Applied Electrical and Electronic Engineering: Construction Project 3 & 4 Lab Report

Project 3 & 4 Lab Report Cover Page

Name: Chunyu Wang	**Student ID:** 20123739
Department: Electrical and Electronic Engineering	**Module Name:** Applied Electronical and Electronic Engineering: Construction Project
Module Code: EEEE1039 UNNC	**Module Convenors:** Dr. Chunyang Gu Dr. Jing Wang
Lab Report Coverage: Lab 3 & Lab 4 session	**Submission Date:** 30 Dec.

ABSTRACT

This report has covered what has been achieved in lab session 3 and 4, the methods adopted to fulfill the given challenges and the theories involved. The introduction section of this report briefly explains that the primary objective for session 3 is to establish wireless control of the vehicle and the measurement of angles and accelerations with MPU-6050 module and the main goal for session 4 is to achieve the line following of the vehicle first with two TCRT5000 optical sensors and bang-bang controller, then with SF-8CHIDTS and PID controller for advanced control. The methods section explains the usage for the sensors and the steps to finish the tasks. In the results section of the report, measurements were recorded and explained how they were measured. In the discussion section, analysis has been made with regard to the results and the methods adopted. The conclusion parts summarizes the achievements of these two lab sessions and the key findings.

INTRODUCTION

This lab report contains the contents of two lab sessions and the introduction part of this report would explain what attempts have been made and what goals have been achieved in these two lab weeks respectively.

The first lab session was done in lab week 3 and the second lab session was done in lab week 4. For the first lab session, the primary goal was to achieve the wireless communication between the vehicle and the HMI control pad, with the integration of MPU-6050 module to measure the angle and the acceleration. The MPU-6050 module contains a 3-axis gyroscope and accelerometer which can be utilized to acquire the rotation, angle and the acceleration of an object in three axes. This functionality was later applied to enable the vehicle to sense the change in its angle with the horizontal line and make responses. In addition, the module nRF24L01+ was applied to establish the serial communication between the vehicle and the control pad to achieve the remote control of the vehicle. In the first week, unidirectional wireless communication and bidirectional wireless communication were established to fulfil different tasks. In the demo verification part of lab 3, unidirectional communication was exercised with the Nano connected with the MPU-6050 being the transmitter and the Nano connected with the LCD (control pad) being the receiver. The transmitter end could obtain the angle information from MPU and transmitted to the other Nano board and displayed on an LCD. In the final challenges of the lab 3, bi-directional communication was built up to remote control the vehicle. The information of the angle and the distance travelled of the vehicle was transmitted to the control pad and displayed on an LCD. The analogue signals of the joystick were transmitted from the control pad to the vehicle so that the vehicle can process to achieve the control. For the second final task in this lab session, MPU-6050 sensor was applied to detect the angle the vehicle makes with the horizontal ground and therefore enable the vehicle to go up a ramp and pause for 2 seconds and rotate for 360 degrees before going down the ramp. The angle and the distance were transmitted from the vehicle to the control pad as information feedback.

For lab session 4 in the following lab week, the main objective was to achieve the line following of the vehicle first with two optical sensors, TCRT5000, and then secondly, achieve the advanced line following with SF-8CHIDTS, an array of 8 optical sensors. There

were two control algorithms adapted, Bang-bang control and PID (proportional, integral and derivative) control respectively. Bang-bang controllers turn off or on an operation when the desired set values have been reached, also known as hysteresis controllers, because there is a lag in response [1]. For example, to controll the room temperature with a thermostat with a bang-bang controller, the thermostat will drive the plant, which in this case, is the heating system, to increase the temperature if the temperature falls below a certain desired value. Then when the temperature rises up to the set point, the heater will be turned off. However, the control is always hysteretic. In the line following of the vehicle, bang-bang controller enables the vehicle to turn left if the left side sensor reads the black line and vice versa for right side sensor. Therefore, the black line always stays in between the two sensors.
For the advanced line following, PID controller was adopted. PID control provides a continuous feedback correction mechanism which generates the output that can reduce the error to a designed point. This controlling method is widely adopted in industries and the control is affected by three terms, proportional, integral and derivative [2]. In this experiment, PID controller generates the regulated output used to control the motor speed to keep the SF-8CHIDTS sensor mid-point always align with the black line.

In these lab sessions, all the challenges were made attempt and the relevant methods will be discussed in the method section of this report.

METHODS

In lab session 3, the first task was to utilize MPU-6050 module to obtain the acceleration, angle of the inclination of the tilts, and the rotational angle. The MPU-6050 module would later be used as a tilt angle sensor and installed onto the vehicle to give feedback of the angle to the remote-control pad and also detects the angle between the ramp and the horizontal ground. MPU-6050 contains a gyroscope and an accelerometer integrated on a chip. The gyroscope measures the rotational velocity by applying MEMS (the micro-electro-mechanical) technology and Coriolis effect and the accelerometer measures the gravitational Acceleration [3]. With a gyroscope, the rotational angular can be obtained by taking the integral of the velocity. For accelerometer the angle with the horizontal level can be obtained by applying the laws of trigonometry. Both the accelerometer and the gyroscope have three axis and can have the measurement in three dimensions. The module itself has an I2C interface with the Nano board and the data is transferred across SDA and SCL bus. In this case, the MPU-6050 acts as a slave and the Nano board is the master. The raw data the sensor obtained cannot be used directly and has to be processed. Therefore, library was used to manipulate these raw data in order to generate accelerations and angles in each direction of axis. Once the accelerations in three axes were obtained, the angle can be calculated with trigonometrical formula 2.1:

$$\theta = tan^{-1}\frac{\sqrt{A_X{}^2+A_Y{}^2}}{A_Z} \tag{2.1}$$

Where θ stands for the angle with the Z axis; A_X , A_y , A_z stand for the acceleration projection in X, Y, Z axis respectively.

Similar approach can be adopted to obtain the angle in X and Y axis. For the “acceleration vs tilt angle plot” task and the later inclination measurement of a ramp, only one direction of angle and acceleration was needed and therefore, the acceleration and the angle in X axis were adopted for the quantitative measurement. For the “acceleration vs tilt angle plot” task, a protractor was used to compare with the angle acquired from MPU-6050 to verify the accuracy of this electronic component. The plane of the protractor was placed perpendicular to the ground and parallel to the Z axis and X axis of the MPU-6050. Then progressively increase the angle between the ground and the X axis of the MPU from 0° to 90° with each increment of 10°. Then the angles and the accelerations read from the MPU Serial Monitor were recorded and discussed in the results section of this report. The angle readings from the protractor and the predicted acceleration at each angle were used to compare with the ones generated by MPU-6050. More detailed results are discussed in the results and discussion section.

After obtaining the angle and the acceleration from the MPU, attempts to establishing the wireless communication between two Nano boards were made. The wireless serial communication was enabled by nRF24L01 module. It has a Serial Peripheral Interface and allows a maximum of 10 Mbps data transfer rate [4]. When connected to Arduino Nano, it should be powered by 3.3 V port because its operating power input is 1.9 V to 3.6 V [4]. The Nano board acts as a master while the module acts as a slave transferring data with another nRF24L01 module. nRF24L01 allows frequencies in 2.4 GHz ISM band and each channel takes a bandwidth of less than 1 MHz, which means there can be 128 channels with adequate frequency spacing to avoid overlapping channels [5]. The channel selected by group 14 was 101 according to the requirement.

