THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
ISDN 2602

Laboratory 6: Routing (5%)

A) Objectives

In this lab, we will utilize one of the most popular routing algorithms, Dijkstra's algorithm, to find the shortest path between two given nodes in a weighted graph. A weighted graph is a graph in which each branch is given a numerical weight. In particular, a weighted graph is a special type of labeled graph whose labels are numbers (usually taken to be positive), representing distance, time, money, or any cost.

Please **work in groups** to accomplish the following two parts in this lab:

Part I: Simulation of Dijkstra's Algorithm by MATLAB

The objective is to understand how to find the shortest path between two nodes using Dijkstra's algorithm.

Part II: Robotic car

The objective is to get familiar with the robotic car to be used for the final project and control it to complete some simple tasks.

B) Lab tasks

Part I: Simulation by MATLAB

Task 1 – Find the shortest path by Dijkstra's Algorithm

In this task, we will use the graph shown in Fig. 1.

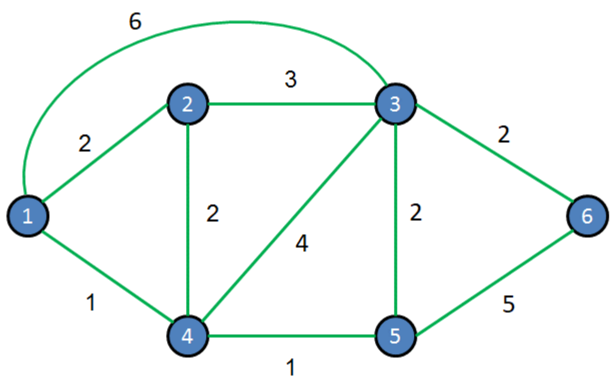


Figure 1: Example of a weighted graph

Open the file "**Task1.m**" in the MATLAB editor window and run the code.

There is an n x n array ***Graph*** whose elements represent the weight/cost, where n denotes the number of nodes in the graph. The cost between nodes ***i*** and ***j*** is given by ***Graph(i,j)***. In particular, ***Graph(i,j)*** = 0 for ***i*** = ***j***, ***Graph(i,j)*** is set to infinity (***inf***) when nodes ***i*** and ***j*** are not adjacent.

Function ***fun_dijkstra()*** is used to find the shortest path by Dijkstra's Algorithm and calculate the corresponding cost with three inputs, i.e., the ***Graph*** array, the source node (***source***), and the destination node (***dest***).

The initial code finds the shortest path from node **1** to node **3** as (**1** > **4** > **5** > **3**), where the cost is 4.

Check Point 1:

1) Modify the code to find the shortest path and the cost from node **2** to node **6**. Please **Write down your answer** in the blank below.

____

Task 2 – Create the Graph

Fig. 2 shows the traffic situation in Hong Kong.

