DESIGN AND IMPLEMENTATION OF NOISE FILTERS IN POSITION AND ORIENTATION ESTIMATION OF AUTONOMOUS AGRICULTURAL VEHICLES

AYTUĞ GÜNER

Master's Thesis

Graduate School
Izmir University of Economics
Izmir
2022

DESIGN AND IMPLEMENTATION OF NOISE FILTERS IN POSITION AND ORIENTATION ESTIMATION OF AUTONOMOUS AGRICULTURAL VEHICLES

AYTUĞ GÜNER

A Thesis Submitted to

The Graduate School of Izmir University of Economics

Master of Science Program in Electrical and Electronics Engineering

Izmir

2022

ABSTRACT

DESIGN AND IMPLEMENTATION OF NOISE FILTERS IN POSITION AND ORIENTATION ESTIMATION OF AUTONOMOUS AGRICULTURAL VEHICLES

Güner, Aytuğ

M.Sc. in Electrical and Electronics Engineering

Advisor: Prof. Dr. N. Süha Bayındır

January, 2022

An autonomous agricultural vehicle is designed using cost effective sensors such as an IMU, a wheel odometry, a magnetometer and a laser sensor. The main aim of this project is to move the autonomous agricultural vehicle on a straight path between the trees in an orchard until it detects a sign plate at the end of the lane. Hence we have concentrated on estimating the orientation of the vehicle along the axis of the lane, rather than trying to estimate the exact location of it. In order to get more accurate orientation information, the advantages and disadvantages of the accelerometer, gyroscope, magnetometer, and wheel odometry are combined with filters such as the complementary, Madgwick, Kalman, and the integrated Kalman filter and the accuracies of the filters in different environmental conditions are compared in terms of the mean error. Using the orientation information obtained from the filters, the autonomous agricultural vehicle can move on a straight path in an orchard.

Furthermore, a different type of approach is developed using a laser distance sensor to move the autonomous agricultural vehicle on a straight path between the trees. Using the laser distance sensor, the distance and the angle between the autonomous agricultural vehicle and the trees are determined and taking the trees as reference, it travels on a straight path between the trees. Various filters are developed and implemented using the ROS platform and the python script on the autonomous vehicle and the performance of these filters are examined on MATLAB.

Keywords: autonomous agricultural vehicle, filter, complementary filter, Madgwick filter, Kalman filter, integrated Kalman filter, wheel odometry

ÖZET

OTONOM TARIM ARAÇLARININ KONUM VE YÖN TAHMİNİ İÇİN GÜRÜLTÜ FİLTRELERİ TASARIMI VE UYGULAMASI

Güner, Aytuğ

Elektrik ve Elektronik Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Prof. Dr. N. Süha Bayındır

Ocak, 2022

Otonom bir tarım aracı, IMU, tekerlek kilometre sayacı, manyetometre ve lazer sensörü gibi uygun maliyetli sensörler kullanılarak tasarlanmıştır. Bu projenin temel amacı, otonom tarım aracını, bir meyve bahçesindeki ağaçların arasında, şeridin sonunda bir işaret levhası algılayana kadar düz bir yol üzerinde hareket ettirmektir. Bu nedenle, aracın tam konumunu tahmin etmeye çalışmak yerine, şeridin ekseni boyunca aracın yönünü tahmin etmeye odaklandık. Daha doğru yön bilgisi elde etmek için ivmeölçer, jiroskop, manyetometre ve tekerlek kilometre sayacının avantaj ve dezavantajları tamamlayıcı, Madgwick, Kalman ve entegre kalman filtresi gibi filtreler uygulanarak birleştirilmis¸ ve filtrelerin farklı çevre koşullarındaki doğrulukları, ortalama hata bulunarak karşılaştırılmıştır. Filtrelerden elde edilen yön bilgisini kullanarak, otonom tarım aracı bir meyve bahçesinde düz bir yolda hareket ettirilmiştir. Ayrıca, otonom tarım aracını ağaçlar arasında düz bir yolda hareket ettirmek için bir lazer mesafe sensörü kullanılarak farklı bir yaklaşım türü geliştirildi. Lazer mesafe sensörü kullanılarak otonom tarım aracı ile ağaçlar arasındaki mesafe ve açı belirlendi¸ ve ağaçları referans alarak ağaçların arasında düz bir yol üzerinde aracın hareketi sağlanmıştır. Otonom araç üzerinde ROS platformu ve python betiği kullanılarak çeşitli filtreler geliştirilip uygulanmakta ve bu filtrelerin performansı MATLAB üzerinde incelenmektedir.

Anahtar Kelimeler: otonom tarım aracı, filtre, tamamlayıcı filtre, Madgwick filtre, Kalman filtre, tümleşik Kalman filtre, tekerlek kilometre sayacı

To my nephew …

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Prof. Süha Bayındır for inspiring me to choose the field of agricultural robotics and for giving me the motivation and encouragement along this journey.

Also, I would like to thank my family for their unconditional love, patience, and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Autonomous agricultural vehicles are the alternatives of tractors to accomplish basic cultivation tasks such as ploughing, seeding and harvesting, with self-driving capabilities (Biber et al., 2012). They are used to decrease the labor force and to accomplish agricultural tasks day or night (Sowjanya et al., 2017). The main problem in autonomous agricultural vehicles is to achieve correct orientation and position estimation with affordable sensors because the agricultural environment could consist of different types of challenges, such as rain, snow or magnetic disturbances which could interfere with the sensors' observations (Hague et al., 2000). Furthermore, the wheel slippage may cause errors in the odometry information. The main aim of this project is to move the autonomous agricultural vehicle on a straight path between the trees in an orchard until it detects a sign plate at the end of the lane. Hence, we have concentrated on estimating the orientation of the vehicle along the axis of the lane, rather than trying to estimate the exact location of it.

The advance of technology in micro electromechanical system (MEMS) have enabled the production of small and affordable IMU's and speed sensors whose are improved over the years (Han et al.,2020). In spite of the advance in technology, each sensor has advantages and drawbacks and using only one sensor, accurate calculation of orientation and position autonomous agricultural vehicle is not possible due to the biases, noises and environmental conditions. Sensor fusion algorithms can be used to acquire accurate orientation and position of autonomous agricultural vehicle from affordable sensors by making use of their advantages (Gui, Tang, and Mukhopadhyay, 2015). The well-known sensor fusion algorithms or noise filters are the complementary filter (Treffers and Wietmarschen, 2016), the Kalman filter (Welch, and Bishop, 2001) and the Madgwick filter (Madgwick et al., 2011).

One of the sensor fusion algorithms is the complementary filter (Treffers and Wietmarschen, 2016). The complementary filter applies high-pass filter and low-pass filter to the given data in order to estimate the orientation of the autonomous agricultural vehicle. The complementary filter algorithm is easy to implement, and it has low computational cost (Gui, Tang, and Mukhopadhyay, 2015). Other sensor fusion algorithm is the Kalman filter which is recursive. The Kalman filter estimates the orientation noise information and the covariance matrix of the inputs (Treffers and Wietmarschen, 2016). It is more complex than the complementary filter in terms of computational cost since it involves matrix inversions and covariance matrix calculations (Madgwick et al., 2011). On the other hand, the Madgwick filter has a simpler algorithm than that of the Kalman filter and it uses the quaternion representation of the given data to estimate the orientation (Madgwick et al., 2011). Besides, the Madgwick filter is designed for only the IMU and the Magnetic Angular Rate and Gravity (MARG) estimations (Madgwick et al., 2011).

In this thesis, the navigation problem is examined using different types of sensors such as the speed sensor, inertial measurement unit (IMU), and Laser distance sensor. In addition, an autonomous agricultural vehicle is developed to plow a field in an orchard using cost effective sensors with reasonable drift from the navigation path. The first aim of this project is to move the autonomous agricultural vehicle on a straight line using the orientation information estimated from the IMU, magnetometer and the odometry sensors and to stop the vehicle when it detects a stop sign using a camera. The second aim of this project is to move the autonomous agricultural vehicle on the lanes between the trees in an orchard using only the laser distance sensor data.

The autonomous agricultural vehicle control system consists of a speed sensor, an inertial measurement unit (IMU), a laser distance sensor, a camera, a motor driver, two DC motors an arduino uno and a raspberry pi 3 modules. The arduino uno is used to drive the motors according to the information coming from the raspberry pi using UART communication. All sensors are directly connected to the raspberry pi and the sensor information are obtained using a python script developed on the robotic operating system platform (ROS).

In order to control the autonomous agricultural vehicle, accurate orientation and position information is necessary. To determine the orientation and position information, a speed sensor and an IMU are used on the autonomous agricultural vehicle. The wheel odometry information is obtained using the speed sensor from which the orientation and position values are calculated. As a second source of information, the IMU is also used to obtain the orientation values. The main problem is that the orientation and position estimation from the IMU and wheel odometry can be inaccurate due to the electrical noises and magnetic disturbances around the sensors. In order to eliminate the noise on the orientation and position data, various filters such as the complementary filter (Treffers and Wietmarschen, 2016), Madgwick filter (Madgwick et al., 2011), Kalman filter (Welch, and Bishop, 2001) and integrated Kalman filter (El-Diasty,2014), are designed and implemented on the IMU and the wheel odometry data.

A laser distance sensor is used on the autonomous agricultural vehicle to obtain the orientation of the vehicle by making use of the distances between the vehicle and the trees around the vehicle. Various methods are developed for calculating the orientation from the distance information which is obtained from the laser distance sensor. In the field, there are four trees around the vehicle and the laser sensor detects the distance between the vehicle and the trees. Taking the trees as a reference, the orientation of the autonomous agricultural vehicle is determined. Using only the orientation information calculated from the laser distance sensor, the autonomous agricultural vehicle travels on a straight path between the trees. However, there is one disadvantage of this method where the data obtained from the laser sensor is not reliable under bright sunlight which interferes with the laser signal and causes errors in the orientation data. A better solution could have been obtained by using a radar distance sensor coupled with a laser distance sensor whose outputs could be fused with a Kalman filter to obtain an accurate orientation information even under bright sunlight conditions.

The algorithms are developed, and the implementations are made using a python script developed on the robot operating system (ROS) platform which is an open source platform on which it is very easy to develop robotic applications (Quigley et al.,2009). In this thesis, ROS platform is used to visualize the sensor data, calculate, and monitor the position and orientation data of the autonomous agricultural vehicle remotely.

The thesis consists of 6 chapters. Chapter 2 introduces general description of the autonomous agricultural vehicle, and of the sensors that are used in autonomous agricultural vehicle. Chapter 3 explains the calibration procedures and implementations on the IMU. In order to reduce noise and to obtain accurate orientation, pre-filter method is implemented on the IMU and the implementation of the filter is explained in Chapter 4. In addition, the implementation of different types of filters, namely the complementary, Madgwick, Kalman and integrated Kalman filters, using the IMU and the wheel odometry data and the comparison of the filters are stated in Chapter 4. Chapter 5 explains how to obtain orientation information using the laser distance sensor and navigate the vehicle on a straight path between the trees with minimum drift from the central axis. Comparison of the performances of the methods developed and the results obtained are included in the last chapter.

CHAPTER 2: SYSTEM DESCRIPTION

The autonomous agricultural vehicle is designed using a speed sensor, an inertial measurement unit (IMU), a laser distance sensor, a camera, a motor driver, two DC motors an arduino uno and a raspberry pi 3 module as shown in Figure 1. Moreover, the designed autonomous agricultural vehicle is shown in Figure 2.

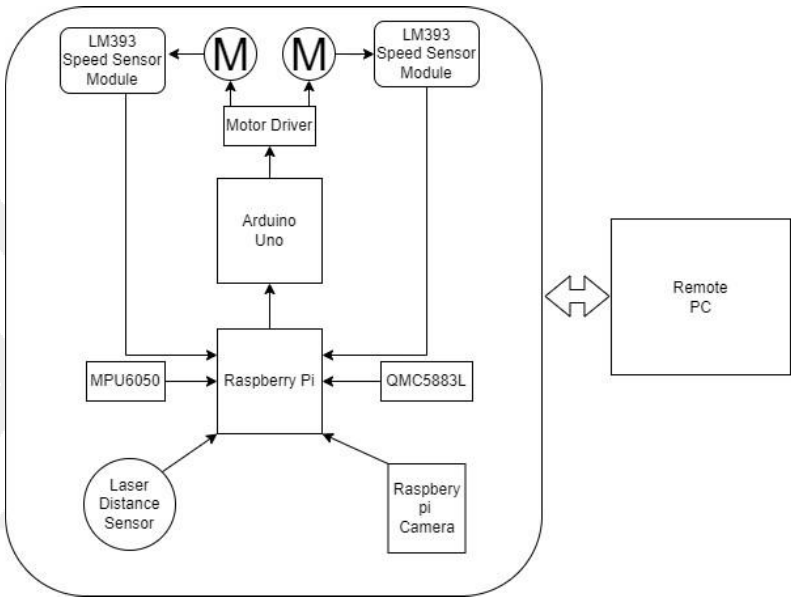


Figure 1. Overall system design

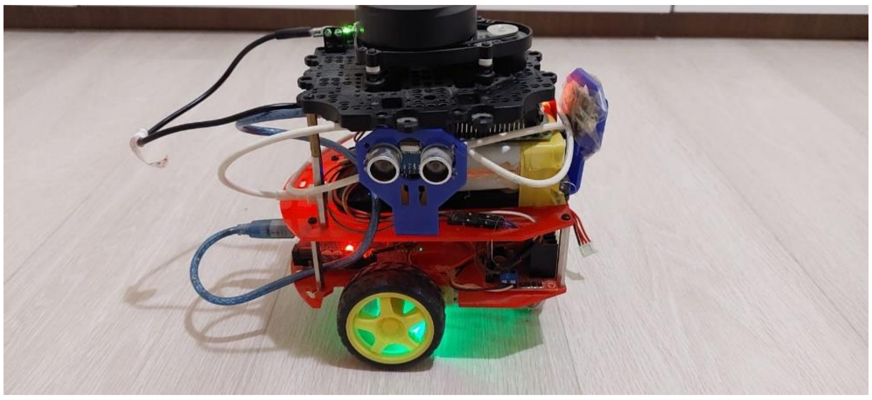


Figure 2. Autonomous agricultural vehicle

A UART communication is established between the arduino and the raspberry pi modules in order to adjust the speed of the motors. The MPU6050, QMC5883L, laser distance sensor, raspberry pi camera, and the speed sensors are connected to the raspberry pi module. The sensor information are obtained using the python script developed in this project, on the robot operating system (ROS). Furthermore, in order to visualize the sensor data and monitor the orientation and position of the autonomous agricultural vehicle, a TCP/IP communication is established between the raspberry pi and the remote pc using the ROS platform.

The autonomous agricultural vehicle system consists of two parts. The first part is the determination of the orientation using the IMU the wheel odometry data in order to move the autonomous agricultural vehicle on a straight line and to stop the vehicle when it detects a stop sign using a camera. The second part is to move the autonomous agricultural vehicle on the lanes between the trees in an orchard using only a laser distance sensor data.

The main aim of this project is to move the autonomous agricultural vehicle on a straight path between the trees in an orchard until it detects a sign plate at the end of the lane. Hence, we have concentrated on estimating the orientation of the vehicle along the axis of the lane, rather than trying to estimate the exact location of it. In order to calculate accurate orientation of the autonomous agricultural vehicle, different types of filters are implemented on the IMU and the wheel odometry data and the filter results are tested to move the autonomous agricultural vehicle on a straight line in an orchard as shown in Figure 3. The test path is 2 meters long and at the end of the field, there is a stop sign. The test field and real-time monitoring of the orientation of the autonomous agricultural vehicle are shown in Figure 4 and Figure 5 respectively.