For the demo verification task in lab 3, a unidirectional communication was built up. One end of the Nano board connects with an MPU-6050 module to obtain the angle information and a nRF24L01 acting as a transmitter to send the information of the angles to the other Nano board. The other end of the Nano board was connected with an LCD and an nRF24L01 acting as a receiver. The information of the angle sent from the MPU-6050 end was received and displayed onto the LCD display. Noticeably, at the transmission end, the Nano board was connected with two slaves, with the one being the MPU-6050 module and the other one being the nRF24L01. Each one was assigned with a different slave address. In the setup function, the function "openWritingPipe()" creates a data transmission pipe and "stopListening()" enables the nRF24L01 to be in transmission mode instead of receiving data. In the main loop, the angle and the acceleration are constantly read from MPU-6050 and were constantly transmitted by function "write()". In the receiving end, however, function "openReadingPipe(pipe, address)" activates the pipe for receiving data in the setup and function "read()" would read the angles and the acceleration from the MPU-6050 in the loop function.

Since there were multiple information to be transmitted from one Arduino Nano to the other, such as the angles and the accelerations, the information was stored in a float array before being sent. Then in the receiving end, a float array was there to receive the data and then each single element was extracted from the array to use. The working principle of nRF24L01+ is that the transmitter transmits the data and when the receiver receives and waits for around 130 μs, an acknowledge signal will be sent to the transmitter indicating the data has been received. Then an interrupt signal will prompt the transmitter that new data is ready to be sent [6]. If the transmitter fails to receive the acknowledgement signal from the receiver within the Auto-Retransmit-Delay (ARD), then the data will be retransmitted in case of data loss [6]. For the remote control of the vehicle, a bidirectional communication is needed. The Nano board on the vehicle and the Nano board on the control pad both act as transceiver. Both the "openWritingPipe()" function and "openReadingPipe(pipe, address)" function were used in each one's setup function. The address for control pad is stored as string "1Node" and the address for the vehicle is “2Node”. In the main loop of both Arduino Nano, there are two external functions alternately being executed. One external function contains the code “stopListening()” and it can enable the nRF24L01+ to enter transmitting mode and the other function contains the code “startListening()” and it can enable the module to be in receiving mode. Two external functions alternate with each other so that each Arduino is constantly transmitting and receiving data. The diagram below shows the how bidirectional communication operates inside the Arduino program. Complete code for the bidirectional communication can be found in the appendix section.

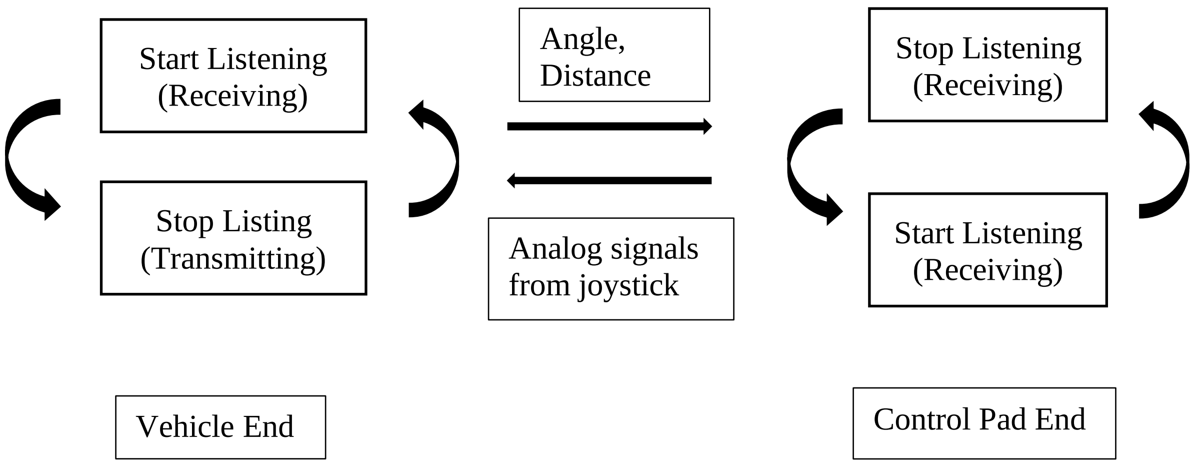
Start Listening (Receiving)
Stop Listing (Transmitting)
Angle, Distance
Analog signals from joystick
Stop Listening (Receiving)
Start Listening (Receiving)
Vehicle End
Control Pad End

Diagram 2.1: The Working Principles of the Bidirectional Communication

The distance the vehicle has travelled was derived from the encoder value from the encoder motor. Since the vehicle may turn left or turn right, it is not reasonable to use the encoder value from only one motor. Determined by the the vehicle firmware, only motor 1 and 2 can give right feedback of the encoder value and they were arranged on the two rear wheels.

The encoder 1 value and the encoder 2 value were added together and divided by two to obtain the average. Noticeably, the ender value is stored as an unsigned long integer and if two encoder value are added together, the sum will exceed its maximum number that can be stored. Therefore, both encoder value 1 and 2 were divided by 10 before adding up and the final result were multiplied by 10 to restore to the correct sum value and was stored in a long unsigned integer variable called "encoderStat". The mathematical formula to obtain the distance were shown by equation 2.2, 2.3 and 2.4 below:

$$Distance = \frac{\pi \times Wheel\ Diameter \times\ Encoder\ Value\ Increment \times \cos{(\theta)}}{Gear\ Ratio \times\ Encoder\ Ratio} \tag{2.2}$$

In this case, the gear ratio is 1:21.296, the encoder ratio is 1:11 and the wheel diameter is 9.53 cm. θ is the angle between the MPU-6050 X-axis and the horizontal level. The increment of the encoder value at each iteration of the loop was calculated in the method below:

$$Encoder\ Value\ Increment = encoderStat - Initial\ Encoder\ Value \tag{2.3}$$

Where the initial encoder value is 2147483648. Therefore, the distance was calculated by substituting equation 2.2 and 2.3:

$$Distance = \frac{0.299 \times\ (encoderStat - 2147483648) \times \cos{(\theta)}}{234.256} \tag{2.4}$$

At the control pad end, the joystick has two analog outputs in two axes respectively. If the joystick is toggled from left to right, the analog output in Y-axis will change from 1023 to 0 since in Arduino, there is a 10-bit analog to digital converter. If the joystick is toggled from up to down, the analog output in X-axis will change from 0 to 1023. Then the X-axis analog output was mapped on-one-on down to the scale of from -80 to 80 as the speed for going forward or going backward, by "map()" function in Arduino. Similarly, Y-axis output was mapped to from -60 to 60 as the speed for turning left or turning right. Therefore, the remote control was established by the methods mentioned above. For the automated path tracking, the complete path was divided into 13 steps for the vehicle to complete. If in step 1 or 4, the MPU detects a change in angle, the vehicle will enter a while loop to finish a series of tasks such as going up a ramp, pausing for two seconds, spinning 360 degree and going down the ramp.

In lab session 4, first bang-bang control with two optical sensors TCRT5000 were employed to finish the basic line following task. TCRT5000 is reflective optical sensor with an infrared light emitter and phototransistor enclosed in a leaded package to filter out the visible light [7]. Infrared light is emitted out from the emitter and bounced back from the object with some of the light being absorbed. The phototransistor detects the intensity of the reflective light and generates an analogue output. In the entire optical sensor module, there are four pins, two LED indicators and a potentiometer. The four pins are VCC pin, ground pin, analogue output pin and digital output pin respectively. One LED is power indicator and the other is logic high indicator. The analogue output pin of the module provides a continuous indication of the intensity of the reflective light. The adjustable potentiometer will determine the threshold for the digital output of the module to be set as logical high. If the analogue output exceeds the threshold, the digital pin will become from logical low to high and the LED indicator will be on. Therefore, this module can be applied to differentiate different background colour and used to control the line following vehicle. There are multiple factors that can affect the output of the module such as distance to ground, ambient light, background colour and material. These factors will be discussed in the results and discussion section of this report. The varying output of the sensor with different distance to reflective object, different colours and different intensity of ambient light are recorded in the results section. The control for the basic line following task employed bang-bang control theory. Two TCRT5000s are installed at the front of the vehicle with a 3D printed frame as a rack. The TCRT5000s are 6.00 cm wide and 2.00 cm above the ground. If the left side of the TCRT5000 detects the black line, which means the vehicle has deviated to the right, turn-left command will make the vehicle to turn left until the TCRT5000 reads white colour back gain. If the right side of the TCRT5000 detects the black line, the vehicle will keep turning right until the right-side sensor detects white colour again. The algorithm for this task was RT-control. All the operations were included in the interrupt function and had a frequency of 200 Hz.

