EEEE1039

Applied Electrical and Electronic Engineering:

Construction Project

Motion Sensing, Remote Control and Line Following

Department of Electrical and Electronic Engineering

University of Nottingham Ningbo China

Student's Name: Jingxiong Yu

Student's ID: 20123738

Group: 7

Tutors' Name: Dr. Chunyang Gu, Dr. Jing Wang,

Dr. Giampaolo Buticchi, Dr. Chiew-Foong Kwong

Submission Date: 31 December 2019

Abstract

The purpose for this experiment was to build a highly advanced vehicle enhanced with motion sensing, remote control and line following. Therefore, in this report, process of developing the advanced vehicle, some concerning theory and possible analysis were displayed. For theory, motion sensing, bidirectional wireless communication, remote control and PID control techniques were presented. All data and graphs collected during the experiment were also covered. Circuits and codes were very important as this experiment was operated on the condition of many modifications and testing of circuits and codes. The reconstructed and integrated vehicle and remote controller were presented in this report. As the result of this experiment, an advanced line follower vehicle was successfully built.

Introduction

In previous lab sessions, a vehicle was built that was able to finish tasks about spinning and running with real-time control and had a human machine interface for operators to input commands before reactions taken. However, the aim of this construction project was to build a highly advanced vehicle that eventually was able to run following the lines automatically and telecontrolled. Therefore, the vehicle built in previous sessions was not advanced enough to fulfil the tasks about line following due to which improvement of both hardware and software were needed. The experiments were conducted to build the highly advanced line following vehicle.

The aims of session 3 and 4 were to enhance the vehicle with motion sensing and remote control to fulfil the tasks about line following.

For lab session 3, some hardware was required to be integrated and assembled on the vehicle built in the previous sessions, including an Arduino Nano board, a nRF24L01+ module and an MPU-6050 module. And some hardware was required to be integrated and assembled on the remote controller, including an Arduino Nano board, a nRF24L01+ module and a re-soldered Human-Machine Interface with an LCD, buttons and a joystick. For lab session 4, some hardware was required to be integrated and assembled on the vehicle built in session 3, including a TCRT5000 sensor module, a Sunfounder line follower module SF-8CHIDTS and three potentiometers.

Coding was also needed for driving the vehicle, the wireless communication, the motion sensing and line following. Real-Time communication and control methods were applied to the software, based on the MsTimer2 Interrupt Service Routine algorithm. Bang-Bang control and PID control algorithms were used for line following.

Lab Session 3:

1. Use MPU-6050 to measure inclination and other data.
2. Use nRF24L01+ to realize wireless communication between two Arduino Nano boards.
3. Joystick, HMI design and integration with remote controller.
4. System integration and challenge (remote control and/or path tracking).

Lab Session 4:

1. Basic line following with two optical sensors.
2. Position measurement using SunFounder line follower module SF-8CHIDTS.
3. PID study and its parameter tunings for advanced line following with SF-8CHIDTS.
4. System integration with HMI & RT, challenge.

Methods

Lab Session 3:

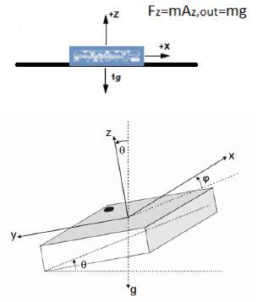
Subject 1: Use MPU-6050 to measure inclination and other data.

The MPU-6050 shown in Figure 1 was the world's first integrated 6-axis Motion Tracking device that combined a 3-axis gyroscope, a 3-axis accelerometer, and a Digital Motion Processor (DMP) all in a small 4mm × 4mm × 0.9mm package.

Figure 1. MPU-6050

The working principle about accelerometer and gyroscope was studied.

As shown in Figure 2, an accelerometer was a device that measured acceleration, which was the rate of change of the velocity of an object, measured in meters per second squared (m / s^2) or in G-forces (g). When a physical body accelerated at a certain direction, it became subject to a force equal to: F=ma in accordance with Newton's Second Law. The acceleration measured by an accelerometer was proper acceleration. In relative theory, proper acceleration was the physical acceleration experienced by an object. It was thus acceleration relative to a free-fall, or inertial.



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

Figure 2. Working principle about accelerometer

For example, an accelerometer at rest on the surface of the Earth would measure a straight upwards acceleration of 9.81 m / s^2 due to Earth's gravity. By contrast, accelerometers in free fall would measure zero. Therefore, accelerometers could be used in tilt sensing. For an accelerometer placed on a ramp, the tilt angle could be calculated from equation shown in Figure 2.

The gyroscope sensor measured rotational velocity (angular velocity, degree/sec) along the Roll, Pitch and Yaw axes. As shown in Figure 3, Roll was the rotation around X-axis, Pitch was the rotation around Y-axis and Yaw was the rotation around Z-axis. Applications of gyroscopes included inertial navigation systems, such as in the Hubble telescope.

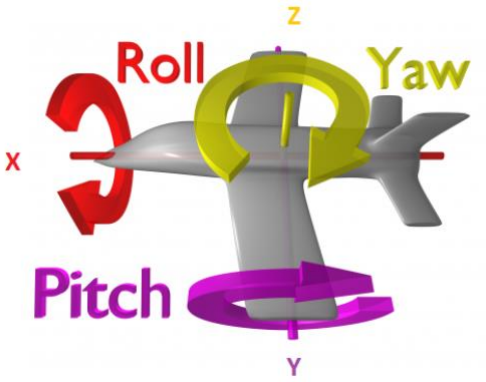


Figure 3. Working principle about gyroscope

Serial communication with I^2C was further studied.

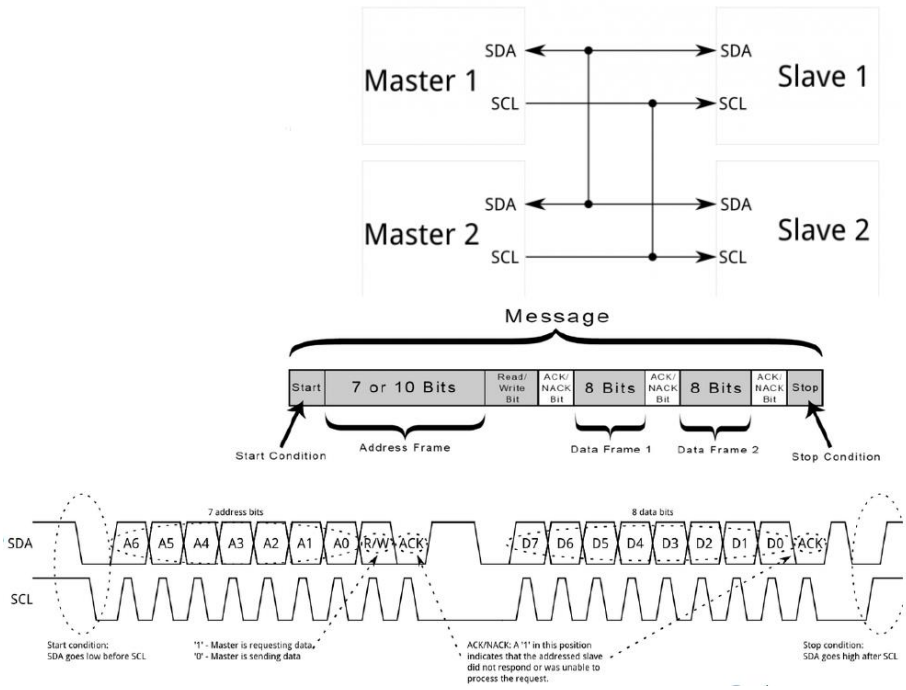


Figure 4. I^2C Communication

As shown in Figure 4, most I^2C devices could communicate two wires for data transmission which was called synchronized serial communication. It had two speed modes including a standard speed mode of a frequency of 100kHz and a fast speed mode of a frequency of 400kHz. It had unlimited number of masters and up to 1008 slaves. The protocol was built into the Arduino libraries and ready to be used. To get the I^2C communication working, it was needed to find unique slave address for each sensor module. Therefore, a very simple sketch I^2C scanner could be used to scan the I^2C-bus for devices as shown in Figure 5. If a device was found, it was reported the Hexadecimal values of 7-bit addresses of the found devices to the Arduino serial monitor.

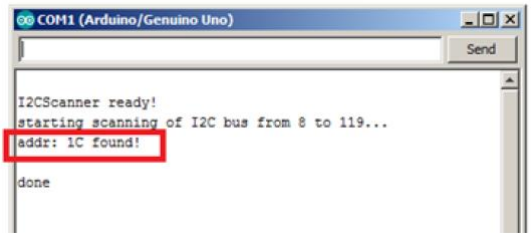


Figure 5. I^2C scanner

MPU-6050 accelerometer and gyroscope were studied. As shown in Figure 1 and Figure 6, the chip of this module was MPU-6050. The operating voltage of this module was 3V to 5V with on board LDO regulator. The communication method was I^2C fast speed mode at 400kHz. It had a built-in 16bit AD converter and 16bit data output. It had four different kinds of gyroscopes ranges: +/- 250, 500, 1000, 2000 degree/sec. The acceleration ranges were: +/- 2g, +/-4g, +/-8g, +/- 16g accordingly. Also, the MPU6050 module had an on-chip 1024 Byte FIFO buffer and a DMP engine for data fusion calculation and sensor synchronization. The module had wide applications, such as Electron and Optical Image Stabilization, portable navigation devices, the zero touch gestures user interface, intelligent mobile phone and tablet devices and handheld game products and 3D remote control.

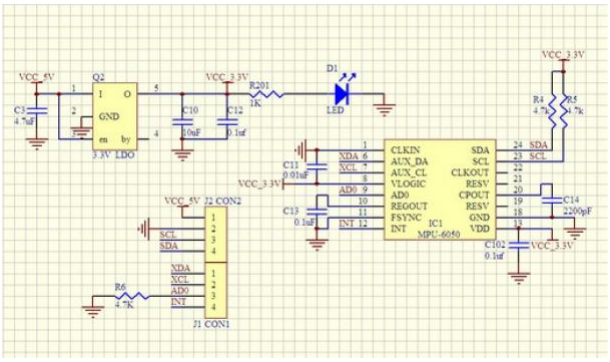


Figure 6. Schematic of MPU-6050

The module was connected to the I^2C bus for example by using one of the I^2C headers on the car baseboard. The minimum connections were shown in Figure 7, but some libraries might take use of the interrupt pin on the module as well. It was needed to collect the yaw, pitch, roll data from the MPU6050 and transmit this to the remote Arduino Nano.

