-1}, where x_t is the input matrix to the cell and h_{t-1} is the hidden state at time step *t-1* (the previous output of the cell).The forget gate has the ability to choose the information to keep. The forget gate is given by equation [1],

$$f_t = \sigma(W_f(h_{t-1}, x_t) + b_f) \quad [1]$$

where,
W_f is weight of the forget gate,
b_f is the bias of the forget gate,
σ is a sigmoid activation function.

The input gate controls the information that is added to the current cell state. First layer of the input gate denoted i_t is a sigmoid activation function with its own weights and biases similarly to the forget gate. The next *tanh* layer is responsible for a new cell state, denoted C'_t and is multiplied by the input gate which controls the amount of information passed from the previous state to the current cell state vector, C_t. The final result of the cell state is given by equation [2],

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C'_t \quad [2]$$

where,

C_{t-1} is the cell state at time step *t-1*,

The output gate, o_t, is composed of a sigmoid layer and a *tanh* layer. The output gate combines the prior gates to decide which parts of the cell state are passed on to the hidden state. The sigmoid layer is given by [3],

$$o_t = \sigma(W_o(h_{t-1}, x_t) + b_o) \quad [3]$$

where,
W_o is weight of the gate,
b_o is the bias of the gate,
σ is a sigmoid activation function.

And the hidden state, h_t is given by [4],

$$h_t = o_t \cdot \tanh(C_t) \quad [4]$$

The weights and biases for each gate are updated individually during training. The gate structure of the LSTM solves the vanishing gradient problem as memory persists in the cell state.

Gated Recurrent Unit (GRU)

The GRU is one of many variants of the LSTM cell but with a simplified structure containing only 2 gates (Britz, 2015). The first gate, called update gate, determines how much information from past time steps is kept using a sigmoid activation layer. The update gate can copy previous information using weight U_z as it is multiplied by the previous hidden state. The update gate is given by [5]

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \quad [5]$$

where,
σ is the sigmoid activation function.
W_z is the weight of the update gate,
U_z is the weight for hidden state of the previous time step,
x_t is input data,
h_{t-1} is previous hidden state.

The second gate, called reset gate, controls how much information is forgotten. Implementation is similar to the update gate and the proportion of the past state to the current input is controlled by weights [6],

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \quad [6]$$

where,
W_r is the weight of the reset gate,
U_r is the weight for hidden state of the previous time step.

The GRU removes the need for a cell state vector by using the hidden state to pass information. The hidden state can store relevant information by combining the output of the reset gate which itself controls previous time step data, see equation [7],

$$h'_t = tanh(W x_t + r_t \odot U h_{t-1}) \quad [7]$$

where,
W is the output weight,
U is weight assigned to previous hidden state,
⊙ is Hadamard product.

Lastly, the cell needs to determine how much information is transferred from the current state and the previous steps. The hidden state of the GRU, h_t is given by equation [8]

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \quad [8]$$

Consider a scenario where the update gate z_t is close to 1, the leftmost term of [8] would converge to h_{t-1} which is the previously stored state. Meanwhile the rightmost term would converge to 0, due to $(1-z_t)$ and so the present state of h'_t is ignored or forgotten. Thus the update gate controls how much information from the previous state is stored in the hidden state. This design of a recurrent neural network is capable of preventing the vanishing gradient problem while also being faster to train than the LSTM as a result of fewer parameters. A diagram of a GRU cell is shown in Figure 3.

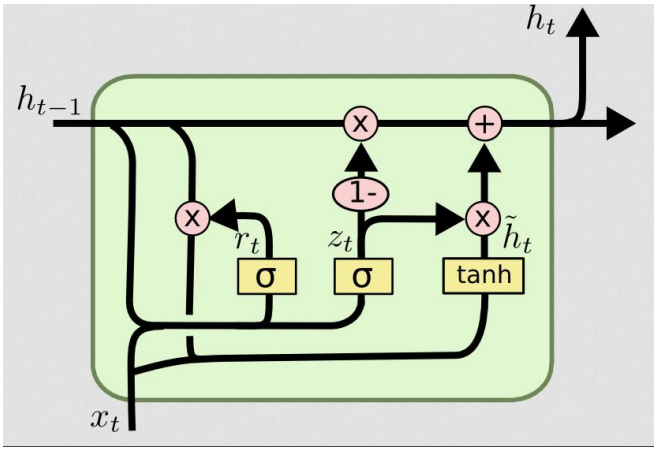

Figure 3 – Structure of a Gated Recurrent Unit (Olah, 2015).

Hardware Setup

The MPU-9255 is a low-cost MEMS IMU designed by Invensense consisting of MPU-6500 gyroscope with accelerometer and AK8963 magnetometer by AsahiKasei (AsahiKasei, 2013). For convenience, the MPU-9255 is surface-mounted on MPU-92/65 breakout board as illustrated in Figure 5 which provides accessible through-hole headers for soldering. The IMU communicates using I^2C bus with an STM32F103C8 microcontroller provided on a popular breakout board called maple mini. The microcontroller has 37 high-speed GPIO pins and a USB interface for connecting to a desktop computer enabling information transfer (STMicroelectronics, 2015).

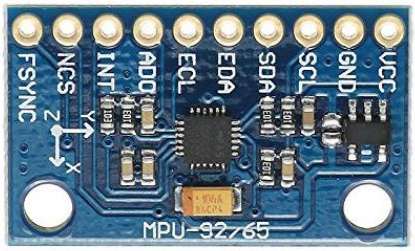

Figure 4 – MPU-92/65 breakout board containing MPU-9255 MEMS IMU with decoupling capacitors.

The MPU-95/65 breakout board is attached to a custom built three axis turntable. Each axis of the turntable is controlled by a 28BYJ-48 stepper motor model which is of the unipolar type powered by 5 volts. The 28BYJ-48 motor (Kiatronics, 2013) has a gear reduction ratio of 1/64 and supports a rotation angle of 0.176 (3 significant figures) degrees per step in full step mode. Occasionally, the gear ratio differs from the manufacturer's datasheet therefore the gear ratio is tested by spinning the motor for 10 full rotations and deviation is measured from the initial mark using a protractor; the value found is 1/63.7 and this is corrected in software. During the test, it was discovered that the shaft has a small amount of looseness where the motors have no holding torque for 2 degrees (measured using a protractor). This could not be corrected as the play is due to the internal mechanism of stepper motors. The turntable mounting solution was designed in AutoCAD Fusion360 and fabricated using a 3D printer in PETG plastic to house the 28BYJ-48 and the MPU-92/65 IMU board. Since the construction is plastic, some level of bending is expected but there were no visible or measurable adverse effects.

The STM32F103C8 controls each stepper motor (and therefore axis of the turntable) separately using 4 GPIO pins (12 in total) connected to ULN2003A Darlington arrays. The stepper motors are not wired directly to the microcontroller as this would damage the internal circuitry. Power is supplied

using a 12V to 5V buck converter which is directed to the stepper motors and ULN2003A arrays. Two separate sources of power are supplied to the motors and MPU-9255 in order to reduce noise and damage on sensors contained within the MEMS IMU. This is supplied from the 5V USB and reduced to 3.3V using maple mini's onboard regulator. The hardware setup is labelled in Figure 5.

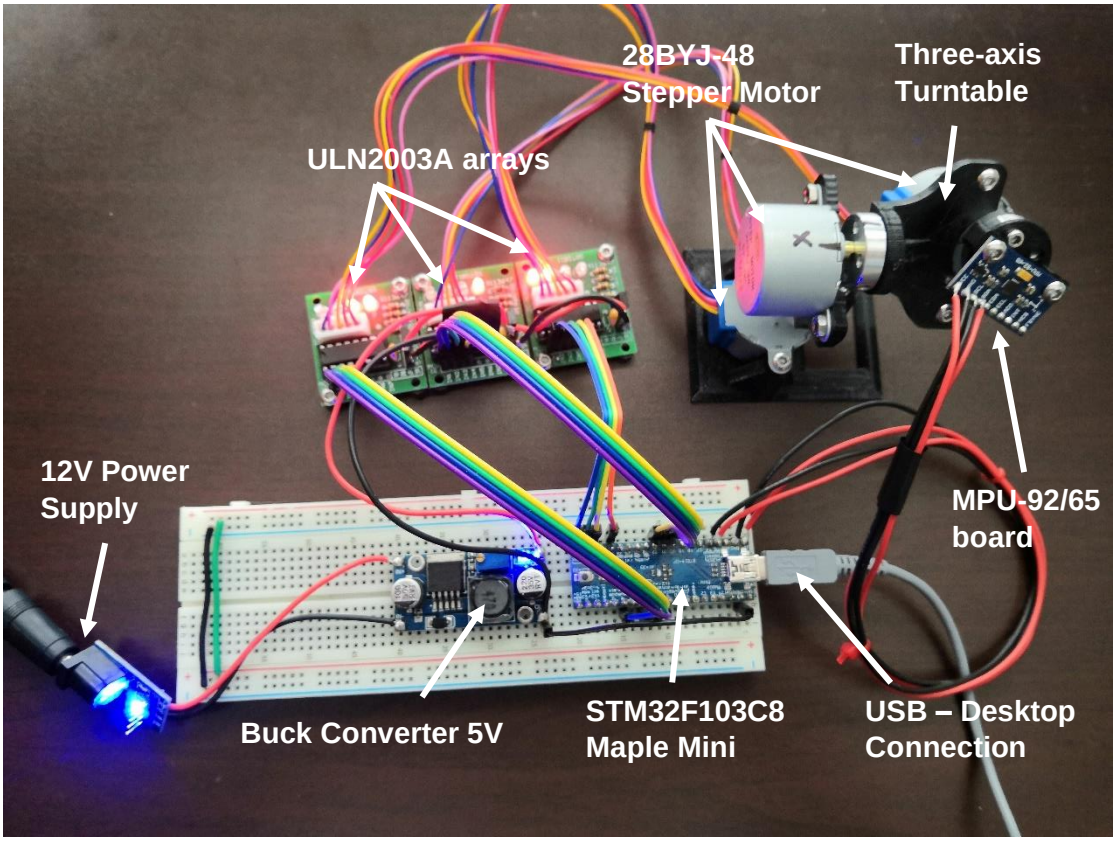

Figure 5 – Setup of the data collection equipment.

Software

Microcontroller Implementation

A program loop written in C++ programming language runs on the STM32F103C8 microcontroller that handles data retrieval from the MPU-9255 IMU, controls turntable movements and communicates with the desktop via USB Serial port. The MPU-9255 configuration is initialised through the I^2C bus and begins collecting data immediately at a sample rate of 250Hz. The MPU-9255 is configured for maximum range of ±8G acceleration and ±500$degs^{-1}$ and data is converted according to the datasheet provided by the manufacturer (STMicroelectronics, 2015). The magnetometer data includes all magnetic sources surrounding the IMU and since the IMU is positioned near magnetic fields of three stepper motors, the magnetometer requires careful calibration to compensate for biases (soft iron and hard iron distortions). Hard iron distortions are caused by constant, fixed magnetic sources that rotate with the magnetometer (Konvalin, 2009). Soft iron sources cause a skewed distortion that changes depending on the orientation of the magnetometer. To compensate for soft iron and hard iron sources, a magnetometer calibration routine is written into the program and performed at start-up by rotating the sensor in all possible directions. The final magnetometer data output is compensated for scales and biases including built-in factory calibration values to maximise accuracy of the data.