For advanced line following, line follower module SF-8CHIDTS was applied. It contains an array of eight TCRT5000 sensors and has an I2C interface. When the module was first installed in the front of the vehicle, its height to ground is 2.60 cm and its width is 9.00 cm and the distance between the neighbouring two sensors is 1.28571 cm. The black tape on the track has a width of 1.85 cm. After the sensor outputs were carefully examined, an algorithm of calibration was defined. Some of the marginal values were cut off by the “constrain()” function in Arduino due to the fluctuation of the analogue value. Then the constrained values were mapped to a scale of from 0 to 255. If the sensor reads background white, then the output would be 0 and it will output 255 if it reads black. These measurements were done in the lab with lab ambience. It may vary with different environment. Then a weighted average algorithm was designed to obtain the distance of black line to the sensor array mid-point. The error for the PID control is based on that distance. Each sensor has different distance to the mid-point and therefore was given different weight for calculating the weighted average. The weight is based on the distance to the mid-point with the unit of milli-meter. Table 2.1 shows how each sensor is invested with weight.

Table 2.1: Weight for each sensor for weighted average calculation

Sensor Index	0	1	2	3	4	5	6	7
Weight (mm)	-45	-32.1	-19.2858	-6.4283	6.4283	19.2858	32.1	45

The mathematical formula to calculate the distance to mid-point based on the weighted average algorithm is shown by equation 2.5:

$$Distance = \frac{\sum_{n=0}^{8} Sensor\ Value \times weight}{\sum_{n=0}^{8} Sensor\ Value} \tag{2.5}$$

The distance was used to generate error. In this case, the error is the distance to mid-point. Then PID controller was applied with three terms, proportional, integral and derivative term to generate an output based on the error and the output was used to control the speed of the left-side and the right-side motors. The mathematical expression can be seen below in equation 2.6:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)d\,t + K_d \frac{de(t)}{dt} \tag{2.6}$$

Where, K_p, K_i and K_d are coefficients for three terms respectively. $e(t)$ is the error function with respect to time and $u(t)$ is the PID output with time. t is the time elapsed.

For PID tuning, proportional term K_p was first adjusted to constantly give a proportional negative feedback to the motor speed. When the K_p is tuned to 8, the vehicle can make the most curved turn without derailing. However, it was oscillating left to right going the straight line. Then the derivative term K_d was adjusted to 3 to reduce the overshooting caused by the proportional term. The integral term K_i not adjusted because the residual error can be neglected in this experiment. Then the vehicle can finish the task with a base speed of 25 percent of the power. Saturation point for the motor speed was set to prevent the case when the output speed exceeds the motor maximum power output. Complete code for the PID control calibration details and weighted average can be found in the appendix of the report.

RESULTS

In lab session 3, the first quantitative measurement is to generate the acceleration versus the tilt angle plot. MPU-6050 was to be installed onto the vehicle to give feedback of the angle and therefore the digital angle and acceleration it obtained with the designed algorithm was used to compare with the angle read from the protractor to determine if the MPU-6050 can give an accurate measurement of the angle. The measurements were chosen from 0 degree to 90 degree with an increment of 10 degree at a time. The predicted acceleration Acc_{Pre} was calculated with the angle An_{Pro} from the protractor by the equation 3.1:

$$Acc_{Pre} = g \times \sin(An_{Pro}) \tag{3.1}$$

Where g is the gravitational acceleration and is considered as 9.8 m/s2.
The results were recorded in table 3.1 as shown below.

Table 3.1: measurements by MPU with reference to protractor

Measurement	**Angles on protractor** $\boldsymbol{An_{Pro}}$	**Angles by MPU** $\boldsymbol{An_{MPU}}$	**Predicted Acceleration** $\boldsymbol{Acc_{Pre}}$ (m/s^2)	**Acceleration by MPU** $\boldsymbol{Acc_{MPU}}$ (m/s^2)
1	0°	-0.21°	0	0.04
2	10°	9.84°	1.70	1.53
3	20°	19.73°	3.35	3.75
4	30°	29.84°	4.90	5.78

5	40°	38.62°	6.30	6.93
6	50°	47.54°	7.51	8.02
7	60°	56.73°	8.49	8.93
8	70°	66.83°	9.21	9.54
9	80°	73.84°	9.65	9.73
10	90°	82.64°	9.80	9.83

An_{Pro} is the value measured from the protractor and Acc_{MPU}, An_{MPU} are values read from MPU-6050. In the second measurement, for instance, $Acc_{Pre}$was calculated by equation 3.2 as shown below:

$$Acc_{Pre} = 9.8 \times \sin(10^{\circ}) = 1.70\, m/s^2 \tag{3.2}$$

Since, the readings from the MPU module are digital values with a resolution of 0.001, therefore the uncertainty associated with it can be calculated by the equation 3.3 below:

$$Uncertainty\ for\ MPU\ readings = \frac{0.0005}{\sqrt{3}} \tag{3.3}$$

Then the angles measured by MPU were used to compare with the angle on protractor and shown below:

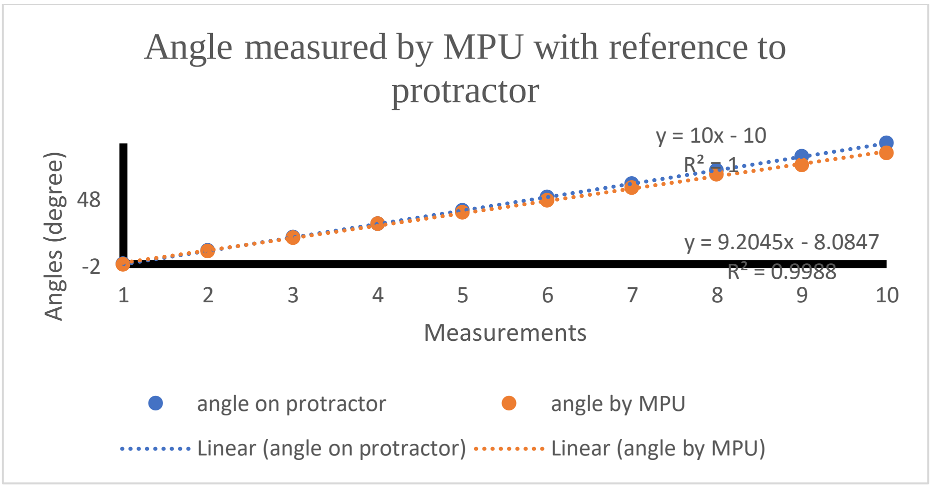

Figure 3.1: Angle measured by MPU-6050 referenced to protractor

As can be seen in figure 3.1, difference between the angle measured by protractor and the angle read from MPU increases as the value of the angle increases. Larger difference will be seen from 80° to 90°. However, the R2 for the MPU angle is 0.9988, close to 1, which proves that the angles have an obvious positive linear trend and the difference between angles from protractor are not considerably large. The code for deriving the angle from the raw data the MPU obtained are shown in the appendix section. Since the difference is not huge when the angle is small, it can be used to measure the angle.