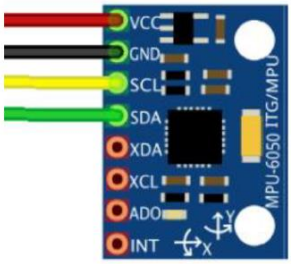


Figure 7. Minimum connections of MPU-6050

The breadboard construction of MPU-6050 with Arduino was shown in Figure 8.

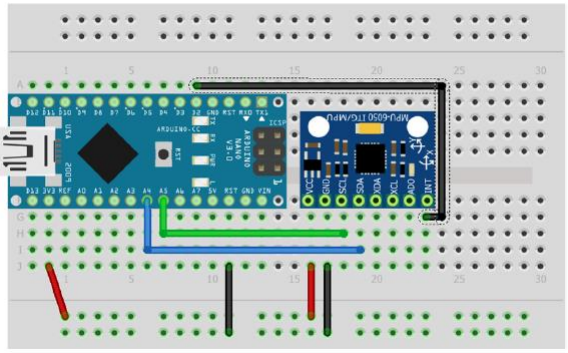


Figure 8. MPU6050 with Arduino

The codes for MPU-6050 with MPU6050 Library were designed to get the measurements of inclination and other data. And the design was verified with Serial Plotter and LCD. The most important part of the code (had slight modifications compared to original) to obtain raw data was shown in Figure 9.

```
Wire.beginTransmission(0x68);//I2C address of the MPU-6050
Wire.write(0x3B);//starting with register 0x3B (ACCEL_XOUT_H)
Wire.endTransmission(false);
Wire.requestFrom(0x68, 14, true);//request a total of 14 registers
AcX = Wire.read() << 8 | Wire.read();//0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
AcY = Wire.read() << 8 | Wire.read();//0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
AcZ = Wire.read() << 8 | Wire.read();//0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
//rough calibration
a_x = (AcX + 300) * 9.8 / 14368;
a_y = (AcY - 104) * 9.8 / 14368;
a_z = (AcZ) * 9.8 / 14368;
```

Figure 9. Code to obtain raw data from MPU6050

The accelerometer output read from MPU6050 was used to measure the inclination. The equation used was shown in Figure 2. The acceleration vs. tilt angle plot based on the measurement was generated and the findings were discussed. The device and the data it provided were used to detect the ramp placed on any straight path.

Subject 2: Use nRF24L01+ to realize wireless communication between two Arduino Nano boards.

The nRF24L01+ with RF24 Library were self-studied. As shown in Figure 10, the nRF24L01+ module was used for wireless system communication. The car controller was modified to allow its movements to be remotely controlled by an operator over a wireless radio frequency (RF) link. The RF modules given used the 2.4GHz band which was split into 128 channels. The unique channel used by group 7 was 115.

Figure 10. nRF24L01+ module

The connections between nRF24L01+ module and Arduino were shown in Figure 11.

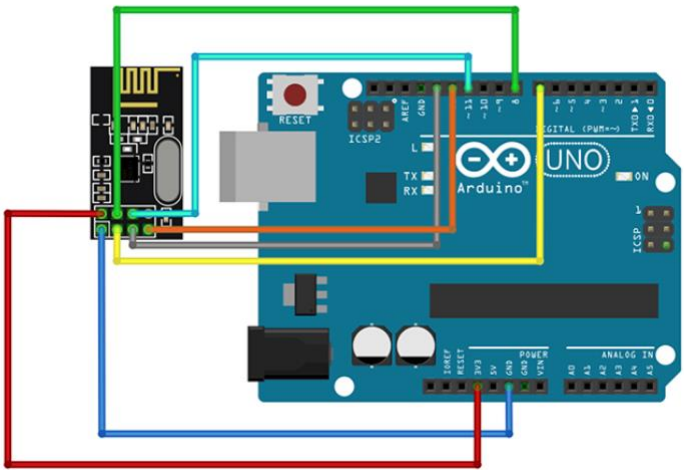


Figure 11. nRF24L01+ module with Arduino

Example codes for transmitting and receiving of wireless communication between two Arduino Nano boards were shown in Figure 12.

Transmit

```
#include <SPI.h>
#include "RF24.h"

RF24 rf24(9,10); // CE, CSN

const byte addr[] = "1Node";
const char msg[] = "Happy Hacking!";

void setup() {
  rf24.begin();
  rf24.setChannel(83);
  rf24.openWritingPipe(addr);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.stopListening();
}

void loop() {
  rf24.write(&msg, sizeof(msg));
  delay(1000);
}
```

Receive

```
#include <SPI.h>
#include "RF24.h"

RF24 rf24(9,10); // CE, CSN

const byte addr[] = "1Node";
const byte pipe = 1;

void setup() {
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(83);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.openReadingPipe(pipe, addr);
  rf24.startListening();
  Serial.println("nRF24L01 ready!");
}

void loop() {
  if (rf24.available(&pipe)) {
    char msg[32] = "";
    rf24.read(&msg, sizeof(msg));
    Serial.println(msg);
  }
}
```

Figure 12. Example codes for wireless communication

Serial communication with SPI was studied.

Unidirectional wireless communication between two Arduino Nano board was implemented by modifying codes shown in Figure 12. The implementation was verified with LCD display of inclination data on remote Arduino.

The codes of bidirectional communication between two Arduino Nano boards were designed. The most important part of the code (had slight modifications compared to original) to realize bidirectional wireless communication was shown in Figure 13.

```
#include<SPI.h>
#include<RF24.h>

void setup() {
  // put your setup code here, to run once:
  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
}

void loop() {
  // put your main code here, to run repeatedly:
  transmit_setup();
  rf24.write(&msg1, sizeof(msg1));
  rf24.write(&b, 20);
  delay(70);

  receive_setup();
  if (rf24.available(&pipe))
  {
    rf24.read(&a, 40);
  }
  delay(70);
}

void transmit_setup()
{
  rf24.openWritingPipe(addr1);
  rf24.stopListening();
}

void receive_setup()
{
  rf24.openReadingPipe(pipe, addr2);
  rf24.startListening();
}
```

Figure 13. Codes for bidirectional wireless communication

Subject 3: Joystick, HMI design and integration with remote controller.

Joystick was self-studied for using. And it was used to build the remote controller. It was shown in Figure 14.

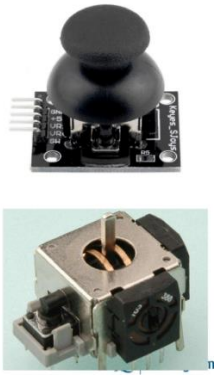


Figure 14. Joystick

The HMI hardware and software with a LCD, a joystick, press buttons and any other components were designed, and the HMI was attached to the remote controller. Soldering work of the remote HMI was done. An “LM2596 step-down power module” was used to generate a separate 5V power supply for the remote controller. Also, a fuse was used to protect the circuit.

Subject 4: System integration and challenge (remote control and/or path tracking).

Task C1 was selected. In the task, path tracking with remote control of joystick was required. The vehicle had to pass the track two times continuously.

Lab Session 4:

Subject 1: Basic line following with two optical sensors.

The working principle of TCRT5000 was studied. The TCRT5000 sensor module and its schematic were shown in Figure 15.

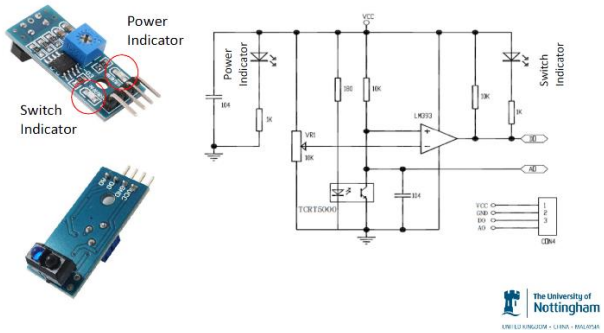


Figure 15. TCRT5000 sensor module

As shown in Figure 16, The output voltages (at A0 and D0) with a certain distance and darkness/colour of the reflection surface were recorded.

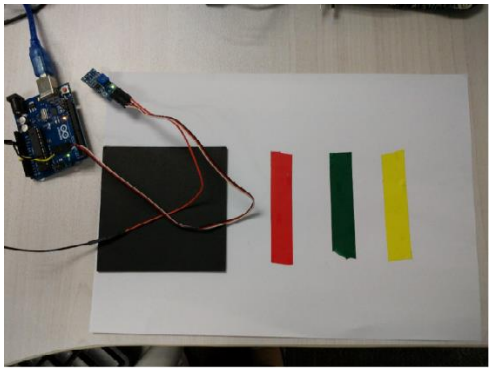


Figure 16. Output Voltages vs. Distance and Colour

The codes for the line following with Bang-bang control were designed and the implementation of basic line following with two of these optical sensors was done. The basic idea of Bang-bang control was shown in Figure 17. The most important part of the code (had slight modifications compared to original) to realize basic line following via Bang-bang control was shown in Figure 18.

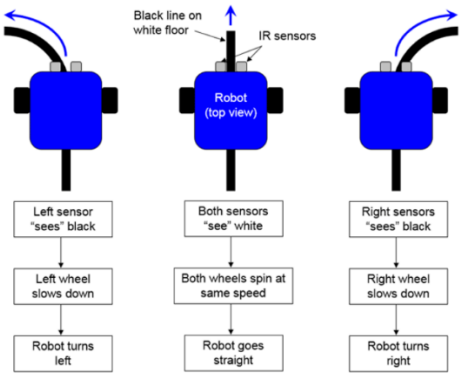


Figure 17. Basic idea of Bang-bang control

```
inline void rtControl()
{
  MsTimer2::set(10, demo);
  MsTimer2::start();
}

void demo()
{
  interrupts();
  int leftValue = analogRead(lf_left_a);
  int rightValue = analogRead(lf_right_a);
  if (leftValue > 500)flag_black_left = 1; else flag_black_left = 0;
  if (rightValue > 500)flag_black_right = 1; else flag_black_right = 0;
  linefollow();
}

void linefollow()
{
  if ((!flag_black_left) && (!flag_black_right))
  {
    //run straight
    Wire.beginTransmission(42);
    Wire.write("baffff");
    for (int i = 1; i <= 4; i++) {
      Wire.write(10);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  else if (flag_black_left && (!flag_black_right))
  {
    //turn left
    Wire.beginTransmission(42);
    Wire.write("barrff");
    for (int i = 1; i <= 2; i++) {
      Wire.write(70);
      Wire.write(0);
    }
    for (int i = 1; i <= 2; i++) {
      Wire.write(90);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  else if ((!flag_black_left) && flag_black_right)
  {
    //turn right
    Wire.beginTransmission(42);
    Wire.write("baffrr");
    for (int i = 1; i <= 2; i++) {
      Wire.write(90);
      Wire.write(0);
    }
    for (int i = 1; i <= 2; i++) {
      Wire.write(70);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
}
```

Figure 18. Codes for Bang-bang control

As shown in Figure 18, the Real-Time communication and control was also applied to the software to restrain the frequency of the detection and movement control of the vehicle to 100Hz.