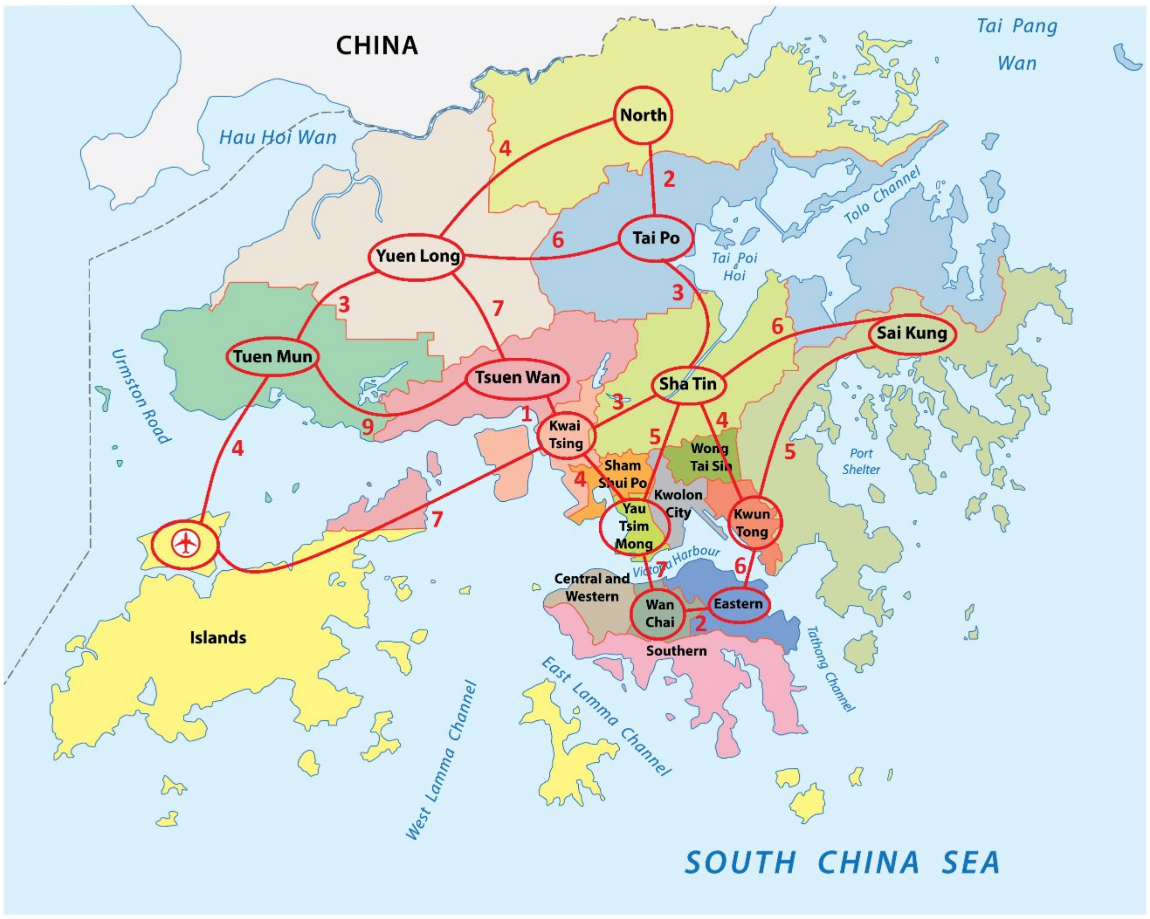


Fig. 2 – Traffic situation in Hong Kong

Let’s denote the districts of Hong Kong by the nodes below:

- Node ***1***: North
- Node ***2***: Tai Po
- Node ***3***: Yuen Long
- Node ***4***: Tuen Mun
- Node ***5***: Tsuen Wan
- Node ***6***: Sha Tin
- Node ***7***: Sai Kung
- Node ***8***: Kwai Tsing
- Node ***9***: Yau Tsim Mong
- Node ***10***: Kwun Tong
- Node ***11***: Wan Chai
- Node ***12***: Eastern
- Node ***13***: Airport

The traffic cost between any two nodes is labeled by the red integer number.

Open the file "**Task2.m**" in the MATLAB editor window. Then, modify the code to create an array to represent the weighted graph for the traffic situation in HK.

Check point 2:

1) Write down the **array you created** to represent the weighted graph.

2) Find the shortest path and the cost from **North** to **Eastern**

Part II: Robotic car

Task 3 – Introduction to the robotic car

In this task, we will introduce the robotic car, which you will use for the final project.
The car is shown below:

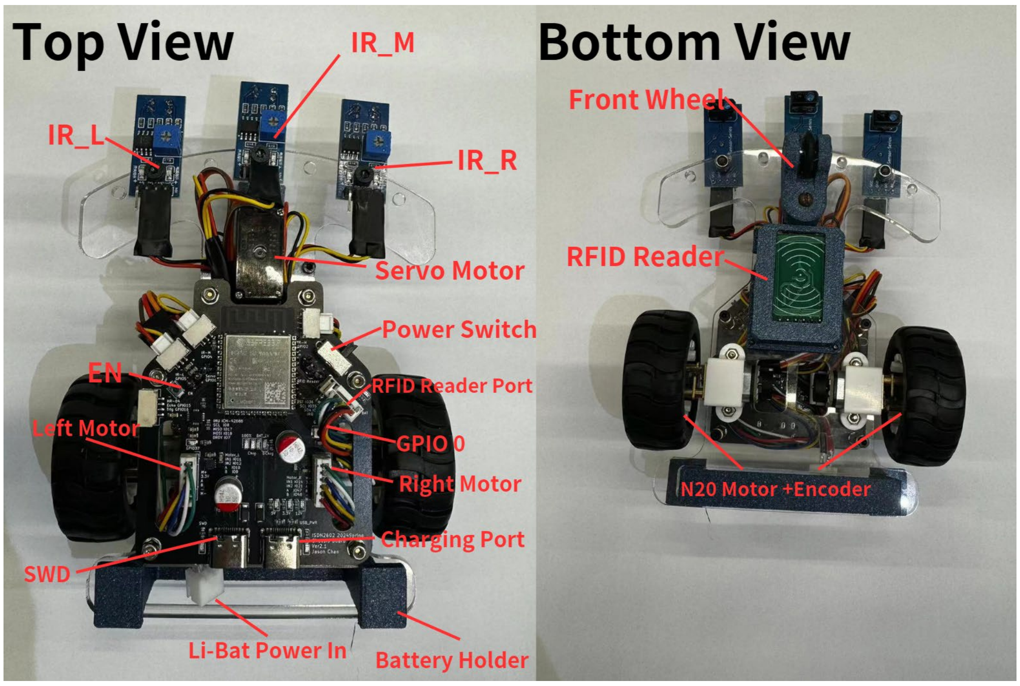


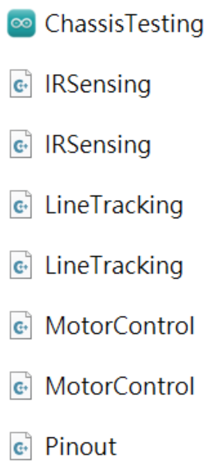
Figure 3: Composition of the robotic car

First, check whether the wires are connected properly.
Then, turn on the car. Check whether all the power indicators are on and whether the battery level is enough to drive the motor.

Download the skeleton codes from GitHub:

*****DO NOT change the code UNLESS it is specified**, especially "Pinout.cpp"***

The skeleton codes should contain the following files:



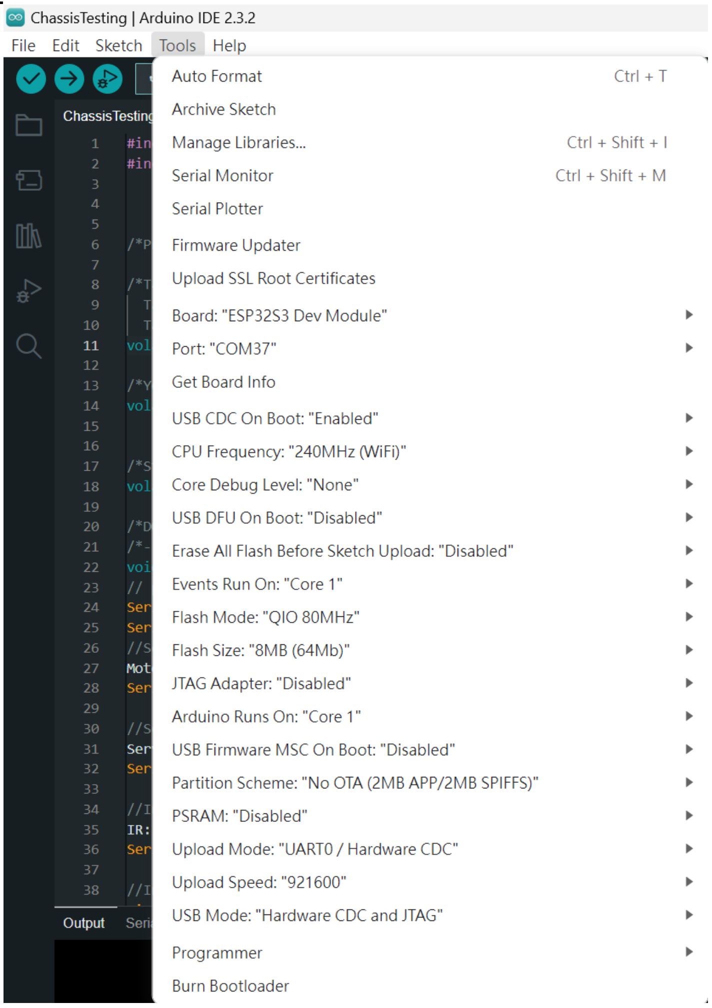
ChassisTesting
IRSensing
IRSensing
LineTracking
LineTracking
MotorControl
MotorControl
Pinout