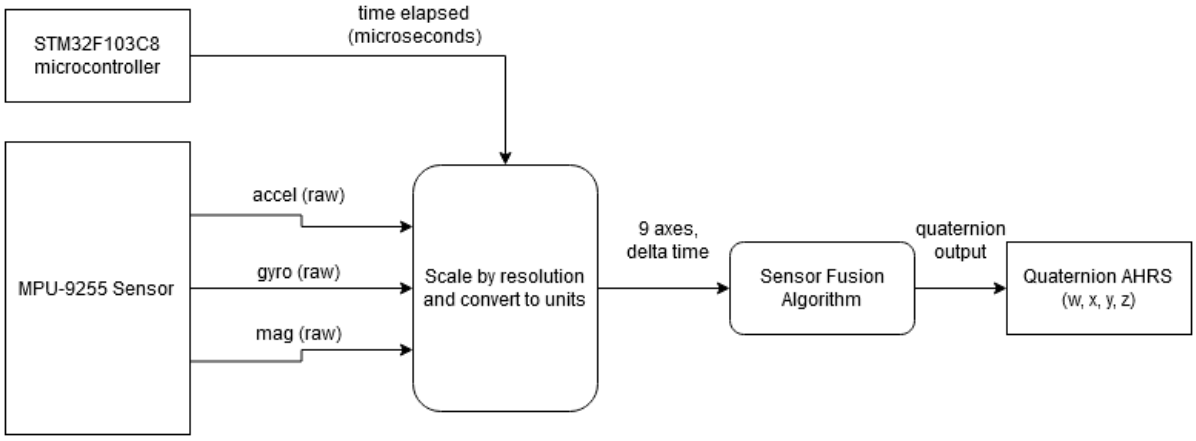

Figure 6 – Implementation of sensor fusion on the STM32F103C8 microcontroller.

The Mahony and Madgwick sensor fusion algorithms are implemented in C++ and run on the microcontroller in real time, Figure 6. The algorithms are updated at the same rate as data is being retrieved from the IMU at a rate of 250*Hz*. The output is stored as a quaternion using 32 byte floating point data type. The sensor fusion algorithms are carefully tuned by comparing response time and difference to the true angles (established by the turntable) to ensure optimal performance.

Since the turntable has an *x, y* and *z* axis, the attitude is naturally a Euler angle representation. The Euler angles are calculated by tracking the position of each motor's step count and converting the steps into degrees using a value calculated from the gear ratio and angle per step. The turntable is calibrated using physical markers and aligned with the magnetic North with the help of software and onboard MEMS IMU to home each axis to zero, according with the AHRS. The output label data is converted from Euler angles to a quaternion representation of AHRS using [9][10][11][12] .

$$q_0 = \cos\left(\frac{r}{2}\right)\cos\left(\frac{p}{2}\right)\cos\left(\frac{y}{2}\right) + sin\ \left(\frac{r}{2}\right)sin\ \left(\frac{p}{2}\right)sin\ \left(\frac{y}{2}\right) \quad [9]$$

$$q_1 = sin\left(\frac{r}{2}\right)\cos\left(\frac{p}{2}\right)\cos\left(\frac{y}{2}\right) - \cos\ \left(\frac{r}{2}\right)sin\ \left(\frac{p}{2}\right)sin\ \left(\frac{y}{2}\right) \quad [10]$$

$$q_2 = \cos\left(\frac{r}{2}\right)sin\left(\frac{p}{2}\right)\cos\left(\frac{y}{2}\right) + sin\ \left(\frac{r}{2}\right)\cos\ \left(\frac{p}{2}\right)sin\ \left(\frac{y}{2}\right) \quad [11]$$

$$q_3 = \cos\left(\frac{r}{2}\right)\cos\left(\frac{p}{2}\right)sin\left(\frac{y}{2}\right) - sin\ \left(\frac{r}{2}\right)sin\ \left(\frac{p}{2}\right)\cos\ \left(\frac{y}{2}\right) \quad [12]$$

where,
p is the pitch Euler angle, rotation around the *x*-axis,
r is the roll Euler angle, rotation around the *y*-axis,
y is the yaw Euler angle, rotation around the *z*-axis,
q_0 is the *w* component of a quaternion,
q_1 is the *x* component of a quaternion,
q_2 is the *y* component of a quaternion,
q_3 is the *z* component of a quaternion,

Quaternion representation is a four-component complex number in the range of -1 to 1 used for describing orientation of an object with respect to a reference system. Quaternions are used in software applications over Euler angles due to several advantages such as faster code execution speed (Gavilan, 2016). Mahony and Madgwick sensor fusion algorithms compute AHRS as a unit quaternion therefore it was considered appropriate to keep the output label data in the form of a quaternion. In addition, a quaternion does not need further normalising to scale labels for the purpose of training neural networks.

The 28BYJ-48 motors have physical limitations that should not be exceeded for proper function of the turntable. Stepper motors can randomly skip steps during operation at maximum stress and this leads to inaccurate position. To ensure that these limitations are not exceeded, the *x* and *y* stepper motors are constrained to a maximum safe acceleration of 35.3$degs^{-2}$ and maximum angular velocity of 176.6$degs^{-1}$ meanwhile the *z* axis is limited to 7.06$degs^{-2}$ and 35.2$degs^{-1}$ due to heavier load on this motor. Skipped steps were not encountered during use and this was ensured at the end of data collection by homing motors to the initial position and measuring deviation from physical markers. Improving on prior research methods the turntable is rotated in pseudo-random fashion, generated by a Perlin noise function that produces smooth transitions between random numbers allowing for gradual varying acceleration and position. Prior work used periodic functions such as a sinusoidal (Wang, et al., 2019) which is predictable and unrepresentative of motion of most real-world applications (such as a quadcopter).

Due to the design resembling a gimbal, an event termed gimbal lock occurs when one of the axes advances to ±90*deg*, or parallel to the other axes (Hanson, 2006). At this stage, the axis at parallel with another axis no longer has unique control of the reference frame axis and one degree of freedom is lost. This is corrected once the locked axis returns to a prior state under ±90*deg*. In theoretical applications quaternions can be used to avert gimbal lock such as in 3D software, however in the real-world gimbal lock occurs because a rotation around an axis is not arbitrary but rather constrained to the physical location of a motor. Engineers in the Apollo program encountered this problem and suggested a fourth axis to correct gimbal lock when a degree of freedom was lost (Hoag, 1963). This solution could not be installed in this design of the turntable therefore rotations of the turntable were constrained to prevent the occurrence of gimbal lock. The maximum angle for *x* and *y* axes is limited to ±90*deg* in software.

Python Data Collection Script

A python 3 script running on an Intel i7 desktop enables communication with the microcontroller to store the MEMS IMU data. When a serial port connection is established the script transmits commands that home turntable axes to their initial position (zero for each axis). Data is transmitted to the python script running on the desktop at a sample rate of 250Hz, to match filter and MPU-9255 sample rates. Data is received as individual frames in hexadecimal format separated by a line ending delimiter and then unpacked into 32 byte floating point type. The received data frame contains 9 axis IMU data, elapsed time in microseconds, 4 sensor fusion quaternions and 4 ground truth quaternions. The script checks for frame length to detect if data is corrupted and in case that it is, the corrupted frame is discarded. Data is collected for one million frames which lasts 67 minutes and then stored in a *csv* file. Two datasets are stored for training, one containing the Mahony sensor fusion filter quaternion data referred to as the Mahony dataset and another containing the Madgwick sensor fusion data referred to as the Madgwick dataset hereinafter. Limitation of microcontroller hardware prevented both Mahony and Madgwick sensor fusion algorithms running at the same time as the computation time was too long to maintain 250Hz rates as data began corrupting frequently near USB Serial port bandwidth.

Neural Network Training

Tensorflow is a free open-source machine learning library for python used in this project along with Keras high level API. Datasets collected from the python 3 script are loaded using Pandas python

library and processed before training. The Mahony/Madgwick filter columns are excluded from the dataset as they are not needed for training.

Pre-processing Datasets

Firstly, the time elapsed column (in microseconds) is transformed into a delta (difference) of times by subtracting each row from the previous row and then scaled linearly to range of 0 to 1. Relative time (time between IMU data samples) is passed to the neural network as relative time is needed to estimate orientation using a gyroscope (integrating angular velocity). The Mahony and Madgwick sensor fusion algorithms require relative time therefore for an equal comparison it was decided appropriate to input the same information to the neural networks.

Each sensor of the MEMS IMU has a different unit of different scale and range which can increase difficulty in training neural networks. For this reason, a pre-processing module by Scikit-Learn is imported to apply a standard scaler method on the input data. The standard scaler function applies the operation given by equation [13] for each sample n to output the result z_n,

$$z_n = \frac{(x_n - u)}{s} \qquad [13]$$

where,

x_n is input data point,
u is mean of training data,
s is the standard deviation of training data.

Pre-processing data is important because activation functions such as the sigmoid converge to 0 for large negative inputs and 1 for positive inputs (Bruha, 1999). The characteristics of data points further from the origin are less emphasised than data points nearer to the origin because there is a greater rate of change of the activation function (not the case for a rectified linear unit). This leads to reduced accuracy and increased training time of neural networks.

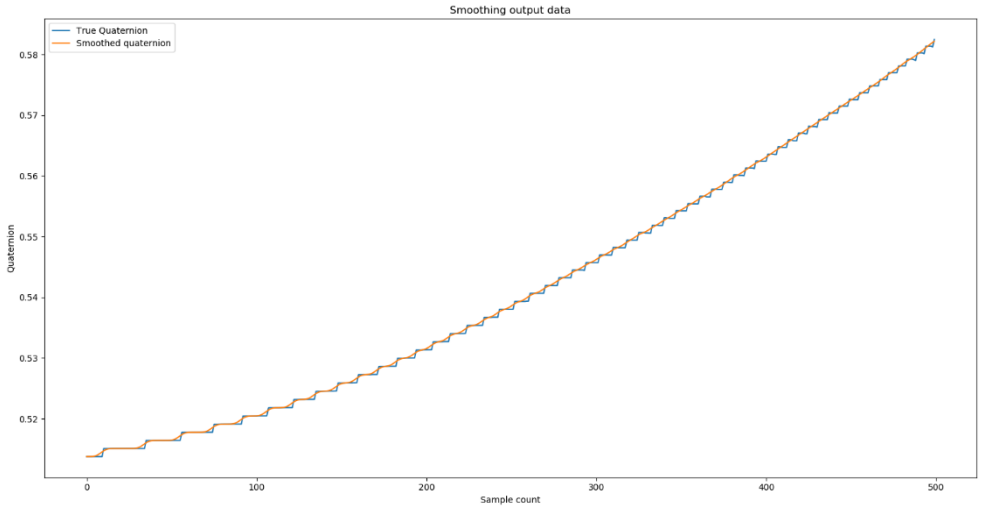

Figure 7 – Plot of w-component of the ground truth quaternion and a rolling mean curve.