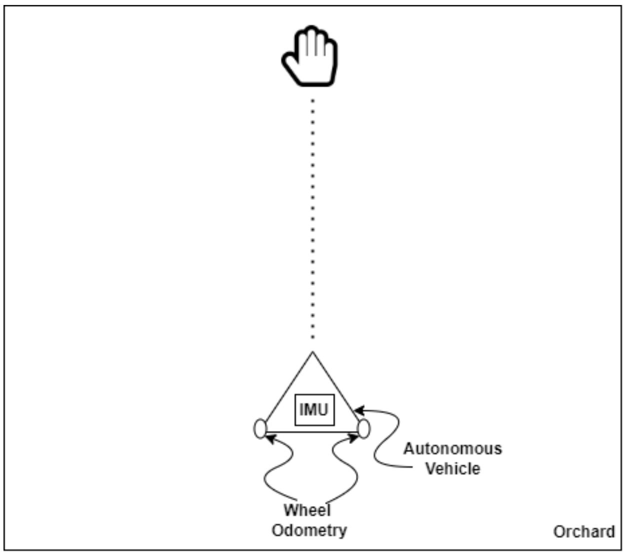


Figure 3. Orchard field

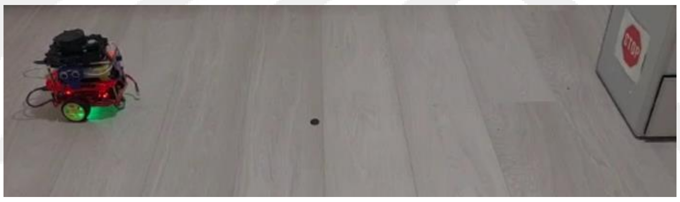


Figure 4. Autonomous agricultural vehicle test field

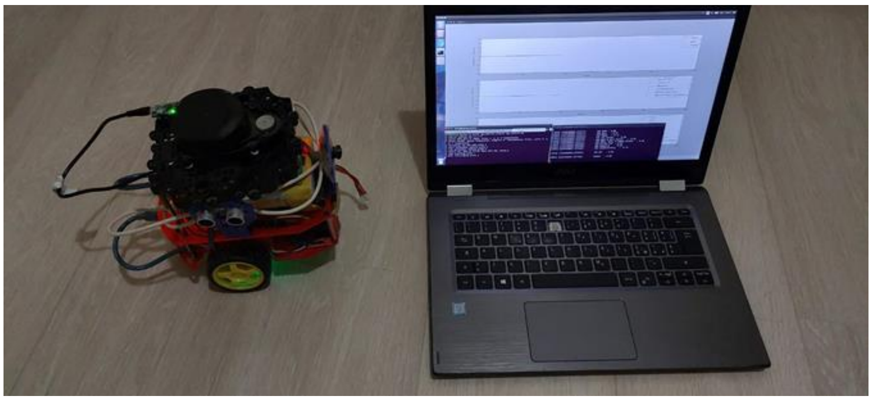


Figure 5. Autonomous agricultural vehicle real-time monitoring

In the first part of this thesis, an algorithm is developed and tested on the autonomous agricultural vehicle using the python script on the ROS platform. The algorithm consists of multiple ROS packages and multiple nodes as shown in Figure 6. The nodes called Imu, filter, agv_filter, motor and odom are defined on the autonomous agricultural vehicle program to collect the sensor data and navigate the vehicle and the nodes called AGV_remote_control, Filter_yaw_plot, and Filter_monitor_and_record, are defined on the remote PC program in order to visualize and monitor the orientation of the autonomous agricultural vehicle. On the autonomous agricultural vehicle side, the Imu node publishes pre-filtered accelerometer, gyroscope, and magnetometer data and the odom node publishes odometry information which is calculated from the speed sensor. The filter node subscribes both to the IMU node and the odom node and it estimates the orientation of the autonomous agricultural vehicle by using different types of filters. The estimated orientation data from the filters are published by the agv_filter node. The agv_filter node determines the state of autonomous agricultural vehicle to keep the autonomous agricultural vehicle on a straight path and it publishes the velocity information to the motor node. The motor node collects the velocity information and using the velocity information, the Arduino Uno adjusts the speed of the motors via the UART communication established between the arduino uno and the raspberry pi modules. On the remote PC side, the orientation information calculated by the filters and sensors are collected with the filter_monitor_and_record node and filter_yaw_plot nodes in order to monitor, record and visualize the orientation data. The AGV_remote_control node is used to initialize the Imu and the Odom nodes, and to start and control the autonomous agricultural vehicle.

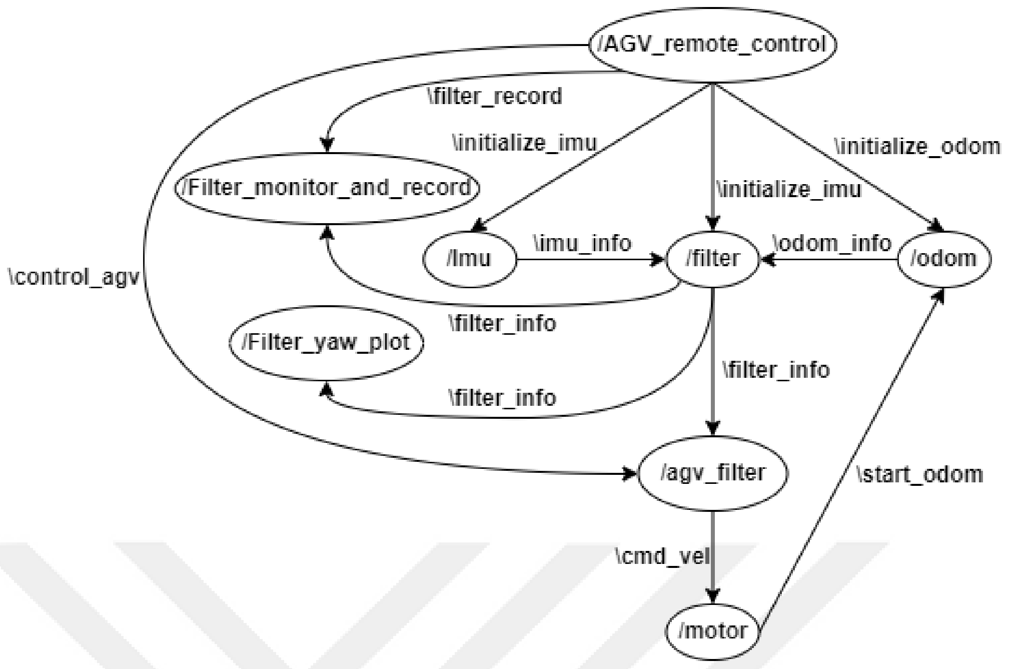


Figure 6. ROS nodes and topics diagram

The agv_filter node consists of 3 states. Initially, the vehicle orientation angle is zero and the first state is to check whether the yaw angle is within the specified limits. The limits for the yaw angle are ±5 degrees. If the vehicle is not in this range, then the autonomous vehicle rotates itself to make the yaw angle zero. The second state is to move the vehicle on a straight path. If the yaw angle is in the range, then the autonomous vehicle’s angular speed is zero and linear speed is 0.5 m/s. The last state is to stop the vehicle. If the stop sign is detected by camera, then the autonomous vehicle stops.

The algorithm, that is implemented on the agv_filter node, is tested with different calculated yaw angles using the odometry, gyroscope and the magnetometer data. The complementary filter, Madgwick filter, Kalman filter and the integrated Kalman filter are implemented on this node. The autonomous vehicle completes its movement successfully with the yaw angles calculated by all the filters, when there is no magnetic disturbance around the vehicle. When there is a magnetic disturbance on the path of the autonomous vehicle, the movement of the autonomous vehicle was not completed correctly using the calculated yaw angles from magnetometer, complementary filter, Madgwick filter, and Kalman filter. On the other hand, when the yaw angles are obtained using the odometry and the gyroscope data fused by the integrated Kalman filter, the autonomous vehicle completes its movement successfully, even when there is a magnetic disturbance on the path of the autonomous vehicle.

In the second part of this thesis, a laser distance sensor is used to move the autonomous agricultural vehicle on the lanes between the trees in an orchard as shown in Figure 7. The field is also shown in Figure 8. This is the alternative way to move autonomous agricultural vehicle on a straight path. In this method, the paths in the orchard and the initial position of autonomous agricultural vehicle are known. Instead of calculating the orientation, the laser distance sensor measures the distance and angle between the autonomous agricultural vehicle and the trees to move the autonomous agricultural vehicle on a straight path.

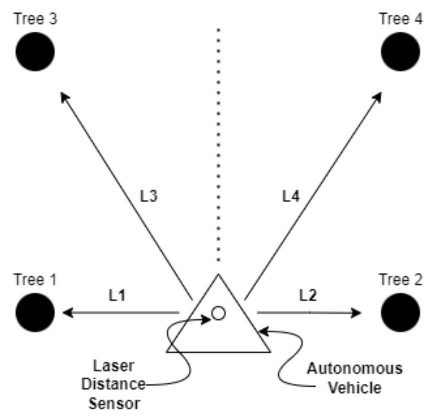


Figure 7. Laser distance sensor tested field

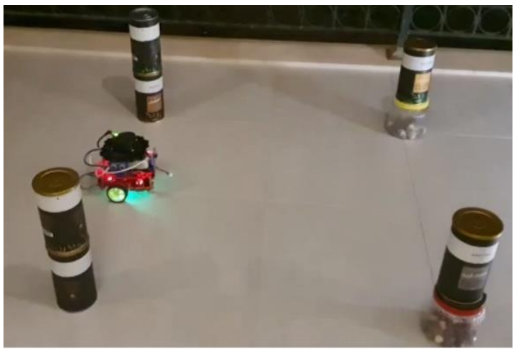


Figure 8. Autonomous agricultural vehicle test field for laser distance sensor

An algorithm is developed and implemented on the autonomous agricultural vehicle using a python script developed on the ROS platform in the second part of this thesis. The developed algorithm consists of multiple ROS packages, and nodes. These nodes lidar, lidar_scan_data, agv_lidar, and the motor, are defined in the autonomous agricultural vehicle program and the nodes agv_lidar_control, and lidar_monitor are defined on remote PC program in order to monitor laser distance sensor data, and control the autonomous agricultural vehicle. The communication between each nodes are shown in Figure9. In the autonomous agricultural vehicle side, the laser distance sensor data is obtained using the lidar node and the laser distance sensor data is configured according to the sensor position on the autonomous agricultural vehicle using the lidar_scan_data node. The lidar_scan_data node publishes the distance information that is acquired from the angles between 0 and 360 degrees. The agv_lidar node collects distance information from all the angles to determine location of the trees. According to the distance and the corresponding angles of trees, it publishes the velocity information to the motor node.

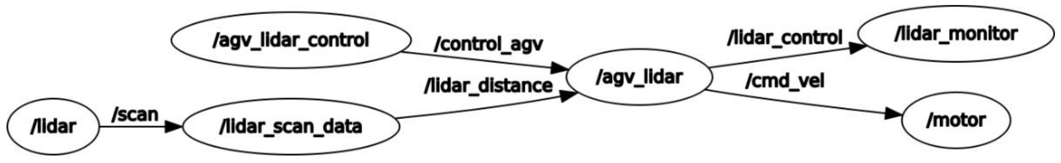


Figure 9. ROS nodes and topics diagram for laser distance sensor

2.1. Wheel Odometry

In order to determine the orientation and position of the autonomous vehicle, wheel odometry information is essential. In this thesis, the wheel odometry is calculated using the motion sensors and the wheel speed encoder that are connected to the two rear wheels of the autonomous vehicle.

The LM393 speed sensor module shown in Figure 10, contains a comparator and an infrared light sensor. The infrared light sensor contains an IR LED and a photo-transistor which are separated with a gap (Aswinth, 2019). On the other hand, the wheel speed encoder, which is shown in Figure 11, contains slots and it is mounted on the motor. The infrared light sensor can detect the slots on the wheel speed encoder and the output of the motion sensor becomes 1 when the slot is detected. Otherwise, the output of the sensor is zero. By counting the number of slots, the speed of the motor can be calculated.

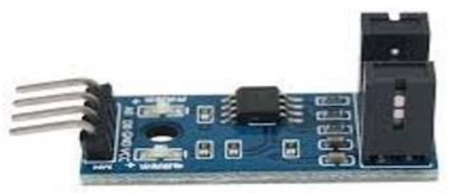


Figure 10. LM393 speed sensor module

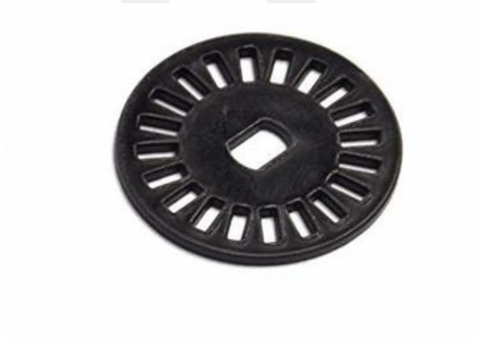


Figure 11. Wheel speed encoder

The motion sensor and raspberry pi connections are made as shown in Appendix A. In order to calculate speed of the motor, the slots that are detected by motion sensor is counted from two motors using the GPIO interrupt of Raspberry pi.

The wheel speed encoder contains 20 slots. By counting number of slots with speed sensor module in short period of time and knowing the radius of the wheel, the speed of motor can be determined. The speed of the one motor can be calculated using formula shown in equation 1. The radius of wheel of autonomous vehicle, denoted as r, is 3.30cm and the calculated speed for the left motor is denoted as V_l In the equation, the $slot_{t1}$ and $slot_{t0}$ are the total number of slots counted by the speed sensor module at time t_1 and t_0 respectively. The slot is the total number of slots that speed encoder has. Similarly, the speed of the right motor, V_r, can be calculated using the formula shown in equation 1.

$$V_l = \frac{2*\pi*r*\frac{slot_{t1}-slot_{t0}}{slot}}{t_1-t_0} \tag{1}$$

Using the calculated speed for both motors, the orientation and position of the autonomous vehicle can be determined. The yaw angle at time t can be calculated as the summation of previous yaw angle $\psi_{odom,t-1}$, and of difference of left and right motor speed as shown in equation 2. The distance between wheels, denoted as L, is 10.5 cm.

$$\psi_{odom,t} = \psi_{odom,t-1} + \frac{V_r - V_l}{L} * \Delta T = \psi_{odom,t-1} + \frac{V_{th}}{L} * \Delta T \tag{2}$$

The rate of change of position in x-axis and y-axis can be calculated as shown in equation 4 and 5. The V_x is the average of the speed of left, and of right motor as shown in equation 3.

$$V_x = \frac{V_l + V_r}{2} \tag{3}$$

$$\Delta x = \cos(V_{th} * \Delta T) * V_x * \Delta T \tag{4}$$

$$\Delta y = -\cos(V_{th} * \Delta T) * V_x * \Delta T \tag{5}$$

The final position in x-axis, $x_{odom,t}$, and y-axis, $y_{odom,t}$, can be obtained using Δx and Δy as shown in equation 6 and 7. In the equation, the $x_{odom,t-1}$ and $y_{odom,t-1}$ are the previous calculated orientations in x-axis and y-axis.

$$x_{odom,t} = x_{odom,t-1} + \sin(\psi_{odom,t-1}) * \Delta x - \sin(\psi_{odom,t-1}) * \Delta y \qquad (6)$$

$$y_{odom,t} = y_{odom,t-1} + \sin(\psi_{odom,t-1}) * \Delta x + \cos(\psi_{odom,t-1}) * \Delta y \qquad (7)$$

2.2. Inertial Measurement Unit (IMU)

The inertial measurement unit, which contains the accelerometer, gyroscope, and magnetometer data, is used in order the determine the orientation of the vehicle. For the accelerometer and gyroscope data, the MPU6050 is used and for the magnetometer data, the QMC5883L is used as shown in Figure 12 and Figure 13.

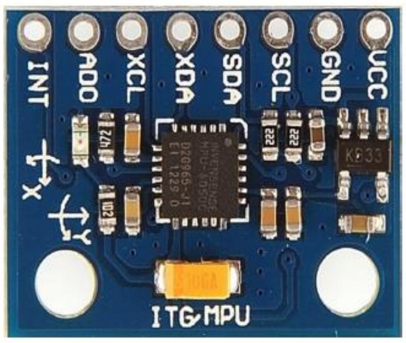


Figure 12. MPU6050

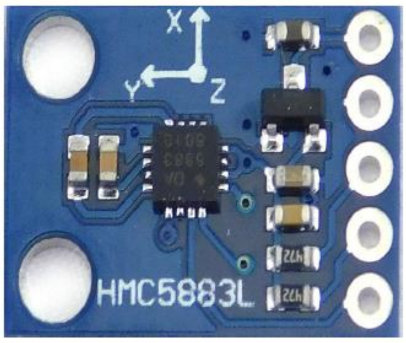


Figure 13. QMC5883L

The MPU6050 has a 16-bit analog to digital converter (ADC) to digitize the accelerometer and gyroscope data and it uses I^2C communication to receive these data (Fedorov et al., 2015). In order to read data from MPU6050, the connection between raspberry pi and MPU6050 are made as shown in Appendix B. Moreover, a package in python is created to read data from the relative register of the MPU6050 and the conversion of the digital data to the relative units are also made in the same package.

For the accelerometer, the full-scale ranges are ±2g, ±4g, ±8g, and ±16g. Since the MPU6050 has a 16-bit ADC, the sensitivity of the sensor can be calculated as shown in the equations 8, 9, 10, and 11 for each full-scale range in terms of $\frac{mg}{LSB}$.

$$S_{2g} = \frac{4}{2^{16}} = 0.061\frac{mg}{LSB} \tag{8}$$

$$S_{4g} = \frac{8}{2^{16}} = 0.122\frac{mg}{LSB} \tag{9}$$

$$S_{8g} = \frac{16}{2^{16}} = 0.244\frac{mg}{LSB} \tag{10}$$

$$S_{16g} = \frac{32}{2^{16}} = 0.488\frac{mg}{LSB} \tag{11}$$

Using these formulas, the sensitivity of the accelerometer is obtained as 0.00060 $\frac{m/_{s^2}}{LSB}$, 0.0012 $\frac{m/_{s^2}}{LSB}$, 0.0024 $\frac{m/_{s^2}}{LSB}$, and 0.0048 $\frac{m/_{s^2}}{LSB}$ for ±2g, ±4g, ±8g, and ±16g respectively.