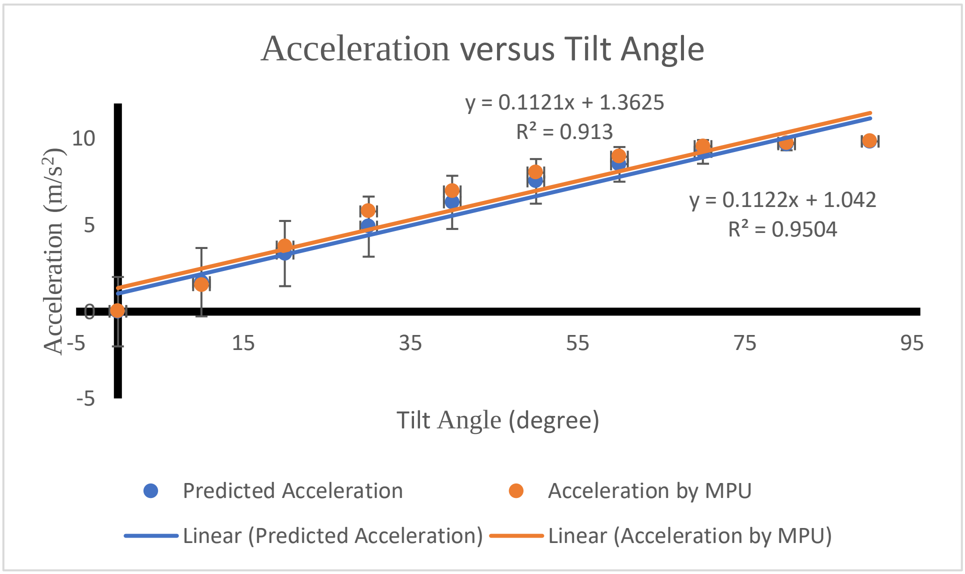

Figure 3.2: Acceleration versus Tilt Angle

Figure 3.2 shows the relationship between the tilt angle and the acceleration both measured by MPU and the protractor. It was found that the curves shown in the figure do not have valid linear form and the R_2 is relatively small, not showing a solid positive linear relationship. Therefore, the sine values of the angles are used to form mathematical relationship with the acceleration because the acceleration is the projection of the gravitational acceleration in MPU's axis.

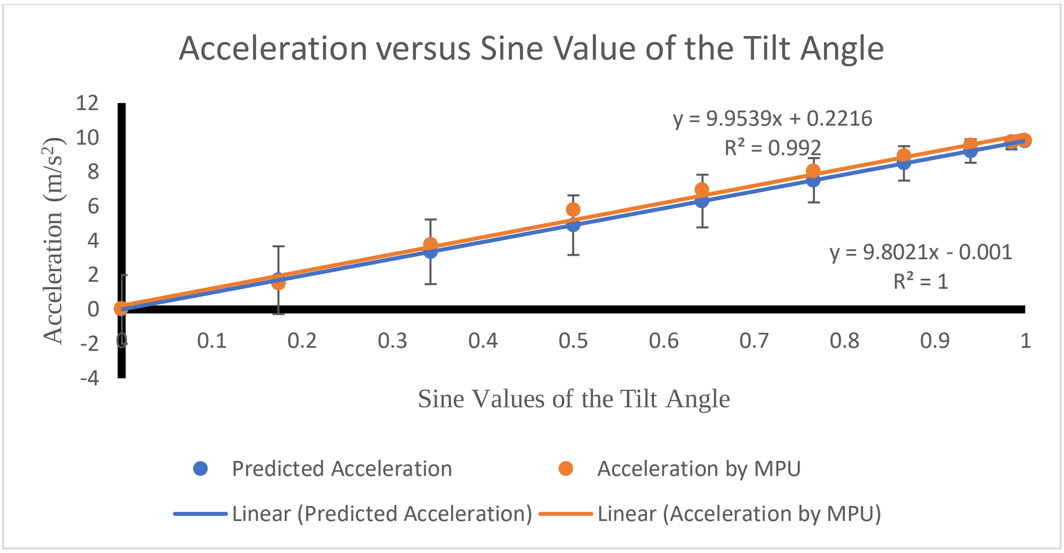

Figure 3.3: Acceleration versus Sine Value of the Tilt Angle

As shown in the figure 3.3, the sine value of the angles and the accelerations have a well-defined linear relationship and the R_2 is close to 1 which consolidates the fact.

At first the angle and acceleration were derived by the algorithm which is to directly read data and process the data to be readable acceleration values, as shown in appendix. The raw data in six axes was processed to obtain the accelerations and angles for the gyroscope and the accelerometer. The algorithm gives a good measurement for angle and acceleration as fore-mentioned, however, when the MPU encounters lateral vibration or when there is an external acceleration applied on the module the value for the angle will fluctuate even when the module is at level state.

The angle fluctuation versus time was plotted in figure 3.4 with the Arduino Serial Plotter and in this case, the time refers to the iteration of the code in the main loop. Its Y axis shows the angle value and the X axis is the iteration.
The MPU-6050 was held level to ground all the time and was given a moderate vibration.

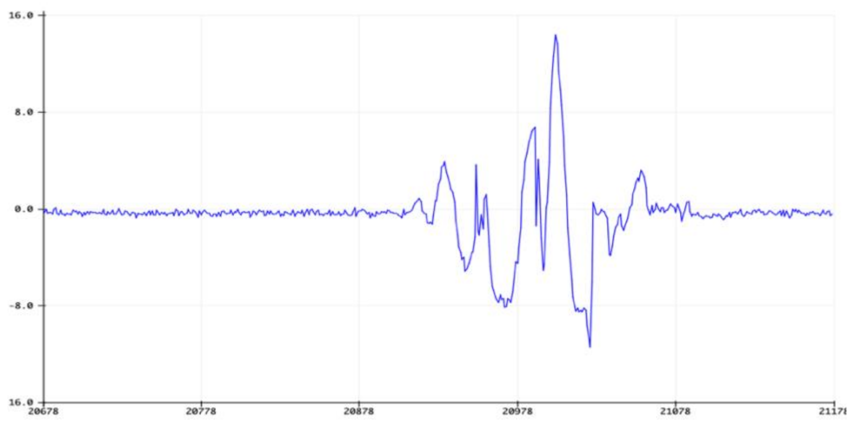

Figure 3.4: Angle fluctuation with slight vibration on MPU-6050

Then another algorithm was adopted. It was modified with the MPU-6050 library example file code. The similar methods were used to make the plot.

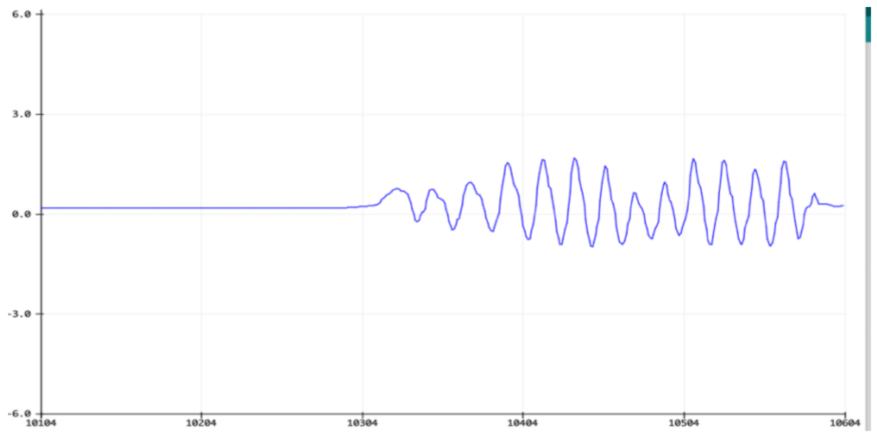

Figure 3.5: Improved Angle fluctuation with slight vibration on MPU-6050

The problem of angle fluctuation with external force was greatly mitigated and the MPU now could be installed onto the vehicle.

In lab 4, factors that affect the output of the TCRT5000 were investigated. Several results were discussed in the results and the discussion section of the report. First the output voltage at the digital pin (D0) and analogue pin (A0), digital output and analogue output were recorded with a distance of 2 cm to the background of different colour and with default laboratory ambient light as shown in table 3.2.