Stepper motors are controlled in discrete steps which means the quaternion label data is also discrete as shown by a plot of the output quaternion in Figure 7. However, in reality the motor has momentum that maintains movement during step transitions in contrast to discrete steps. For this reason, the output data is smoothed using a moving average with a period of 5 samples. LSTM and GRU network accuracy reflected positively after this change.

Recurrent neural networks are passed sequences (sometimes referred to as time steps) of data to predict the next target in the sequence thus the dataset is split into equal sequences of 30 frames. The sequence length is not chosen arbitrarily, a longer sequence improves accuracy of the model at the expense of higher computation cost and memory. A sequence length of 30 is chosen in consideration of computing performance of embedded hardware and portable devices. The sequences are shuffled randomly to reduce the chance of the neural network memorising the dataset and overfitting. Overfitting is a term to describe a situation where a neural network optimises solely to the training data such that when tested on new, unseen data (validation data) the model underperforms (Srivastava, et al., 2014). To measure how well the neural network model is performing on unseen data (and whether the neural network is overfitting), the dataset is split into 80% training set and 20% validation set.

Neural Network Model

Two recurrent neural network models are implemented, the LSTM RNN and GRU RNN. The difference between the final models is the type of neuron used in the hidden layers. A smaller sample of a dataset is taken containing 50,000 data frames for the purpose of tuning model parameters. Testing different parameters and configurations on a smaller sample reduces time spent waiting for training to complete. The parameters changed were learning rate, batch size, number of layers, sequence length, loss function, optimisers and hidden layer neuron type, then compared using mean absolute error (MAE) and mean squared error (MSE) metrics.

The final model has the following structure: 10 inputs, 2 hidden layers and a final output layer of 4 neurons corresponding to 4 quaternion outputs. The final RNN and LSTM models differ only in the hidden layer neuron type. During training, dropout layers are added after hidden layers 1 and 2 to reduce overfitting which works by randomly setting input elements to zero to force neurons to learn independently rather than rely on other neurons. The inputs are: 3 accelerometer axes, 3 gyroscope axes, 3 magnetometer axes and delta time; these are the same inputs that Mahony and Madgwick algorithms receive. A training run consists of 100 epochs of batch size 500.

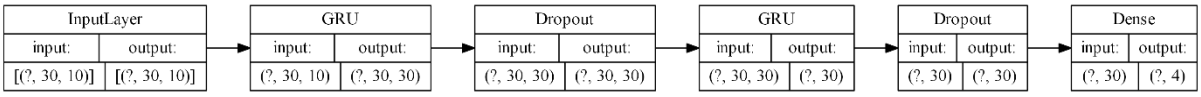

Figure 8- Training model of the GRU neural network.

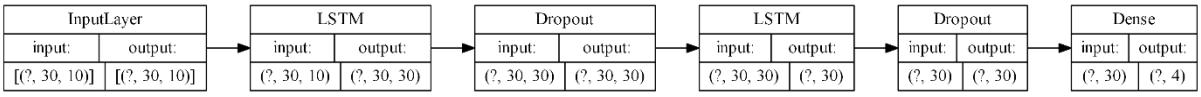

Figure 9 – Training model of the LSTM neural network.

The final model selected for training contains 30 cells for the hidden layers each with a RNN neuron with an additional dropout layer of value 0.1, *Nadam* optimizer with MAE loss function, learning rate of 10^{-3} and batch size of 500. The sequence length of 30 was selected for its quick computation and good prediction trade-off. The model diagram is displayed in Figure 8 and Figure 9 for the GRU and LSTM respectively.

Analysis methods

Data analysis is performed on validation data because training data is not usually representative of how a neural network will perform on new, unseen data. The validation set of 200,000 samples is passed through the neural network and predictions are stored in an array. Performance of the neural network model is assessed by several metrics calculated against the ground truth data provided by the turntable in quaternion representation. Data analysis performed on Mahony and Madgwick sensor fusion algorithms is also compared against the ground truth. The metrics used for measuring error and accuracy are described in this section.

Mean squared error (MSE) is a measure of how far data lies from the actual values by taking the square of errors and averaging over the entire set, given by [14]. MSE is often used in loss functions of neural networks because the model converges quickly when errors are large and is easily differentiable for backpropagation (Kawaguchi, 2000).

$$MSE = \frac{1}{N}\sum_{n=0}^{N}(y_n - y'_n)^2 \quad [14]$$

where,
y_n is the estimated value,
y'_n is the true value,
N is the sample size.

A model that fits the actual data perfectly will have zero MSE as there are no errors (Mishra, 2019). Root mean squared error (RMSE) is preferred in measuring errors over MSE because it has the advantage of being in the same base unit as the data which makes it easier to interpret the result. RMSE is a quadratic score rule and is given by taking the square root of MSE, equation [15].

$$RMSE = \sqrt{MSE} \quad [15]$$

MSE and RMSE range from 0 to infinity and are negatively-oriented (the lower the better). Unfortunately, both MSE and RMSE suffer from the same drawback that large errors of a few outliers are emphasised more than the rest of the data (due to squaring of errors) and not how well the rest of the data fits with observations. For this reason, other metrics are used in conjunction with RMSE to improve knowledge about the model's performance.

Mean absolute error (MAE) represents the average magnitude of errors between predictions and actual observations. Similarly to RMSE, MAE is negatively-oriented and ranges from 0 to infinity but the key difference is that less weight is given to outliers and MAE values increase with variance of errors while RMSE can remain relatively fixed (variance expresses how far values are from the mean). MAE is a linear score which means that differences between observations are weighted equally in average. MAE is given by equation [16].

$$MSE = \frac{1}{N}\sum_{n=0}^{N} |y_n - y'_n| \quad [16]$$

where,
y_n is the estimated value,
y'_n is the true value,
N is the sample size.

The coefficient of determination, R^2 indicates how much of the variation in data is due to input values. Values of R^2 can express the percentage of data points passing through the observations which can indicate the goodness of fit. R^2 is positively-orientated in the range of 0 to 1, with 1 indicating that all variations in data are explained by the input (Swalin, 2018). Sometimes negative values of R^2 are calculated which means that predicted values are not correlated with observations. Equation for the coefficient of determination is given by equation [17]

$$R^2 = 1 - \frac{\sum_n^N (y_n - y'_n)^2}{\sum_n^N (y_n - \bar{y}_n)^2} \quad [17]$$

where,
y_n is the estimated value,
y'_n is the true value,
$\bar{y}_n$ is the mean of true values,
N is the sample size.

There are limitations of R^2 that should be considered when using the metric, R^2 can increase when new variables are added to the model without detecting this change. Additionally, R^2 can increase with an increased number of data points which can seem that the model is improving when it is not (Swalin, 2018). Also, even when R^2 shows a strong correlation between variables it does not prove causality. This is addressed by using adjusted R^2 which is given by equation [18]

$$R^2_{adj} = \frac{(1 - R^2)(1 - N)}{N - k - 1} \quad [18]$$

where,
k is the number of independent variables,
N is the sample size.

Results and Discussion

Parameter Tuning

Prior to training of the final neural network models, the dataset is reduced to a sample count of 50,000 and tested with various configurations and parameters. The results for an RNN hidden layer model are shown in Figure 10 with four different parameters tested.

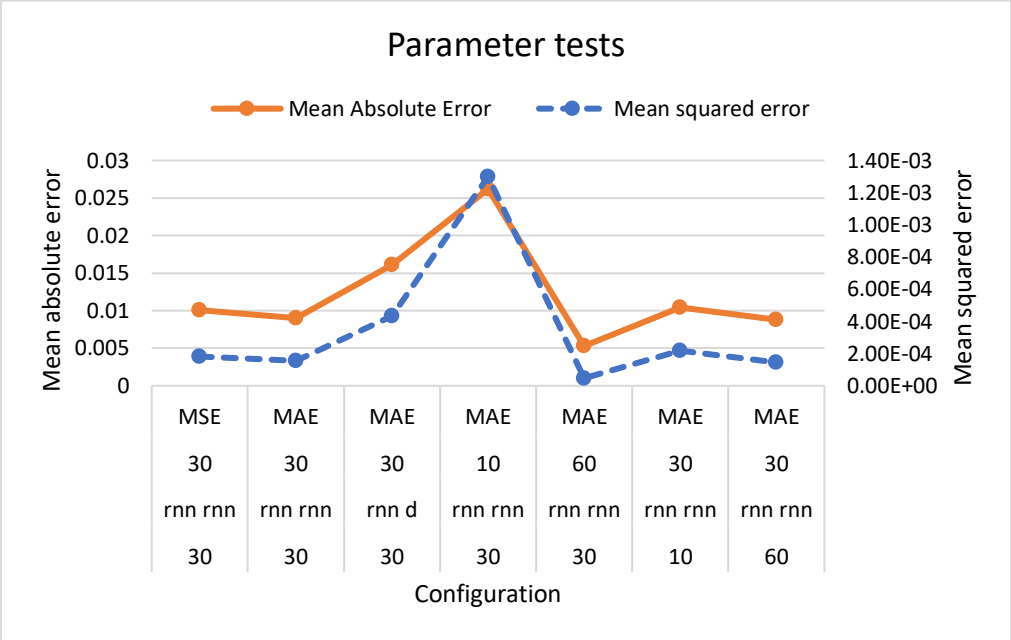

Figure 10 – A plot of MSE and MAE comparing differences in parameters after 100 epochs of training.

First, the baseline model contains two GRU layers (denoted *rnn*) of 30 neurons each, applying MSE as the loss function and sequence length of 30 time steps. The first model had an MSE of 1.83×10^{-4} and MAE of 0.0101. The next test had the loss function changed to MAE which resulted in improved reduction of errors as the best model over 100 epochs converged to MSE of 1.56×10^{-4} and MAE of 0.00903. In this particular implementation, results demonstrate that MAE reduces errors in fewer epochs than MSE and for further runs MAE was selected as the loss function, this configuration is selected as the baseline. One key note of interest was to check if using a GRU layer and a layer of densely connected neurons (perceptron) improved performance of the model as researchers of paper (Gonzaleza & Catania, 2019) used MLP neural networks with good results. Unfortunately, both MSE and MAE showed a substantially reduced performance with the perceptron layer(denoted *d*) and thus future runs replaced the perceptron with the same type GRU in hidden layer 2. The greatest effect on MSE and MAE reduction is seen when the number of neurons in hidden layers was increased to 60 (brings the total to 120) and a neuron count of 10 increased errors by the largest amount with MSE of 1.30×10^{-3} and MAE of 0.0262. Finally, reducing sequence length to 10 time steps caused a slim increase in errors and doubling sequence length to 60 had little change, as MAE improved by only 3% and MSE by 8% over the baseline.

In conclusion, the greatest improvement was achieved by increasing the neuron count in hidden layers and increasing sequence length, however this comes at a cost of model complexity and higher computational cost. With consideration of embedded hardware and battery powered devices, the number of neurons in hidden layers was selected to 30 and sequence length of 30 as a good trade-off between accuracy and complexity. The loss function selected was MAE as it offered the best convergence and two LSTM/GRU hidden layers.

Pre-processing Data

Pre-processing data played a crucial role in training neural networks as the raw input data had varying scales and ranges. Figure 11 shows an example of raw sensor data from the MPU-9255. In this snapshot, the accelerometer values vary about 20 ms^{-2} but the magnetometer values vary a staggering 2,400$microT$.