In this lab, we will only use **motor drivers** and **IR Sensors** for the **line tracking**.

Open “**ChassisTesting.ino**”.

****IMPORTANT****

Since the development board does not have a pre-set environment in the Arduino default board library, please follow the Upload Setting shown below. Otherwise the serial port and flash in MCU may not work properly.

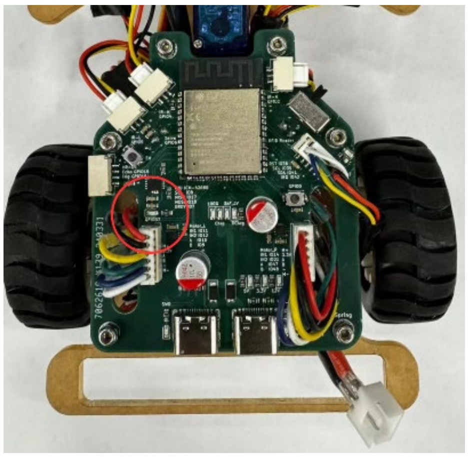


Board: "ESP32S3 Dev Module"
Port: "COM37"
USB CDC On Boot: "Enabled"
CPU Frequency: "240MHz (WiFi)"
Core Debug Level: "None"
USB DFU On Boot: "Disabled"
Erase All Flash Before Sketch Upload: "Disabled"
Events Run On: "Core 1"
Flash Mode: "QIO 80MHz"
Flash Size: "8MB (64Mb)"
JTAG Adapter: "Disabled"
Arduino Runs On: "Core 1"
USB Firmware MSC On Boot: "Disabled"
Partition Scheme: "No OTA (2MB APP/2MB SPIFFS)"
PSRAM: "Disabled"
Upload Mode: "UART0 / Hardware CDC"
Upload Speed: "921600"
USB Mode: "Hardware CDC and JTAG"

Figure 4: Upload setting

Click “**Upload**” and wait for the code to be uploaded to MCU.

The LED highlighted in the following photo will start blinking.