Table 3.2: TCRT5000 Outputs with Different Reflection Surface

Colour	**Analogue Output**	**Digital Output**	**Voltage at A0 Pin (V)**	**Voltage at D0 Pin (V)**
Black	554 - 559	1	2.9067	4.701
White	37-40	0	0.1886	0.1678
Green	38-40	0	0.1889	0.1678
Blue	40-44	0	0.1897	0.1678
Red	40-41	0	0.1713	0.1670

Then the relationship of TCRT5000 digital and analogue output with ambient light intensity was examined and recorded in table 3.3. The distance at this point was still held at 2 cm height. The ambient light intensity was varied by adding additional smart phone flashlight to the background. Noticeably the recording of table 3.3 was referred to the condition when the sensor read black colour background.

Table 3.3: TCRT5000 Outputs with Different Ambient Light Intensity

Light Intensity	**Analog Output**	**Digital Output**

No additional light	670-700	1
Medium additional light	655-656	1
Strong additional light	655-690	1

The factor distance was also investigated by placing TCRT5000 at different height to the ground and then the digital and analogue outputs were recorded in table 3.4.

Table 3.4: TCRT5000 Outputs with Different distance to reflective object

	White reflective background		**Black reflective background**	
Distance (cm)	**Analog output**	**Digital output**	**Analog output**	**Digital output**
1.00	32-36	0	290-328	0
2.00	39-40	0	690-713	1
3.00	52-53	0	678-725	1
5.00	552	1	790-890	1

For advanced line following task, every sensor analogue output derived from SF-8CHIDTS was calibrated and with weighted average, a distance to mid-point was obtained, which was used for PID controller algorithm. Table 3.5 displays the output position with different placement of the vehicle. Negative value indicates that the mid-point is left to the black line, while positive value indicates the mid-point is on the right side of black line. The recording was done in lab condition with lab ambient light.

Table 3.5: Output position with different placement of the vehicle

Measurement	**Displacement (mm)**	**Output distance (mm)**
1	-45.0	-45.00

2	-32.1	-31.87
3	-19.3	-19.21
4	-6.4	-6.37
5	-3	-1.31
6	0	0.21
7	3	1.45
8	6.4	6.40
9	19.3	19.28
10	32.1	32.21
11	45.0	45.01

Then a diagram was plotted with the X axis being the displacement measured by a 0.01 cm resolution ruler and Y axis being the reading of the distance output, as shown in figure 3.6 below.

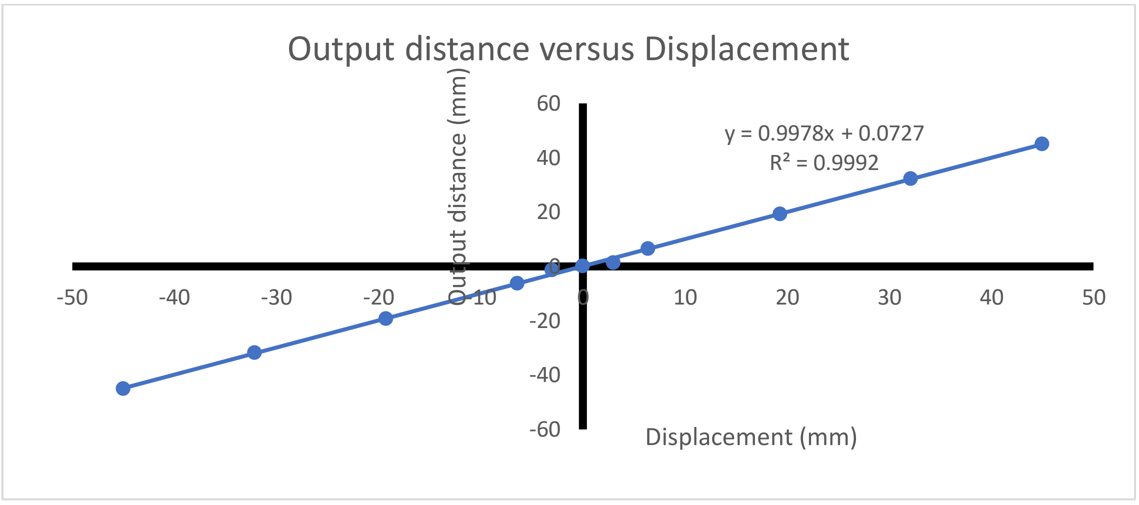

Figure 3.6: Output distance versus Displacement

The output distance by large conform with the displacement measured with the ruler considering with the uncertainties. Therefore, the calibration and the weighted average can be used for vehicle PID controller input.

DISCUSSION

This section will mainly analyse the quantitative results and discuss the problems that have been encountered during the lab session with the methods to fulfil the given lab 3, 4 tasks. In lab 3, as mentioned in the result section, there were two difference types of code with different algorithm to obtain the angle and acceleration. At the beginning, the MPU-6050 library example file was not included, and the angle was directly derived from the raw data. For the accelerometer, the angle obtained can have a valid representation of the real angle the MPU-6050 is at. The error is small as mentioned in the result section and shown in figure 3.1. However, when it is given a lateral vibration or when there is an external force applied to the module, the reading will fluctuate, leading to a relatively large error. As can be seen in figure 3.4 in the results section, the largest error has a magnitude of around 15 degree. The vibration given to the MPU-6050 was of only mild intensity. In reality, when the vehicle is accelerating or decelerating, considerably larger resultant acceleration will be experienced by the vehicle, and the vehicle will constantly read false angle produced by the MPU-6050. In addition, it was measured that the up ramp of the track has an angle of 25.5° and the down-hill ramp has an angle of 17°, measured by the MPU module. The fore-mentioned problem would cause two problems for the automatic path-tracking control of the vehicle. The first is when the vehicle accelerates, the angle will experience a spike due to the value fluctuation in acceleration. The controller will mistakenly process that as a sign for the vehicle to go up a ramp and pause for two seconds and spin before going down hills, because in automatic path-tracking, when the angle is around 25.5°, the vehicle will be going up a hill and doing a series of performance as the task requirement. Second, the distance calculated by encoder increment multiplied by cosine angle will lead to an accumulative error. The distance is an important feedback for the vehicle and if it is faulty, the control will have relatively huge error. As for the angle calculated by the gyroscope with this algorithm, there is another problem. The error caused by the gyroscope algorithm will increase with time because it is an integral of the velocity of the rotation. Even though, it is resilient to fluctuation due to vibration, the error is constantly increasing with time. Therefore, this algorithm tracking is not appropriate for the automatic path. Then the MPU-6050 library example file was used to obtain the angle. As shown in figure 3.5, the largest magnitude of the fluctuation captured by the serial monitor is less then 2°. This could address the two control problems measured above. However, it was found that the angle obtained with this algorithm will have an error of 9° once the MPU-6050 begins working. The error will last for around 1 minute before going back to 0. This may be related to the electromechanical configuration of the MPU. The electric static force at the beginning may cause the internal structure of the MPU to work abnormally for a small period of the time. However, this minor issue does not have a significant effect over control. With this algorithm for angle, the vehicle controller can perform a series of actions if the angel becomes larger than 20°.