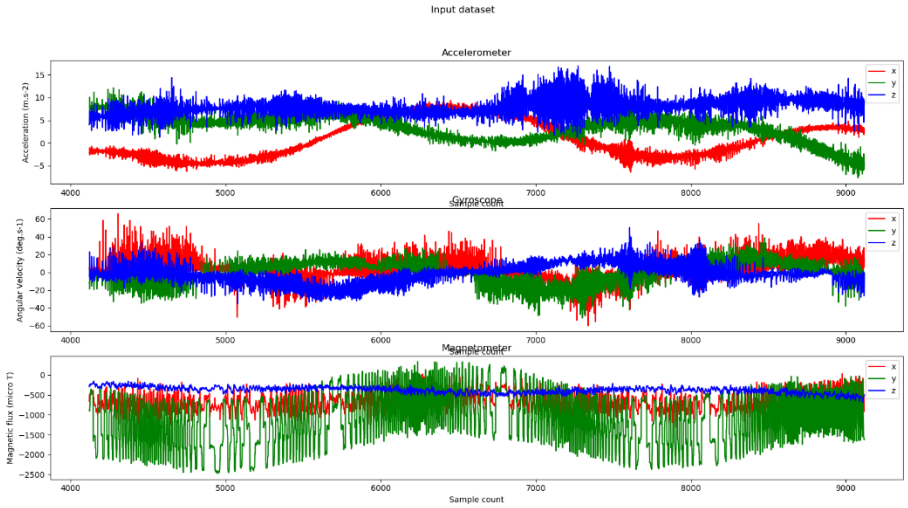

Figure 11- Plot of raw input data (using a sample of 5000 at random) from the MPU-9255.

Additionally, there is a substantial amount of noise on both the accelerometer and gyroscope plots which was probably due to the mechanical vibrations of stepper motors, each moving at different velocities. The magnetometer data has an exceptional amount of noise, the most out of all sensors, particularly on the *y* axis (indicated by the green line). It is suspected that the source is the *y* axis stepper motor as its placement is the closest to the MPU-9255 mounting position, as seen in Figure 5. The noise appears to vary in amplitude and frequency as it is being condensed in some areas and expanded in others, this corresponds with the changes in acceleration of the *y*-axis motor.

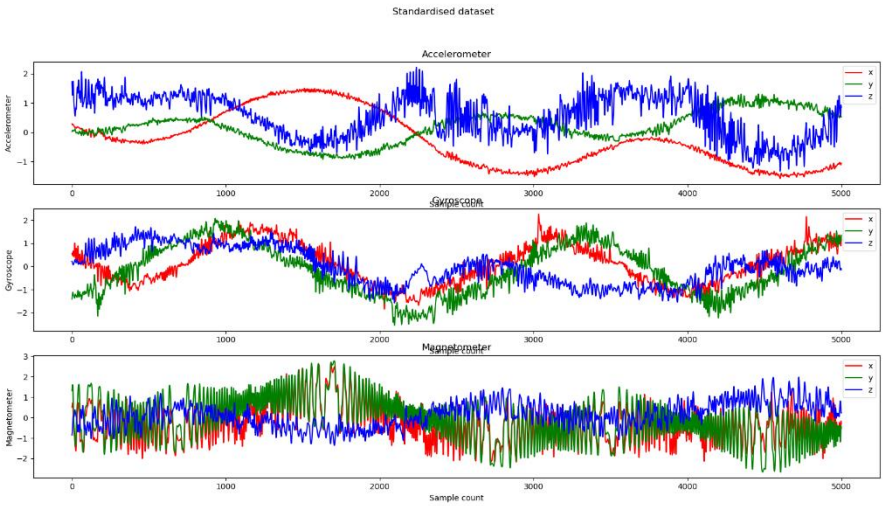

Figure 12- Input data after pre-processing function.

Figure 12 shows a random sample of input dataset after the pre-processing function was applied. The inputs are now scaled on each axes and centred around origin. Additionally, the amount of clutter on the graphs is reduced versus the raw sensor information data from Figure 11. Features of the dataset remain recognisable and original characteristics were preserved. Pre-processing data had the effect of improving training accuracy and reducing the time it took to converge to a solution.

Training Neural Networks

The graph in Figure 14 shows validation and training accuracy over 100 epochs of the LSTM neural network. After the first epoch, training accuracy is at 85.0% and validation accuracy at 88.3% and eventually begins converging to an accuracy of 92.7% and 93.4% on the last epoch for training and validation respectively.

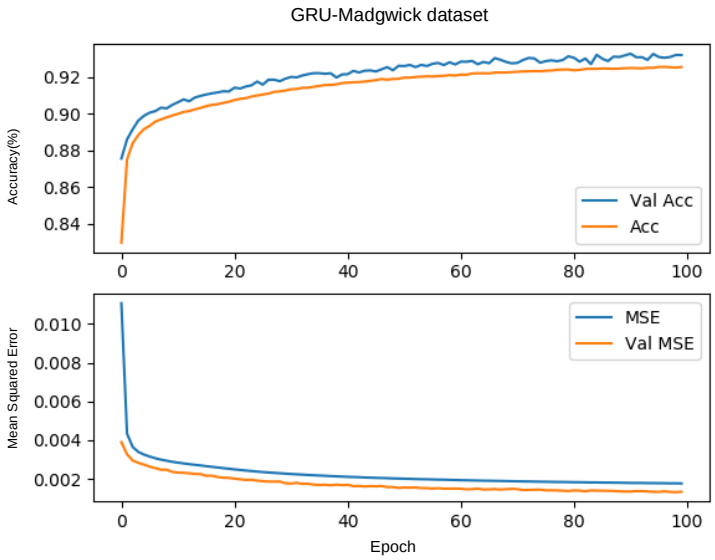

Figure 13 – GRU neural network training history over 100 epochs on the Madgwick dataset.

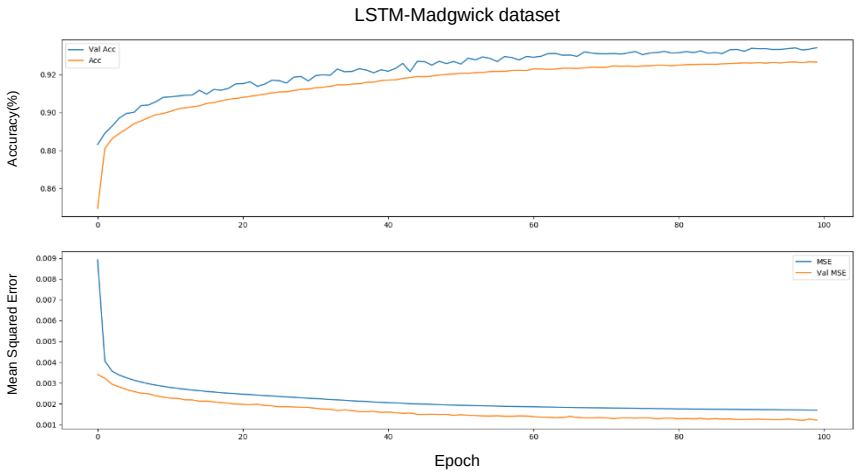

Figure 14 – LSTM neural network training history over 100 epochs on the Madgwick dataset.

Upon inspection, it may seem counter-intuitive that validation accuracy is greater than training accuracy, after all the model has never seen validation data. However, the dropout layers and other types of regularisation are turned off during validation which means that the neural network encounters fewer erroneous datapoints. In the opposite scenario, where training accuracy is greater than validation accuracy and the curves diverged would be indication of overfitting. This graph does not contain such features and instead the accuracy converges. Figure 13 shows mean squared error of training and validation of a GRU model on the Madgwick dataset. Again, the validation error is lower than training error and there is no indication that the model is overfitting. Instead, it appears that the model has not yet converged, and further training epochs are needed to minimise error.

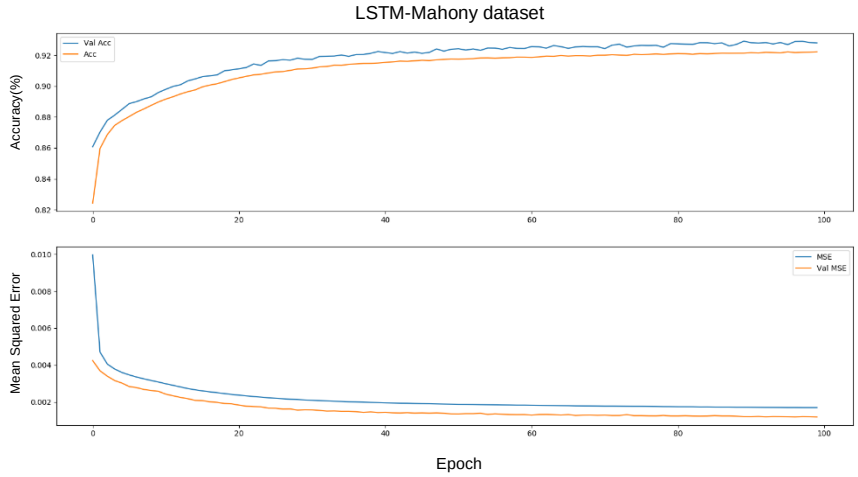

Figure 15- LSTM neural network training history over 100 epochs on the Mahony dataset.

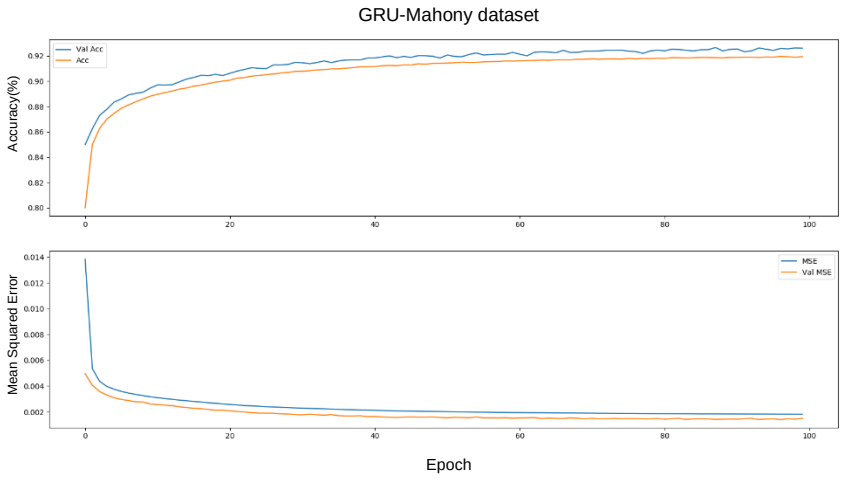

Figure 16- GRU neural network training history over 100 epochs on the Mahony dataset.

Figure 15 and Figure 16 display the validation MSE of the LSTM and GRU neural networks on the Mahony dataset. The LSTM model displays a slightly decreased error over the GRU model which suggests that the LSTM is outperforming the GRU on the Mahony dataset. The LSTM is outperforming the GRU by a narrow margin when trained on the Madgwick dataset. Further epochs are needed to verify whether the LSTM is better suited over the GRU. Despite this, the LSTM converged sooner and to a lower error value under both datasets.