The full-scale ranges are ±250 $°/_S$, ±500 $°/_S$, ±1000 $°/_S$, and ±2000 $°/_S$ for the gyroscope. The sensitivity of the sensor can be calculated as shown in equations 12, 13, 14, and 15 for each full-scale ranges in terms of $\frac{°/_s}{LSB}$.

$$S_{\pm 250} = \frac{500}{2^{16}} = 0.0076\frac{°/s}{LSB} \tag{12}$$

$$S_{\pm 500} = \frac{1000}{2^{16}} = 0.0153\,\frac{{}^{\circ}/s}{LSB} \tag{13}$$

$$S_{\pm 1000} = \frac{2000}{2^{16}} = 0.0305\,\frac{{}^{\circ}/s}{LSB} \tag{14}$$

$$S_{\pm 2000} = \frac{4000}{2^{16}} = 0.0610\,\frac{{}^{\circ}/s}{LSB} \tag{15}$$

Similarly, the QMC5883L has a 16-bit ADC and it uses I^{2}C communication. The connection between the sensor and raspberry pi are made as shown in Appendix C and a package is developed to read sensor data in python. Moreover, the conversion of the raw digital data to relative units are made in the same package.

The QMC5883L offers two full-scale ranges which are ±2Gauss(G), and ±8G for the magnetometer. The sensitivity of the sensor are $\frac{1}{1200}$, and $\frac{1}{3000}$ for ±2G, and ±8G respectively in terms of $\frac{G}{LSB}$. The sensitivity for ±2G and ±8G can be obtained as 0.0083 and 0.033 respectively in terms of $\frac{\mu T}{LSB}$.

In this thesis, for the accelerometer, the ±2g full-scale range is selected, and according to that the sensitivity of the measurement is 0.00060 $\frac{m/s^2}{LSB}$. For the gyroscope, the full-scale range is taken as ±250 $^{\circ}/_{S}$ and the sensitivity is obtained as 0.0076 $\frac{{}^{\circ}/s}{LSB}$. Lastly, the magnetometer full-scale range is chosen as ±2G with a sensitivity of 0.0083 $\frac{\mu T}{LSB}$.

2.3. Laser Distance Sensor

The laser distance sensor, is shown in Figure 14, is used to collect distance information around the vehicle and to calculate orientation of the vehicle.

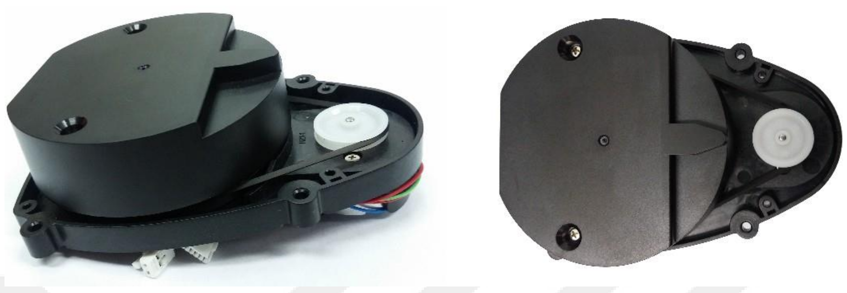


Figure 14. 360-degree laser distance sensor (LDS-1)

The laser distance sensor has full scan rate of 360 degrees with 1 degree resolution and the laser distance sensor supports UART communication (David, 2020). The distance accuracy of the sensor is ±15mm in between 120mm and 499mm and ±5.0% in between 500mm and 3500mm (David, 2020).

The communication between the laser distance sensor and raspberry pi is established with UART communication. The sensor data is obtained using ROS package that is called HLS-LFCD2 (David, 2020). The package gives distance information that are taken from in all degrees as an array with publishing the data under the scan topic. In this thesis, the distance information, which is in terms of meter, is taken using scan topic and it is used on the autonomous vehicle implementation to find orientation of the vehicle.

2.4. Camera

Raspberry pi Camera module, is shown in Figure 15, is used to detect a traffic sign and the autonomous vehicle rotates in one direction when the traffic sign is detected.

Figure 15. Raspberry Pi camera module v2

The raspberry pi camera module has 8 Megapixels camera and it has video resolution of 480p, 720p, and 1080p with frame rate of 90, 60, and 30 respectively (Pagnutti et al., 2017). In this thesis, the raspberry pi camera module is used with video resolution of 480p and the frame rate of 30. The configuration of the camera module is done by using picamera library and the video capturing property is done by using python script with picamera library on ROS platform.

In order to detect a traffic sign, which is stop sign, haar cascade classifier method (Khan,2019) is used. The xml file is created that contains information to detect stop sign. Using opencv (Khan,2019) library on python script and using xml file, the stop sign is detected. In autonomous agricultural vehicle implementation, the information is taken with subscribing cam topic when the stop sign is detected.

2.5. Robotic Operating System (ROS)

Robotic operating system (ROS) is an open-source platform which provides hardware abstraction, messaging between devices and package management to develop and improve function definitions (Yoonseok, 2017). ROS is actually a meta-operating system unlike the conventional operating systems such as Windows, Linux, and Android. A meta-operating system can operate processes like scheduling, monitoring, and visualization of data including error inspection (Yoonseok, 2017).

The main concepts used while developing ROS programs, are master, node, package, message, topic, publisher, and subscriber. Master is the main node that provides connection and message communication between the nodes. In order to run the master, roscore command is used and when it is executed, the connection between all nodes could be established and the data could be transferred between the nodes (Yoonseok, 2017). Node is the smallest unit on ROS ecosystem (Yoonseok,2017) which is an executable program that can be written in python or C++. Furthermore, a ROS ecosystem is developed with packages where each package could consist of a launch file, a message file, and multiple nodes. A ROS package also includes ROS dependencies, libraries, and configuration files in order to run the nodes properly. In order to transfer data between nodes, messages are used which consist of variables of type integer, floating, array, and Boolean (Yoonseok, 2017). In order to get information or publish information from one node to another, topics are used. In ROS, publishers are used to transfer a message with the corresponding topic to the nodes which subscribe to the related topic. On the other hand, a subscriber subscribes a topic which is published by a publisher to receive a message. In the ROS ecosystem, there could be multiple publishers and subscribers to transmit and receive messages under various topics.

In ROS, there are some commands, such as roscore, rosrun, and roslaunch to execute an application. Initially, the roscore command is used to run the master node which communicates with the other nodes using the XML-Remote Procedure Call (XML- RPC) which is an HTPP-based protocol (Yoonseok, 2017) and other nodes can register to the master node using the XMLRPC. In this thesis, the roscore command is executed on the remote PC and the autonomous agricultural vehicle is on the same network to register the nodes to the master. When the registration of nodes is completed with the master, the nodes can communicate between each other using the TCPROS of the TCP/IP protocols (Yoonseok, 2017). Furthermore, the rosrun command is used to run a node in any ROS package and the roslaunch command is used to run multiple nodes using a single launch file. In order to run the roslaunch command, the launch file should be included in the ROS package.

The message communication system between nodes is shown in Figure 16. This type of communication is called Topic communication in ROS. This communication is asynchronous, and it is unidirectional. In this type of communication, the publisher and subscriber nodes can communicate between each other using the same type of message and topic. In this example, the publisher node name is wheel odometry which publishes the wheel odometry information under the odom topic with the messages of x, y, ϕ. The subscriber receives this information from the publisher by registering to the odom topic. As you can see in Figure 16, there could be multiple subscribers to receive this information from the publisher node.

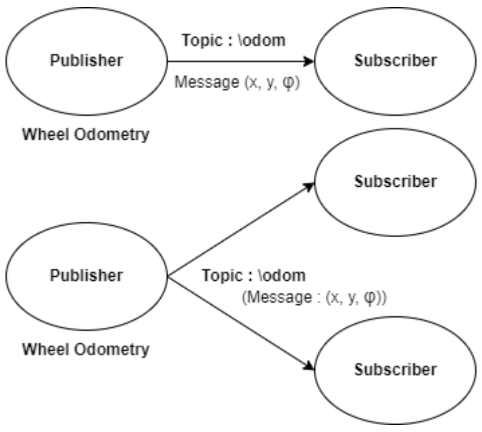


Figure 16. ROS message communication

The message communication system is started by running the master node with the roscore command. In the ROS ecosystem, the name of the nodes, topics and the message types are registered to the master node using the XMLRPC as shown in Figure 17. In this case a subscriber node sends its name, topic name, and message type to the master node using the XMLRPC.

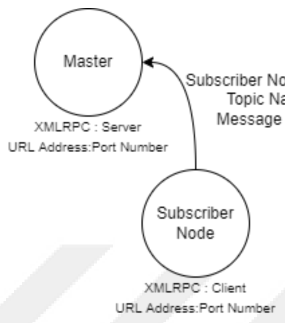


Figure 17. Master node – subscriber node

Similarly, the publisher node sends its node name, topic name, and message type to the master node using the XMLRPC as shown in Figure 18, when the node is executed by the rosrun command. If the topic name and the message type of the subscriber, and the publisher are matched, then the master node sends the publisher’s information to subscriber.

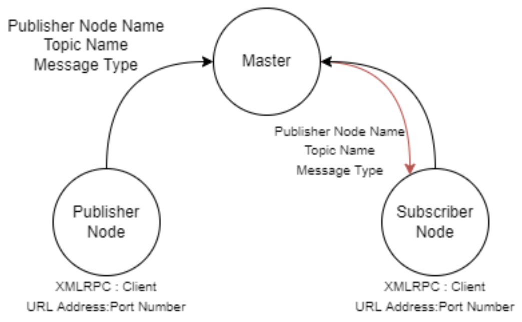


Figure 18. Master node, subscriber node, and publisher node

Furthermore, the subscriber node sends a request to the publisher in order to establish direct communication with the publisher using the XMLRPC and the publisher node sends a response that contains its TCP server information. After that, a client server is created on the subscriber node in order to communicate with the publisher node using the TCP/IP protocol, namely the TCPROS as shown in Figure 19.

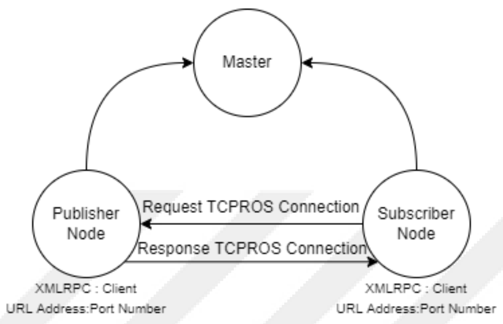


Figure 19. TCPROS request and response

When the TCPROS communication is established between the publisher node and the subscriber node, the messages sent by the publisher node, are transferred to the subscriber node as shown in Figure 20.

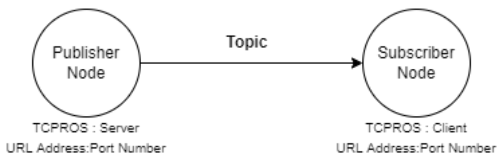


Figure 20. TCP/IP message transmission

In this thesis, all algorithms are developed using python script on the ROS platform. The first application developed on the ROS platform moves the autonomous agricultural vehicle on a straight path using the IMU and the wheel odometry data. The general layout of the communication between the remote PC and the autonomous agricultural vehicle, the nodes and the topics, that are defined on both sides, are shown in Figure 21. The master node is run on the remote PC side and the nodes AGV_remote_control, Filter_monitor_and_record, and Filter_yaw_plot are defined on the remote PC side in order to monitor, visualize and control the autonomous agricultural vehicle. On the other hand, the nodes, that are implemented on the autonomous agricultural vehicle side, are Imu, filter, odom, motor, and agv_filter. The communication between nodes are established using the topic based communication and different type of messages are defined to transfer data between nodes. The messages that are defined under the topics of imu_info, odom_info and filter_info are shown in Appendices D, E, and F respectively.

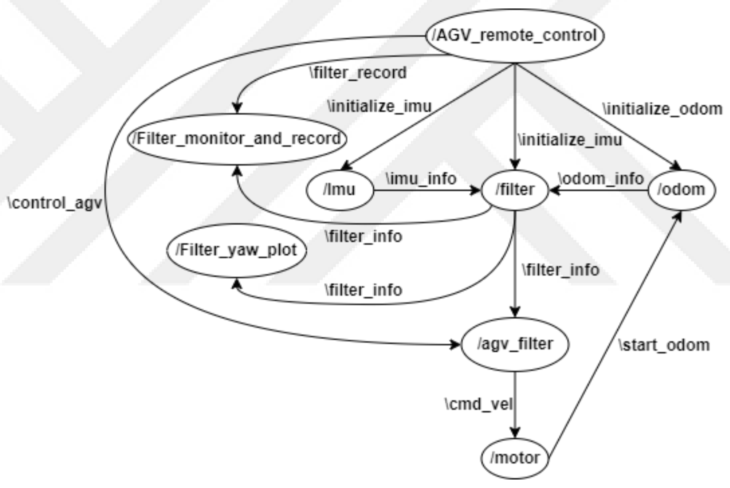


Figure 21. ROS nodes and topics for IMU and wheel odometry

In the second application, an algorithm is developed, using a laser distance sensor on the ROS platform, to move the autonomous agricultural vehicle on the lanes between the trees in an orchard. The nodes and topics, that are used on this application, are shown in Figure 22. Similarly, the master node is run on the remote PC side and the nodes agv_lidar_control and lidar_monitor, are defined on the remote PC side. In order to move the autonomous agricultural vehicle, lidar, lidar_scan_data, agv_lidar and motor nodes are defined on the autonomous agricultural vehicle side.

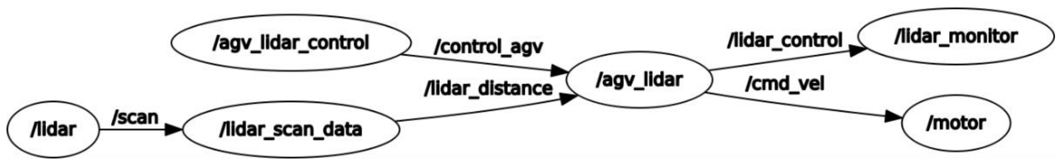


Figure 22. ROS nodes and topics for laser distance sensor

Furthermore, each of the filters are implemented separately on the ROS platform in order to evaluate the CPU and the memory usages. Some of the filters, mentioned in Chapter 4, use only the IMU information, and others use both the wheel odometry, and the IMU information in order to estimate the orientation of the autonomous agricultural vehicle. For that reason, two different cases are created and implemented on the ROS platform as shown in Figure 23, and Figure 24.

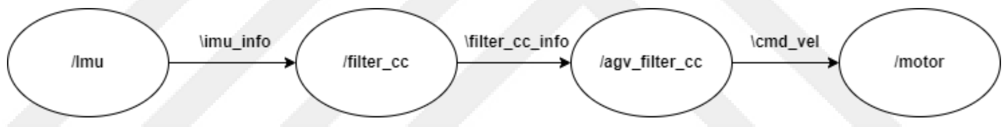


Figure 23. Case 1: ROS nodes and topics to evaluate computational costs

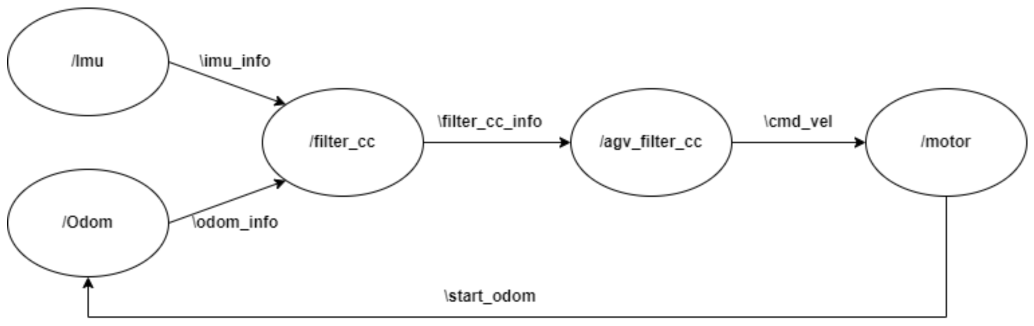


Figure 24. Case 2: ROS nodes and topics to evaluate computational costs

In the first case, the filter_cc node estimates the orientation of the autonomous vehicle using only one of the developed sensor fusion algorithms. The node is executed separately for each of the filters that use only the IMU information and it publishes the filter result to the agv_filter_cc node. The agv_filter_cc node decides the state of the autonomous agricultural vehicle and publishes the velocity information to the motor node in order to move the vehicle on a straight line.

In the second case, the filter_cc node, which estimates the orientation of the autonomous agricultural vehicle, is developed to calculate the computational cost of the filters that use both the wheel odometry, and the IMU information. Similarly, the node is executed separately for each of the filters which publishes the filter result to the agv_filter_cc node. The agv_filter_cc node decides the state of the autonomous agricultural vehicle and publishes the velocity information to the motor node in order to move the vehicle on a straight line.

CHAPTER 3: IMU CALIBRATION

In this part, the accelerometer, gyroscope, and magnetometer offsets are explained. The accelerometer and gyroscope values are acquired from the MPU6050, and the magnetometer values are obtained from the QMC5883L.

The signals obtained from the sensors have a DC offset in the measurement. The mean values of these signals give information about the DC offset. If the mean value of a signal is zero, then the DC offset is zero. When we measure the outputs of the sensors and calculate the mean value of each signal, we can easily observe that the mean values are not zero. Due to the mechanical and electrical factors, each of these sensors have offset which should be eliminated in order to get more reliable results in order to determine the orientation of the vehicle accurately. In order to eliminate the DC offset, each sensors' output should be monitored, and the correction method should be determined for each sensor separately.