In lab 4, factors that can affect TCRT5000's output have been investigated. As shown in table 3.2, the sensor analog output for black color background is significantly higher than the other colors. The sensor can easily distinguish black and white colors but the outputs for white color, green, blue and red color do not differ considerably. Therefore, other colors may not easily interfere the vehicle following the black line. It seems like that the vehicle would ignore the other color and only read black line. Then the influence of the ambient light intensity to the TCRT5000 sensor was examined. Different intensities of smart phone light were used to test the sensor while the sensor was above the black line at a height of 2 cm. The analogue outputs were compared against each other. It was found that the least intensity of light would generate the greatest output but the difference between three intensities of the flashlight does not differ significantly to each other. This indicates the effect of ambient light does not play a major role to the analogue output of the sensor. However, it has been stated by the datasheet [8], that when designing the control with TCRT5000 and calibrating its value, ambient light should be taken into account. The reason why the three outputs measured do not differ considerably to each other might be that the ambient light was already strong enough that it reaches the sensor value saturation point. The sensor becomes less sensitive to that additional change. Second, the measurement conducted in the lab has a flaw because only black color was tested with different intensity of the light. It might be the case that white color is more sensitive to the change of light intensity due to the fact that white color reflects most of the light while black color absorbs the light. Third, the distance from the light source to the ground was not fixed. The smart phone was held by an operator and the distance may vary, leading to a varying light intensity. In addition, the analogue output displayed on the Arduino serial monitor was constantly fluctuating and it was fairly hard to require the precise value for the sensor reading and so was the voltage output at the analogue pin. Therefore, the values recorded were given a range to represent the value measured at the time being. Then the distance to reflective object was taken into account and the relationship between the distance and the sensor reading was investigated. It was found that the sensor cannot work properly if the distance is too short such as less than 1 cm nor can it work if the distance is too large such as greater than 5. Combing with the working diagram of the sensor and the measurement [7], it was found that the sensor can work normally if the distance is between 2 cm and 3 cm. The sensor was once installed under the chassis in a way that the distance is less than 1 cm and the sensor cannot give a useable output. Then a pair of pillars was installed to step up the height to around 2 cm and then the sensor can work.

In the advanced line following task, even though as shown in table 3.5 and figure 3.5, the module gives a good measurement of the distance and the output distance is in linear relationship with the different placement of the module. However, the resolution for the measurement is quite coarse and the sensor only gives a good representation of the distance when the black line is right below the sensor. This may be caused by the limited number of the sensors on the module. This, however, can be partially mitigated by adjusting the potentiometer on the module. Lowering the sensitivity of the module can have a finer resolution. For faster speed in completing the track in the advanced line following task, the motor base speeds can be increased and the proportional can be increased at the same time to provide timely negative correction and also increase the derivative term to prevent overshooting.

CONCLUSION

In lab 3, MPU-6050 was mainly applied to measure the angle and was used as feedback for the distance calculation and for going up a ramp and doing series of performance in automatic path tracking. The raw data obtained by the MPU-6050 was processed to get the acceleration and then the angle was obtained. At first, the processed angle by the accelerometer will be influenced by lateral vibration and the angle processed by the gyroscope will increase over time. They cannot provide a good measurement for the angle. Then MPU-6050 example library file was modified and used to get the angle. When the angle calculated by the new algorithm was resilient to external force or resultant acceleration, MPU-6050 was implemented on the vehicle and bidirectional communication was establish for vehicle control. In lab 4, TCRT5000 was utilized for the line following tasks of the vehicle. The digital and analogue output of the sensor was recorded with different ambient condition. It was found that the sensor can distinguish white colour and black colour with a distance of around 2 cm to the reflective surface. Bang-bang controller was employed for the basic line control with only two optical sensors. The vehicle will always be at the either side of the sensor and the make direction change at each turn. Therefore, the line the vehicle has passed does not strictly conform with the black track. Therefore, for the advanced path tracking, SF-8CHIDTS which is an array of eight optical sensors was utilized to give a measure of the distance of the black line to the mid-point. Then the distance was used to generate an output to control the left-side and right-side motor speed by a PID control algorithm. The base speed was set as 25 percent of the maximum power and the saturation point was set to prevent PID controller generating speed over the maximum value. In tuning the controller in the final stage, the proportional term of the PID controller was first adjusted to give a steady-state correction for two-side motor speeds. It was found that in order to turn at the most curved bend without derailing, the proportional term must be greater than 8. Then it was found that there is overshooting problem when doing the trial. The derivative term of the controller was then increased to 3. Then the vehicle can finish the task without vibrating dramatically and without hitting the roadblocks. Therefore, the integral part was not adjusted and set to 0. For both the basic and advanced line following tasks, RT-control was adopted and had a frequency of 200 Hz. All the challenges in these two lab sessions were attempted.

REFERENCES

[1] R. Bertrand and R. Epenoy, "New smoothing techniques for solving bang–bang optimal control problems—numerical results and statistical interpretation," *Optimal Control Applications and Methods,* vol. 23, no. 4, pp. 171-197, 2002.

[2] S. Bennett, "Development of the PID controller," *IEEE Control Systems Magazine,* vol. 13, no. 6, pp. 58-62, 1993.

[3] D. Fedorov, A. Y. Ivoilov, V. Zhmud, and V. Trubin, "Using of measuring system MPU6050 for the determination of the angular velocities and linear accelerations," *Automatics & Software Enginery,* vol. 11, no. 1, pp. 75-80, 2015.

[4] N. Semiconductor, "nrf24l01+ single chip 2.4 ghz transceiver," *Datasheet, September,* 2008.

[5] Z.-p. Liu and G.-l. Zhao, "Short-range wireless data transmission based on nRF24L01 [J]," *Applied Science and Technology,* vol. 3, 2008.

[6] Z.-y. SHI, J.-p. GAI, D.-h. WANG, and Z.-j. ZHANG, "A new kind of high speed wireless RF transceiver-nRF24L01 and its application [J]," *International Electronic Elements,* vol. 8, pp. 42-44, 2007.

[7] V. Semiconductors, "TCRT5000, TCRT5000L," ed: Aug, 2009.

[8] V. Semiconductors, "Reflective Optical Sensor with Transistor Output," *Folha de dados. Ago,* 2009.

APPENDICES

Code for measuring angle and acceleration by MPU-6050 (read raw data directly and process to be readable accelerations and angles):

```
#include <Wire.h>
const int MPU = 0x68; // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;
void setup() {
  Serial.begin(115200);
  Wire.begin();                    // Initialize comunication
  Wire.beginTransmission(MPU);       // Start communication with MPU6050 // MPU=0x68
  Wire.write(0x6B);                  // Talk to the register 6B
  Wire.write(0x00);                  // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true);        //end the transmission
  /*
  // Configure Accelerometer Sensitivity - Full Scale Range (default +/- 2g)
//  Wire.beginTransmission(MPU);
//  Wire.write(0x1C);                  //Talk to the ACCEL_CONFIG register (1C hex)
//  Wire.write(0x10);                  //Set the register bits as 00010000 (+/- 8g full scale range)
//  Wire.endTransmission(true);
//  // Configure Gyro Sensitivity - Full Scale Range (default +/- 250deg/s)
//  Wire.beginTransmission(MPU);
//  Wire.write(0x1B);                   // Talk to the GYRO_CONFIG register (1B hex)
//  Wire.write(0x10);                   // Set the register bits as 00010000 (1000deg/s full scale)
//  Wire.endTransmission(true);
//  delay(20);
  */
  // Call this function if you need to get the IMU error values for your module
  calculate_IMU_error();
  delay(20);
}
void loop() {
  // === Read acceleromter data === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2 registers
  //For a range of +-2g, we need to divide the raw values by 16384, according to the datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
```

```
    Serial.println(AccX);
  // Calculating Roll and Pitch from the accelerometer data
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - 0.58; // AccErrorX ~(0.58) See the 
calculate_IMU_error()custom function for more details
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.58; // AccErrorY ~(-1.58)
  // === Read gyroscope data === //
  previousTime = currentTime;        // Previous time is stored before the actual time read
  currentTime = millis();            // Current time actual time read
  elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000 to get seconds
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2 registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divide first the raw value by 131.0, according to the 
datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
  // Correct the outputs with the calculated error values
  GyroX = GyroX + 0.56; // GyroErrorX ~(-0.56)
  GyroY = GyroY - 2; // GyroErrorY ~(2)
  GyroZ = GyroZ + 0.79; // GyroErrorZ ~ (-0.8)
  // Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by sendonds (s) to get the angle in degrees
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;
  // Complementary filter - combine acceleromter and gyro angle values
  roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;
  AccY = AccY*10;
  // Print the values on the serial monitor
//  Serial.print("roll: ");
//  Serial.println(roll);
//  Serial.print("\t");
//  Serial.print("pitch: ");
   // Serial.println(pitch);
//  Serial.print("\t");
//  Serial.print("yaw: ");
  //Serial.println(yaw);

//  Serial.print(" ");
//  Serial.println(AccY);
}
void calculate_IMU_error() {
  // We can call this funtion in the setup section to calculate the accelerometer and gyro data error. From here we will get the error values 
used in the above equations printed on the Serial Monitor.
  // Note that we should place the IMU flat in order to get the proper values, so that we then can the correct values
  // Read accelerometer values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    // Sum all readings
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
    c++;
  }
  //Divide the sum by 200 to get the error value
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Read gyro values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
    GyroZ = Wire.read() << 8 | Wire.read();
    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX / 131.0);
    GyroErrorY = GyroErrorY + (GyroY / 131.0);
    GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
    c++;
  }
  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
}
```