Analysis of Results

This subsection compares and discusses the results of recurrent neural networks versus Mahony and Madgwick sensor fusion algorithms. Metrics from the methodology section are presented in tables and model predictions are plotted against sensor fusion algorithm estimates. Pre-processing of the output quaternion was disabled during metric calculation.

	Mahony dataset		
Metric	**Mahony Sensor Fusion**	**GRU**	**LSTM**
MSE	0.392	0.0129	0.00954
RMSE	0.626	0.114	0.0977
MAE	0.386	0.0776	0.0656
$\mathbf{R^2_{adj}}$	0.0429	0.553	0.664

Table 1 – Results of the error metrics on the Mahony dataset (3 significant figures).

	Madgwick dataset		
Metric	**Madgwick Sensor Fusion**	**GRU**	**LSTM**
MSE	0.0192	0.0150	0.00711
RMSE	0.139	0.122	0.0843
MAE	0.103	0.0824	0.0567
$\mathbf{R^2_{adj}}$	0.231	0.304	0.694

Table 2 – Results of the error metrics on the Madgwick dataset (3 significant figures).

Table 1 and Table 2 display errors and metrics calculated for the Mahony and Madgwick dataset in comparison with the GRU and LSTM neural networks. In Table 1, the neural networks scored substantially better in RMSE versus the Mahony algorithm with a percentage improvement of 82% and 84% for the GRU and LSTM respectively. The MAE was reduced by 80% and 83% by the GRU and LSTM. The Mahony dataset scored a particularly low score of RMSE 0.626 with regards to a quaternion representation. The errors are considerably large and would most likely lead to unreliable predictions. Moreover, the difference between RMSE and MAE is positive and the largest out of the other models which indicates that outliers present in the predictions contributed to large errors. The RMSE scored 0.626 while MAE scored 0.386, a difference of 62%. Furthermore, the adjusted R^2 value for the Mahony suggests a weak fit and quaternion estimates may not correlate with the true turntable quaternion, reasons for this are examined using graphical plots. The GRU and LSTM score an R^2_{adj} value of 0.55 and 0.66, which suggests that over half the data points fall on the true attitude line. While this is not ideal, it is certainly an improvement over the Mahony algorithm.

Table 2, shows a good performance by the Madgwick sensor fusion algorithm with RMSE and MAE of 0.139 and 0.103 respectively. The neural networks outperformed the Madgwick in RMSE as the GRU scored 0.122 and LSTM scored 0.0843, an improvement of 12% and 39% respectively. The three methods provide similar performance and adequate AHRS estimation; the RMSE values correspond to an expected mean quaternion error of about 7% for Madgwick, 6% for Mahony and 4% for the LSTM model. Again, the difference between RMSE - MAE indicates a presence of outliers in the data that increased overall errors and reduced goodness of fit. The adjusted R^2_{adj} is only 0.23 for the Madgwick sensor fusion algorithm despite similar RMSE and MAE values to GRU and LSTM. The LSTM and GRU outperformed Madgwick in the R^2_{adj} score with a value of 0.69 and 0.30 for the GRU.

Although the Madgwick and neural networks perform similarly in terms of errors, the LSTM neural network solution consistently beats the other methods in every metric and dataset.

Four graphs were plotted containing four quaternion components to investigate estimations by algorithmic sensor fusion against predictions of the neural networks. The graphs were plotted using a random sample of 10,000 frames from the validation dataset. Every graph contains the neural network prediction (indicated by red line), the sensor fusion algorithm estimation (blue), ground truth (green) and a final smoothed out curve using a moving average of 100 steps (black). Smoothing was added to visualise the average trend of the neural network predictions due to fluctuations and noise.

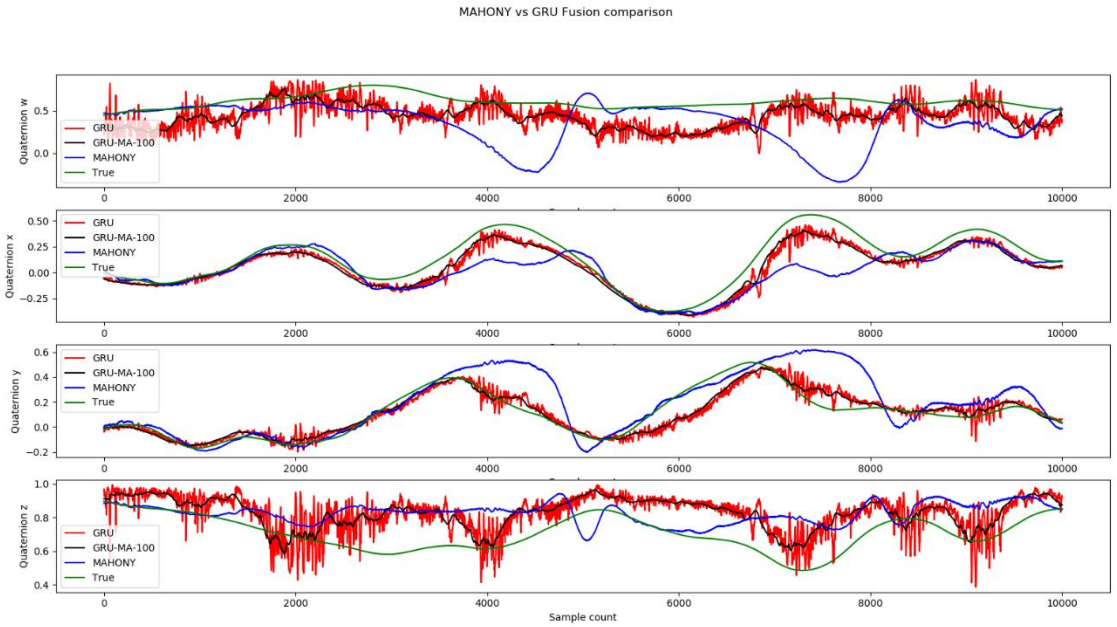

Figure 17 – Quaternion AHRS plot of Mahony vs GRU sensor fusion.

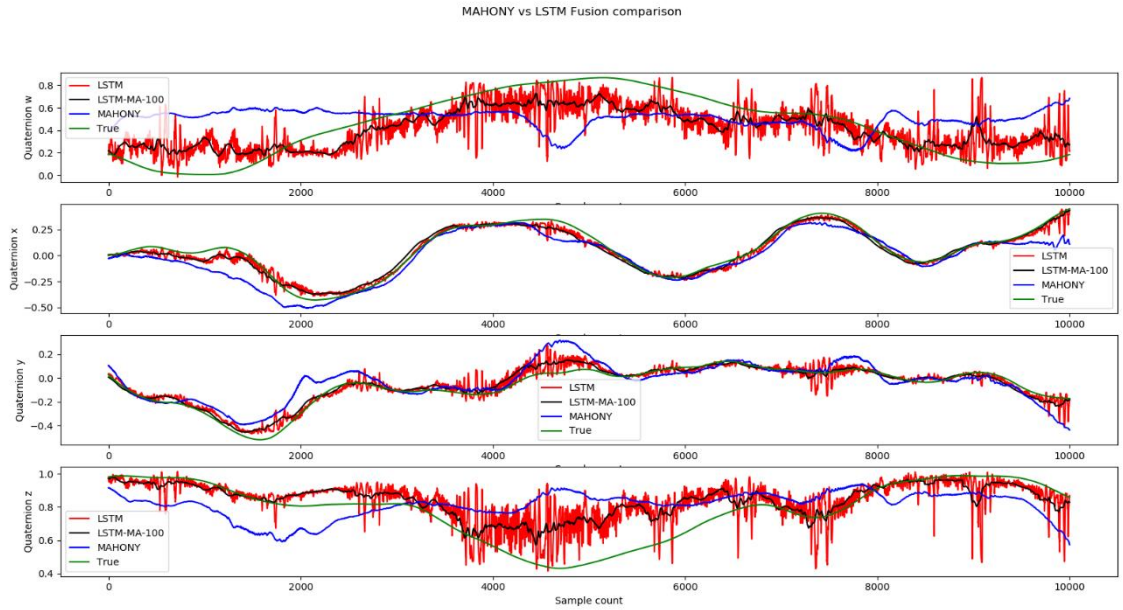

Figure 18 - Quaternion AHRS plot of Mahony vs LSTM sensor fusion.

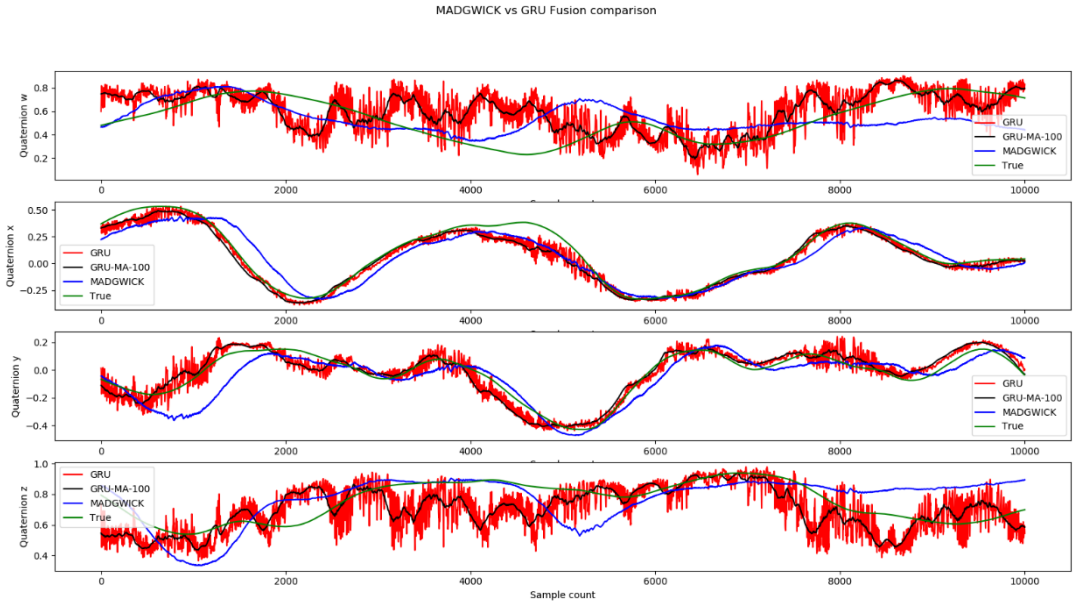

Figure 20- Quaternion AHRS plot of Madgwick vs GRU sensor fusion.