Subject 2: Position measurement using SunFounder line follower module SF-8CHIDTS.

The I^2C serial interface and SF-8CHIDTS were studied. As shown in Figure 19, the Sunfounder Line Follower Module SF-8CHIDTS was used for advanced line following. The module was connected to the Arduino via I^2C serial interface. The output data represented amount of reflected light seen by each sensor. The processing of the analog voltage to a digital value was done by the sensor module which meant less effort for Arduino.

Figure 19. Sunfounder Line Follower Module SF-8CHIDTS

The application of the weighted average algorithm was understood. The weighted average algorithm and calibration method for the sensor based on given requirement were designed. Each sensor was calibrated to return a min and max value, for example 0 over white, 1000 over black. And each sensor value was given a weighting, for example distance from a reference point such as centre point of the vehicle. The weighting used for the algorithm was shown in Table 1.

Table 1. Weighting for 8 sensors of SF-8CHIDTS

Sensor ID	0	1	2	3	4	5	6	7
Weight (mm)	43.75	31.25	18.75	6.25	-6.25	-18.75	-31.25	-43.75

The distance from centre point was then given by the equation shown in Figure 20.

$$Distance = \frac{\sum_{i=1}^{8} sensor\ value_i . weight_i}{\sum_{i=1}^{8} sensor\ value_i}$$

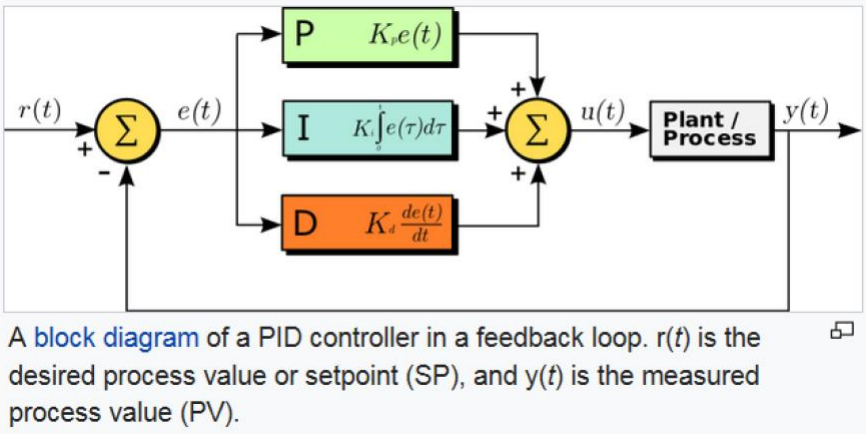
Figure 20. Equation for weighted average calculation

The output positions with different placements of the vehicle were recorded to verify the strategy.

Subject 3: PID study and its parameter tunings for advanced line following with SF-8CHIDTS.

The basis of traditional control theory was self-studied. As shown in Figure 17, Bang-bang control was one of the traditional control methods to realize line following. However, this method was not highly advanced.

The basic ideas of PID control were self-studied. If the magnitude of the error (e.g. distance from the reference point) was accurately known, a control scheme could be implemented known as Proportional Integral Derivative (PID). The advantages using PID control were that the robot was properly implemented to be able to follow the line quickly and in a stable manner. The basic principle of PID control was shown in Figure 21.



A block diagram of a PID controller in a feedback loop. r(*t*) is the desired process value or setpoint (SP), and y(*t*) is the measured process value (PV).

Figure 21. Basic of PID

The digital PID control method was self-studied. There were three steps for PID control:

a) An error value e(t) was difference between a desired setpoint SP=r(t) and a measured process variable PV=y(t) which was continuously calculated: e(t)=SP-PV.
b) As shown in Figure 22, a correction was applied to a control variable u(t) based on proportional, integral and derivative terms to minimize the error over time.
c) The final measured process variable was stable and with minimum error with the accurate and optimal control.

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,d\tau + K_\mathrm{d} \frac{de(t)}{dt}$$

K_p,K_i, K_d are non-negative the coefficients for the proportional, integral, and derivative terms

Figure 22. Equation for PID

The PID was separated into three terms. Term P was proportional to the current value of the SP-PV error e(t). Using proportional control alone would always result in an error between the setpoint and the actual process value. Term I accounted for past values of the SP-PV error e(t) and integrated them over time to produce the I term. The integral term sought to eliminate the residual error by adding a control effect due to the historic cumulative value of the error. Term D was a best estimate of the future trend of the SP-PV error e(t), based on its current rate of change. The derivative term sought to reduce the effect of the SP-PV error e(t) by exerting a control influence generated by the rate of error change.

The PID control codes were designed and the parameters were able to be tuned with the HMI including an LCD, buttons and three potentiometers.

The PID tuning approach followed these:

- First set K_p, K_i and K_d to zero.
- Started tuning K_p only to try and get the robot to follow the line (this was a P controller).
- Now tuned K_d to tune the robot response to the rate of change of error (this was a PD controller).
- Then tuned K_i to improve performance as much as possible (this was now a PID controller).
- If it all went wrong, started again.

Subject 4: System integration with HMI & RT, challenge.

Tasks about the basic line following with two single sensors and the advanced line following with SF-8CHIDTS and PID control were challenged.

The LCD on the remote controller built in session 3 was used to display system status including two sides' wheels' speeds, PID parameters and error via unidirectional wireless communication with nRF24L01+ module.

Results and Discussions

Lab Session 3:

Subject 1: Use MPU-6050 to measure inclination and other data.

The design was verified with Serial Plotter as shown in Figure 23.

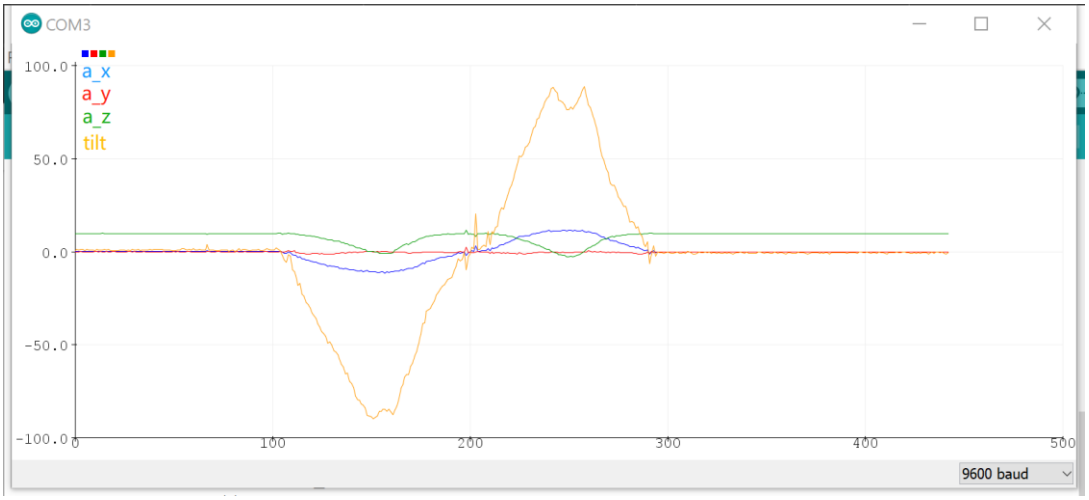


Figure 23. Verification with Serial Plotter

As shown in Figure 24, 25 and 26, the acceleration vs. tilt angle plots were generated base on the measurement via Serial Monitor.