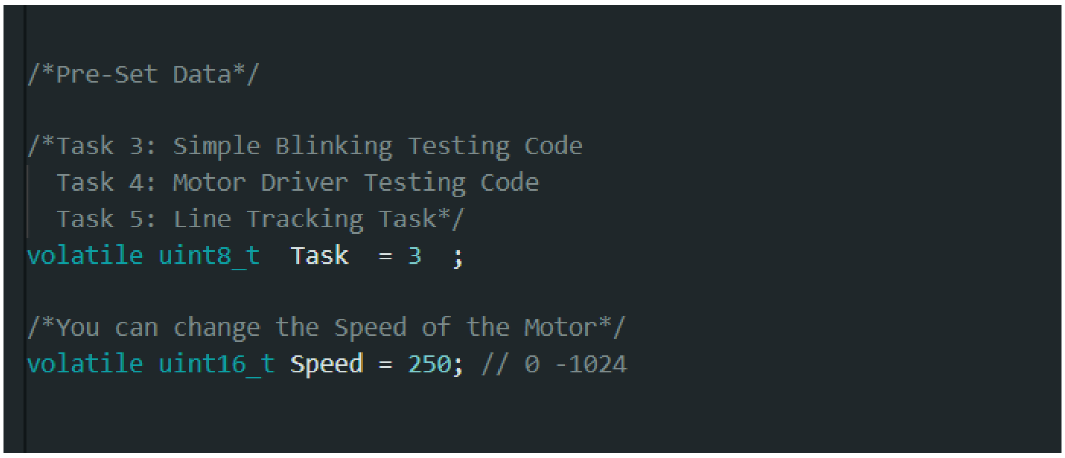
<u>Check point 3:</u>

1) Check all the wires are connected.
2) Plug the Li-Bat into the Power Plug.
3) Turn on the car.
4) Upload the testing code.
5) Check whether the LED on the Chassis is blinking or not.

Show the results to the TA.

<u>Task 4 – Test the movement of the car</u>

Go to “**ChassisTesting.ino**”,
Set the volatile uint8_t Task = 4;

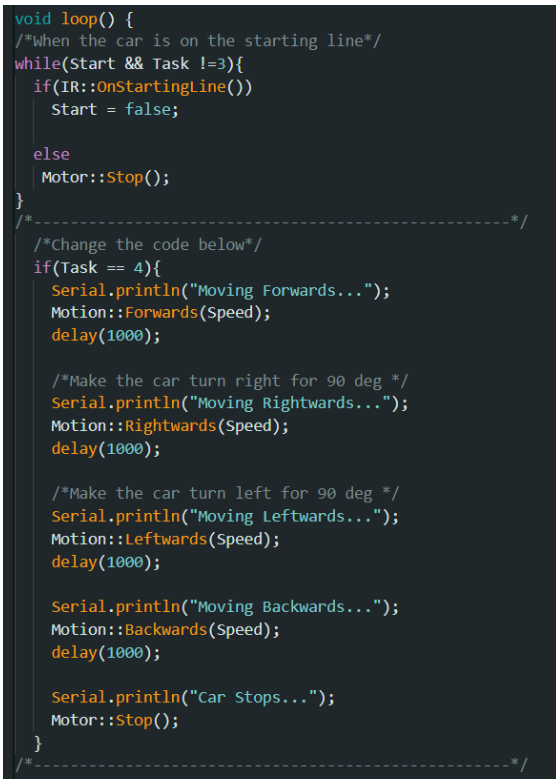


```
/*Pre-Set Data*/

/*Task 3: Simple Blinking Testing Code
  Task 4: Motor Driver Testing Code
  Task 5: Line Tracking Task*/
volatile uint8_t  Task  = 3  ;

/*You can change the Speed of the Motor*/
volatile uint16_t Speed = 250; // 0 -1024
```

Functions given:

`void Motion::Forwards(uint16_t Speed)` : The car will move forward when this function is called.
`void Motion::Backwards(uint16_t Speed):` The car will move backward when this function is called.
`void Motion::Rightwards(uint16_t Speed):` The car will turn right when this function is called.