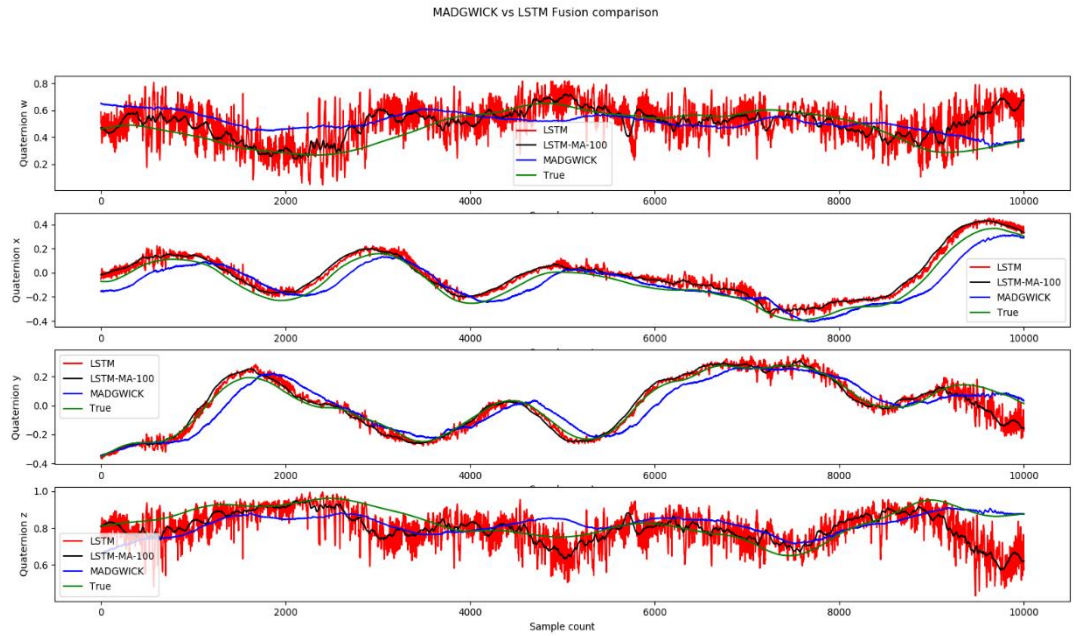

Figure 19 - Quaternion AHRS plot of Madgwick vs LSTM sensor fusion.

Figure 17 shows a Mahony filter versus GRU plot of quaternion estimates. Upon reviewing the graphical plots, it appears that the Mahony sensor fusion algorithm occasionally diverged from the ground truth curve in various places, for instance at the 4,000-5,000 sample count range where it can be seen that the Mahony curve deviates and then gradually returns to the true quaternion values. These erroneous estimations in the data clarify the reason for large error RMSE values and weak fit indicated by R^2_{adj}. The explanation for these deviations is likely magnetic disturbances of the stepper motors which the Mahony filter relied on, despite the unreliable magnetic sensor readings. It seems the Mahony filter performs poorly with low quality input data. The GRU sensor fusion model performed best in predicting the *x* and *y* quaternions however, likely due to the same reason the Mahony filter struggled, quaternion *z* and *w* components veer off from ground truth and contain significantly more noise than *x* and *y* components.

The GRU and LSTM estimations contain a lot of noise in general but when smoothed using a moving average (window of 100 samples) to remove most of the fluctuations, the trend of the curve trails the true quaternion. The plots have a striking resemblance to the paper (Liu, et al., 2018) in which the output Euler angles also encounter random noise. There are places where the GRU estimations deviate from the actual but eventually correct to the true quaternion line in a similar fashion to the Mahony although less prominently.

A plot of LSTM versus Mahony can be seen in Figure 18 which demonstrates similar behaviour encountered by the GRU. Even though the LSTM scored better error values all around, there is still a significant amount of noise and uncertainty particularly on the *w* and *z* quaternion components. In addition, further confirmation of the Mahony filter struggling at some places such as 0 to 2,000 where every subplot shows a deviation of the estimate from the ground truth. The difference between neural networks and the Mahony filter is visible the most when compared to the *x* and *y* components where neural networks manage to salvage noisy sensor information and output an attitude that if filtered, could still be used. In a scenario where a quadcopter drone encounters a lot of magnetic disturbances due to motor noise, the GRU and LSTM could estimate pitch and roll from the quaternion *x* and *y* components enough for the drone to stabilise albeit following a magnetic North would prove to be difficult.

Investigating the Madgwick versus GRU plot in Figure 20 illustrates a significant improvement over the Mahony filter. The Madgwick fusion algorithm performs a good estimation of true quaternions including the *w* and *z* components where the Mahony filter encountered difficulty. Outliers are still present in certain areas such as in the range of 4,500-5,000 of the *w* and *z* components but are shortly corrected. The GRU fails to estimate the *w* and *z* components as shown by excessive noise and random walk. On the other hand, the LSTM model (Figure 19) manages to predict the *w* and *z* components with a decent fit, which agrees with the result of the R^2_{adj} metric. The LSTM and Madgwick perform similarly in estimating the w and z components with the occasional outliers, however the Madgwick does so with much less turbulence. The weak R^2_{adj} score for the Madgwick could be due to lagging of the true quaternion. This can be seen best in the *x* and *y* components, the Madgwick filter is lagging the true value by about 50-100 samples while the filtered GRU and LSTM curves respond sooner.

The plots indicate a generally good fit by the neural networks in *x* and *y* components with arguably better performance than the Madgwick. The result is reversed when comparing *w* and *x* quaternions as the Madgwick demonstrates a fantastic resistance to noise with occasional deviations that are corrected promptly. The LSTM learned to overcome the distortions and noise in *w* and *z* components over long term, albeit fluctuations in the output persist and would require filtering in a real world application. Further work should aim to reduce and discover the source of these errors.

Sources and Discussion of Errors

Systematic Errors

The primary source of errors in this project was the magnetometer data. The graph plots of the input data show large varying magnetic disturbances in the *x* axis and *y* axis. The source of these disturbances is attributed to the turntable's stepper motors which produce strong magnetic fields when rotating. The position of the MEMS IMU mount was too close to the internal magnets and coils of the stepper motors due to the design of mounts holding everything together. During magnetic calibration routine, soft iron and hard iron biases were compensated for but not dynamic changes caused by the motors. The turntable mounts were fabricated in 3D printed PETG material which is durable but relatively flexible compared to PLA and ABS therefore the size of the turntable was limited. The accelerometer and gyroscope sensors showed vibrations and fluctuations in the inputs which further increased noise and errors. The source of vibrations is likely the stepper motors because when the turntable was not moving, the variation in input data was reduced. This suggests that this particular

model of stepper motors is not ideal for the turntable and in future improvements another motor should be used for the design. During turntable design, it was noted that each stepper motor has 2 degrees of looseness where the motor has no holding torque. This means that when a stepper motor changed directions there may have been a discrepancy between actual position and measured position of the shaft.

Random Errors

Random types of error are unavoidable and unpredictable. Although power supplied to the motors and MEMS IMU was kept separate to reduce electrical noise, there are always small variations in the voltage of linear regulators and buck converters. For example, temperature changes and electrical load will affect the voltage supplied to the MEMS IMU. MEMS IMUs are also affected by temperature changes which can affect sensor readings such as the gyroscope (El-Diasty, et al., 2007). This project did not measure or control temperature therefore it is unknown how much temperature changes affected the final result.

Stepper motors are known to skip steps when stressed to their maximum working conditions. Skipped steps are random and can happen when the motor is forced to move faster than the manufacturer recommends. Skipped steps are difficult to measure but if they occur frequently then there is a difference between the motor home position and initial position. To ensure that this did not happen, the turntable was constrained to maximum speeds and accelerations and at the end of each run tested if the home position deviated from the initial markers.

Measurement Errors

Several measurements were taken to ensure proper operation of the turntable. The gear ratio was calculated by using a protractor to measure the difference between initial position and ten full rotations. The gear ratio was adjusted in software, however because the measurement was taken using a protractor with degree increments, the measurement error is ±1.0 degree. Moreover, the physical markers for calibrating initial position were measured using a straight edge of a ruler. This method of measurement is imperfect as some measurement errors originate due to parallax and the ruler. Time was measured for the calculation of AHRS using sensor fusion algorithms and neural networks, but time was measured using a microcontroller that relies on a quartz crystal for time keeping. The microcontroller's ability to maintain accurate time is not perfect and measurement precision was limited to ±1microsecond, therefore some variation is expected.

Summary

The quaternion estimation plots indicate a similar performance for the LSTM and GRU in the x and y components although both neural networks exhibit a large amount of noise. This noise can be filtered using a simple moving average which demonstrates that the average of noisy values is on track with the true AHRS. This data noise could be due to the large amount of vibrations and disturbances of the magnetometer data that the neural networks were trained on. Each neural network processes a time step length of only 30 frames while moving average filtered 100 data points therefore it may be difficult to output a smooth result by observing short sequences. Nevertheless, further work is required to determine the true cause of the fluctuating output.

Noise in the magnetometer data proved a significant challenge for all tested methods to estimate the yaw direction as can be seen by noise and drift on the quaternion outputs of *w* and *z* plots. Determining the *x* and *y* components contained less errors likely because there is accurate data from the gyroscope and accelerometer that can be combined for estimation of pitch and roll. One interesting characteristic of the LSTM and GRU plots is the quick response to changes in attitude and heading where the Mahony and Madgwick algorithms lagged ground truth. The LSTM and the Madgwick algorithms performed better in the *w* and *z* components despite noise and disturbances that

the other methods found challenging. These results suggest that the LSTM model and Madgwick algorithm are less susceptible to noisy environments than the Mahony algorithm and the GRU model.

Conclusion

Further work

Data Collection

The turntable design should be revised by extending the length of the brackets and mounts such that the MEMS IMU placement is far enough from stepper motors that the magnetic field disturbances are negligible. The design could incorporate higher quality motors capable of faster rotations and smoother movement as well as an extra axis for four degrees of freedom to solve the gimbal lock problem. Initial position calibrations can be performed using a more accurate method such as using an LED and receiver diode. These changes could extensively test a wider range of movements such as full spins on various axes. Alternatively, a high quality commercially available turntable could be used to mount one or more MEMS IMU models at the same time. Different types of MEMS IMUs will have varying drifts therefore testing how well the neural network adapts to these biases will be important for deployment on various devices. Testing edge cases, such as free fall and extreme angular velocities is critical in applications of quadcopter drones where drastic movements are common. A temperature sensor could be included in the MEMS IMU such that compensation of drift and biases due to temperature is included in the neural network. These improvements will enhance the quality and diversity of data which will be crucial for training reliable neural networks for sensor fusion of MEMS IMUs.

Neural Network Models

This research project trained two recurrent neural networks of two hidden layers however other neural network types such as convolutional neural networks could be tested and compared. Further work should test the effect of changing the number of hidden layers on the errors and accuracy of the final model. Investigate the cause of noise in the output quaternion data and examine methods to reduce fluctuations in the output of the LSTM and GRU models. Further exploration of neural networks in AHRS estimation could be combined with a GPS module to provide enhanced INS position estimates, particularly when a GPS signal is lost or spurious.

Conclusion

The purpose of this research project was to implement and compare machine learning solutions to sensor fusion algorithms of MEMS IMU AHRS estimation. The data was collected using a three-axis turntable in order to rotate the MEMS IMU in various orientations, speeds and accelerations for long periods of time. Over 2 hours of data was collected at a sampling rate of 250Hz using a low cost MPU-9255 sensor. Two recurrent neural networks (LSTM and GRU) were tested against two popular sensor fusion algorithms (Mahony and Madgwick). Although the input data was noisy and affected by magnetic distortions, both the sensor fusion algorithms and neural networks were capable of estimating attitude with the LSTM outperforming other methods in MSE, RMSE, MAE and R^2_{adj} metrics. The graph plots of output quaternion data support these results however the Madgwick and Mahony sensor fusion algorithms had significantly fewer fluctuations compared to the GRU and LSTM. From the graphical comparison of LSTMs and GRUs there was a discernible difference of convergence time to true quaternion over the Madgwick and Mahony algorithms. The results indicate that recurrent neural networks are capable of sensor fusion but may require help from filtering tools to smooth out the data for use in real world applications. Further research should experiment with optimising the neural networks in order to reduce fluctuations and noise in the output data as well as improve performance in the estimation of *w* and *z* components of a quaternion representation of AHRS.