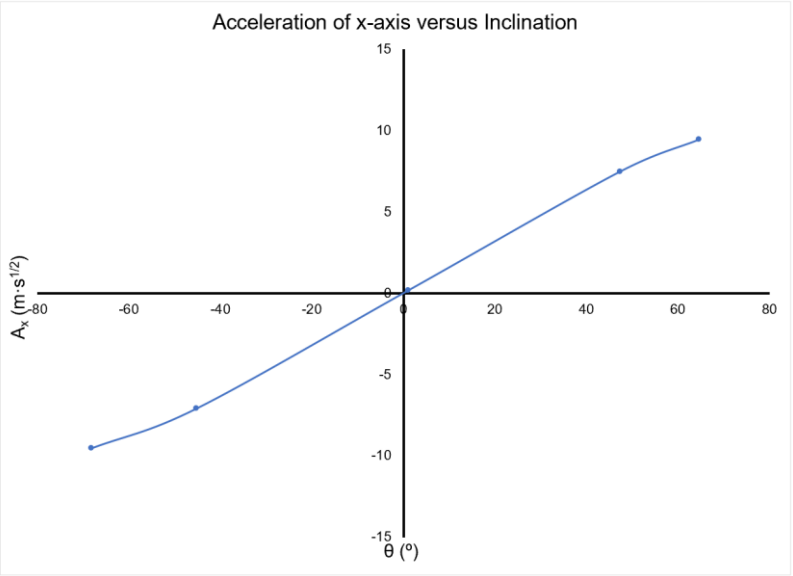


Figure 24. Acceleration of x-axis vs. tilt

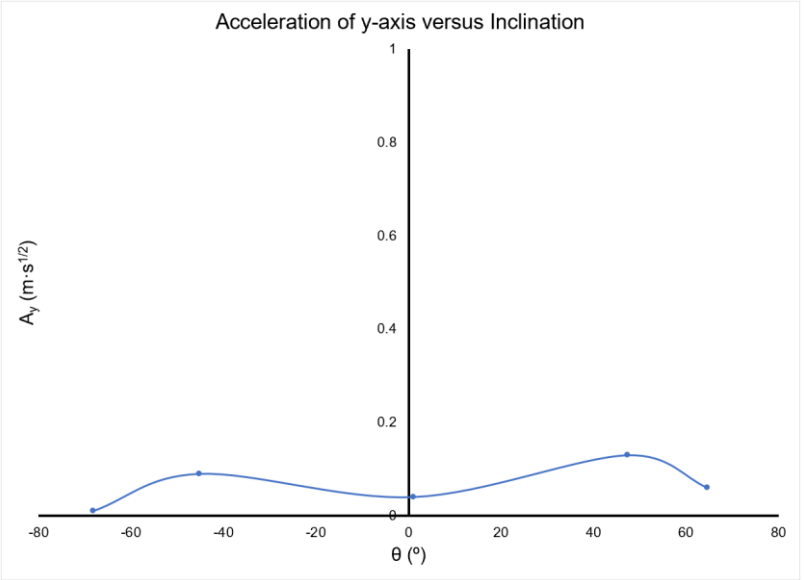


Figure 25. Acceleration of y-axis vs. tilt

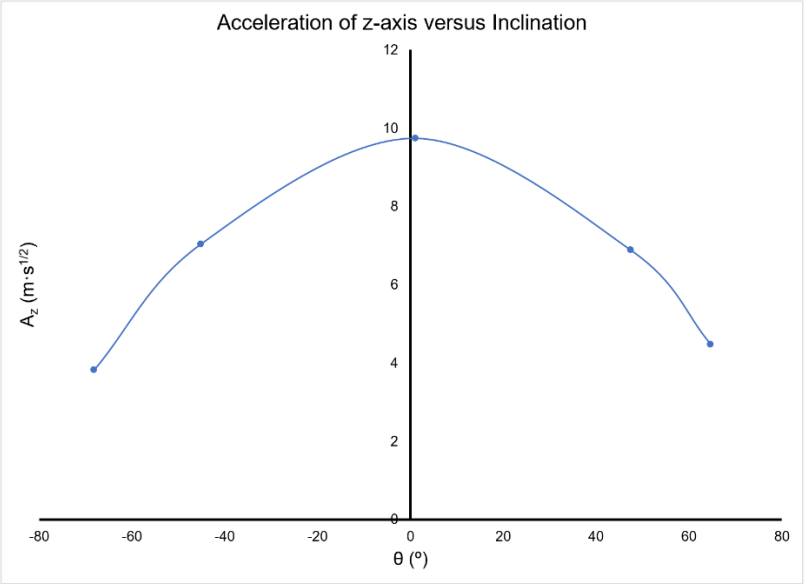


Figure 26. Acceleration of z-axis vs. tilt

As shown in Figure 24, though the response about acceleration of x-axis vs. tilt seemed like a linear function but it was not actually. The response was trigonometric. It seemed linear because the range of the tilt angle was restrained in (-90°, 90°). It was actually a sinusoidal function. As shown in Figure 25, though the response about acceleration of y-axis vs. tilt seemed having fluctuations, however, all the values of acceleration tended to be 0. Therefore, the response was close to y=0 which was linear. As shown in Figure 26, the response about acceleration of z-axis vs. tilt was not linear but trigonometric. The response was a cosine function.

Findings of the above responses were discussed. The response of y-axis was close to y=0 because when the tilt angle was detected, there was no transverse rotation of the vehicle, therefore the acceleration of y-axis remained to be 0. When the absolute value of the tilt angle became larger, the

acceleration of x-axis became larger and the acceleration of z-axis became smaller, therefore, there was a positive nonlinear correlation between acceleration of x-axis and tilt and there was a negative nonlinear correlation between acceleration of z-axis and tilt.

The MPU-6050 sensor was not only able to measure acceleration of three axes to obtain tilt angle but also able to measure rotational velocity along the roll, pitch and yaw axes due to the build-in gyroscope. The temperature could also be obtained using MPU6050 because of the build-in sensor.

Subject 2: Use nRF24L01+ to realize wireless communication between two Arduino Nano boards.

Because when the remote controller was controlling the vehicle remotely, the remote controller was used as transmitting terminal and the vehicle was used as receiving terminal. However, when the status of the vehicle was obtained by the remote controller remotely to display on the LCD screen integrated onto the controller, the vehicle was used as the transmitting terminal and the controller was used as the receiving terminal. Thus, the unidirectional wireless communication used while doing verification was not applicable in this situation. Therefore, the bidirectional wireless communication method was used to realize it.

As shown in Figure 13, the delay between transmitting and receiving for each terminal was set to 70 milliseconds. That was the final decision. If the delay was set too high, the movement control of vehicle was always delayed after the command was given to the controller which made it difficult to control the vehicle to run following the line precisely. And if the delay was set too low, then the bidirectional wireless communication simply went on strike because the too high frequency of changing the transmitting and receiving modes conflicted the pipes for communication.

Subject 3: Joystick, HMI design and integration with remote controller.

Because the contents of session 3 and session 4 were both released at the beginning of session 3, the Arduino's pins were arranged for both sessions and components were soldered onto the stripboard in which case the reconstruction during session 4 was not needed. Thus, the efficiency of time arrangement was improved.

When testing for the joystick, the design disadvantage was discovered that the input range was small. It meant that if the joystick was toggled to be near the four edges, the values obtained from the build-in encoder were all the maximum value (e.g. 1023), which made the joystick too sensitive to control the vehicle easily.

The integrated remote controller was shown in Figure 27.

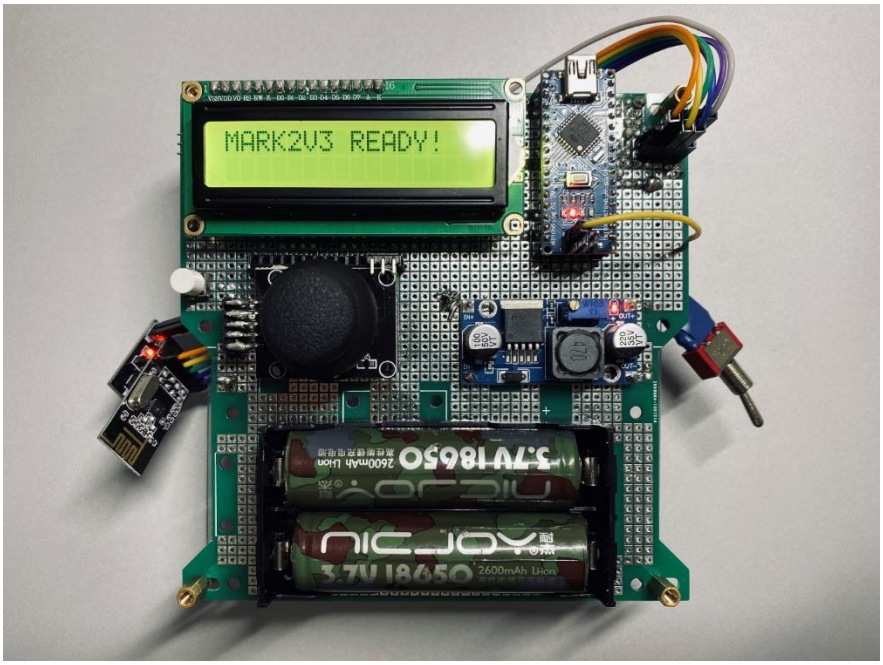


Figure 27. Integrated Remote Controller

Subject 4: System integration and challenge (remote control and/or path tracking).

The most important and difficult part was the debugging for both hardware and software. The last lab session was a failure due to the debugging problem for both the hardware and the software. The reason resulted in the failure was that it was not clear either the hardware or the software had problems, owing to which it was impossible to debug for both the hardware and the software at the same time. Therefore, the lesson learnt from last lab session which was applied into this lab session was that the hardware was debugged first to make sure that the hardware worked properly before debugging for the software. In that case, it was the software problem if any problem occurred after debugging for the hardware.

The demo was successfully challenged which was to remotely control the vehicle to run a track that had twists and turns as well as a bridge twice continuously, following the black line.

The time costed to run a track twice was ranked a bit behind due to two significant reasons which were lacking remote driving experience and lacking a second joystick to control the vehicle's speed and direction separately. Though one joystick was enough for controlling both the speed and direction of the vehicle, but the joystick was too sensitive to differ the different directions input therefore it was difficult to rotate the vehicle at corners smoothly without stopping, causing a lot unnecessary time.

However, the bonus task was not finished in time because a lot time cost during hardware soldering and debugging. The balance of time arrangement between hardware and software needed to be improved. To conclude, session 3 was successful in certain things, but a few things still needed improvement.

Lab Session 4:

Subject 1: Basic line following with two optical sensors.

The output voltages (at A0 and D0) with a certain distance and darkness/colour of the reflection surface were recorded. The data obtained from serial monitor was shown in Table 2 and Table 3. In this experiment, TCRT5000 modules were used to obtain data via serial monitor. And 1 mm and 20 mm two different kinds of the distance between sensor and coloured lines were applied in this experiment because 20 mm was the actual distance between the sensor on vehicle and the black line on ground.

Table 2. TCRT5000 feedback at 1 mm

	Black	Blue	Green	Red
A0	1007	402	55	50
D0	1	0	0	0

Table 3. TCRT5000 feedback at 20 mm

	Black	Blue	Green	Red
A0	550	45	40	40
D0	1	0	0	0

As shown in Table 2 and Table 3, the sensors were sensitive to black but not sensitive to other colours. That was because the sensor was not detecting the different colours but detecting the reflected light which not only related to the colours but also the materials and other possible factors. And it was shown that the sensor was more sensitive near the coloured lines than on the vehicle, which meant the larger the distance between the sensor and the coloured lines, the less sensitive the sensor was. Therefore, it was used to detect the black lines, but it was almost impossible to differ from other coloured lines.

In this part of the subject, both analog pin and digital pin were connected to the Arduino, but the analog pin was used for convenience. At the very first, when the demo 1 basic line following was challenged, the bang-bang control algorithm was applied to the software. Therefore, even though the analog pin was used to obtain data, the data obtained was transferred to digital form 0 and 1 to show whether the black line was detected by the sensor.

About the demo 1 basic line following, two different kinds of line following algorithm using bang-bang control were developed. The very original one was the shown in Figure18. The idea of the original control algorithm was that when either side of the algorithm detected the black line, the vehicle started to rotate a very small angle to get the vehicle back onto the line. And if both the sensors didn't detect the black line, the vehicle would go straight.

When challenged for the demo, the two TCRT5000 sensors were placed as shown in Figure 28. However, it was difficult for the vehicle to detect the large degree angles (e.g. right angle) due to which it was unable to finish the demo. Therefore, the two optical sensors were placed as shown in

Figure 29. Then the vehicle was able to run the track twice continuously. However, the speed for line following was extremely low because if the speed was set too high, the vehicle would easy run off the track.