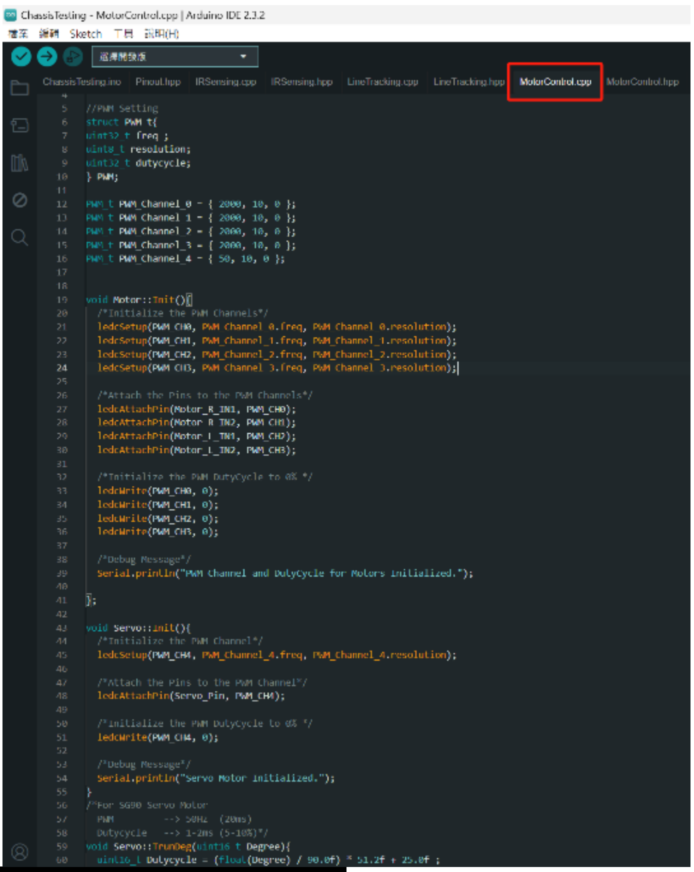
`void Motion::Leftwards(uint16_t Speed)`:The car will turn left when this function is called.
`void Motor::Stop()`: The car will stop when this function is called.



```
void loop() {
/*When the car is on the starting line*/
while(Start && Task !=3){
  if(IR::OnStartingLine())
    Start = false;

  else
   Motor::Stop();
}
/*---------------------------------------------------*/
  /*Change the code below*/
  if(Task == 4){
    Serial.println("Moving Forwards...");
    Motion::Forwards(Speed);
    delay(1000);

    /*Make the car turn right for 90 deg */
    Serial.println("Moving Rightwards...");
    Motion::Rightwards(Speed);
    delay(1000);

    /*Make the car turn left for 90 deg */
    Serial.println("Moving Leftwards...");
    Motion::Leftwards(Speed);
    delay(1000);

    Serial.println("Moving Backwards...");
    Motion::Backwards(Speed);
    delay(1000);

    Serial.println("Car Stops...");
    Motor::Stop();
  }
/*---------------------------------------------------*/
```

If the two motors do not output the same speed, you may need to fine-tune the code (calibrition).

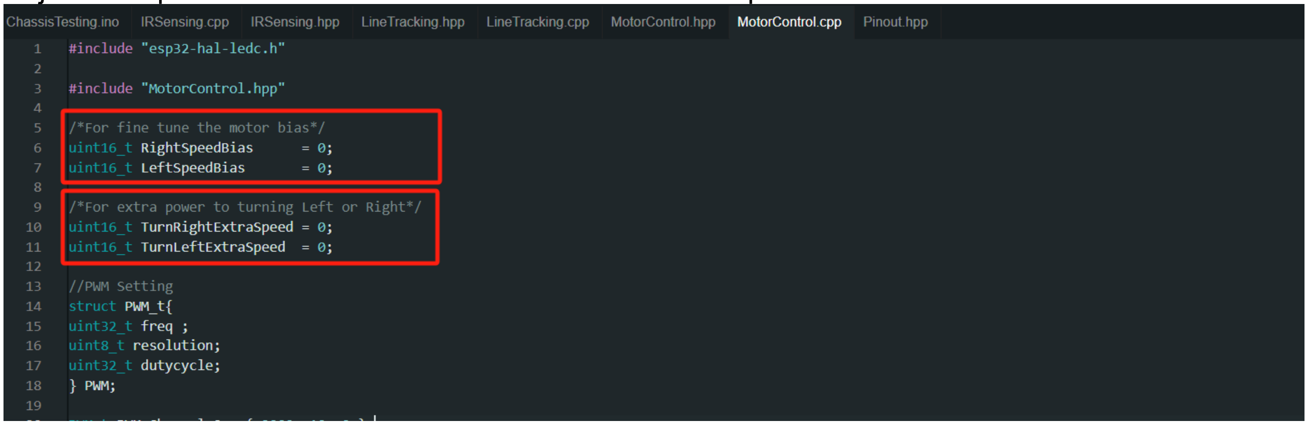
Open the “**MotorControl.cpp**”