MPU-6050 library example files for getting the angle and the acceleration:

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
```

```
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

#define OUTPUT_READABLE_YAWPITCHROLL
#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector
float angle;
// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}
void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
    // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.
    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }


}
```

```
void loop() {
  // if programming failed, don't try to do anything
  if (!dmpReady) return;

  // wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {
    if (mpuInterrupt && fifoCount < packetSize) {
      // try to get out of the infinite loop
      fifoCount = mpu.getFIFOCount();
    }

  }

  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  // get current FIFO count
  fifoCount = mpu.getFIFOCount();

  // check for overflow (this should never happen unless our code is too inefficient)
  if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >= 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();
    Serial.println(F("FIFO overflow!"));

  // otherwise, check for DMP data ready interrupt (this should happen frequently)
  } else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;

    #ifdef OUTPUT_READABLE_YAWPITCHROLL
      // display Euler angles in degrees
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);
      mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
//        Serial.print("ypr\t");
//        Serial.print(ypr[0] * 180/M_PI);
//        Serial.print("\t");
//        Serial.print(ypr[1] * 180/M_PI);
//        Serial.print("\t");
```

```
            angle = ypr[2] * 180/M_PI;


            Serial.println(angle);
        #endif
    }
    mpu.resetFIFO();
}
```

Code for bidirectional communication (Control pad end):

```
#include <SPI.h>
#include "RF24.h"
RF24 rf24(9,10); // CE, CSN
const byte addr[][5] = {"1Node", "2Node"};
const byte pipe = 1;
int Send[2];float Receive[2];
//////////////////////////////////////////////////////////////////
int TurnSpeed = 80;int RunSpeed = 80;int VRx = A0;int VRy = A1;int mapX = 0;int mapY = 0;
///////////////////////////////////////////////////////////////////
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
int angle;float distance;int runStat;
/////////////////////////////////////////////
void setup() {
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(101);
  rf24.openWritingPipe(addr[0]);
  rf24.openReadingPipe(pipe, addr[1]);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  pinMode(VRx, INPUT);
  pinMode(VRy, INPUT);
  Serial.println("nRF24L01 T ready");
  ////////////////////////////////////////////////
   lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Angle:");
  lcd.setCursor(0,1);
   lcd.print("Distance: ");
   ///////////////////////////////////////////////
}

void loop() {
  sendData();
  readData();

   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("Angle:");
   lcd.print(angle);
    lcd.print("  C1 ");
    if(runStat == 1){
      lcd.print("R ");
    }else{
      lcd.print("S ");
    }
   lcd.setCursor(0,1);
   lcd.print("Distance: ");
   lcd.print(distance);
}

inline void sendData(){

   rf24.startListening();
  mapX = analogRead(VRx);
  mapY = analogRead(VRy);
  mapX = map(mapX, 0, 1023, TurnSpeed, -TurnSpeed);
  mapY = map(mapY, 0, 1023, -RunSpeed, RunSpeed);
    if(mapX < -10 || mapX > 10 || mapY < -10 || mapY > 10){
            if(mapX < -10 || mapX > 10){
      Send[1] = mapX ;
      Send[0] = 0;}

          if(mapY < -10 || mapY > 10){
              Send[0] = mapY ;
              Send[1] = 0;}
      runStat = 1;
}else{
  Send[0] = 0 ;
  Send[1] = 0;
  runStat = 0;
}
    rf24.stopListening();
    rf24.write(&Send, sizeof(Send));
    rf24.startListening();
}

inline void readData(){

  rf24.startListening();
  if(rf24.available(&pipe)){
    rf24.read(&Receive, sizeof(Receive));
    angle = Receive[0];
    distance = Receive[1];
    Serial.print("Xangle: ");
    Serial.print(angle);
    Serial.print("distance: ");
    Serial.println(distance);
  }
}
```

Code for bidirectional communication (Vehicle end):

```
#include <SPI.h>
#include "RF24.h"
RF24 rf24(9,10); // CE, CSN
const byte addr[][5] = {"1Node", "2Node"};
const byte pipe = 1;
float Send[2];
int Receive[2];
////////////////////////////////////////////////////////////////
#include <Wire.h>
const int MPU = 0x68; // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;
int angle;

/////////////////////////////////////////////////////////////////
long unsigned int encoder1Value = 0;
long unsigned int encoder2Value = 0;
long unsigned int target = 0;
//long unsigned int initial = 2147483648;
byte RunningSpeed = 40;
byte TurningSpeed = 60;
float distance = 0.0;
byte RightTurn[14] = {0x62, 0x61, 0x66, 0x66, 0x72, 0x72, TurningSpeed , 0x00, TurningSpeed , 0x00, TurningSpeed , 0x00,
TurningSpeed , 0x00};
// Turn Right   F: 66  R: 72 .  1: 090   2 : 090  3 : 086  4 : 086
byte LeftTurn[14] = {0x62, 0x61, 0x72, 0x72, 0x66, 0x66,TurningSpeed , 0x00,TurningSpeed , 0x00, TurningSpeed , 0x00, TurningSpeed ,
0x00};
// Turn Right   F: 66  R: 72 .  1: 090   2 : 090  3 : 086  4 : 086
byte RunForward[14] = {0x62, 0x61, 0x66, 0x66, 0x66, 0x66,RunningSpeed, 0x00, RunningSpeed, 0x00, RunningSpeed, 0x00,
RunningSpeed, 0x00};
// Run Forward   1 : 040  2 : 040    3 : 039    4 : 039
byte RunBackward[14] = {0x62, 0x61, 0x72, 0x72, 0x72, 0x72, RunningSpeed, 0x00, RunningSpeed, 0x00, RunningSpeed, 0x00,
RunningSpeed, 0x00};
```

```
// Run Forward   1 : 040  2 : 040    3 : 039    4 : 039
byte haltAll[2] = {0x68, 0x61};

///////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);
  Wire.begin();                      // Initialize comunication
  Wire.beginTransmission(MPU);       // Start communication with MPU6050 // MPU=0x68
  Wire.write(0x6B);                  // Talk to the register 6B
  Wire.write(0x00);                  // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true);        //end the transmission
  ///////////////////////////////////////////////////////////////
  rf24.begin();
  rf24.setChannel(101);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.openWritingPipe( addr[1]);
  rf24.openReadingPipe(pipe, addr[0]);
  Serial.println("nRF24L01 T ready");