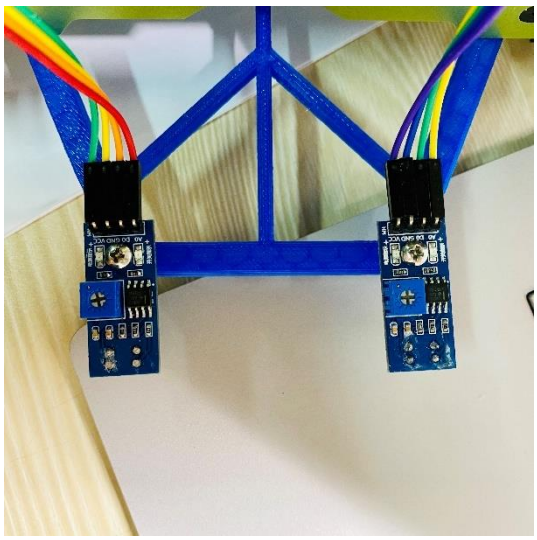


Figure 28. TCRT5000 placement before

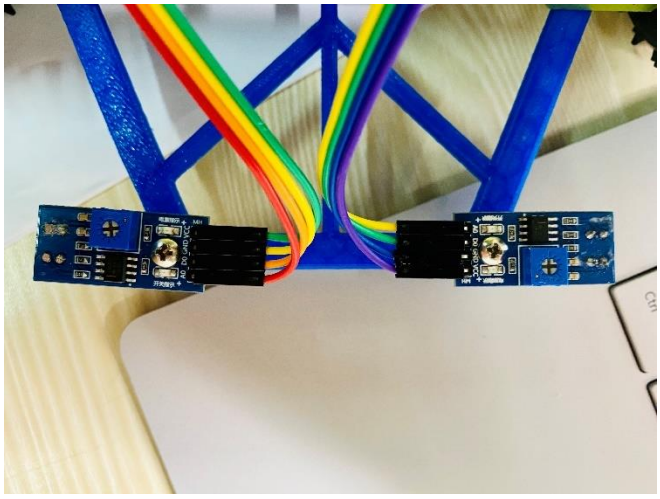
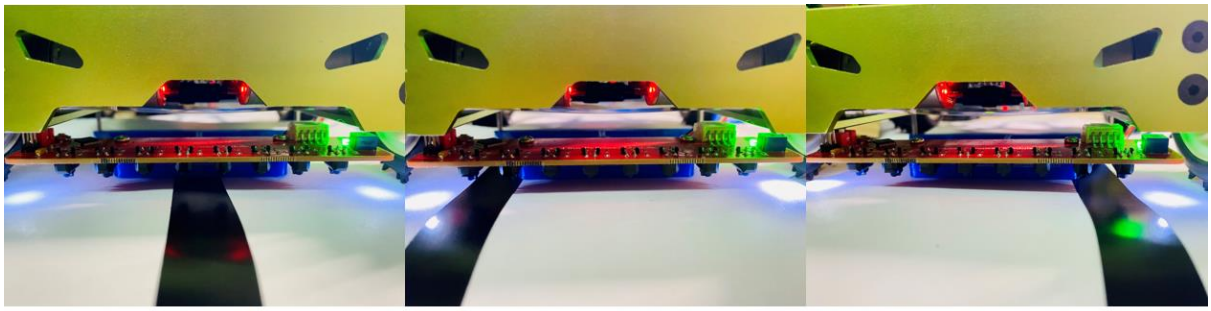


Figure 29. TCRT5000 placement after

Therefore, another idea for judgement was generated. Using the previous algorithm, the vehicle didn't know whether the rotation fitted the needs to get itself back onto the black line, in which case, it was either too much or not enough. And due to the same reason, the line following speed was not added up. However, if a flag was set that the vehicle did not stop rotating until another side detector detected the black line and the vehicle started rotating towards another side. And the vehicle instead of rotating at a same point but running forward while rotating by setting the absolute speed of wheels of two sides differently. Then when the demo was challenged, the vehicle was able to run following the lines quickly and harder to run out off the track. However, the shaking of vehicle was increased, and the line following was not smoothly. Thus, the PID control algorithm was applied into the software.

Subject 2: Position measurement using SunFounder line follower module SF-8CHIDTS.

The output position with different placement of the vehicle was recorded as shown in Figure 30 to verify the strategy.



Output Position: 1.50 **Output Position: 28.00** **Output Position: -30.00**

Figure 30. Output position with different placement of the vehicle

Using the weighted average algorithm, the output position obtained was in the range between -30 and 28, which was smaller than the actual range between -43.75 and 43.75 in millimetres. In that case, the algorithm used to calculate the output position had some imperfection. The output position was supposed to be equal to the actual distance between the black line and the referenced point which was the centre of the vehicle. And the output position obtained from the sensor was not stable due to certain undiscovered reason. These problems were not solved before integrated the weighted average algorithm with the PID control algorithm which might have influence onto the PID control. Thus, further experiment was to be done to perfect the algorithm.

Subject 3: PID study and its parameter tunings for advanced line following with SF-8CHIDTS.

The most difficult part of this session was PID control. It was applied to the algorithm to reduce the shaking of the vehicle so that the car was able to follow the line smoothly and precisely without touching the barriers. The hard part was finding the three variables for the PID control for which a lot of try-outs were done.

When the parameters were tested at first, all three parameters were set at the same time, and the vehicle was far not able to run the track following the line. In further discovery, it was realized that the K_i and K_d were only used for perfection of the line following and the K_p was the only parameter that was necessary for line following. Therefore, K_p=1.9 was found to make the vehicle to run the track. If K_p was smaller than 1.9, the vehicle was running off the track when overcame large degree corners. And if the K_p was larger than 1.9, the shaking of vehicle was increased therefore the barriers at the corners were touched by the vehicle. Then, the other two parameters were tested. The integral term would become too large if the K_i was too large because it was the cumulative sum of the error, therefore, the K_i was set to 0. K_d was set to 0.03 to perfect the line following. Thus, the vehicle was able to run the track for three times continuously without touching the barriers placed at the corners.

Subject 4: System integration with HMI & RT, challenge.

It was required to have switches on vehicle to modify PID variables. To fulfil this requirement, one potentiometer and one button switch were firstly soldered onto the vehicle. However, in further testing, it was discovered that the voltage rising in the button switch was not precisely captured by the Arduino due to which it was complicated to switch between different kinds of PID variable. But that was not the most significant reason why two more potentiometers had to be soldered to set three PID variables. It was mainly because the Arduino was unable to remember any of the PID variables every time it was rebooted in which case the PID had to be set each time which was difficult. So, three potentiometers were soldered onto the vehicle to control three PID variables separately. And to save Arduino pins for future enhancement, the LCD screen was not soldered onto the vehicle but the old one was used which was integrated onto the remote controller. And the LCD was used to display vehicle's status and different PID variables as well as PID error remotely using unidirectional wireless communication. That was the integration of the HMI for this session.

The HMI integrated on the vehicle was shown in Figure 31.

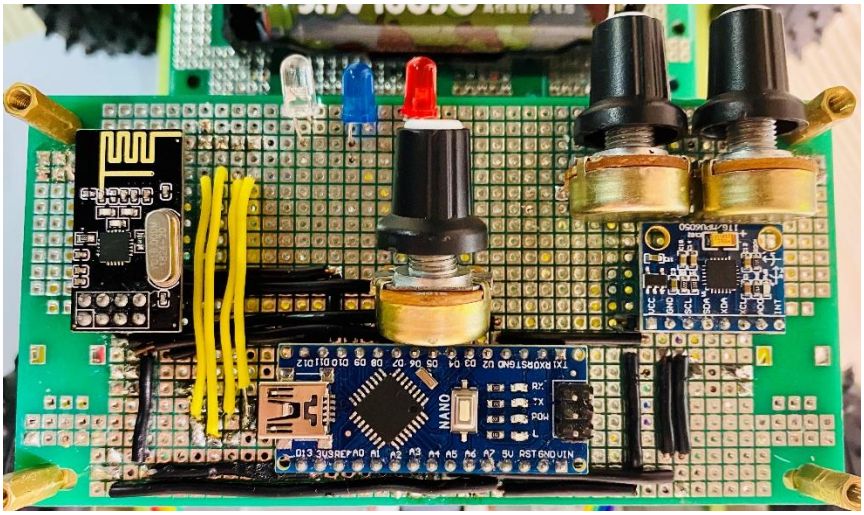


Figure 31. Integrated HMI on vehicle

The demo was successfully challenged, and the vehicle was enhanced with PID control to reduce the shaking therefore run a track for three times continuously without touching the barriers at corners.

During the session, the basic line following vehicle with two optical sensors TCRT5000 was developed at first using Bang-bang control algorithm to pass the demo 1. Secondly, the vehicle was enhanced with position measurement using Sunfounder line follower module SF-8CHIDTS connected via I^2C serial communication method. When the demo 2 was challenged, bang-bang control was used at first. However, it was unavoidable that the vehicle touched the barriers because of the shaking. Therefore, PID control was used to reduce the shaking.

Real-Time control was also applied into the software algorithm to restrain the frequency of the detection and the movement control of the vehicle to 100Hz. The frequency was verified using

oscilloscope, but it was a negligence that the screenshot of oscilloscope was not recorded. Therefore, the verification was not concluded in the report which needed to be avoid for future experiment.

Although the vehicle ran the track for three times continuously precisely following the line successfully, but the speed was quite low. Therefore, improvement of software with advanced PID control was needed.

The final integrated line follower vehicle Mark2_v4 was shown in Figure 32 and Figure 33.

Figure 32. Integrated Line Follower Vehicle (Top)

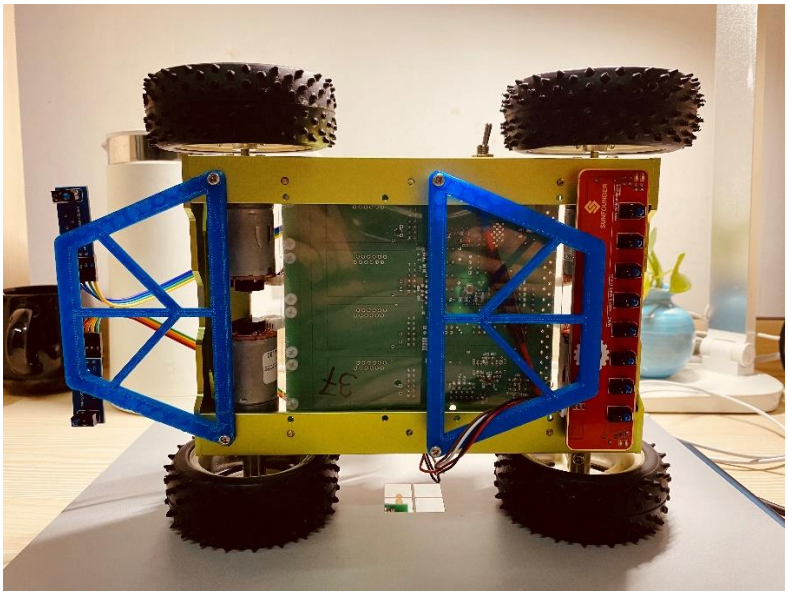


Figure 33. Integrated Line Follower Vehicle (Bottom)

Conclusion

During the experiment, session 3 was about basis of motion sensing with MPU-6050 and wireless communication with nRF24L01+ module as well as joysticks to realize remote control. In the first session, basic understanding of accelerator and bidirectional wireless communication were learnt, and the vehicle was reconstructed and integrated for session 3 and 4. And an additional remote controller was integrated with HMI and a joystick to realize remote controlled line following. Session 4 offered several methods to realize automatic line following. In this session, the vehicle was enhanced with basic line following with two optical TCRT5000 sensor modules and bang-bang control algorithm at first. The advanced line following was developed using Sunfounder line follower module SF-8CHIDTS and PID control algorithm to realize fast and accurate line following.