1. Accelerometer DC offset : The DC offset the accelerometer can be observed by collecting raw data from the MPU6050 sensor. While collecting the raw data, the sensor must be locked in a fixed place in order to reduce error due to the effect of movement of the sensor. After locking the sensor in a fixed place, the mean value of the signal can be calculated from each axis to determine the DC offset. Then, the calculated offset can be subtracted from each axis separately. Moreover, the gravitational force or gravitational field of the earth should be considered in the z-axis to use the accelerometer data in the calculation of orientation (Treffers and Wietmarschen, 2016). When the sensor is placed upwards facing the z-axis, the raw value collected from the sensor on the z-axis should be observed as 1g (9.8 $m/_{s^2}$). The DC offset calculated on the z-axis can be subtracted from the raw value, but the gravity component should be left as 1g.

2. Gyroscope DC offset : The gyroscope offset calculation is similar to that of the accelerometer offset. In order to collect gyroscope data correctly, the sensor must be locked in a fixed place. When the sensor is locked in a fixed place, the angular velocity should be observed as zero. If it is not zero, then this is because of the DC offset. In order to eliminate the offset, the mean value of the signal which is collected from each axis should be subtracted separately from the raw data (Treffers and Wietmarschen, 2016).

3. Magnetometer DC offset : The magnetometer sensor QMC5883L senses the earth’s magnetic field (Treffers and Wietmarschen,2016) and the offset can cause error in the calculation of orientation relative to the earth’s magnetic field. The electronic devices near the magnetometer sensor can contain components that disturb the measurement of the earth’s magnetic field. This causes the offset in the magnetometer data. The errors or offset on the magnetometer can be categorized as hard iron and soft iron errors (Renaudin, Afzal, and Lachapelle, 2010). The hard iron errors occur when the magnetic material is permanently magnetized which can add constant values in all axes which are the offsets on the magnetometer data (Kok and Schön, 2016). On the other hand, soft iron errors occur when the material is magnetized by an external magnetic field (Kok and Schön, 2016). The hard iron errors can be eliminated easily by collecting raw magnetometer data where the sensor is locked in a fixed place and rotated 180 degrees. When the sensor is locked in a fixed place, the magnetometer measurements contain positive or negative earth magnetic field and the hard iron offset. When the sensor is rotated 180 degrees in the same place, the magnetometer measurements consist of both the hard iron offset and the opposite sign of the earth’s magnetic field which is measured before being rotated 180 degrees. The summation of both measurements give us twice the hard iron offset (Treffers and Wietmarschen, 2016). Dividing the obtained value by two and subtracting the value from the raw magnetometer data, we can

remove the hard iron offset. The process can be summarized mathematically as follows:

$$X = earthmagneticfield + offset \tag{16}$$

$$Y = -earthmagneticfield + offset \tag{17}$$

$$HardIronOffset = (\mathrm{X} + \mathrm{Y})/2 \tag{18}$$

Another way of calculating the hard iron offset is to measure the raw magnetometer data while rotating the sensor 360 degrees on all three axes. For each axes, summation of the maximum and minimum values of the magnetometer data give us twice the hard iron offset. Similarly, half of the calculated value can be subtracted from the raw magnetometer data in order to get rid of the hard iron offset. In this thesis, this process is applied using the python script on the Robot Operating System (ROS) platform. Furthermore, the soft iron errors can be obtained by rotating the sensor in all directions while collecting the raw magnetometer data in all axes. The soft iron errors can be observed by plotting the magnetometer data in any axes with respect to any other axes of the magnetometer. If the obtained graph is a perfect sphere, then there is no soft iron offset. If the graph is an ellipsoid, then there is a soft iron offset; in other words, there is a scaling error in data. In order to eliminate soft iron errors, the obtained ellipsoid must be compared with the perfect sphere and the results gives us the scaling vector which should be applied to the raw magnetometer data (Treffers and Wietmarschen, 2016). In this thesis, the process of soft iron offset detection is implemented, but the scaling vector is not obtained and is not applied to the raw magnetometer data since the acquired signals from the magnetometer sensor do not contain substantial soft iron errors.

3.1. Implementation

The calibration procedure is applied using a Raspberry Pi module and the ROS platform. The accelerometer, gyroscope, and magnetometer data are collected from the MPU6050 and the QMC5883L sensors. Then the mean value of each signal is calculated and saved in an excel file. Moreover, the collected data from the sensors are sent to a remote PC using the ROS. On the remote PC side, the collected data is saved in a text file in order to visualize the signal in MATLAB. While collecting data, the data are also visualized on the remote PC thanks to the ROS platform.

1. Gyroscope DC offset: The raw gyroscope data is obtained from the MPU6050, while it is locked in a fixed place. On all three axes, 1000 samples are acquired, and the mean value of each signal is calculated. The calculated mean values of each axis are stored in an excel file. In the normal operation of the sensor, the calculated mean values are read from the excel file and subtracted from raw gyroscope data in order to eliminate the DC offset.

2. Accelerometer DC offset: The raw accelerometer data is acquired from the MPU6050, while it is locked in a fixed place where the z-axis points upwards. Similar to the gyroscope offset procedure, 1000 samples are collected. The mean value of each signal is calculated and the mean values are saved into the excel file for further usages. The calculated mean values are subtracted from the raw accelerometer data to get rid of the DC offset in the normal operation of the sensor. Since the sensor's z-axis points upward, the calculated mean value for the z-axis should be around 1g. In order to use the accelerometer data in orientation calculation, the calculated offset is subtracted from the z-axis and 1g is added to the corrected the value.

3. Magnetometer DC offset: The raw magnetometer data is obtained from the QMC5883L, while the sensor is on the autonomous vehicle. In

order to obtain the hard iron offset, firstly, the vehicle is rotated around each axis and for each rotation 1000 samples are taken separately with a 50 Hz sampling rate. The collected data is saved in a NumPy array, and the maximum and minimum values of each sample are determined. Then, the summation of the maximum and minimum values are obtained to get twice the offset. Lastly, the obtained value is divided by two to get hard iron offset in each axes. The calculated hard iron offsets are subtracted from the raw magnetometer to completely get rid of hard iron errors in the normal operation of the sensor.

3.2. Results

The raw accelerometer data, which is obtained from MPU6050, is in terms of $m/_{s^2}$ as shown in Figure 25. It can be easily seen that the x-axis and y-axis values are not zero. The z-axis values are around 1g (9.8 $m/_{s^2}$), but with a small offset. When the DC offset is removed from the accelerometer data, we can see that x-axis and y-axis values are around zero, and z-axis values are around 9.8 $m/_{s^2}$ (1g) as shown in Figure 26.

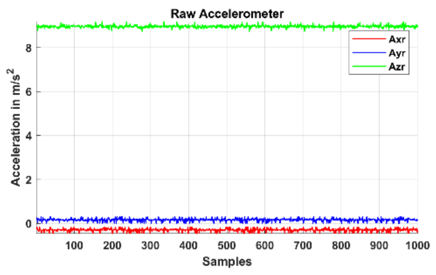


Figure 25. Raw accelerometer data

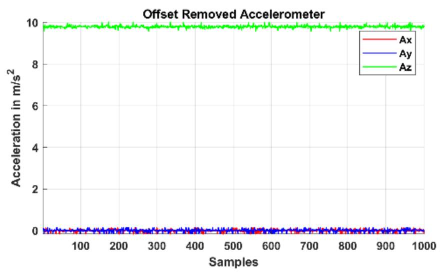
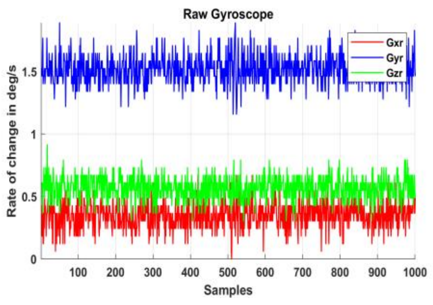


Figure 26. Offset Removed accelerometer data

The raw gyroscope data and the offset removed gyroscope data are shown in Figure 27 and Figure 28 respectively. In all three axes of the raw gyroscope data, the values are not around zero which means that there is a DC offset. When the procedure to remove the offset from the gyroscope data is applied, it can be seen that the values are centered around zero as shown in Figure 28.



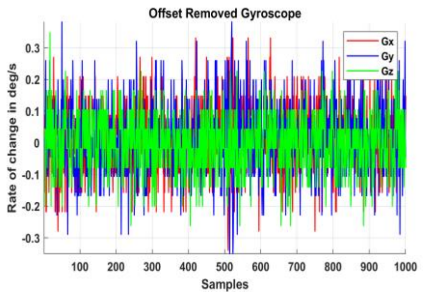
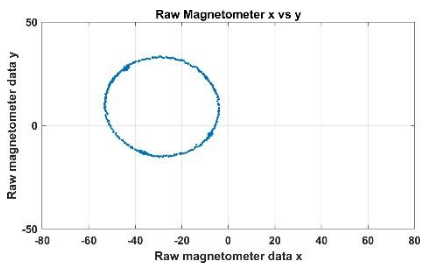


Figure 27. Raw gyroscope data

Figure 28. Offset Removed gyroscope data

The raw magnetometer data x vs y, y vs z, and x vs z are shown in Figure 29, Figure 31, and Figure 33 respectively. It can be easily see that the data is not around the center (0, 0) due to the hard iron offset. On the other hand, the hard iron offset removed magnetometer data x vs y, y vs z and x vs z, are centered around as can be seen in Figure 30, Figure 32, and Figure 34 respectively.



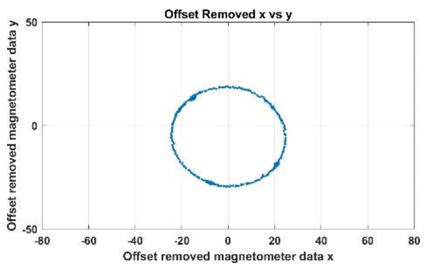


Figure 29. Raw magnetometer data

Figure 30. Offset Removed magnetometer data

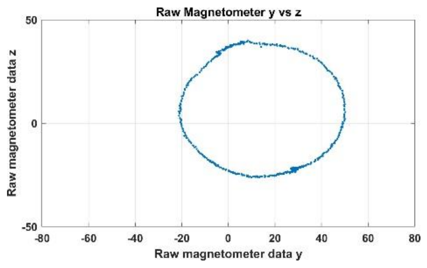


Figure 31. Raw magnetometer data

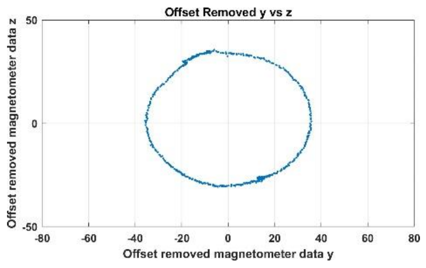


Figure 32. Offset Removed magnetometer data

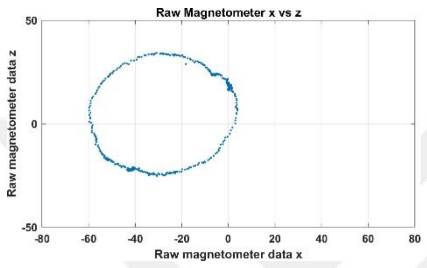


Figure 33. Raw magnetometer data

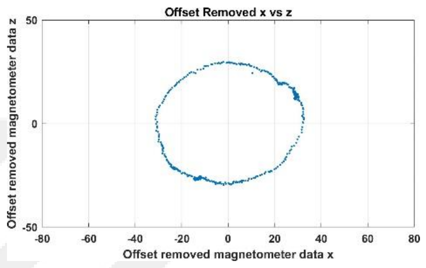


Figure 34. Offset Removed magnetometer data

CHAPTER 4: NOISE FILTERS

Determination of orientation is the key for autonomous vehicles and the orientation of the autonomous vehicle is determined using the roll (ϕ), pitch (θ), and yaw (ψ) angles. The yaw angle, also called the heading, is the most important angle to control the vehicle. It shows the rotation around the z-axis. The roll and pitch angles represent the rotations around the x-axis and y-axis respectively. The roll (ϕ), pitch (θ), and yaw (ψ) are shown in Figure 35.

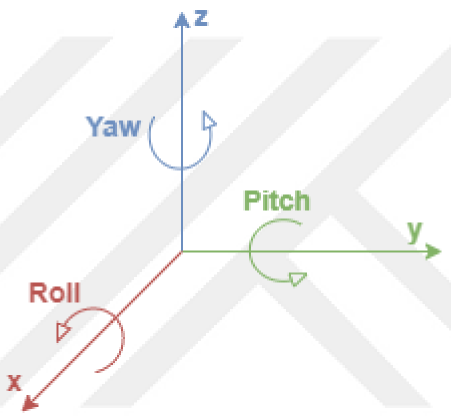


Figure 35. Roll(ϕ), Pitch(θ) and Yaw (ψ) angles

The roll, pitch, and yaw angles can be calculated using the accelerometer, gyroscope, and magnetometer data. Using the accelerometer data, the roll and pitch angles can be calculated, but the calculated angles can be inaccurate. Since the accelerometer data is collected in the autonomous car, the vibration of the car while it is not moving or vibrations while it is moving can affect the measurement and cause errors in the calculation of the angles. Therefore, the calculation of the roll and pitch angles using accelerometer data is not sufficient (Gui, Tang, and Mukhopadhyay, 2015) to determine the orientation angle accurately. The gyroscope data also could be used in the calculation of the roll, pitch, and yaw angles since the gyroscope measurements provide the angular velocities in all three axes. In order to calculate the angles, the integral of these angular velocities obtained from the gyroscope data should be taken. This can cause a problem in the long term since the gyroscope data tends to drift. On the other hand, the gyroscope information is only reliable in a short period of time (Gui, Tang, and Mukhopadhyay,2015). Furthermore, the magnetometer data can also be used in the yaw angle calculation. The magnetometer data is accurate in a long period of time whereas it can be not accurate in the short term due to the effect of external magnetic disturbances and noise.

Moreover, the wheel odometry provides orientation and position information of the autonomous vehicle as mentioned in section2. In order to get more accurate orientation information, the advantages and disadvantages of the accelerometer, gyroscope, magnetometer and wheel odometry can be combined with filters. In this thesis, different types of filters, which are complementary, Madgwick, Kalman, and integrated Kalman filter, are implemented on IMU data. Moreover, the Kalman filter is implemented on both the IMU and the wheel odometry data.

In order to reduce noise on the IMU data, a pre-filter is implemented. The pre-filter response on the IMU data is used as the input for the noise filters namely the complementary, Madgwick, Kalman and the integrated Kalman filter. Moreover, the filters are implemented using the ROS platform and python script on the autonomous vehicle and the performance of the filters are examined on MATLAB.

4.1. Pre-Filter

In this section, the pre-filter method is explained. The data obtained from the accelerometer, gyroscope, and the magnetometer are noisy and in order to reduce the effect of noise in the orientation calculation, a pre-filter is applied on the IMU data. The pre-filter method is taken from Charlotte Treffer’s master thesis (Treffers and Wi-etmarschen, 2016). In my thesis, I used the ROS platform and python scripts to develop my application. The original method was written in C++, therefore, I developed a python version of the method as presented below.

```
def NF(value, CV, NP, NPS, NPL):
    if(value<CV-NP or value>CV+NP):
        filtered_value = value
    else:
        if(value<CV-NPS or value>CV+NPS):
            filtered_value=(value+CV)/2
        else:
            filtered_value=(1-NPL)*CV +NPL*value
    return filtered_value
```

In this method, there are two variables called value (V) and compared value (CV), and three parameters called noise parameter (NP), noise parameter second (NPS), and noise parameter low (NPL). In the function, value is the current raw accelerometer, gyroscope or magnetometer data on the x-axis, y-axis or z-axis. The first if-statement checks whether the value is smaller than the compared value minus the noise parameter or greater than compared value plus the noise parameter. If this statement is true, then there is a real change in the data and the filter's output is equal to the value. If there is no significant change in the value, then the value is compared with compared value plus and minus noise parameter second. If the case is true, then the filter output is the average of the value and previous value which is compared value. Otherwise, a low pass filter is applied using the parameter which is noise parameter low.

In summary, there are two threshold parameters which are NP and NPS. If the current data, accelerometer, gyroscope, or magnetometer, is above the first threshold (NP), then the filter output is the current data. If it is below the NP and above the NPS, then the filter output is the average of the current data and of previous filter output. If the current data is below the two-threshold parameter that means that the data is not changed significantly on the current state, then the filter applies low-pass filter to the data. The NP and NPS values are determined separately for the accelerometer, gyroscope, and the magnetometer data while the autonomous vehicle is stationary. If the vehicle is stationary, the data should not change too much. If it changes then the pre-filter applies low-pass filter to the signal to obtain more accurate results and to get rid of the noise.

4.1.1. Implementation

The pre-filter is implemented using the python script on the ROS platform. The pre-filter is implemented on a single python package for the accelerometer, gyroscope and the magnetometer.