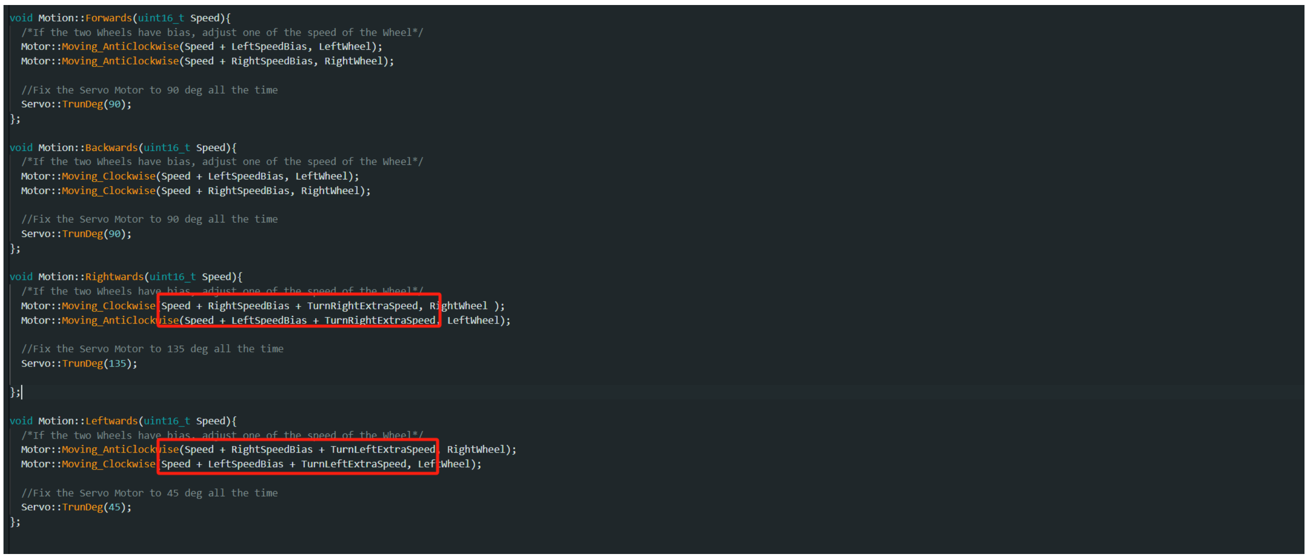
ChassisTesting - MotorControl.cpp | Arduino IDE 2.3.2

```
//PWM Setting
struct PWM_t{
uint32_t freq ;
uint8_t resolution;
uint32_t dutycycle;
} PWM;

PWM_t PWM_Channel_0 = { 2000, 10, 0 };
PWM_t PWM_Channel_1 = { 2000, 10, 0 };
PWM_t PWM_Channel_2 = { 2000, 10, 0 };
PWM_t PWM_Channel_3 = { 2000, 10, 0 };
PWM_t PWM_Channel_4 = { 50, 10, 0 };


void Motor::Init(){
  /*Initialize the PWM Channels*/
  ledcSetup(PWM_CH0, PWM_Channel_0.freq, PWM_Channel_0.resolution);
  ledcSetup(PWM_CH1, PWM_Channel_1.freq, PWM_Channel_1.resolution);
  ledcSetup(PWM_CH2, PWM_Channel_2.freq, PWM_Channel_2.resolution);
  ledcSetup(PWM_CH3, PWM_Channel_3.freq, PWM_Channel_3.resolution);

  /*Attach the Pins to the PWM Channels*/
  ledcAttachPin(Motor_R_IN1, PWM_CH0);
  ledcAttachPin(Motor_R_IN2, PWM_CH1);
  ledcAttachPin(Motor_L_IN1, PWM_CH2);
  ledcAttachPin(Motor_L_IN2, PWM_CH3);

  /*Initialize the PWM DutyCycle to 0% */
  ledcWrite(PWM_CH0, 0);
  ledcWrite(PWM_CH1, 0);
  ledcWrite(PWM_CH2, 0);
  ledcWrite(PWM_CH3, 0);

  /*Debug Message*/
  Serial.println("PWM Channel and DutyCycle for Motors initialized.");

};

void Servo::init(){
  /*Initialize the PWM Channel*/
  ledcSetup(PWM_CH4, PWM_Channel_4.freq, PWM_Channel_4.resolution);

  /*Attach the Pins to the PWM Channel*/
  ledcAttachPin(Servo_Pin, PWM_CH4);

  /*initialize the PWM DutyCycle to 0% */
  ledcWrite(PWM_CH4, 0);

  /*Debug Message*/
  Serial.println("Servo Motor initialized.");
}
/*For SG90 Servo Motor
  PWM          --> 50Hz  (20ms)
  Dutycycle   --> 1-2ms (5-10%)*/
void Servo::TrunDeg(uint16_t Degree){
  uint16_t Dutycycle = (float(Degree) / 90.0f) * 51.2f + 25.0f ;
```