References

Aboelmagd Noureldin, T. B. K. M. D. E. A. E.-S., 2008. Performance Enhancement of MEMS-Based INS/GPS Integration for Low-Cost Navigation Applications. *IEEE Transactions on Vehicular Technology,* 58(3), pp. 1077-1096.

Acarnley, P., 2002. *Stepping Motors a guide to theory and practice.* 4 ed. London, UK: The Institution of Engineering and Technology.

Alpaydin, E., 2020. *Introduction to Machine learning.* 4 ed. Cambridge, Massachusetts: MIT Press Ltd .

AsahiKasei, 2013. *AK8963.* [Online]
Available at: https://download.mikroe.com/documents/datasheets/ak8963c-datasheet.pdf
[Accessed 1 February 2020].

Ayodele, T. O., 2010. Types of Machine Learning Algorithms. In: Y. Shang, ed. *New Advances in Machine Learning.* London: IntechOpen, pp. 19-48.

Barbic, M., 2002. Magnetic wires in MEMS and bio-medical applications. *Journal of Magnetism and Magnetic Materials,* 249(1-2), pp. 357-367.

Bengio, Y., Simard, P. & Frasconi, P., 1994. Learning Long-Term Dependencies with Gradient Descent is Difficult. *Transactions of Neural Networks ,* 5(2), pp. 157-166.

Beverini, N. et al., 2015. High-Accuracy Ring Laser Gyroscopes: Earth Rotation Rate and Relativistic Effects. *Journal of Physics: Conference Series.*

Boden, M., 2001. *A guide to recurrent neural networks and backpropagation,* Halmstad, Sweden: School of Information Science, Computer and Electrical Engineering, Halmstad University.

Britz, D., 2015. *Recurrent Neural Network Tutorial – Implementing a GRU/LSTM RNN with Python and Theano.* [Online]
Available at: http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/
[Accessed 1 February 2020].

Brossard, M., Barrau, A. & Bonnabel, S., 2020 . AI-IMU Dead-Reckoning. *IEEE Transactions on Intelligent Vehicles* .

Bruha, I., 1999. From machine learning to knowledge discovery: Survey of preprocessing and postprocessing. *Intelligent Data Analysis,* 4(3-4), pp. 363-374.

Budker, D., Kimball, J. & F., D., 2013. *Optical Magnetometry.* 1 ed. Cambridge: Cambridge University Press.

Cavallo, A. et al., 2014. *Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation.* Cape Town, South Africa, The International Federation of Automatic Control.

Chong, S. et al., 2016. Temperature drift modeling of MEMS gyroscope based on genetic-Elman neural network. *Mechanical Systems and Signal Processing,* 72-73(1), pp. 897-905.

El-Diasty, M., El-Rabbany, A. & Pagiatakis, S., 2007. Temperature variation effects on stochastic characteristics for low-cost MEMS-based inertial sensor error. *Measurement Science and Technology,* 18(11), p. 3321.

Gavilan, D., 2016. *Performance of quaternions in the GPU.* [Online]
Available at: https://tech.metail.com/performance-quaternions-gpu/
[Accessed 3 February 2020].

Gebre-Egziabher, D., Hayward, R. & Powell, J., 2002. *A low-cost GPS/inertial attitude heading reference system (AHRS) for general aviation applications.* Palm Springs, CA,USA, Institute of Electrical and Electronics Engineers.

Gonzaleza, R. & Catania, C. A., 2019. Time-delayed multiple linear regression for de-noising MEMS inertial sensors. *Computers & Electrical Engineering,* Volume 76, pp. 1-12.

Grewal, M. S. & Andrews, A., 2010. Applications of Kalman Filtering in Aerospace 1960 to the Present. *IEEE control systems,* 30(3), pp. 69 - 78.

Gui, P., Tang, L. & Mukhopadhyay, S., 2015. *MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion.* Auckland, New Zealand, Institute of Electrical and Electronics Engineers.

Hanson, A. J., 2006. *Visualizing Quaternions.* 22 ed. San Francisco, CA, USA: Elsevier Science.

He Li, L. Z., 2010. The study and implementation of mobile GPS navigation system based on Google Maps. *International Conference on Computer and Information Application,* pp. 87-90.

Hellmers, H., Norrdine, A., Blankenbach, J. & Eichhorn, A., 2013. *An IMU/magnetometer-based Indoor positioning system using Kalman filtering.* Montbeliard-Belfort, France, Institute of Electrical and Electronic Engineers.

Hoag, D., 1963. *Apollo Guidance and Navigation: Considerations of Apollo IMU Gimbal Lock,* Cambridge, Massachusetts, USA: MIT Instrumentation Laboratory .

Hochreiter, S. & Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Computation,* 9(8), pp. 1735-1780.

InvenSense Inc., 2014. *MPU-9255 Product Specification.* [Online]
Available at: https://stanford.edu/class/ee267/misc/MPU-9255-Datasheet.pdf
[Accessed 1 February 2020].

Iurato, G., 2015. On the historical evolution of gyroscopic instrumentation: a very brief account. 28 March.

Jaw-Kuen Shiau, C.-X. H. M.-Y. C., 2012. Noise Characteristics of MEMS Gyro Null Drift andTemperature Compensation. *Journal of Applied Science and Engineering,* 15(3), pp. 239-246.

Jiang, C. et al., 2018. Performance Analysis of a Deep Simple Recurrent Unit Recurrent Neural Network (SRU-RNN) in MEMS Gyroscope De-Noising. *Sensors,* 18(12), p. 4471.

Judy, J. W., 2001. Microelectromechanical systems (MEMS): fabrication, design and applications. *Smart Materials and Structures,* 10(6), pp. 1115-1134.

Jung, S. & Kim, H., 2017. Analysis of Amazon Prime Air UAV Delivery Service. *Journal of Knowledge Information Technology and Systems,* 12(2), pp. 253-265.

Kalman, R. E., 1960. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME--Journal of Basic Engineering,* Volume 82, pp. 35-45.

Kawaguchi, K., 2000. *Backpropagation Learning Algorithm 2.4.4.* [Online]
Available at: http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-

html/node22.html
[Accessed 3 February 2020].

Kelleher, J. & Dobnik, S., 2017. *What is not where: the challenge of integrating spatial representations into deep learning architectures.* Gothenburg, Sweden, University of Gothenburg.

Kiatronics, 2013. *28BYJ-48 – 5V Stepper Motor.* [Online]
Available at: https://datasheetspdf.com/pdf-file/1006817/Kiatronics/28BYJ-48/1
[Accessed 1 February 2020].

Kim, K., Kim, J. S. & Kim, Y. J., 2015. *Application of nonlinear complementary filters to human motion analysis.* Boston, MA, USA, Institute of Electrical and Electronics Engineers.

King, A. D., 1998. Inertial Navigation - Forty Years of Evolution. *GEC Review,* pp. 140-149.

Kolanowski, K. et al., 2018. Multisensor data fusion using Elman neural networks. *Applied Mathematics and Computation,* Volume 319, pp. 236-244.

Kolanowski, K. et al., 2013. *Nine-Axis IMU sensor fusion using the AHRS algorithm and neural networks.* Pilsen, Czech Republic, University of West Bohemia, pp. 23-24.

Kong, X., 2004. INS algorithm using quaternion model for low cost IMU. *Robotics and Autonomous Systems,* 46(4), pp. 221-246.

Konvalin, C., 2009 . *Compensating for Tilt, Hard-Iron, and Soft-Iron Effects.* [Online]
Available at: https://www.fierceelectronics.com/components/compensating-for-tilt-hard-iron-and-soft-iron-effects
[Accessed 1 February 2020].

Kostadinov, S., 2018. *Recurrent Neural Networks with Python Quick Start Guide.* Birmingham, UK: Packt Publishing Ltd..

Lenz, J. & Edelstein, S., 2006. Magnetic sensors and their applications. *IEEE Sensors Journal,* 6(3), pp. 631-649.

Liu, Y., Zhou, Y. & Li, X., 2018. *Attitude Estimation of Unmanned Aerial Vehicle Based on LSTM Neural Network.* Rio de Janeiro, Brazil, International Joint Conference on Neural Networks .

Ludwig, S. A. & Burnham, K. D., 2018. *Comparison of Euler Estimate using Extended Kalman Filter, Madgwick and Mahony on Quadcopter Flight Data.* Dallas, TX, USA , Institute of Electrical and Electronics Engineers.

Madgwick, S. O. H., 2010. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays,* Bristol: Report x-io and University of Bristol .

Mahony, R., Hamel, T. & Pflimlin, J.-M., 2008. Nonlinear Complementary Filters on the Special Orthogonal Group. *IEEE Transactions on Automatic Control ,* 53(5), pp. 1203-1218.

Mandic, D. P. & Chambers, J., 2001 . *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability.* 1 ed. Hoboken, New Jersey: John Wiley and Sons Ltd .

Mishra, D., 2019. *Regression: An Explanation of Regression Metrics And What Can Go Wrong.* [Online]
Available at: https://towardsdatascience.com/regression-an-explanation-of-regression-metrics-and-what-can-go-wrong-a39a9793d914
[Accessed 3 February 2020].

Mizell, D., 2003. *Using Gravity to Estimate Accelerometer Orientation,* Seattle: Seventh IEEE International Symposium on Wearable Computers.

Nawrat, A., Jędrasiak, K., Daniec, K. & Koteras, R., 2012. *Inertial Navigation Systems and Its Practical Applications.* London: IntechOpen.

Norhafizan Ahmad, R. A. R. G. N. M. K., 2013. Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications. *International Journal of Signal Processing Systems,* 1(2), pp. 256 -262.

Ogee, A., Ellis, M., Scibilia, B. & Pammer, C., 2013. *Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?.* [Online]
Available at: https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit
[Accessed 3 February 2020].

Olah, C., 2015. *Understanding LSTM Networks.* [Online]
Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[Accessed 3 February 2020].

O'Shea, T., Hitefield, S. & Corgan, J., 2016. *End-to-End Radio Traffic Sequence Recognition with Deep Recurrent Neural Networks.* Washington, DC, USA, IEEE.

P., V. & M., O., 1999. Does a navigation algorithm have to use a Kalman filter?. *Canadian Aeronautics and Space Journal,* 45(3), pp. 292-296.

Qi, H. & Moore, J., 2002. Direct Kalman filtering approach for GPS/INS integration. *IEEE Transactions on Aerospace and Electronic Systems ,* 38(2), pp. 687-693.

Reed, R. & Marks, R. J., 2014. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks.* Massachusetts, United States: MIT Press.

Soo, Y., 2003. *Attitude estimation using low cost accelerometer and gyroscope.* Ulsan, South Korea, Institute of Electrical and Electronics Engineers.

Srivastava, N. et al., 2014. Dropout: A Simple Way to Prevent Neural Networks fromOverfitting. *Journal of Machine Learning Research ,* Volume 15, pp. 1929-1958.

STMicroelectronics, 2015. *STM32F103C8.* [Online]
Available at: https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html
[Accessed 1 February 2020].