1. Pre-Filter implementation in accelerometer data: The accelerometer data obtained from the MPU6050 are noisy. In order to use the accelerometer data in the orientation calculations, the noise should be reduced. The pre-filter is used to reduce the measurement noise in the accelerometer data. In this method, the value is the accelerometer value, and the compared value is the previous output of the pre-filter. If the current accelerometer value is smaller than the previous output of the filter minus NP or greater than the previous output of the filter plus NP, then the output of the filter is set to the current accelerometer value. If this is not the case, the current accelerometer value is compared with the previous accelerometer value plus and minus NPS. If the current accelerometer value is smaller than the previous output of the filter minus NPS or greater than the previous output of the filter plus NPS, then the filter output is set to the average of the current accelerometer value and previous output of the filter. If the current accelerometer value is much smaller, then the low-pass filter is applied. The cut-off frequency of the low-pass filter is $F_c = \frac{NPL}{(1-NPL)*2*\pi*\Delta\mathrm{T}} = 0.20\ Hz$ where ΔT is the sampling time which is about 0.02 seconds and the NPL is 0.025. Furthermore, the NP and NPS parameters are determined by observing the sensor output when the sensor is stationary as 0.2095 $m/_{s^2}$ and 0.1796 $m/_{s^2}$ respectively.

2. Pre-Filter implementation in gyroscope data: The gyroscope data which are acquired from the MPU6050 are very noisy and this noise causes drift. In order to prevent the drift problem, the pre-filter is applied. In the pre-filter method, the value is the gyroscope data and the compared

value is zero. Since the compared value is zero, averaging can be avoided by taking the NP and NPS parameters as the same. Moreover, the NPL value is taken as zero and the low-pass filter is not applied. Basically, the pre-filter output is the current gyroscope data when the current value is smaller than the negative NP or greater than the NP or NPS. Otherwise, the filter output is zero. Similarly, after the observation of gyroscope data, the NP or NPS parameters are determined as 0.267 $^{\circ}/_{s}$.

3. Pre-Filter implementation in magnetometer data: The magnetometer data which are obtained from the QMC5883L are noisy even when the sensor is at the stationary position. In order to get more reliable results in the orientation calculation, the pre-filter is also implemented on the magnetometer data. A similar procedure is applied as we did on the accelerometer data. First, the offset removed magnetometer data are collected, while the sensor is stationary and the collected data are plotted in MATLAB. As a result, the NP and NPS values are determined as 3 µT. In the pre-filter implementation, the variable value is the current magnetometer data, and the compared value is the previous output of the filter. The NPL coefficient is the same as the NPL parameter calculated for the accelerometer. Also, the working principle of this pre-filter is same as that of the pre-filter implemented for the accelerometer data.

The pre-filter variables and parameters are summarized in Table 1.

Table 1. The variables and parameters used in the filter

Data	Value	Compared Value	NP	NPS	NPL
Accelerometer	Curr. Acc. Data	Prev. filter output	0.2095	0.1796	0.025
Gyroscope	Curr. Gyro. Data	0	0.267	0.267	0
Magnetometer	Curr. Magne. Data	Prev. filter output	3	3	0.025

4.1.2. Results

The offset removed accelerometer, gyroscope, and magnetometer data in each axes are compared with the pre-filter results. The sensor outputs and the filter outputs were collected while the vehicle is stationary and when the vehicle is rotated in one direction.

The pre-filter output for the accelerometer and the offset removed accelerometer data on the x-axis, y-axis, and z-axis are shown in Figure 36, Figure 37, and Figure 38 respectively. The green line in the graphs represent the offset removed accelerometer data and the red line represents pre-filter output. The results show that pre-filter has reduced the noise considerably, whereas the offset removed accelerometer signal is quite noisy. For the offset removed accelerometer data, the absolute mean errors are 0.031169 ± 0.045067, 0.042499 ± 0.060542, and 0.041376 ± 0.055739 on the x-axis, y-axis, and z-axis respectively. On the other hand, the pre-filtered accelerometer data has the absolute mean errors of 0.013518 ± 0.021376 on the x-axis, 0.009036 ± 0.006746 on the y-axis, and 0.033788 ± 0.048860 on the z-axis.

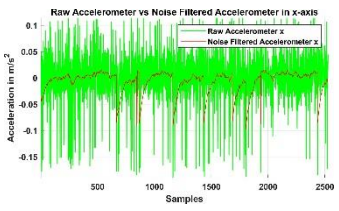


Figure 36. Pre-Filtered acc. in x-axis

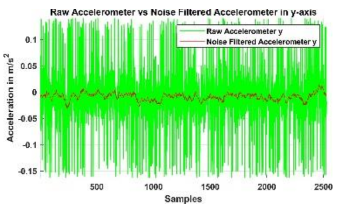


Figure 37. Pre-Filtered acc. in y-axis

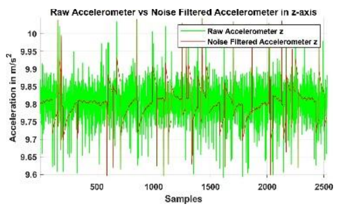


Figure 38. Pre-Filtered acc. in z-axis

The comparison of the offset removed gyroscope data with the pre-filter output for the gyroscope data on all axes, while the vehicle is stationary, are shown in Figure 39, Figure 40, and Figure 41. We can see that the pre-filter output for the gyroscope data is around zero. These results also show that the pre-filtered gyroscope data can be used in the orientation calculation without drift problems since the noise is reduced significantly. For the offset removed gyroscope data, the absolute standard deviations are 0.084670 ± 0.103286, 0.08188 ± 0.106602, and 0.073284 ± 0.093619 on the x-axis, y-axis, and z-axis respectively. On the other hand, the pre-filtered gyroscope data have the absolute mean errors 0.003407 ± 0.032598 on the x-axis, 0.005872 ± 0.041098 on the y-axis, and 0.000363 ± 0.010549 on the z-axis.

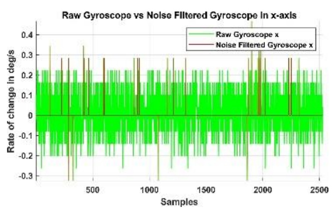


Figure 39. Pre-Filtered gyro. in x-axis

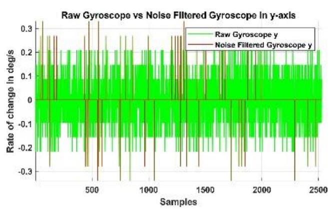


Figure 40. Pre-Filtered gyro. in y-axis

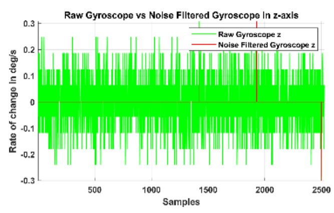


Figure 41. Pre-Filtered gyro. in z-axis

Figure 42, Figure 43, and Figure 44 show the pre-filter output values for the magnetometer data and the offset eliminated magnetometer data on all three axes while the vehicle is stationary. We can see that the pre-filter has reduced the noise in the magnetometer data significantly. For the offset removed magnetometer data, the standard deviations are 0.236397, 0.303134, and 0.695758 on the x-axis, y-axis, and z-axis respectively. On the other hand, the pre-filtered magnetometer data have the standard deviations of 0.033477 on the x-axis, 0.044426 on the y-axis, and 0.240160 on the z-axis.

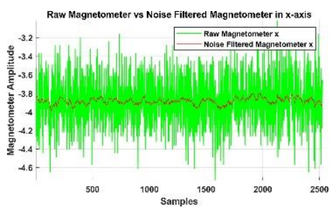


Figure 42. Pre-Filtered magne. in x-axis

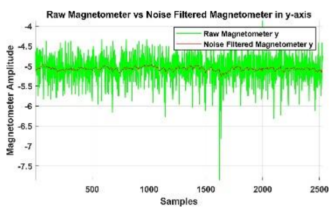


Figure 43. Pre-Filtered magne. in y-axis

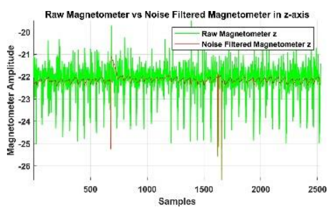


Figure 44. Pre-Filtered magne. in z-axis

The pre-filter output values for the accelerometer and the offset removed accelerometer data, while the vehicle is rotating in one direction, are shown in Figure

45, Figure 46, and Figure 47. The graphs show that the pre-filter response is very fast while the vehicle is rotating in one direction and when the vehicle is stationary. We can also see that the pre-filter applies a low-pass filter to the noisy signal when the vehicle is stationary. On the other hand, the filter output is equal to the measured accelerometer data, when the vehicle changes its direction.

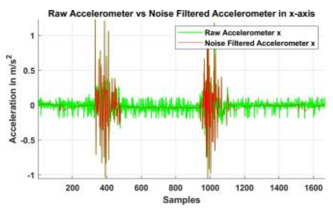


Figure 45. Pre-Filtered acc. in x-axis

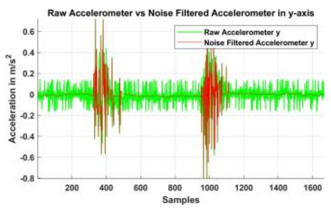


Figure 46. Pre-Filtered acc. in y-axis

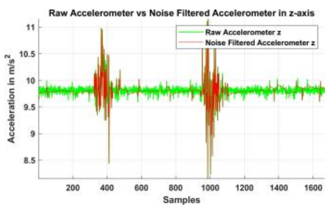


Figure 47. Pre-Filtered acc. in z-axis

The pre-filter results for the gyroscope data on all axes, while the vehicle is rotating in one direction, are shown in Figure 48, Figure 49, and Figure 50. We can see that the pre-filter output is zero when the vehicle is stationary, and it is equal to the current gyroscope data while the vehicle is rotating in one direction.

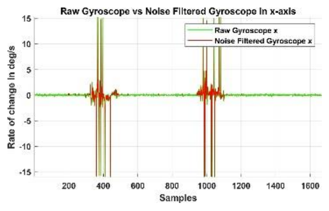


Figure 48. Pre-Filtered gyro. in x-axis

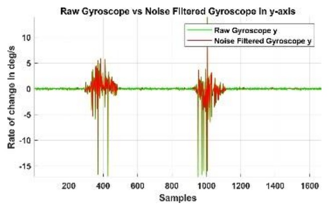


Figure 49. Pre-Filtered gyro. in y-axis

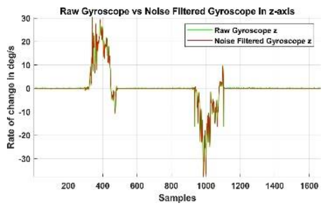
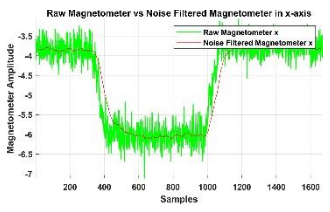
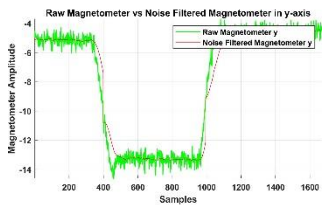


Figure 50. Pre-Filtered gyro. in z-axis

Figure 51, Figure 52, and Figure 53 show the pre-filter output values for the magnetometer data on all axes while vehicle is rotating in one direction. It can be easily seen that the pre-filter output is equal to the current magnetometer output while the vehicle is rotating, and the pre-filter applies a low-pass filter while the vehicle is stationary.





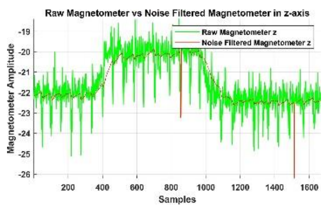


Figure 51. Pre-Filtered magne. in x-axis

Figure 52. Pre-Filtered magne. in y-axis

Figure 53. Pre-Filtered magne. in z-axis

As a result, the pre-filter implementation on offset removed accelerometer, gyro- scope, and magnetometer data reduces the noise significantly. Since the noise is reduced significantly when the vehicle is stationary, the pre-filter also solves the drift problem on the orientation calculation. While the vehicle is rotating in one direction, the pre-filter can quickly respond and give an output which is equal to the currently measured sensor data. Moreover, the pre-filter output for the accelerometer can be equal to the average of the currently measured data and the previous data according to the NP and NPS values.

4.2. Complementary Filter

The complementary filter applies a low-pass and a high-pass filter to the given data. The roll, pitch, and yaw angles can be calculated using the accelerometer, gyroscope, and magnetometer data. The accelerometer and magnetometer data are reliable in a long period of time whereas the gyroscope data is accurate in the short term. Using the complementary filter, the two data can be combined, and more accurate results can be obtained for the roll, pitch, and yaw angles.

To calculate the roll, and pitch angles, the accelerometer and gyroscope data can be used. On the accelerometer data, a low pass filter should be applied whereas a high-pass filter should be applied on the gyroscope data, since the gyroscope data is more reliable on short period of time due to the drift problem and the accelerometer is more reliable in the long period of time due to the possible vibration errors in the short period of time. Before applying the complementary filter, the roll and pitch angles should be calculated from accelerometer and gyroscope data. In order to calculate the roll and pitch angle from gyroscope data, the gyroscope data should be integrated over time to determine the angles from the angular speeds. On the other hand, the roll and pitch angles can be calculated using the accelerometer data as shown in equation 19 and 20 (Treffers and Wietmarschen,2016). In these equations, the variables ax, ay, and az, represent the accelerometer data in the x-axis, y-axis, and z-axis respectively.

$$\phi_{acc} = arctan\left(\frac{ay}{\sqrt{ax^2+az^2}}\right) \tag{19}$$

$$\theta_{acc} = arctan\left(\frac{-ax}{\sqrt{ay^2+az^2}}\right) \tag{20}$$

Using the calculated ϕ_{acc} and θ_{acc} and the gyroscope data, the complementary filter can be applied to obtain the roll and pitch angles. The complementary filter equations are shown in equation 21 and 22. The variables gx, and gy are the gyroscope data in the x-axis and y-axis respectively. ΔT is sampling time which is 0.02 seconds and α is the complementary filter parameter which can be chosen between zero and one. If α is selected closer to zero, then the roll and pitch angles are calculated based on the accelerometer data in the long term. If it is taken closer to one, then the roll and pitch angles are obtained based on the gyroscope data in the short term. In order to get reliable results while taking the advantage of the accelerometer and gyroscope data, the α parameter is selected closer to one. In other words, by selecting the α closer to one, the accelerometer data is used in a long period of time and the gyroscope data is used in a short period of time.

$$\phi = \alpha * (g_x * \Delta T + \phi) + (1-\alpha) * \phi_{acc} \tag{21}$$

$$\theta = \alpha * \left(g_y * \Delta T + \theta\right) + (1-\alpha) * \theta_{acc} \tag{22}$$

The yaw angle can be calculated using the gyroscope and the magnetometer data. Similarly, the gyroscope data is accurate in a short period of time whereas the magnetometer data is accurate in a long period of time. By using the complementary filter, the advantages of both gyroscope and magnetometer can be combined to get an accurate yaw angle. In order to calculate yaw angle from the magnetometer data, equation23is used. The Mx, and My are the magnetometer data on the x-axis and y-axis respectively.

$$\psi_{mag} = arctan\left(\frac{M_y}{M_x}\right) \tag{23}$$

The complementary filter is applied to calculate ψ_{mag}, and the gyroscope data is used to obtain an accurate yaw angle as shown in equation 24 where gz is the gyroscope data on the z-axis and α is the complementary filter parameter. Similarly, α is selected closer to one to rely on the magnetometer data in the long term and the gyroscope data in the short term.

$$\psi = \alpha * (g_z * \Delta T + \psi) + (1 - \alpha) * \psi_{mag} \tag{24}$$

The algorithm that is developed for complementary filter implementation is shown in Appendix G. The Complimentary class is called by the filter node, that is developed on ROS platform, in every iteration in order to estimate orientation using gyroscope and magnetometer.

4.3. Madgwick Filter

The Madgwick filter, was proposed by Madgwick in 2011 (Madgwick et al., 2011), proposes a low-cost and accurate orientation estimation using accelerometer, gyroscope, and magnetometer data. Compare to the Kalman filter, the Madgwick filter provides less computational cost.