To summarize, these two lab sessions were successful. The basic demos and verifications of two sessions were successfully challenged. However, the bonus task for session 3 was not finished in time due to a lot time cost integrating and debugging for the hardware. In session 4, the PID control algorithm was imperfect that the vehicle was only able to run the track for three times continuously without touching the barriers placed at corners at a pretty low speed. Time allocation and clarity of tasks still needed improvement. Some record of the data was incomplete including the frequency verification of RT control which needed improvement.

References

[1]*H61AEE: Project Session III Introduction.* Department of Electrical and Electronic Engineering, University of Nottingham Ningbo China, 2019 [Online].

Available:

https://moodle.nottingham.ac.uk/pluginfile.php/5346450/mod_resource/content/20/Session_3_Introduction%201920%20V2.pdf

[Accessed: 31 December 2019]

[2]*H61AEE: Project Session IV Introduction.* Department of Electrical and Electronic Engineering, University of Nottingham Ningbo China, 2019 [Online].

Available:

https://moodle.nottingham.ac.uk/pluginfile.php/5346481/mod_resource/content/18/Session_4_Introduction%201920%20V1.pdf

[Accessed: 31 December 2019]

Appendices

Code for verification 1 of session 3 (obtain acceleration of three axes and tilt angle):

verify1

```
//verification 1
#include<Wire.h>
#include<Math.h>

const int MPU_addr = 0x68;//I2C address of the MPU-6050
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
double a_x = 0, a_y = 0, a_z = 0;
double d_xz = 0, d_xy = 0, d_yz = 0;

void setup() {
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);//PWR_MGMT_1 register
  Wire.write(0);//set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}

void loop() {
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);//starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true);//request a total of 14 registers
  AcX = Wire.read() << 8 | Wire.read();//0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read();//0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read();//0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  //Tmp=Wire.read()<<8|Wire.read();//0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  //GyX=Wire.read()<<8|Wire.read();//0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  //GyY=Wire.read()<<8|Wire.read();//0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  //GyZ=Wire.read()<<8|Wire.read();//0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  //rough calibration
  a_x = (AcX + 300) * 9.8 / 14368;
  a_y = (AcY - 104) * 9.8 / 14368;
  a_z = (AcZ) * 9.8 / 14368;
  Serial.print("a_x = "); Serial.print(a_x);
  Serial.print(" | a_y = "); Serial.print(a_y);
  Serial.print(" | a_z = "); Serial.print(a_z);

  //degrees
  int flag = 0;
  if (a_x > 0)flag = 1;
  else flag = -1;
  d_xz = acos(a_z / 9.8) / 3.14 * 180 * flag;
  Serial.print(" | degree = "); Serial.println(d_xz);
  delay(333);
}
```

Codes for verification 2 of session 3 (verify unidirectional wireless communication):

Code for remote controller:

2.3.1_RC

```
//code for mark2_v3.1 remote controller
//channel for wireless communication of group 7: 129-2*7=115
#include<SPI.h>
#include<RF24.h>
#include<LiquidCrystal.h>

RF24 rf24(7, 8); //ce,csn
const byte addr1[] = "addr1"; //address for rc transmit and robot receive
const byte addr2[] = "addr2"; //address for robot transmit and rc receive
const char msg1[] = "Message sent by RC.";
const byte pipe = 1;
//initialize the library by associating any needed lcd interface pin
//with the arduino pin number it is connected to
const int rs = 10, en = 9, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
double a[10];

void setup() {
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  receive_setup();
  Serial.println("mRF24L01 on RC ready!");
  //set up the lcd's number of columns and rows
  lcd.begin(16, 2);
  lcd.print("MARK2V3 READY!");
  delay(1000);
}

void loop() {
  if (rf24.available(&pipe))
  {
    rf24.read(&a, 40);
    lcd.clear();
    lcd.setCursor(0, 0); lcd.print(a[0]);
    lcd.setCursor(5, 0); lcd.print(a[1]);
    lcd.setCursor(10, 0); lcd.print(a[2]);
    lcd.setCursor(0, 1); lcd.print(a[3]);
  }
}

void receive_setup()
{
  rf24.openReadingPipe(pipe, addr2);
  rf24.startListening();
}
```

Code for vehicle:

2.3.1_Robot

```
//code for mark2_v3.1 robot//channel for wireless communication of group 7: 129-2*7=1
#include<SPI.h>
#include<RF24.h>
#include<Wire.h>
#include<Math.h>
RF24 rf24(7, 8); //ce,csn
const byte addr1[] = "addr1"; //address for rc transmit and robot receive
const byte addr2[] = "addr2"; //address for robot transmit and rc receive
const char msg2[] = "Message sent by Robot.";
const byte pipe = 1;
//definition for accelerator
const int MPU_addr = 0x68; //I2C address of the MPU-6050
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
double a_x = 0, a_y = 0, a_z = 0;
double degree_x = 0, degree_y = 0, degree_z = 0;
double a[10];
void setup() {
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.openWritingPipe(addr2);
  rf24.stopListening();
  //set up for accelerator
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);//PWR_MGMT_1 register
  Wire.write(0);
  Wire.endTransmission(true);
}
void loop() {
  //obtain data from accelerator
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);//starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true); //request a total of 14 registers
  AcX = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_YOUT_H) & 0x3C (ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_ZOUT_H) & 0x3C (ACCEL_ZOUT_L)
  //rough calibration
  a_x = (AcX - 500) * 9.8 / 14500;
  a_y = (AcY - 100) * 9.8 / 14500;
  a_z = (AcZ) * 9.8 / 14500;
  degree_x = (atan(a_x / sqrt(a_y * a_y + a_z * a_z))) * 180 / 3.14;
  delay(333);
  a[0] = a_x; a[1] = a_y; a[2] = a_z; a[3] = degree_x;
  //transmit data from accelerator to rc
  rf24.write(&a, 40);
  delay(1000);
}
```

Codes for demo of session 3 (remote controlled line following):

Code for remote controller:

2.3.1_RC

```
//code for mark2_v3.1 remote controller
//channel for wireless communication of group 7: 129-2*7=115

#include<SPI.h>
#include<RF24.h>
#include<LiquidCrystal.h>
#include<Math.h>
#include<PinChangeInterrupt.h>//for button switch

RF24 rf24(7, 8); //ce,csn

const byte addr1[] = "addr1"; //address for rc transmit and robot receive
const byte addr2[] = "addr2"; //address for robot transmit and rc receive
const char msg1[] = "Message sent by RC.";
const byte pipe = 1;

//initialize the library by associating any needed lcd interface pin
//with the arduino pin number it is connected to
const int rs = 10, en = 9, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

double a[10];
int b[10];

int flag_button = 1;

//definition of joystick
double joy_x = 0, joy_y = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  lcd.begin(16, 2);
  lcd.print("MARK2V3 READY!");
  delay(1000);
  //  //button switch
  //  pinMode(A3, INPUT_PULLUP);
  //  attachPinChangeInterrupt(digitalPinToPCINT(A3), right_corner, RISING);
}
```

```
void loop() {
  //joystick
  joy_x = analogRead(A0);
  joy_y = analogRead(A1);
  if (joy_y < 512)b[5] = 1;
  else if (joy_y > 514)b[5] = 2;
  else if (b[5] != 5)b[5] = 0; //set direction
  if (b[5] != 5)b[0] = b[1] = b[2] = b[3] = (int)abs((joy_y - 513) * 80 / 513);
  //spin on a point
  if (joy_x < 50)b[5] = 3;//anti-clockwise
  else if (joy_x > 950)b[5] = 4;//clockwise

  transmit_setup();
  rf24.write(&msg1, sizeof(msg1));
  rf24.write(&b, 20);
  delay(70);
  receive_setup();
  if (rf24.available(&pipe))
  {
    rf24.read(&a, 40);
    lcd.clear();
    lcd.setCursor(0, 0); lcd.print(b[0]);
    lcd.setCursor(3, 0); lcd.print(b[1]);
    lcd.setCursor(6, 0); lcd.print(b[2]);
    lcd.setCursor(9, 0); lcd.print(b[3]);
    lcd.setCursor(12, 0); lcd.print(b[5]);
    lcd.setCursor(0, 1); lcd.print(a[5]);
    lcd.setCursor(6, 1); lcd.print(a[3]);
  }
  delay(70);
}

void transmit_setup()
{
  rf24.openWritingPipe(addr1);
  rf24.stopListening();
}

void receive_setup()
{
  rf24.openReadingPipe(pipe, addr2);
  rf24.startListening();
}
```

Code for vehicle:

2.3.1_Robot

```
//code for mark2_v3.1 robot
//channel for wireless communication of group 7: 129-2*7=115

#include<SPI.h>
#include<RF24.h>
#include<Wire.h>
#include<Math.h>

RF24 rf24(7, 8); //ce,csn

const byte addr1[] = "addr1"; //address for rc transmit and robot receive
const byte addr2[] = "addr2"; //address for robot transmit and rc receive
const char msg2[] = "Message sent by Robot.";
const byte pipe = 1;

//definition for accelerator
const int MPU_addr = 0x68; //I2C address of the MPU-6050
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
double a_x = 0, a_y = 0, a_z = 0;
double degree_x = 0, degree_y = 0, degree_z = 0;
double a[10];
int b[10];

long unsigned int encoder1Value = 0;
long unsigned int encoder2Value = 0;
long unsigned int encoder3Value = 0;
long unsigned int encoder4Value = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);

  //set up for accelerator
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);//PWR_MGMT_1 register
  Wire.write(0);
  Wire.endTransmission(true);
  delay(10);
}
```

```
void loop() {
  transmit_setup();
  //obtain data from accelerator
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);//starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true); //request a total of 14 registers
  AcX = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_YOUT_H) & 0x3C (ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_ZOUT_H) & 0x3C (ACCEL_ZOUT_L)
  //rough calibration
  a_x = (AcX - 500) * 9.8 / 14500;
  a_y = (AcY - 100) * 9.8 / 14500;
  a_z = (AcZ) * 9.8 / 14500;
  degree_x = (atan(a_x / sqrt(a_y * a_y + a_z * a_z))) * 180 / 3.14;
  //define array
  a[0] = a_x; a[1] = a_y; a[2] = a_z; a[3] = degree_x;
  a[5] = readEncoder();
  //transmit data from accelerator to rc
  rf24.write(&a, 40);
  delay(70);
  receive_setup();
  if (rf24.available(&pipe))
  {
    char msg1[32] = "";
    rf24.read(&msg1, sizeof(msg1));
    rf24.read(&b, 20);
    drive();
  }
  delay(70);
}

void transmit_setup()
{
  rf24.openWritingPipe(addr2);
  rf24.stopListening();
}

void receive_setup()
{
  rf24.openReadingPipe(pipe, addr1);
  rf24.startListening();
}
```