Swalin, A., 2018. *Choosing the Right Metric for Evaluating Machine Learning Models.* [Online]
Available at: https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4
[Accessed 3 February 2020].

Takeda, M., 2001. *Applications of MEMS to industrial inspection.* Interlaken, Switzerland, IEEE International Conference on Micro Electro Mechanical Systems.

Tao, D. et al., 2006. *Supervised tensor learning.* Houston, TX, USA, Institute of Electrical and Electronics Engineers.

Wang, J., Lou, W., Liu, W. & Liu, P., 2019. *Calibration of MEMS Based Inertial Measurement Unit Using Long Short-Term Memory Network.* Bangkok, Thailand, Thailand, Institute of Electrical and Electronics Engineers.

Winer, K., 2018. *Arduino sketches for MPU9250 9DoF with AHRS sensor fusion.* [Online]
Available at: https://github.com/kriswiner/MPU9250
[Accessed 3 February 2020].

Wrigley, W., 1977. The History of Inertial Navigation. *Journal of Navigation,* 30(1), pp. 61-68.

Zou, A.-M., Hou, Z.-G., Fu, S.-Y. & Tan, M., 2006. Neural Networks for Mobile Robot Navigation. In: W. Jun, et al. eds. *Lecture Notes in Computer Science.* Berlin: Springer, p. 1219.

Appendix A – Ethics Form

Research Ethics Form

Section 1
Your details

a) Researcher

Benjamin Kalmus
(17010358)

c) Title of Proposed Project

Application of recurrent neural networks in inertial measurement units

d) Programme Title and Level of Study

COMPUTER SCIENCE

e) Research Dates

Start Date:

2019-11-19

End Date:

2020-04-30

f) Department

First department

MATHEMATICS, COMPUTER SCIENCE AND ENGINEERING

First Supervisor Name

Dr Neil Buckley

g) Professional Guidelines Referenced

No Guidelines Referenced

Section 2

Who will be taking part in your research?

a) Will other people be taking part in your research?

No human participants

Section 3

This section is for research which does not involve human participants

a) Does the research present a risk to you as the researcher?

The research involves no risk to me as the researcher

b) Summary of Research Project

Please provide a brief but clear 200-word summary of the project

Applying several neural network algorithms to a 9-axis inertial measurement unit (IMU) consisting of a gyroscope, an accelerometer and a magnetometer. Inertial measurement units are used to find orientation, rotation and even compass bearing however the down side is they are prone to noise and vibrations which are unwanted in time-critical applications where orientation of a device is required such as that in a quad-copter drone. This can cause a drone to drift from a particular position or even fail mid-air . The purpose of this project will be to compare neural network solution versus classical algorithms as well as efficiency/execution speed of the code. This neural network could be applied to a variety of devices such as a balancing robot arm, mobile phones, remote-controlled planes/quadcopters and any other implementations of IMU

Section 4

Consent Forms, Research Information Sheet and Additional Documentation

c) Research Information Sheet

Uploaded Files:

No Files Uploaded

d) Additional Documentation

Uploaded Files:

No Files Uploaded

Appendix B – Sensor Fusion Implementation

Mahony and Madgwick algorithm implementations by Kris Winer (Winer, 2018) in C++ for Arduino.

```
void SensorFusion::MadgwickQuaternionUpdate(float ax, float ay, float az, float gx, float gy,
float gz, float mx, float my, float mz, float deltat)
{
    float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local variable for
readability
    float norm;
    float hx, hy, _2bx, _2bz;
    float s1, s2, s3, s4;
    float qDot1, qDot2, qDot3, qDot4;

    // Auxiliary variables to avoid repeated arithmetic
    float _2q1mx;
    float _2q1my;
    float _2q1mz;
    float _2q2mx;
    float _4bx;
    float _4bz;
    float _2q1 = 2.0f * q1;
    float _2q2 = 2.0f * q2;
    float _2q3 = 2.0f * q3;
    float _2q4 = 2.0f * q4;
    float _2q1q3 = 2.0f * q1 * q3;
    float _2q3q4 = 2.0f * q3 * q4;
    float q1q1 = q1 * q1;
    float q1q2 = q1 * q2;
    float q1q3 = q1 * q3;
    float q1q4 = q1 * q4;
    float q2q2 = q2 * q2;
    float q2q3 = q2 * q3;
    float q2q4 = q2 * q4;
    float q3q3 = q3 * q3;
    float q3q4 = q3 * q4;
    float q4q4 = q4 * q4;

    // Normalise accelerometer measurement
    norm = sqrtf(ax * ax + ay * ay + az * az);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f / norm;
    ax *= norm;
    ay *= norm;
    az *= norm;

    // Normalise magnetometer measurement
    norm = sqrtf(mx * mx + my * my + mz * mz);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f / norm;
    mx *= norm;
    my *= norm;
    mz *= norm;

    // Reference direction of Earth's magnetic field
    _2q1mx = 2.0f * q1 * mx;
    _2q1my = 2.0f * q1 * my;
    _2q1mz = 2.0f * q1 * mz;
    _2q2mx = 2.0f * q2 * mx;
    hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2 + _2q2 * my * q3 + _2q2 * mz * q4 -
mx * q3q3 - mx * q4q4;
    hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx * q3 - my * q2q2 + my * q3q3 + _2q3 *
mz * q4 - my * q4q4;
    _2bx = sqrtf(hx * hx + hy * hy);
    _2bz = -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx * q4 - mz * q2q2 + _2q3 * my * q4 -
mz * q3q3 + mz * q4q4;
    _4bx = 2.0f * _2bx;
    _4bz = 2.0f * _2bz;

    // Gradient decent algorithm corrective step
    s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f * q1q2 + _2q3q4 - ay) - _2bz * q3
* (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q4 + _2bz * q2) * (_2bx
* (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 * (_2bx * (q1q3 + q2q4) + _2bz *
(0.5f - q2q2 - q3q3) - mz);
```

```
    s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q2 *
(1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz *
(q2q4 - q1q3) - mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) -
my) + (_2bx * q4 - _4bz * q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
    s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q3
* (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3 - _2bz * q1) * (_2bx * (0.5f - q3q3 -
q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) * (_2bx * (q2q3 - q1q4) + _2bz *
(q1q2 + q3q4) - my) + (_2bx * q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 -
q3q3) - mz);
    s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f * q1q2 + _2q3q4 - ay) + (-_4bx * q4
+ _2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz
* q3) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q2 * (_2bx * (q1q3 + q2q4)
+ _2bz * (0.5f - q2q2 - q3q3) - mz);
    norm = sqrtf(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);    // normalise step magnitude
    norm = 1.0f / norm;
    s1 *= norm;
    s2 *= norm;
    s3 *= norm;
    s4 *= norm;

    // Compute rate of change of quaternion
    qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
    qDot2 = 0.5f * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
    qDot3 = 0.5f * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
    qDot4 = 0.5f * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;

    // Integrate to yield quaternion
    q1 += qDot1 * deltat;
    q2 += qDot2 * deltat;
    q3 += qDot3 * deltat;
    q4 += qDot4 * deltat;
    norm = sqrtf(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise quaternion
    norm = 1.0f / norm;
    q[0] = q1 * norm;
    q[1] = q2 * norm;
    q[2] = q3 * norm;
    q[3] = q4 * norm;

}



void SensorFusion::MahonyQuaternionUpdate(float ax, float ay, float az, float gx, float gy,
float gz, float mx, float my, float mz, float deltat)
{
    float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local variable for
readability
    float norm;
    float hx, hy, bx, bz;
    float vx, vy, vz, wx, wy, wz;
    float ex, ey, ez;
    float pa, pb, pc;

    // Auxiliary variables to avoid repeated arithmetic
    float q1q1 = q1 * q1;
    float q1q2 = q1 * q2;
    float q1q3 = q1 * q3;
    float q1q4 = q1 * q4;
    float q2q2 = q2 * q2;
    float q2q3 = q2 * q3;
    float q2q4 = q2 * q4;
    float q3q3 = q3 * q3;
    float q3q4 = q3 * q4;
    float q4q4 = q4 * q4;

    // Normalise accelerometer measurement
    norm = sqrtf(ax * ax + ay * ay + az * az);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f / norm;        // use reciprocal for division
    ax *= norm;
    ay *= norm;
    az *= norm;

    // Normalise magnetometer measurement
    norm = sqrtf(mx * mx + my * my + mz * mz);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f / norm;        // use reciprocal for division
```

```
    mx *= norm;
    my *= norm;
    mz *= norm;

    // Reference direction of Earth's magnetic field
    hx = 2.0f * mx * (0.5f - q3q3 - q4q4) + 2.0f * my * (q2q3 - q1q4) + 2.0f * mz * (q2q4 +
q1q3);
    hy = 2.0f * mx * (q2q3 + q1q4) + 2.0f * my * (0.5f - q2q2 - q4q4) + 2.0f * mz * (q3q4 -
q1q2);
    bx = sqrtf((hx * hx) + (hy * hy));
    bz = 2.0f * mx * (q2q4 - q1q3) + 2.0f * my * (q3q4 + q1q2) + 2.0f * mz * (0.5f - q2q2 -
q3q3);

    // Estimated direction of gravity and magnetic field
    vx = 2.0f * (q2q4 - q1q3);
    vy = 2.0f * (q1q2 + q3q4);
    vz = q1q1 - q2q2 - q3q3 + q4q4;
    wx = 2.0f * bx * (0.5f - q3q3 - q4q4) + 2.0f * bz * (q2q4 - q1q3);
    wy = 2.0f * bx * (q2q3 - q1q4) + 2.0f * bz * (q1q2 + q3q4);
    wz = 2.0f * bx * (q1q3 + q2q4) + 2.0f * bz * (0.5f - q2q2 - q3q3);

    // Error is cross product between estimated direction and measured direction of gravity
    ex = (ay * vz - az * vy) + (my * wz - mz * wy);
    ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
    ez = (ax * vy - ay * vx) + (mx * wy - my * wx);
    if (Ki > 0.0f)
    {
        eInt[0] += ex;      // accumulate integral error
        eInt[1] += ey;
        eInt[2] += ez;
    }
    else
    {
        eInt[0] = 0.0f;     // prevent integral wind up
        eInt[1] = 0.0f;
        eInt[2] = 0.0f;
    }

    // Apply feedback terms
    gx = gx + Kp * ex + Ki * eInt[0];
    gy = gy + Kp * ey + Ki * eInt[1];
    gz = gz + Kp * ez + Ki * eInt[2];

    // Integrate rate of change of quaternion
    pa = q2;
    pb = q3;
    pc = q4;
    q1 = q1 + (-q2 * gx - q3 * gy - q4 * gz) * (0.5f * deltat);
    q2 = pa + (q1 * gx + pb * gz - pc * gy) * (0.5f * deltat);
    q3 = pb + (q1 * gy - pa * gz + pc * gx) * (0.5f * deltat);
    q4 = pc + (q1 * gz + pa * gy - pb * gx) * (0.5f * deltat);

    // Normalise quaternion
    norm = sqrtf(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);
    norm = 1.0f / norm;
    q[0] = q1 * norm;
    q[1] = q2 * norm;
    q[2] = q3 * norm;
    q[3] = q4 * norm;

}
```