In order to calculate roll, pitch, and yaw angles using accelerometer, gyroscope, and magnetometer data, firstly, an array, denoted as S_{w_t}, is created to store raw gyroscope data at time t as shown in 25. The gyroscope data on the x-axis, y-axis, and z-axis which are on the sensor frame, are defined as gx, gy, and gz respectively. Then, the quaternion derivative, ${}^{S}_{E}\dot{q}_t$ at time t is calculated to obtain rate of change in orientation from sensor frame to earth frame relative to the sensor frame as shown in 26 (Madgwick, 2010). The ${}^{S}_{E}\hat{q}_{est,t-1}$is the estimation of the orientation at time t-1 and the quaternion product with the S_{w_t} give us the quaternion derivative. By taking numerical integration of the ${}^{S}_{E}\dot{q}_{w,t}$, the orientation, ${}^{S}_{E}q_{w,t}$, at time t can be calculated with respect to earth frame as shown in 27 (Madgwick, 2010). In the equation, the ΔT is the sampling time and it is 0.02 seconds.

$$s_{w_t} = \left[0 \; g_x \; g_y \; g_z\right] \tag{25}$$

$${}^{S}_{E}\dot{q}_{w,t} = 0.5 * \; {}^{S}_{E}\hat{q}_{est,t-1} \otimes \; s_{w_t} \tag{26}$$

$${}^{S}_{E}q_{w,t} = {}^{S}_{E}\hat{q}_{est,t-1} + \; {}^{S}_{E}\dot{q}_{w,t} \; \Delta T \tag{27}$$

Similarly, the accelerometer and magnetometer data, obtained at time t, are stored in two different arrays as shown in equation 28, and 29. Then accelerometer, and magnetometer data are normalized as shown in equation 30, and 31.

$$s_{a_t} = \left[0 \; a_x \; a_y \; a_z\right] \tag{28}$$

$$s_{m_t} = \left[0 \; m_x \; m_y \; m_z\right] \tag{29}$$

$$s_{\hat{a}_t} = \frac{s_{a_t}}{\sqrt{a_x{}^2 + a_y{}^2 + a_z{}^2}} = \begin{bmatrix}0.0 \ \hat{a}_x \ \hat{a}_y \ \hat{a}_z\end{bmatrix} \tag{30}$$

$$s_{\hat{m}_t} = \frac{s_{m_t}}{\sqrt{m_x{}^2 + m_y{}^2 + m_z{}^2}} = \begin{bmatrix}0.0 \ \hat{m}_x \ \hat{m}_y \ \hat{m}_z\end{bmatrix} \tag{31}$$

The previous orientation estimation matrix which is at time t-1 is defined as shown in equation 32. The next step is to determine the Jacobian matrix and objective function using the previous orientation estimation and accelerometer data. The Jacobian matrix and objective function are calculated as shown in equation 33, and 34 (Madgwick, 2010).

$${}^S_E\hat{q}_{est,t-1} = [q_1 \ q_2 \ q_3 \ q_4] \tag{32}$$

$$J_g({}^S_E\hat{q}_{est,t-1}) = \begin{bmatrix} -2q_3, \ 2q_4, -2q_1, \ 2q_2 \\ 2q_2, \ 2q_1, 2q_4, \ 2q_3 \\ 0, -4q_2, -4q_3, \ 0 \end{bmatrix} \tag{33}$$

$$f_g({}^S_E\hat{q}_{est,t-1}, \ s_{\hat{a}_t}) = \begin{bmatrix} 2(q_2q_4 - q_1q_3) - \hat{a}_x \\ 2(q_1q_2 - q_3q_4) - \hat{a}_y \\ 2(\frac{1}{2} - q_2{}^2 - q_3{}^2) - \hat{a}_z \end{bmatrix} \tag{34}$$

The rotation matrix and measured direction with respect to earth’s magnetic field in the earth frame should be calculated for magnetometer data. The rotation matrix is defined as shown in equation 35 (Madgwick, 2010). The measured direction can be calculated with matrix multiplication of rotation matrix and normalized magnetometer data as shown in equation 36. The obtained result is stored in an array to calculate Jacobian matrix and objective function. Furthermore, one horizontal axis and one vertical axes can represent the earth’s magnetic field. In order to obtain this, the ${}^S_E h_t$ can be used as shown in equation 37 (Madgwick, 2010). The jacobian matrix and objective funtion calculations are shown in equation 38 and 39 (Madgwick, 2010).

$$ {}_{E}^{S}R_{\,t-1} = \begin{bmatrix} 2{q_1}^2 - 1 + 2{q_2}^2, & 2(q_2q_3 + q_1q_4), & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4), & 2{q_1}^2 - 1 + 2{q_3}^2, & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 - q_1q_3), & 2(q_3q_4 - q_1q_2), & 2{q_1}^2 - 1 + 2{q_4}^2 \end{bmatrix} \tag{35} $$

$$ {}_{E}^{S}h_t = {}_{E}^{S}R_{\,t-1} * \begin{bmatrix} \hat{m}_x \\ \hat{m}_y \\ \hat{m}_z \end{bmatrix} = \left[h_x,\ h_y,\ h_z\right] \tag{36} $$

$$ E_{b_t} = \left[0,\ \sqrt{{h_x}^2 + {h_y}^2},\ 0,\ h_z\right] = [0,\ b_x,\ 0,\ b_z] \tag{37} $$

$$ J_b = 2\begin{bmatrix} -b_zq_3,\ b_zq_4, -2b_xq_3 - b_zq_1, -2b_xq_3 - b_zq_1 \\ -b_xq_4 + b_zq_2, b_xq_3 + b_zq_1, b_xq_2 + b_zq_4, -b_xq_1 + b_zq_3 \\ b_xq_3,\ b_xq_4 - 2b_zq_2, b_xq_1 - 2b_zq_3,\ b_xq_2 \end{bmatrix} \tag{38} $$

$$ f_b(E_{b_t}) = \begin{bmatrix} 2b_x(0.5 - {q_3}^2 - {q_4}^2) + 2b_z(q_2q_4 - q_1q_3) - \hat{m}_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 - q_3q_4) - \hat{m}_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z(0.5 - {q_2}^2 - {q_3}^2) - \hat{m}_z \end{bmatrix} \tag{39} $$

The Jacobian matrix, $J_g({}_{E}^{S}\hat{q}_{est,t-1})$, that is calculated from accelerometer and the Jacobian matrix, J_b, which is calculated from magnetometer are concatenated as shown in equation 40. Also, the objective junctions which are $f_g({}_{E}^{S}\hat{q}_{est,t-1},\ s_{\hat{a}_t})$, and $f_b(E_{b_t})$ are concatenated as shown in equation 41.

$$ J_{g,b}({}_{E}^{S}\hat{q}_{est,t-1}, E_{b_t}) = \begin{bmatrix} J_g\left({}_{E}^{S}\hat{q}_{est,t-1}\right) \\ J_b \end{bmatrix} \tag{40} $$

$$f_{g,b}({}^S_E\hat{q}_{est,t-1},\ E_{b_t}, s_{\hat{m}_t}, s_{\hat{a}_t}) = \begin{bmatrix} f_g\left({}^S_E\hat{q}_{est,t-1},\ s_{\hat{a}_t}\right) \\ f_b(E_{b_t}) \end{bmatrix} \tag{41}$$

Finally, the current orientation, ${}^S_E q_{\nabla,t}$, can be calculated using the previous orientation, ${}^S_E\hat{q}_{est,t-1}$, and objective function gradient, ∇f (Madgwick, 2010). ∇f is calculated as shown in the equation 42, since the accelerometer and magnetometer data are used to acquire roll, pitch and yaw angles. Moreover, the filter gain, β, shows that mean measurement errors on gyroscope data and the errors can be sensor noise, calibration errors, and quantisation errors (Madgwick, 2010). The calculation of the filter gain is shown in the equation 44. The $\dot{w}_\beta$ is the mean gyroscope measurement error for each axis (Mario, 2019).

$$\nabla f = J^T_{g,b}({}^S_E\hat{q}_{est,t-1}, E_{b_t}) f_{g,b}({}^S_E\hat{q}_{est,t-1},\ E_{b_t}, s_{\hat{m}_t}, s_{\hat{a}_t}) \tag{42}$$

$${}^S_E q_{\nabla,t} = {}^S_E\hat{q}_{est,t-1} - \beta \frac{\nabla f}{\|\nabla f\|} \tag{43}$$

$$\beta = \sqrt{\frac{3}{4}}\ \dot{w}_\beta \tag{44}$$

Using the estimated orientation in quaternion form, the roll, pitch and yaw angles can be calculated in Euler form as shown in equation 45, 46, and 47.

$$\phi = \arctan\left(\frac{2q_1q_2+2q_3q_4}{{q_1}^2+{q_4}^2-{q_2}^2-{q_3}^2}\right) \tag{45}$$

$$\theta = -\arcsin(2(q_2q_4 - q_1q_3)) \tag{46}$$

$$\psi = \arctan(\frac{2q_2q_3+2q_1q_4}{q_1{}^2+ q_2{}^2-q_3{}^2-q_4{}^2}) \tag{47}$$

4.4. Kalman Filter

The Kalman filter estimates the orientation of the autonomous vehicle using measurements from accelerometer, gyroscope, magnetometer, and wheel odometry, and with knowledge of the noises (Treffers and Wietmarschen, 2016). The noises are the measurement noise, which is the noise of the input and process noise that is noise of the system itself (Sloth, 2017).

The Kalman filter algorithm is recursive and can be grouped into three steps as prediction, measurement, and update. In the prediction state, there are two equations which are dynamic model equation, and predictor covariance equation. The dynamic model equation consists of state transition matrix A with previous state vector x_{k-1}, control matrix B with current input vector u_k, and process noise vector w_k as shown in equation 48. The second equation, the predictor covariance, consists of state transition matrix A, previous predicted covariance matrix P_{k-1}, and process noise uncertainty matrix Q_k as shown in equation 49. The coefficients, matrices, equations, and their definitions that are used on prediction step are summarized on Table 2.

The predicted covariance matrix P_k shows that the current predicted state vector estimation is reliable or not reliable. If the small values are obtained, then the current predicted state estimation is more reliable (Sloth, 2017). Moreover, the process noise vector w_k is the noise of the system and it is gaussian distributed with a zero mean and covariance matrix Q_k at time k. In this thesis, the covariance matrix Q_k represents the estimation variance of the magnetometer or wheel odometry and variance of bias. These variances depend on the cases which are mentioned in Kalman filter implementation. The variance of the measurement and bias can be defined according to the system response. For example, if the estimated angle tends to drift, the variance of the bias should be chosen high. On the other hand, the measurement variance should be chosen low if the Kalman filter response lasts long. In this thesis, variance of measurement and the bias is selected after the observation of the filter response.

$$\hat{x}_k = Ax_{k-1} + Bu_k + w_k \tag{48}$$

$$\hat{P}_k = AP_{k-1}A^T + Q_k \tag{49}$$

Table 2. Kalman Filter Prediction Step Coefficients, Matrices, and Definitions

Coefficients, Matrices, and Equations	**Definitions**
$\hat{x}_k = Ax_{k-1} + Bu_k + w_k$	Dynamic model equation
$\hat{P}_k = AP_{k-1}A^T + Q_k$	Predictor covariance equation
$\hat{x}_k$	Predicted state vector matrix or vector
$\hat{P}_k$	State vector matrix or vector
P_k	Covariance matrix
x_{k-1}	Previous state matrix or vector
P_{k-1}	Previous predictor covariance matrix
u_k	Input vector or variable
w_k	Process noise vector
Q_k	Process noise uncertainty matrix
A	State transition matrix
B	Control matrix

In the measurement step, there are three equations. The first one is innovation or measurement residual equation, the second one is innovation covariance equation and the last one Kalman gain equation. The innovation equation consists of measurement matrix or vector z_k, observation model matrix H, and predicted state vector $\hat{x}_k$ as shown in equation 50. As shown in equation 51, the innovation covariance matrix S_k is calculated using observation model matrix H, predicted covariance matrix $\hat{P}_k$, and measurement uncertainty vector R. In addition, the kalman gain K_k is defined in equation 52. The coefficients, matrices, equations, and their definitions that are used on measurement step are summarized on Table 3.

The innovation covariance matrix predicts that how much the current measurement is reliable based on the predicted covariance matrix $\hat{P}_k$. The measurement uncertainty vector R is the variance of the measurement. If the R is taken high, the measurement is not reliable and the effect of measurement in angle calculation is not much. In addition, the Kalman gain determines that how much we trust innovation matrix y_k.

$$y_k = z_k + H\hat{x}_k \tag{50}$$

$$S_k = H\hat{P}_k H^T + R \tag{51}$$

$$K_k = \hat{P}_k H^T S^{-1} \tag{52}$$

Table 3. Kalman Filter Measurement Step Coefficients, Matrices, and Definitions

Coefficients, Matrices, and Equations	**Definitions**
$y_k = z_k + H\hat{x}_k$	Innovation or measurement residual equation
$S_k = H\hat{P}_k H^T + R$	Innovation covariance equation
$K_k = \hat{P}_k H^T S^{-1}$	Kalman gain equation
y_k	Innovation matrix or vector
S_k	Innovation covariance matrix
K_k	Kalman gain
z_k	Measurement matrix or vector
H	Observation model matrix or vector
R	Measurement uncertainty vector

In the update step, the state vector x_k, and covariance matrix P_k are updated using calculated predicted state vector, predicted covariance matrix, kalman gain, observation matrix, and innovation matrix as shown in equations 53 and 54. The updated state vector ${x_k}^+$, and updated covariance matrix ${P_k}^+$ are used on the next iteration of Kalman algorithm as, x_{k-1} and P_{k-1} respectively. The coefficients, matrices, equations and their definitions that are used on the update step are summarized on Table 4.

$${x_k}^+ = \hat{x}_k + K_k y_k \tag{53}$$

$${P_k}^+ = (1 - K_k H)\hat{P}_k \tag{54}$$

Table 4. Kalman Filter Update Step Coefficients, Matrices, and Definitions

Coefficients, Matrices, and Equations	**Definitions**
${x_k}^+ = \hat{x}_k + K_k y_k$	Dynamic model update equation
${P_k}^+ = (1 - K_k H)\hat{P}_k$	Predictor covariance update equation
${x_k}^+$	Updated state vector
${P_k}^+$	Updated covariance matrix

4.4.1. Implementation

The Kalman filter is implemented on the accelerometer, gyroscope, magnetometer, and wheel odometry data in order to calculate orientation of the autonomous agricultural vehicle. Three different cases are developed using various data in order to estimate orientation with Kalman filter. In the first case, the Kalman filter algorithm is used with the gyroscope and magnetometer data. The state vector x_k is defined as shown in equation 55. The first element of state vector is yaw angle ϕ and the second element is bias $\dot{\phi}_b$. The yaw angle, that is calculated from magnetometer, is used as measurement vector z_k. The state transition matrix A, and control matrix B are defined as shown in equations 56 and 57. Furthermore, the input variable u_k is the gyroscope data in z-axis in terms of $^{\circ}/_{s}$. For all equations, sampling time ΔT is 0.02 seconds.

$$x_k = \begin{bmatrix} \phi \\ \dot{\phi}_b \end{bmatrix} \tag{55}$$

$$A = \begin{bmatrix} 1 & -\Delta T \\ 0 & 1 \end{bmatrix} \tag{56}$$

$$B = \begin{bmatrix} \Delta T \\ 0 \end{bmatrix} \tag{57}$$

In the second case, the Kalman filter algorithm is used with the gyroscope and wheel odometry data. In this case, the measurement vector z_k is defined as yaw angle that is calculated from wheel odometry as shown in equation 2. The state vector, state transition matrix, and control matrix are defined as same with the first case. Also, the input variable is the gyroscope data in z-axis in terms of $^{\circ}/_{s}$.

In the last case, gyroscope, magnetometer, and wheel odometry data are fused with Kalman filter. Similarly, the yaw angle is taken as measurement vector z_k. In this case, the state transition matrix is same with the first and second case. The control matrix is defined as variable equals to ΔT, and the input vector u_k is defined as shown in equation 59. The $w_z{}^{est}$ is the change rate of yaw angle, that is calculated from magnetometer. The calculation of $w_z{}^{est}$ is mentioned in integrated Kalman filter method.

$$B = \Delta T \tag{58}$$

$$u_k = \begin{bmatrix} g_z \\ {w_z}^{est} - g_z \end{bmatrix} \tag{59}$$

In all three cases, initial values are defined for state vector, covariance matrix, process noise uncertainty matrix, and measurement uncertainty vector as shown in equations 60, 61, 62, and 63 respectively.

$$x_k = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{60}$$

$$P_k = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{61}$$

$$Q_k = \begin{bmatrix} 0.002 & 0 \\ 0 & 0.005 \end{bmatrix} \tag{62}$$

$$R = 0.03 \tag{63}$$

Developed algorithm for Kalman filter implementation is shown in Appendix H. The Kalman filter algorithm is applied to all three cases with different inputs on ROS platform using python script. In order to calculate orientation using Kalman filter, the Kalman class is called by the filter node in every iteration to update the state vector, and the covariance vector for the three cases.

4.5. Integrated Kalman Filter

The integrated Kalman filter, is proposed by El-Diasty (El-Diasty, 2014), suggest that more accurate orientation calculation with reducing possible magnetic disturbance errors. The Kalman filter, is mentioned in the section 4, is designed to reduce measurement and bias errors. However, the magnetometer can be easily affected by magnetic disturbances, and this can cause an error in orientation calculation. In this method, the Kalman filter is used to reduce magnetic disturbance error caused by the presence of iron or magnetic material around the magnetometer sensor (El-Diasty, 2014).