//Remaining code on next page

```
void drive()
{
  Wire.beginTransmission(42);
  if (!b[5])
  {
    Wire.write("ha");
    Wire.endTransmission();
  }
  else if (b[5] == 1 || b[5] == 2)
  {
    if (b[5] == 1)Wire.write("baffff");
    else if (b[5] == 2)Wire.write("barrrr"); //set direction
    for (int i = 0; i < 4; i++)
    {
      Wire.write((unsigned char)b[i]);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  else if (b[5] == 5)
  {
    Wire.write("baffff");
    for (int i = 0; i < 4; i++)
    {
      Wire.write(b[i]);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  else
  {
    if (b[5] == 3) {
      Wire.write("barrff");
      for (int i = 0; i < 4; i++)
      {
        Wire.write(50);
        Wire.write(0);
      }
      Wire.endTransmission();
    }
    if (b[5] == 4) {
      Wire.write("baffrr");
      for (int i = 0; i < 4; i++)
      {
        Wire.write(50);
        Wire.write(0);
      }
      Wire.endTransmission();
    }
  }
}
```

```
double readEncoder()
{
  long unsigned int encoder1 = 0;
  long unsigned int encoder2 = 0;
  long unsigned int encoder3 = 0;
  long unsigned int encoder4 = 0;
  Wire.beginTransmission(42);
  Wire.write("i");
  Wire.endTransmission();
  Wire.requestFrom(42, 8);
  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;
  encoder2Value = encoder2;
  Wire.requestFrom(42, 8);
  encoder3 = (long unsigned int)Wire.read();
  encoder3 += ((long unsigned int)Wire.read() << 8);
  encoder3 += ((long unsigned int)Wire.read() << 16);
  encoder3 += ((long unsigned int)Wire.read() << 24);
  encoder4 = (long unsigned int)Wire.read();
  encoder4 += ((long unsigned int)Wire.read() << 8);
  encoder4 += ((long unsigned int)Wire.read() << 16);
  encoder4 += ((long unsigned int)Wire.read() << 24);
  encoder3Value = encoder3;
  encoder4Value = encoder4;
  if (encoder1Value > 0x80040000)
    encoder1Value = 0x80000000;
  else if (encoder1Value < 0x7FFC0000)
    encoder1Value = 0x80000000;
  if (encoder2Value > 0x80040000)
    encoder2Value = 0x80000000;
  else if (encoder2Value < 0x7FFC0000)
    encoder2Value = 0x80000000;
  if (encoder3Value > 0x80040000)
    encoder3Value = 0x80000000;
  else if (encoder3Value < 0x7FFC0000)
    encoder3Value = 0x80000000;
  if (encoder4Value > 0x80040000)
    encoder4Value = 0x80000000;
  else if (encoder4Value < 0x7FFC0000)
    encoder4Value = 0x80000000;
  double E1 = (encoder1Value - 2147483648) * 0.078 * 3.14 / (21.296 * 11);
  double E2 = (encoder2Value - 2147483648) * 0.078 * 3.14 / (21.296 * 11);
  double E3 = (encoder3Value - 2147483648) * 0.078 * 3.14 / (21.296 * 11);
  double E4 = (encoder4Value - 2147483648) * 0.078 * 3.14 / (21.296 * 11);
  return (E1 + E2 + E3 + E4) * 0.25;
}
```

Code for demo 1 of session 4 for vehicle only (basic line following):

2.4.1_Robot_original

```
//code for mark2_v4.1 remote controller
#include<SPI.h>
#include<RF24.h>
#include<Wire.h>
#include<Math.h>
#include<MsTimer2.h>//for RT control

//definition for line follower (2 detectors)
//left
const int lf_left_a = A0, lf_left_d = 2;
//right
const int lf_right_a = A1, lf_right_d = 3;

int flag_black_left = 0, flag_black_right = 0;

void setup() {
  Wire.begin();
  delay(10);
  Serial.begin(9600);
  rtControl();
  interrupts();
}

inline void rtControl()
{
  MsTimer2::set(10, demo);
  MsTimer2::start();
}

void loop() {
}

void demo()
{
  interrupts();
  int leftValue = analogRead(lf_left_a);
  int rightValue = analogRead(lf_right_a);
  if (leftValue > 500)flag_black_left = 1; else flag_black_left = 0;
  if (rightValue > 500)flag_black_right = 1; else flag_black_right = 0;
  linefollow();
}
```

//Remaining code on next page

```
void linefollow()
{
  if ((!flag_black_left) && (!flag_black_right))
  {
    //run straight
    Wire.beginTransmission(42);
    Wire.write("baffff");
    for (int i = 1; i <= 4; i++)
    {
      Wire.write(10);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  else if (flag_black_left && (!flag_black_right))
  {
    //turn left
    Wire.beginTransmission(42);
    Wire.write("barrff");
    for (int i = 1; i <= 2; i++)
    {
      Wire.write(70);
      Wire.write(0);
    }
    for (int i = 1; i <= 2; i++)
    {
      Wire.write(90);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  else if ((!flag_black_left) && flag_black_right)
  {
    //turn right
    Wire.beginTransmission(42);
    Wire.write("baffrr");
    for (int i = 1; i <= 2; i++)
    {
      Wire.write(90);
      Wire.write(0);
    }
    for (int i = 1; i <= 2; i++)
    {
      Wire.write(70);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
}
```

Codes for demo 2 of session 4 (advanced line following):

Code for vehicle:

2.4.4

```
#include <Wire.h>
#include <MsTimer2.h>
#include<SPI.h>
#include<RF24.h>

//wireless comm
RF24 rf24(7, 8); //ce csn
const byte addr[] = "addr1"; //address for robot transmit
const char msg[] = "Mark2_4.3 Ready!";
const byte pipe = 1;
double a[100] = {};

#define uchar unsigned char
uchar t;
int i, SPEED, button;
int Potentiometer1 = A2, Potentiometer2 = A3, Potentiometer3 = A7;
float BACK, DA, error;
unsigned int n, flag = 0;
unsigned char sensorData[8];
uchar TYPE;
uchar sensor[16];
uchar value[8];
int state[8];
float distance[8];

typedef struct
{
  float Target_value;
  float Current_value;
  float PWM;
  float Err;
  float Last_Err;
  float Kp, Ki, Kd;
  float output;
  float integral;
} PID_TypeDef;

PID_TypeDef PID;

void PID_Init();
float PID_operation(float value);
```

```
void PID_Init()
{
  PID.Target_value = 0.0;
  PID.Current_value = 0.0;
  PID.PWM = 0.0;
  PID.Err = 0.0;
  PID.Last_Err = 0.0;
  PID.output = 0.0;
  PID.integral = 0.0;
  PID.Kp = 1.9;
  PID.Ki = 0.0;
  PID.Kd = 0.03;
}

float PID_operation(float value)
{
  PID.Target_value = value;
  PID.Err = PID.Target_value - PID.Current_value;
  PID.integral += PID.Err;
  PID.output = PID.Kp * PID.Err + PID.Ki * PID.integral + PID.Kd * (PID.Err - PID.Last_Err);
  PID.Last_Err = PID.Err;
  PID.PWM = PID.output;
  PID.Current_value = PID.Current_value + PID.PWM;
  return PID.Current_value;
}

void setup()
{
  PID_Init();
  Wire.begin();
  t = 0;
  flag = 0;
  button = 0;
  pinMode(9, INPUT_PULLUP);
  noInterrupts();
  Serial.begin(9600);
  Wire.begin();
  MsTimer2::set(10, Timer2ISR);
  MsTimer2::start();
  interrupts();

  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.openWritingPipe(addr);
  rf24.stopListening();
}
```

//Remaining code on next page

```
void Edetect() {
  DA = 0;
  i = 0;
  Wire.requestFrom(9, 16);
  while (Wire.available())
  {
    sensor[t] = Wire.read();
    if (t < 15) {
      t++;
    }
    else {
      t = 0;
    }
  }
  for (n = 0; n < 8; n++)
  {
    sensorData[n] = sensor[n * 2] << 2;
    sensorData[n] += sensor[(n * 2) + 1];
    sensorData[n] = 1020 - sensorData[n];
  }
  for (n = 0; n < 8; n++) {
    if (sensorData[n] < 100)state[n] = 0;
    else state[n] = 1;
  }
  error = 500;
  if (state[3] == 1 || state[4] == 1) {
    error = 500;
  }
  if (state[2] == 1) {
    error = 400;
  }
  else if (state[1] == 1) {
    error = 150;
  }
  else if (state[0] == 1) {
    error = 0;
  }
  if (state[5] == 1) {
    error = 600;
  }
  else if (state[6] == 1) {
    error = 850;
  }
  else if (state[7] == 1) {
    error = 1000;
  }
}
```

```
void loop() {
  a[0] = PID.Kp;
  a[1] = PID.Ki;
  a[2] = PID.Kd;
  a[3] = PID.Err * 58.5 / 1000;
  delay(300);
  rf24.write(&msg, sizeof(msg));
  rf24.write(&a, 40);
}

void Timer2ISR() {
  interrupts();
  Edetect();
  Wire.beginTransmission(42);
  button = 1;
  if (digitalRead(9) == LOW) {
    button++;
  }
  PID.Kp = ((analogRead(Potentiometer1)) * (2.5 / 1023.0) + 0);
  if (button) {
    int P, I, D;
    P = analogRead(Potentiometer1);
    I = analogRead(Potentiometer2);
    D = analogRead(Potentiometer3);
    PID.Kp = (P * (2.5 / 1023.0) + 0);
    PID.Ki = (I * (10 / 1023.0) + 0);
    PID.Kd = (D * (0.05 / 1023.0) + 0);
  }
  if (flag == 0) {
    PID.Current_value = error;
    flag = 1;
    if (error < 500) {
      TYPE = 'L';
    }
    else if (error > 500) {
      TYPE = 'R';
    }
    else if (error = 500) {
      TYPE = 'M';
    }
  }
```

```
  if (TYPE == 'L') {
    BACK = PID_operation(500);
    SPEED = 20 + (abs(BACK - 500)) / 5;
    Wire.beginTransmission(42);
    Wire.write("barrff");
    for (int i = 1; i <= 4; i++)
    {
      Wire.write(SPEED);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  if (TYPE == 'R') {
    BACK = PID_operation(500);
    SPEED = 20 + (abs(BACK - 500)) / 5;
    Wire.beginTransmission(42);
    Wire.write("baffrr");
    for (int i = 1; i <= 4; i++)
    {
      Wire.write(SPEED);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  if (TYPE == 'M') {
    BACK = PID_operation(500);
    SPEED = 15 + (abs(BACK - 500)) / 6;
    Wire.beginTransmission(42);
    Wire.write("baffff");
    for (int i = 1; i <= 4; i++)
    {
      Wire.write(SPEED);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
  if (BACK > 450 && BACK < 550) {
    PID.Current_value = 500;
    flag = 0;
  }
}
```