}

void loop() {
  getAngles();
  getDistance();
  readData();
  sendData();
    if(Receive[0] > 10 || Receive[0] < -10 || Receive[1] > 10 || Receive[1] < -10 ){
                if(Receive[0] > 10){
            speedChange(RunForward,Receive[0]);
            runForward();
          }
          if(Receive[0] < -10){
            speedChange(RunForward,-Receive[0]);
            runBackward();
          }
                  if(Receive[1] > 10){
            speedChange(RunForward,Receive[1]);
            turnRight();
          }
          if(Receive[1] < -10){
            speedChange(RunForward,-Receive[1]);
            turnLeft();
          }
    }else{
        stopAll();
    }
}


inline void readData(){
  rf24.startListening();
  if(rf24.available(&pipe)){
   rf24.read(&Receive, sizeof(Receive));
   }
}

inline void sendData(){
    rf24.stopListening();
    Send[0] = angle;
    Send[1] = distance;
    rf24.write(&Send, sizeof(Send));
    rf24.startListening();
}




inline void getAngles(){
    // === Read acceleromter data === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2 registers
  //For a range of +-2g, we need to divide the raw values by 16384, according to the datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
  // Calculating Roll and Pitch from the accelerometer data
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - 0.58; // AccErrorX ~(0.58) See the
calculate_IMU_error()custom function for more details
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.58; // AccErrorY ~(-1.58)
  // === Read gyroscope data === //
  previousTime = currentTime;        // Previous time is stored before the actual time read
  currentTime = millis();            // Current time actual time read
  elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000 to get seconds
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2 registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divide first the raw value by 131.0, according to the
datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
  // Correct the outputs with the calculated error values
  GyroX = GyroX + 0.56; // GyroErrorX ~(-0.56)
  GyroY = GyroY - 2; // GyroErrorY ~(2)
  GyroZ = GyroZ + 0.79; // GyroErrorZ ~ (-0.8)
  // Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by sendonds (s) to get the angle in degrees
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;
  // Complementary filter - combine acceleromter and gyro angle values
  roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;

  // Print the values on the serial monitor
  angle = accAngleX;
}




/////////////////////////////////////////////////////////////////////////////

inline void readEncoder()
{
  long unsigned int encoder1 = 0;
  long unsigned int encoder2 = 0;
  long unsigned int encoder3 = 0;
  long unsigned int encoder4 = 0;
  Wire.requestFrom(42,8);

  while(Wire.available())
  {
    encoder1 = (long unsigned int) Wire.read();
    encoder1 += ((long unsigned int) Wire.read() <<8);
    encoder1 += ((long unsigned int) Wire.read() <<16);
    encoder1 += ((long unsigned int) Wire.read() <<24);
    encoder2 = (long unsigned int) Wire.read();
    encoder2 += ((long unsigned int) Wire.read() <<8);
    encoder2 += ((long unsigned int) Wire.read() <<16);
    encoder2 += ((long unsigned int) Wire.read() <<24);
  }


  encoder1Value = encoder1;
  Serial.println(encoder1Value);
  encoder2Value = encoder2;
  Serial.println(encoder2Value);
}

inline void turnRight(){
  Wire.beginTransmission(42);
  Wire.write(RightTurn, 14);
  Wire.endTransmission();
}
```

```
inline void turnLeft(){
  Wire.beginTransmission(42);
  Wire.write(LeftTurn, 14);
  Wire.endTransmission();
}

inline void stopAll(){
  Wire.beginTransmission(42);
  Wire.write(haltAll, 2);
  Wire.endTransmission();
}

inline void runForward() {
  Wire.beginTransmission(42);
  Wire.write(RunForward, 14);
  Wire.endTransmission();
}

inline void runBackward() {
  Wire.beginTransmission(42);
  Wire.write(RunBackward, 14);
  Wire.endTransmission();
}

inline void getDistance(){
  readEncoder();
   long unsigned int encoderStat = (encoder1Value / 10 + encoder2Value / 10) / 2;
   distance  = (encoderStat * 10 - 2147483648) * 0.3 / 234.256 ;
   Serial.print("encoderStat: ");
  Serial.print(encoderStat * 10);
  Serial.print("\tDistance: ");
  Serial.println(distance);

}

inline void ResetEncoder(){
  if (encoder1Value > 0x80040000)
  {
    encoder1Value=0x80000000;
  }
  else if (encoder1Value < 0x7FFC0000)
  {
    encoder1Value=0x80000000;
  }
}

inline void speedChange(byte *Command, byte Speed){
  Command[6] = Speed;
  Command[8] = Speed;
  Command[10] = Speed;
  Command[12] = Speed;
}
```

Code for PID control calibration and weighted average part (All external functions):

```
void readSensorData(void)
{
  unsigned char n;           // Variable for counter value
  unsigned char dataRaw[16];  // Array for raw data from module

  // Request data from the module and store into an array
  n = 0;  // Reset loop variable
  Wire.requestFrom(9, 16);  // Request 16 bytes from slave device #9 (IR Sensor)
  while (Wire.available())  // Loop until all the data has been read
  {
    if (n < 16)
    {
      dataRaw[n] = Wire.read();  // Read a byte and store in raw data array
      n++;
    }
    else
    {
      Wire.read();  // Discard any bytes over the 16 we need
      n = 0;
    }
  }

  // Loop through and covert two 8 bit values to one 16 bit value
  // Raw data formatted as "MSBs 10 9 8 7 6 5 4 3", "x x x x x x 2 1 LSBs"

  for(n=0;n<8;n++)
  {
    sensorData[n] = dataRaw[n*2]<< 2;   // Shift the 8 MSBs up two places and store in array
    sensorData[n] += dataRaw[(n*2)+1];  // Add the remaining bottom 2 LSBs

    // Apply the calibration values here!

  }
  calibrate();

  sum = sensorData[0] + sensorData[1] + sensorData[2] + sensorData[3] + sensorData[4] + sensorData[5] + sensorData[6] + sensorData[7];
  if(sum != 0 ) {
  dis =( -45.00 * sensorData[0]  - 32.1 * sensorData[1]   - 19.2858 * sensorData[2]  - 6.4283 * sensorData[3]  + 6.4283 * sensorData[4]  +
19.2858 * sensorData[5] + 32.1 * sensorData[6] + 45.00 * sensorData[7]) / sum;
  }
  else{
    dis  = 0;
  }

//  for(int n = 0 ; n < 8; n++){
//    Serial.print("\t");
//    Serial.print( sensorData[n]);
//  }
//
//
//  Serial.print("\t\t");
//  Serial.print( sum);
//  Serial.print("\t\t");
//  Serial.print( dis);
//  Serial.println();
}


inline void calibrate(){
  sensorData[0] = constrain(sensorData[0], 250, 950);
  sensorData[0] = map( sensorData[0] ,250, 950, 255, 0);
//
   sensorData[1] = constrain(sensorData[1], 265, 970);
  sensorData[1] = map( sensorData[1] , 265, 970, 255, 0);

  sensorData[2] = constrain(sensorData[2], 290,980 );
  sensorData[2] = map( sensorData[2] ,290,980 , 255, 0);
//

   sensorData[3] = constrain(sensorData[3], 290, 980);
  sensorData[3] = map( sensorData[3] , 290, 980, 255, 0);
//////////////////////////////////////////////////////////////////////////////
  sensorData[4] = constrain(sensorData[4], 345, 980);
  sensorData[4] = map( sensorData[4] , 345, 980 ,255, 0);
////

   sensorData[5] = constrain(sensorData[5], 295, 980);
  sensorData[5] = map( sensorData[5] , 295, 980,  255, 0);

  sensorData[6] = constrain(sensorData[6], 280, 980);
  sensorData[6] = map( sensorData[6] , 280, 980, 255, 0);
////

   sensorData[7] = constrain(sensorData[7], 250, 980);
  sensorData[7] = map( sensorData[7] , 250, 980, 255, 0);
}

float PID(float lineDist)
{
  // PID loop

  error = 0 - lineDist;

    if(error > 45 ){
    error = 45;
```

```
 }
 if(error < -45){
  error = -45;
 }
    errorOld = error;
      // Save the old error for differential component
 // Calculate the error in position
 errorSum += error;

 float proportional = error * Kp;  // Calculate the components of the PID
 float integral = errorSum * Ki;
 float differential = (error - errorOld) * Kd;
 float output = proportional + integral + differential;  // Calculate the result
 return output;
}
```