The orientation is calculated without magnetic disturbance error, taking derivative of the roll, pitch and yaw angles that are calculated from magnetometer as shown in the equations 64, 65, and 66, since the earth magnetic field changes slowly in time (El-Diasty,2014). The ${m_x}^{mes}$, ${m_y}^{mes}$, and ${m_z}^{mes}$are the magnetometer data that are acquired from pre-filter output for raw magnetometer data. Taking the derivative of the each angle, the rate of change of the roll, pitch, and yaw angles can be calculated.

$$ {w_x}^{est} = \frac{d\phi_{mag}}{dt} = \frac{\mathrm{d}(\mathrm{arctan}(\frac{{m_y}^{mes}}{{m_z}^{mes}}))}{dt} \tag{64} $$

$$ {w_y}^{est} = \frac{d\theta_{mag}}{dt} = \frac{\mathrm{d}(\mathrm{arctan}(\frac{{m_x}^{mes}}{{m_z}^{mes}}))}{dt} \tag{65} $$

$$ {w_z}^{est} = \frac{d\,\psi_{mag}}{dt} = \frac{\mathrm{d}(\mathrm{arctan}(\frac{{m_y}^{mes}}{{m_x}^{mes}}))}{dt} \tag{66} $$

The derivative of the roll, pitch and yaw angles can be simplified as shown in the equations 67, 68, and 69. In the equation, ${\dot{m}_x}^{est}$, ${\dot{m}_y}^{est}$, and ${\dot{m}_z}^{est}$ are the estimated derivative of the magnetometer data in time. The estimated derivative of magnetometer data is calculated as taking the difference of current magnetometer data minus the previous magnetometer data divided by the ΔT. The ΔT is the sampling time and it is 0.02 seconds.

$$w_x{}^{est} = \frac{\dot{m}_y{}^{est} m_z{}^{mes} - \dot{m}_z{}^{est} m_y{}^{mes}}{(m_y{}^{mes})^2 + (m_z{}^{mes})^2} \tag{67}$$

$$w_y{}^{est} = \frac{\dot{m}_x{}^{est} m_z{}^{mes} - \dot{m}_z{}^{est} m_x{}^{mes}}{(m_x{}^{mes})^2 + (m_z{}^{mes})^2} \tag{68}$$

$$w_z{}^{est} = \frac{\dot{m}_y{}^{est} m_x{}^{mes} - \dot{m}_x{}^{est} m_y{}^{mes}}{(m_x{}^{mes})^2 + (m_y{}^{mes})^2} \tag{69}$$

The estimated rate of changes of roll, pitch and yaw angles, which are $w_x{}^{est}$, $w_y{}^{est}$, and $w_z{}^{est}$, are given to Kalman filter like gyroscope data. On the other hand, the roll, pitch and yaw angles, are calculated from gyroscope data with taking the integral of gyroscope data in time, are given as measurement to Kalman filter.

4.6. Results

The results are taken for all the filters simultaneously. The filter performances are tested for three different operating conditions: while the autonomous vehicle is stationary, while it is rotating, and while it is stationary with magnetic disturbance. In all conditions, 1000 samples are collected with sampling time of 0.02 seconds. In the rotation case, the vehicle is rotated at 20 degrees and at 50 degrees by controlling the rotation of the vehicle with the Agv_remote_control node at remote PC. Furthermore, in the rotation case, the vehicle stays stationary for a while and after that the vehicle is rotated 20 degrees. Then, the vehicle stays stationary at 20 degrees for a while and afterwards the vehicle is rotated at 50 degree. The vehicle completes its rotation approximately 0.28 seconds for 20 degrees, and 0.32 seconds for 50 degrees.

The complementary filter results are shown in Figure 54, Figure 55, and Figure 56. While the autonomous vehicle is stationary, the yaw angles calculated from gyroscope, magnetometer and complementary filter are plotted with red, blue and green lines respectively as shown in Figure 54. As you can see, the yaw angle calculated from the gyroscope data does not tend to drift, since the pre-filter reduces the noise of the gyroscope data. The yaw angle obtained from the magnetometer does not change too much and it is around zero. The yaw angle that is calculated from the complementary filter is also zero when the vehicle is stationary. When the autonomous vehicle is rotated 20 degrees and 50 degrees, the gyroscope, magnetometer, and complementary filter results are similar. On the other hand, the filter follows the magnetometer data with an absolute mean error of 7.22 degree, when there is magnetic disturbance around the vehicle.

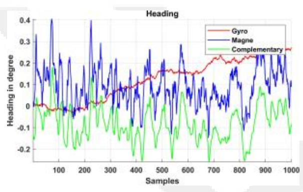


Figure 54. Complementary Filter Stationary

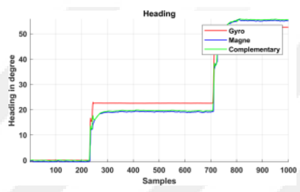


Figure 55. Complementary Filter Movement

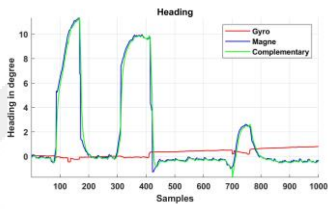


Figure 56. Complementary Filter Magnetic Disturbance

The Madgwick filter results are shown in Figure 57, Figure 58, and Figure 59. The Madgwick filter results are similar to the complementary filter results. When the vehicle is stationary, the calculated yaw angle is around zero and when the vehicle is rotated 20 degrees, the calculated angle is around 20 degrees with an absolute mean error of 0.35 degree. On the other hand, the magnetic disturbance error cannot be eliminated by the Madgwick filter and the absolute mean error is 7.09 degree when there is magnetic disturbance.

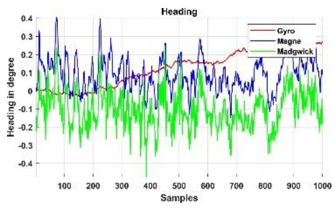


Figure 57. Madgwick Stationary

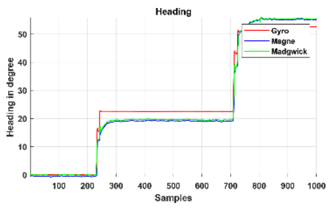


Figure 58. Madgwick Movement

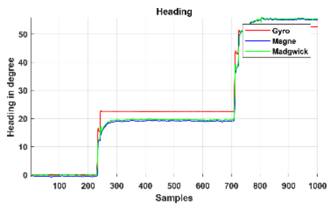


Figure 59. Madgwick Magnetic Disturbance

The Kalman filter results developed using the magnetometer and gyroscope data, are shown in Figure 60, Figure 61, and Figure 62. The Kalman filter results are similar to the complementary filter and Madgwick filter results when the autonomous vehicle is rotating and when it is stationary. The absolute mean error is 0.32 and 5.63 degrees at a yaw angle of 20 degrees and 50 degrees respectively. Similarly, the Kalman filter does not eliminate the errors due to the magnetic disturbances where the absolute mean error is 7.99 when there is a magnetic disturbance.

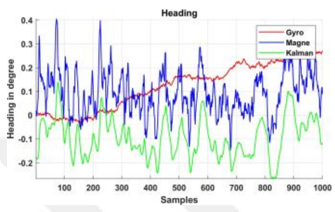


Figure 60. Kalman Stationonary

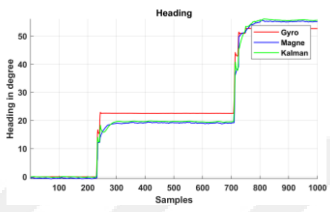


Figure 61. Kalman Movement

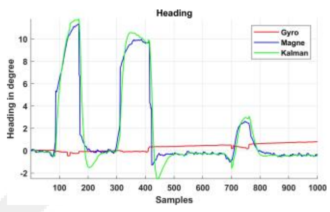


Figure 62. Kalman Magnetic Disturbance

The Kalman filter results developed using the wheel odometry and the gyroscope data, are shown in Figure 63, Figure 64, and Figure 65. When the yaw angle calculated from the odometry is used as the measurement input to the Kalman filter, the absolute mean error is smaller than that of the complementary filter, Madgwick filter, and Kalman filter which use the gyroscope and magnetometer data, when the vehicle is stationary, and when there is magnetic disturbance around the vehicle.

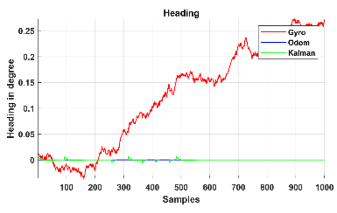


Figure 63. Kalman Odom and Gyro Stationary

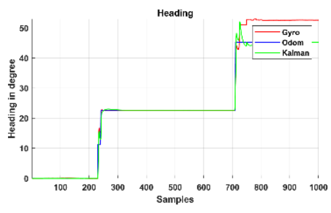


Figure 64. Kalman Odom and Gyro Movement

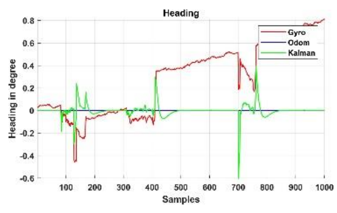


Figure 65. Kalman Odom and Gyro Magnetic

The Kalman filter results developed using the wheel odometry, gyroscope and magnetometer data are shown in Figure 66, Figure 67, and Figure 68. The filter results are similar to those of the Kalman filter implementation using the odometry and gyroscope data, when the vehicle is stationary, and while the vehicle is rotating. However, the absolute mean error, that is calculated when there is magnetic disturbance around the vehicle, is higher.

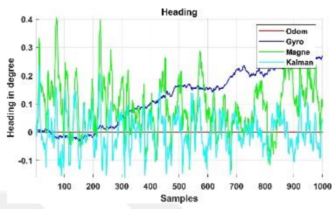


Figure 66. Kalman Odom, Magne and Gyro Stationary

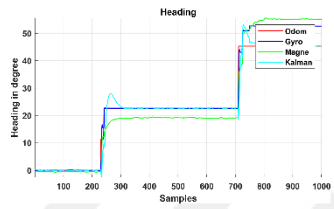


Figure 67. Kalman Odom, Magne, and Gyro Movement

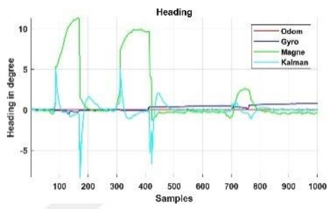


Figure 68. Kalman Odom, Magne, and Gyro Magnetic

The integrated Kalman filter results are shown in Figure 69, Figure 70, and Figure 71. The filter result is similar to the Kalman filter results when the vehicle is stationary. When the vehicle is rotated 20 degree, the absolute mean error of the integrated Kalman filter is 2.56 degrees. At 50 degrees, the absolute mean error is 2.63. When there is magnetic disturbance around the vehicle, the integrated Kalman filter produces better results than those of the Kalman filter using the wheel odometry, magnetometer, and gyroscope data. The filter is affected by the magnetic disturbance in short period of time and the filter output follows the yaw angle that is calculated from gyroscope data as shown in Figure 71. The absolute mean error is 0.92 degree for integrated Kalman filter, when there is magnetic disturbance.

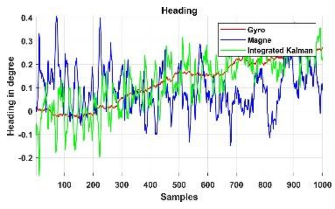


Figure 69. Integrated Kalman Stationary

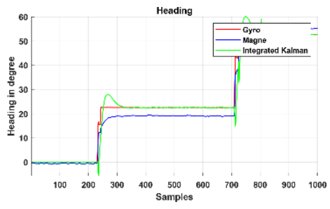


Figure 70. Integrated Kalman Movement

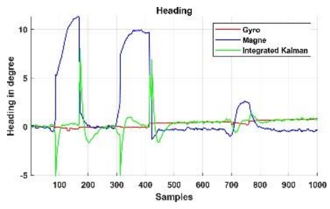


Figure 71. Integrated Kalman Magnetic

The absolute mean error is calculated for each filter and it is summarized on Table 5. For the sake of simplicity, three different Kalman filter implementations are defined as follows:

- **Kalman IMU** uses the gyroscope and magnetometer data
- **Kalman Odom-Gyro** uses the wheel odometry and gyroscope data
- **Kalman Odom-Gyro-Magne** uses the wheel odometry, gyrocope, and magnetometer data

When the vehicle is stationary, the absolute mean error is around zero for all filters. When the vehicle is rotated 20 degrees, Kalman Odom-Gyro, Kalman Odom-Gyro-Magne, and integrated kalman method have similar absolute mean error which is higher than the absolute mean error calculated from the complementary filter, Madgwick filter and Kalman IMU. On the other hand, the integrated kalman filter method has a smaller absolute mean error of 2.63 degrees than those of the other filter implementations when the vehicle is rotated at 50 degree. When there is magnetic disturbance around the vehicle, the mean errors are different in the filters. The Kalman Odom-Gyro, Kalman Odom-Gyro-Magne, and integrated Kalman filter give better solution on the effect of magnetic disturbance. Other filters are directly affected by the magnetic disturbance and the absolute mean error for yaw angle that is calculated from the complementary filter, Madgwick filter, and Kalman IMU, is around 7 degrees. In summary, we can say that all the filters have similar results when the vehicle is stationary and while the vehicle is rotating on the other hand magnetic disturbance error cannot be eliminated with complementary filter, Madgwick filter and Kalman IMU.

Table 5. The Absolute Mean Errors in Filters

Filter	**Stationary**	**20 degrees**	**50 degrees**	**Magne. Dist.**
Complementary	0.096560	0.345533	5.603426	7.224797
Madgwick	0.120812	0.327598	5.440069	7.090184
Kalman	0.106867	0.318881	5.625903	7.991807

IMU				
Kalman Odom-Gyro	0.000482	2.662896	4.674352	0.054352
Kalman Odom-Gyro-Magne	0.051540	2.660897	4.666560	1.091601
Integrated Kalman	0.145179	2.556146	2.627122	0.921807

The computational costs of the filters are summarized in Table 6. In order to evaluate the computational costs of the filters, the filter_cc node, mentioned in Chapter 2, is executed separately for each of the filters and the computational costs of each filter are obtained using the ROS UI in terms of the CPU usage, and the memory usage. In Table 6, we can see that, the complementary filter has the lowest CPU and memory usage as compared to those of the other filters. On the other hand, the CPU and memory usages of the Madgwick, Kalman IMU, Kalman Odom-Gyro, Kalman Odom-Gyro- Magne, and integrated Kalman are similar. In addition, the Kalman Odom-Gyro and Kalman Odom-Gyro-Magne use the wheel odometry information in order to estimate the orientation of the autonomous agricultural vehicle. Since the wheel odometry information is required in the execution of these filter nodes, the Odom node is also executed and the results are presented in Table 6. The computational costs of above filters are actually higher than those of the madgwick and the integrated kalman filters as shown in Table 6.

Table 6. Computational Costs of the Filters and Nodes

Filters and Nodes	**CPU Usage (%)**	**Memory Usage (%)**
Complementary	19.80	2.84
Madgwick	28.10	3.55
Kalman IMU	24.30	3.58
Kalman Odom-Gyro	27.20	3.59

Kalman Odom-Gyro-Magne	27.30	3.59
Integrated Kalman	25.00	3.57
IMU Node	12.80	5.27
Odom Node	6.70	4.75

CHAPTER 5: LASER DISTANCE SENSOR

In this chapter, the usage and implementation of the laser distance sensor is mentioned. The laser distance sensor measures the distance between target and itself through light waves from a laser (Shawn,2020). It can be used in simultaneous localization and mapping (SLAM) (Ocando et al.,2017). Using the laser distance, the map of the environment can be created in 2D and 3D.

In this thesis, the laser distance sensor is used to move the autonomous vehicle in straight road. The autonomous vehicle is tested in a field as shown in Figure 72. In the field, there are four trees, and the autonomous vehicle is standing between two trees at start.

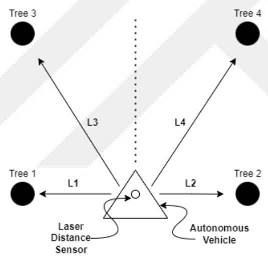


Figure 72. The laser sensor tested field

The location of the trees and the initial location of the autonomous are known. Firstly, the developed algorithm checks the possible location of the trees using threshold distance value and possible angle. For example, the location of tree 1 is at 270 degrees according to the autonomous car and the possible distance is about 40cm between the autonomous car and tree 1. If the distance is above the threshold and angle is around 270 degrees, then there is first tree in this location and it is defined as L1. In order to determine there is tree on the location, the distance that is measured around 270 degrees should also repeat itself between ±10degree with the change of distance as ±5cm. Similarly, the trees 2, 3 and 4 are detected and the distance between the autonomous car and trees are defined as L2, L3, and L4 respectively.

The autonomous vehicle adjusts its speed according to the distance information L1, L2, L3, and L4 to move in straight road. The angular speed of the autonomous vehicle is determined as shown in equation 70, and 71. When the distance obtained from L1 is above the 50cm, the angular speed of the autonomous vehicle is adjusted using obtained distances L3 and L4 as shown in equation 71. The parameter K is 0.01.

$$V_{angular} = K * (L_1 - L_2) \tag{70}$$

$$V_{angular} = K * (L_3 - L_4) \tag{71}$$

5.1. Implementation

The laser distance sensor data is obtained by subscribing the scan topic on ROS platform as mentioned in Chapter 2. The obtained data is an array with size of 360 and each element in the array represents the measured distance with corresponding angle in terms of meter. For example, the first element of the array is the measured distance at angle one in other word the distance measured by in front of the autonomous car.