Code for remote controller:

2.4.3remote

```
//code for mark2.4.3 rc
//channel:115

#include<SPI.h>
#include<RF24.h>
#include<LiquidCrystal.h>

RF24 rf24(7, 8);

const byte addr[] = "addr1"; //for receive
const byte pipe = 1;
char msg[32] = "";

//lcd
const int rs = 10, en = 9, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

double a[100] = {};

void setup() {
  // put your setup code here, to run once:
  rf24.begin();
  rf24.setChannel(115);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.openReadingPipe(pipe, addr);
  rf24.startListening();

  //lcd
  lcd.begin(16, 2);
  delay(10);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (rf24.available())
  {
    rf24.read(&msg, sizeof(msg));
    rf24.read(&a, 40);
  }

  lcd.setCursor(0, 1); lcd.print(a[0]);
  lcd.setCursor(5, 1); lcd.print(a[1]);
  lcd.setCursor(10, 1); lcd.print(a[2]);
  lcd.setCursor(0, 0); lcd.print(a[3]);
}
```

Code for bonus 2 of session 4 for vehicle only (run a track for three times continuously):

mark2.4.2

```
#include<SPI.h>
#include<RF24.h>
#include<Wire.h>
#include<Math.h>
#include<MsTimer2.h>

unsigned int  sensorData[8];
unsigned char n;
long unsigned int encoder1Value = 0;
long unsigned int encoder2Value = 0;
long unsigned int encoder3Value = 0;
long unsigned int encoder4Value = 0;
long unsigned int encoder1 = 0;
long unsigned int encoder2 = 0;
long unsigned int encoder3 = 0;
long unsigned int encoder4 = 0;
long unsigned int target1 = 0;
long unsigned int target2 = 0;
int Distance = 100;
int route_1 = 10000;
int buffer_s;
int flag = 0;

float distances[8] = {43.75, 31.25, 18.75, 6.25, -6.25, -18.75, -31.25, -43.75};
//PID Loop Constant analog inputs
#define Kp_in A2
#define Ki_in A3
#define Kd_in A7

//Default speed of the motors, this is modified by the PID output
#define leftMotorBaseSpeed 20
#define rightMotorBaseSpeed 20
#define BaseSpeed 20

//Speed limits of the motors
//#define min_speed -50 //30 when base 20
#define max_speed 50

float error, errorSum, errorOld;//Variables for the PID loop
float leftMotorSpeed, rightMotorSpeed;//Variables to hold the current motor speed (+-100%)

float Kp, Ki, Kd;//Variables to store the PID constants

void loop() {
}

void setup() {
  // put your setup code here, to run once:
  Wire.begin();
  Serial.begin(9600);
  Kp = 5;
  Ki = 0;
  Kd = 0;
```

//Remaining code on next page

```
  leftMotorSpeed = 0;
  rightMotorSpeed = 0;

  error = 0;
  errorSum = 0;
  errorOld = 0;

  rtControl();
  interrupts();
}

inline void rtControl()
{
  MsTimer2::set(10, demo);
  MsTimer2::start();
}

void readSensorData(void)
{
  unsigned char dataRaw[16];
  n = 0;
  Wire.requestFrom(9, 16);
  while (Wire.available())
  {
    if (n < 16)
    {
      dataRaw[n] = Wire.read();
      n++;
    }
    else
    {
      Wire.read();
      n = 0;
    }
  }
  for (n = 0; n < 8; n++)
  {
    sensorData[n] = dataRaw[n * 2] << 2;
    sensorData[n] += dataRaw[(n * 2) + 1];
    sensorData[n] = 1000 - sensorData[n];
  }
}

void demo()
{
  interrupts();
  readSensorData();
  linefollow();
}
```

```
float Dist_weightedAverage(void)
{
  float lineDist = 0;
  int sum_sensorData = 0;
  for (n = 0; n < 8; n++)
  {
    lineDist += sensorData[n] * distances[n];
    sum_sensorData += sensorData[n];
  }
  lineDist = lineDist / sum_sensorData;
  return lineDist;
}

float PID(float lineDist)
{
  errorOld = error;
  error = lineDist;
  errorSum += error;

  float proportional = error * Kp;
  float integral = errorSum * Ki;
  float differential = (error - errorOld) * Kd;
  float output = proportional + integral + differential;
  return output;
}

void linefollow()
{
  float WA = Dist_weightedAverage();
  float outputraw = PID(WA);
  int output = outputraw;

  if ((WA > -4) && (WA < 4))
  {
    Wire.beginTransmission(42);
    Wire.write("baffff");
    for (int i = 1; i <= 4; i++)
    {
      Wire.write(BaseSpeed);
      Wire.write(0);
    }
    Wire.endTransmission();
  }
```

```
else if (((WA <= -4) && (WA > -8)) || ((WA >= 4) && (WA < 8)))
{
  Wire.beginTransmission(42);
  Wire.write("baffff");
  for (int i = 1; i <= 2; i++)
  {
    if (leftMotorBaseSpeed - output / 2 < 0)
    {
      output = 2 * leftMotorBaseSpeed;
    }
    else if (leftMotorBaseSpeed - output / 2 > 30)
    {
      output = 2 * (leftMotorBaseSpeed - 30);
    }
    Wire.write(leftMotorBaseSpeed - output);
    Wire.write(0);
  }
  for (int i = 1; i <= 2; i++)
  {
    if (rightMotorBaseSpeed + output / 2 < 0)
    {
      output = -2 * rightMotorBaseSpeed;
    }
    else if (rightMotorBaseSpeed + output / 2 > 30)
    {
      output = 2 * (30 - rightMotorBaseSpeed);
    }
    Wire.write(rightMotorBaseSpeed + output / 2);
    Wire.write(0);
  }
  Wire.endTransmission();
}
else if (((WA <= -8) && (WA >= -15)) || ((WA >= 8) && (WA <= 15)))
{
  Wire.beginTransmission(42);
  Wire.write("baffff");
  for (int i = 1; i <= 2; i++)
  {
    if (leftMotorBaseSpeed - output < 0)
    {
      output = leftMotorBaseSpeed;
    }
    else if (leftMotorBaseSpeed - output > 30)
    {
      output = leftMotorBaseSpeed - 30;
    }
    Wire.write(leftMotorBaseSpeed - output);
    Wire.write(0);
  }
```

```
  for (int i = 1; i <= 2; i++)
  {
    if (rightMotorBaseSpeed + output < 0)
    {
      output = -rightMotorBaseSpeed;
    }
    else if (rightMotorBaseSpeed + output > 30)
    {
      output = 30 - rightMotorBaseSpeed;
    }
    Wire.write(rightMotorBaseSpeed + output);
    Wire.write(0);
  }
  Wire.endTransmission();
}
else if ((WA > 15) && (WA <= 20))
{
  Wire.beginTransmission(42);
  Wire.write("barrff");
  for (int i = 1; i <= 2; i++)
  {
    if (leftMotorBaseSpeed + output > max_speed)
    {
      output = max_speed - leftMotorBaseSpeed;
    }
    Wire.write(leftMotorBaseSpeed + output);
    Wire.write(0);
  }
  for (int i = 1; i <= 2; i++)
  {
    if (rightMotorBaseSpeed + output > max_speed)
    {
      output = max_speed - rightMotorBaseSpeed;
    }
    Wire.write(rightMotorBaseSpeed + output);
    Wire.write(0);
  }
  Wire.endTransmission();
}
else if ((WA < -15) && (WA >= -20))
{
  Wire.beginTransmission(42);
  Wire.write("baffrr");
  for (int i = 1; i <= 2; i++)
  {
    if (leftMotorBaseSpeed - output > max_speed)
    {
      output = leftMotorBaseSpeed - max_speed;
    }
    Wire.write(leftMotorBaseSpeed - output);
    Wire.write(0);
  }
```

```
  for (int i = 1; i <= 2; i++)
  {
    if (rightMotorBaseSpeed - output > max_speed)
    {
      output = rightMotorBaseSpeed - max_speed;
    }
    Wire.write(rightMotorBaseSpeed - output);
    Wire.write(0);
  }
  Wire.endTransmission();
}
else if  (WA < -20)
{
  Wire.beginTransmission(42);
  Wire.write("baffrr");
  for (int i = 1; i <= 2; i++)
  {
    Wire.write(70);
    Wire.write(0);
  }
  for (int i = 1; i <= 2; i++)
  {
    Wire.write(60);
    Wire.write(0);
  }
  Wire.endTransmission();
  if ((WA >= -20) && (WA <= -10))
  {
    WA = -21;
  }
  else if (WA > -10)
  {
    WA = WA;
  }
}
else if (WA > 20)
{
  Wire.beginTransmission(42);
  Wire.write("barrff");
  for (int i = 1; i <= 2; i++)
  {
    Wire.write(60);
    Wire.write(0);
  }
  for (int i = 1; i <= 2; i++)
  {
    Wire.write(70);
    Wire.write(0);
  }
  Wire.endTransmission();
```

```
    if ((WA >= -20) && (WA <= -10))
    {
      WA = -21;
    }
    else if (WA > -10)
    {
      WA = WA;
    }
  }
  else if (WA > 20)
  {
    Wire.beginTransmission(42);
    Wire.write("barrff");
    for (int i = 1; i <= 2; i++)
    {
      Wire.write(60);
      Wire.write(0);
    }
    for (int i = 1; i <= 2; i++)
    {
      Wire.write(70);
      Wire.write(0);
    }
    Wire.endTransmission();
    if ((WA <= 20 ) && (WA >= 10))
    {
      WA = 21;
    }
    else if (WA < 10)
    {
      WA = WA;
    }
  }
}
```