Since the initial location of the autonomous vehicle and the location of trees are known, the developed algorithm, firstly, checks the location of trees at the known location. When it detects trees, the angle and distance are stored into different variable for each of the trees separately. Then, the vehicle starts to move with the angular speed according to the distance L1 and L2 as shown in equation 70. When the autonomous vehicle starts to move, the developed algorithm starts to check the location of tree 1 at angle smaller than 270 degree and to check the location of tree 2 at angle greater than 180 degrees. For the tree 1, if the angle is smaller than 270 and the measured distance is repeated itself with ±5cm around the previous angle, then new location of tree 1 is taken as new distance and angle. Similarly, the distance and angle values for trees 2, 3, and 4 are updated. When the distance between the autonomous vehicle and tree 1 or tree 2, the angular speed of the autonomous vehicle is calculated using the distance information obtained from L3 and L4.

In order to move autonomous vehicle in straight road, the algorithm is implemented on autonomous vehicle and the vehicle is completed its movement on daylight successfully.

CHAPTER 6: CONCLUSION

In this thesis, the navigation problem is examined on an autonomous agricultural vehicle using different types of sensors such as the odometry, inertial measurement unit (IMU), magnetometer and Laser distance sensor. The performance of the sensors are investigated to determine the orientation of the autonomous agricultural vehicle and various filters such as the complementary, Madgwick, Kalman and the integrated Kalman are applied to the IMU, magnetometer and the wheel odometry data to obtain accurate orientation information. Using the calculated orientation from the sensors and filters, the autonomous vehicle moves on a straight path and stops when it detects the stop sign at the end of the path, using a camera. Furthermore, a laser distance sensor is used to move the autonomous agricultural vehicle on the lanes between the trees in an orchard.

The complementary, Madgwick, Kalman, and integrated Kalman filters are applied on the IMU, magnetometer and the wheel odometry data to get rid of the noises due to the gyroscope drift, and magnetic disturbances. The filters are developed and implemented using a python script developed on the ROS platform. Three cases are investigated to compare the performance of the filter for the orientation estimation. The first case is when the autonomous vehicle is stationary where the performance of the filters are compared in terms of the absolute mean error. All filters have a very small absolute mean error around 0.1 degree. The second case is when the vehicle is rotated 20 degrees and 50 degrees without any magnetic disturbance. The absolute mean errors that are calculated for the complementary filter, Madgwick filter, and Kalman IMU are similar and they have relatively less absolute mean error than those of the other filters. The third case is to measure the performance of the filters, when there is a magnetic disturbance around the vehicle. The mean error is the lowest for the Kalman Odom-Gyro, Kalman Odom-Gyro-Magne, and integrated kalman filter with around 1 degree when there is magnetic disturbance around the vehicle. Other filters are directly affected by the magnetic disturbance and the absolute mean error for yaw angle that is calculated from the complementary filter, Madgwick filter, and Kalman IMU, is around 7 degrees. The results show that all the filters have similar results when the vehicle is stationary and while the vehicle is rotating. On the other hand, magnetic disturbance error cannot be eliminated with the complementary filter, Madgwick filter and Kalman IMU.

Moreover, the computational costs of the filters are compared in terms of CPU usage and memory usage. The computational cost of the complementary filter is the lowest compared to other filters. Other filters are similar computational costs except Kalman Odom-Gyro, Kalman Odom-Gyro-Magne. Since these filters are required the wheel odometry information, the computational costs of the filters are obtained as higher than Kalman Odom-Gyro, Madgwick filter and integrated Kalman filter.

The computational cost of the filters are summarized in Table 6. In order to evaluate the computational cost of the filters, the filter_cc, is mentioned in Chapter 2, is executed separately for each of the filters and the computational cost of each filter is obtained using ROS UI in terms of CPU usage, and memory usage. In the Table 6, we can see that, the complementary filter has lowest CPU and memory usage compared to the other filter. On the other hand, the CPU and memory usages of the Madgwick, Kalman IMU, Kalman Odom-Gyro, Kalman Odom-Gyro-Magne, and integrated Kalman are similar. In addition, the Kalman Odom-Gyro and Kalman Odom-Gyro-Magne are used the wheel odometry information in order to estimate orientation of the autonomous agricultural vehicle. Since the wheel odometry information is required for these filters, the Odom node is executed, and the computational costs of these filters are actually higher than Madgwick and integrated Kalman filter.

Furthermore, a laser distance sensor is used on the autonomous agricultural vehicle to obtain the orientation of the vehicle in the field. The field model consists of four trees and the laser distance sensor is used to detect the distance between the autonomous agricultural vehicle and the trees. Taking the trees as a reference, the orientation of the autonomous agricultural vehicle was determined. Using only the orientation information calculated from the laser distance sensor, the autonomous agricultural vehicle is moved on a straight path between the trees successfully.

However, there is one disadvantage of this method where the data obtained from the laser distance sensor is not reliable under bright sunlight which interferes with the laser signal and causes errors in the orientation data. A better solution could have been obtained by using a radar distance sensor coupled with a laser distance sensor whose outputs could be fused with a Kalman filter to obtain an accurate orientation information even under bright sunlight conditions.

Finally, an algorithm is developed on the autonomous agricultural vehicle to move the autonomous agricultural vehicle on a straight path between the trees in an orchard. In this algorithm, the yaw angles calculated from the odometry, gyroscope and magnetometer data are applied to various filters such as the complementary filter, Madgwick filter, Kalman filter and integrated Kalman filter to estimate the orientation of the autonomous agricultural vehicle and the performances of these filters are also tested. The autonomous agricultural vehicle has completed its movement successfully using the yaw angles calculated from Kalman Odom-Gyro, Kalman Odom-Gyro-Magne, and integrated Kalman filter when there is magnetic disturbance on the path of the autonomous agricultural vehicle. On the other hand, the autonomous agricultural vehicle cannot move on the straight road using the yaw angles that are calculated from other filters and magnetometer when there is magnetic disturbance on the path. In addition, the autonomous agricultural vehicle can complete its movement successfully, using yaw angles that are calculated from the odometry, gyroscope, magnetometer data which are applied on all the filters when there is no magnetic disturbance on the path.

In summary, different types of filters are implemented to estimate the orientation of the autonomous agricultural vehicle. Furthermore, an alternative algorithm is developed using the laser distance sensor in order to move the autonomous agricultural vehicle on the lanes between the trees in the orchard. However, it is observed that the information obtained from the laser distance sensor can be inaccurate under bright sunshine. All of the filters, implemented on the autonomous agricultural vehicle, can accurately estimate the orientation of the autonomous agricultural vehicle while the vehicle is stationary and rotating. However, the orientation estimated by some of these filters are not accurate when there is a magnetic disturbance in the environment. The error due to the magnetic disturbance can be eliminated by using the Kalman Odom- Gyro, the Kalman Odom-Gyro-Magne, and the integrated Kalman filters. Among all of the filters, the integrated Kalman filter algorithm is the best solution for the correct orientation estimation of the autonomous agricultural vehicle, since it has low computational cost, and it only requires the IMU information to estimate the orientation. Since the orientation of the vehicle is determined by the integrated Kalman filter correctly, the vehicle only requires the IMU sensor. Therefore, the best sensor is IMU sensor for the correct orientation estimation of the vehicle among all the sensors.

REFERENCES

Aswinth, R. (2019). *Distance and angle measurement for mobile robots using Arduino and LM393 Speed Sensor* [Online]. Available at: https://circuitdigest.com/microcontroller-projects/speed-distance-and-angle-measurement. (Accessed : 14 January 2022).

Biber, P., Weiss U., Dorna, M., and Albert, A. (2012) *Navigation system of the autonomous agricultural robot Bonirob,* Workshop on Agricultural Robotics: Enabling Safe, Efficient, and Affordable Robots for Food Production, pp. 1–7.

David, P. (2020). *Robotis e-Manual* [Online]. Available at: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/. (Accessed : 12 January 2022).

El-Diasty, M. (2014) *An accurate heading solution using MEMS-based gyroscope and magnetometer integrated system (preliminary results)*, ISPRS annals of the photogrammetry, remote sensing and spatial information sciences, Vol. 2, pp. 75-78.

Fedorov, D.S., Ivoilov, A.Y., Zhmud, V.A., and Trubin, V.G. (2015) *Using of measuring system MPU6050 for the determination of the angular velocities and linear accelerations*, Automatics & Software Enginery, Vol. 12, pp. 16–80.

Gui, P., Tang, L., and Mukhopadhyay, S. (2015) *MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion*, IEEE 10th conference on Industrial Electronics and Applications, pp. 2004–2009.

Hague, T., Marchant, J.A., and Tillett, N.D. (2000) *Ground based sensing systems for autonomous agricultural vehicles*, Computers and Electronics in Agriculture, Vol. 25, pp. 11-28.

Han, J.H., Park, C.H., Kwon, J.H., Lee, J., Kim, T.S., and Jang, Y.Y. (2020) *Performance evaluation of autonomous driving control algorithm for a crawler-type agricultural vehicle based on low-cost multi-sensor fusion positioning*, Applied Sciences, Vol. 10, pp. 4667-4673.

Khan, T. (2019). *Computer vision : detecting objects using haar cascade classifier* [Online]. Available at: https://towardsdatascience.com/computer-vision-detecting-objects-using-haar-cascade-classifier-4585472829a9. (Accessed : 12 January 2022).

Kok, M., and Schön, T.B. (2016) *Magnetometer calibration using inertial sensors*, IEEE Sensors Journal, Vol. 16, pp. 5679-5689.

Madgwick, S.O.H. (2010) *An efficient orientation filter for inertial and inertial/-magnetic sensor arrays*, Citado, Vol. 5, pp. 9– 19.

Madgwick, S.O.H, Harrison, A., and Vaidyanathan, A. (2011) *Estimation of IMU and MARG orientation using a gradient descent algorithm*, IEEE international conference on rehabilitation robotics, pp. 1-7.

Mario, G. (2019). *Madgwick orientation filter* [Online]. Available at: https://ahrs.readthedocs.io/en/latest/filters/madgwick.html. (Accessed : 15 January 2022).

Ocando, M.G., Certad, N., Alvarado, S., and Terrones, Á. (2017) *Autonomous 2D SLAM and 3D mapping of an environment using a single 2D LIDAR and ROS*, Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR), pp. 1–6.

Pagnutti, M.A., Ryan, R.E., Cazenavette, G.J.V., Gold, M.J., Leggett, R.H.E., and Pagnutti, J.F. (2017) *Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes*, Journal of Electronic Imaging, Vol. 26, pp. 1-13.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A.Y. (2009) *ROS: an open-source Robot Operating System*, ICRA workshop on open source software, Vol. 3, pp. 1-6.

Renaudin, V., Afzal, M.H., and Lachapelle, G. (2010) *Complete triaxis magnetometer calibration in the magnetic domain*, Journal of sensors, Vol. 2010, pp. 1-10.

Shawn, A. (2020). *Types of Distance Sensors and How to Select One* [Online]. Available at: https://www.seeedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/. (Accessed : 15 January 2022).

Sloth, L. (2017). *TKJ Electronics A practical approach to Kalman filter and how to implement it* [Online]. Available at: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/. (Accessed : 11 January 2022).

Sowjanya, K.D., Sindhu, R., Parijatham, M., Srikanth, K., and Bharhav, P. (2017) *Multipurpose autonomous agricultural robot*, 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), Vol. 2, pp. 696–699.

Treffers, C., and Wietmarschen, L.V. (2016) *Position and orientation de- termination of a probe with use of the IMU MPU9250 and a ATmega328 microcontroller*, Master Thesis, Delft University of Technology.

Welch, G., and Bishop, G. (2001) *An Introduction to the Kalman filter*, Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques, pp. 12–17.

Yoonseok, P., Hancheol C., Leon, J., and Darby, L. (2017). *ROS Robot Programming (English).* Available at: http://wiki.ros.org/Books/ROS_Robot_Programming_English. (Accessed 15 January 2022)

APPENDICES

Appendix A – Raspberry Pi and LM393 Connection

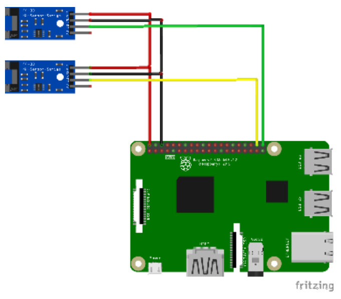


Figure 73. Raspberry Pi and LM393 connection

Appendix B – Raspberry Pi and MPU6050 Connection

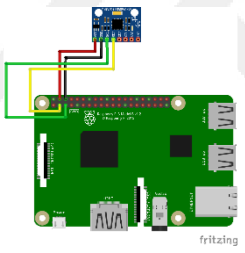


Figure 74. Raspberry Pi and MPU6050 connection

Appendix C – Raspberry Pi and QMC5883L Connection

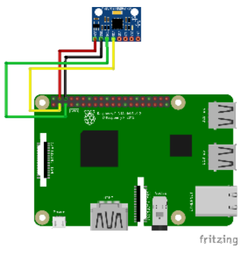


Figure 75. Raspberry Pi and QMC5883L connection

Appendix D – IMU Messages

```
float32 delta_t
float32 time_stamp
float32 axr
float32 ayr
float32 azr
float32 mxr
float32 myr
float32 mzr
float32 gxr
float32 gyr
float32 gzr
float32 ax
float32 ay
float32 az
float32 mx
float32 my
float32 mz
float32 gx
float32 gy
float32 gz
float32 aroll
float32 apitch
float32 groll
float32 gpitch
float32 gyaw
float32 myaw
float32 cmyaw
```

Appendix E – Odometry Messages

```
float32 DL
float32 DR
float32 VL
float32 VR
float32 yaw_odom
```

Appendix F – Filter Messages

```
float32 delta_t
float32 time_stamp
float32 yaw_odometry
float32 yaw_gyroscope
float32 yaw_magnetometer
float32 yaw_kalman
float32 yaw_kalman_odom_gyro
float32 yaw_kalman_odom_gyro_magne
float32 yaw_kalman_integration
float32 yaw_madgwick
float32 yaw_complementary
float32 yaw_ekf
```

Appendix G – Complementary Filter Class

```
class Complimentary():

      def __init__(self, gain = 0.9):

            self.Roll = 0
            self.Pitch = 0
            self.Yaw = 0
            self.Gain = Gain

      def setRoll(self, Roll):
            self.Roll = Roll

      def setPitch(self, Pitch):
            self.Pitch = Pitch

      def setYaw(self, Yaw):
            self.Yaw = Yaw

      def setGain(self, Gain):
            self.Gain = Gain

      def update_roll_pitch_and_yaw_angles(self,
Measured_Roll_Acc, Measured_Pitch_Acc, Measured_Yaw_Magne, gx,
gy, gz, dt):
            self.Roll =
self.complementary_filter_update(self.Roll, Measured_Roll_Acc,
gx, dt)
```

```
            self.Pitch =
self.complementary_filter_update(self.Pitch,
Measured_Pitch_Acc, gy, dt)

            self.Yaw =
self.complementary_filter_update(self.Yaw, Measured_Yaw_Magne,
gz, dt)


      def complementary_filter_update(self, angle_prev,
MeasuredData, InputData, dt):


            ComplementaryResponse = (angle_prev +
InputData*dt)*(self.Gain)  + (1-self.Gain)*(MeasuredData)

            return ComplementaryResponse
```

Appendix H – Kalman Filter Class

```
import numpy as np


class Kalman:


      def __init__(self):


            self.Xk = np.vstack((0.0, 0.0)) # State Matrix

            self.yaw = 0 # Initial Yaw

            self.Pk = np.array([0.1, 0.1])*np.identity(2) # Pk

            self.Q11 = 0.002 # Q11

            self.Q22 = 0.005 # Q22

            self.R = 0.03 # R


      def update_yaw_angle(self, input_vector_variable, dt,
measuredYaw):


            self.yaw, self.Xk, self.Pk = self.update(self.Xk, \
                      measuredYaw, self.Pk, \

                      self.Q11, self.Q22, \

                      self.R, input_vector_variable, dt)



    def update(self, Xk_prev, measurement, Pk_prev, error,
driftError, MeasurementUncertainty, input_vector_variable
,dt):



           StateTransitionMatrix = np.array([[1,-dt],[0,1]])#A

           ControlMatrix =
np.vstack((input_vector_variable,0.0))#B

           ProcessNoise = dt*(np.array([error,
driftError])*np.identity(2))#Q

           ObservationMatrix = np.array([1.0, 0.0])#H


           # Prediction

           DynamicModel = np.matmul(StateTransitionMatrix,
Xk_prev) + dt*ControlMatrix

           PredictorCovariance =
np.matmul(np.matmul(StateTransitionMatrix, Pk_prev),
(StateTransitionMatrix.T)) + ProcessNoise



           # Measurement

           Innovation = measurement -
np.matmul(ObservationMatrix, DynamicModel)

           InnovationCovariance =
np.matmul(np.matmul(ObservationMatrix, PredictorCovariance),
ObservationMatrix.T) + MeasurementUncertainty

           KalmanGain = np.matmul(PredictorCovariance,
np.vstack((1.0, 0.0))) / InnovationCovariance



           # Update
```

```
            StateUpdate = DynamicModel + KalmanGain*Innovation

            CovarianceUpdate = np.matmul( np.identity(2) - 
np.matmul(KalmanGain, np.array([1.0, 0.0]).reshape((1,2))), 
PredictorCovariance)


            return StateUpdate[0,0], StateUpdate, 
CovarianceUpdate


      @property

      def yaw(self):

            return self._yaw


      @yaw.setter

      def yaw(self, yaw):

            self._yaw = yaw

            self.currentYawState[0,0] = yaw
```