Go to `void Motion::Forwards(uint16_t Speed)`.
Adjust the speed so that the two wheels run with similar speed.



```
#include "esp32-hal-ledc.h"

#include "MotorControl.hpp"

/*For fine tune the motor bias*/
uint16_t RightSpeedBias       = 0;
uint16_t LeftSpeedBias        = 0;

/*For extra power to turning Left or Right*/
uint16_t TurnRightExtraSpeed = 0;
uint16_t TurnLeftExtraSpeed  = 0;

//PWM Setting
struct PWM_t{
uint32_t freq ;
uint8_t resolution;
uint32_t dutycycle;
} PWM;
```



```
void Motion::Forwards(uint16_t Speed){
  /*If the two Wheels have bias, adjust one of the speed of the Wheel*/
  Motor::Moving_AntiClockwise(Speed + LeftSpeedBias, LeftWheel);
  Motor::Moving_AntiClockwise(Speed + RightSpeedBias, RightWheel);

  //Fix the Servo Motor to 90 deg all the time
  Servo::TrunDeg(90);
};

void Motion::Backwards(uint16_t Speed){
  /*If the two Wheels have bias, adjust one of the speed of the Wheel*/
  Motor::Moving_Clockwise(Speed + LeftSpeedBias, LeftWheel);
  Motor::Moving_Clockwise(Speed + RightSpeedBias, RightWheel);

  //Fix the Servo Motor to 90 deg all the time
  Servo::TrunDeg(90);
};

void Motion::Rightwards(uint16_t Speed){
  /*If the two Wheels have bias, adjust one of the speed of the Wheel*/
  Motor::Moving_Clockwise(Speed + RightSpeedBias + TurnRightExtraSpeed, RightWheel );
  Motor::Moving_AntiClockwise(Speed + LeftSpeedBias + TurnRightExtraSpeed, LeftWheel);

  //Fix the Servo Motor to 135 deg all the time
  Servo::TrunDeg(135);

};

void Motion::Leftwards(uint16_t Speed){
  /*If the two Wheels have bias, adjust one of the speed of the Wheel*/
  Motor::Moving_AntiClockwise(Speed + RightSpeedBias + TurnLeftExtraSpeed, RightWheel);
  Motor::Moving_Clockwise(Speed + LeftSpeedBias + TurnLeftExtraSpeed, LeftWheel);

  //Fix the Servo Motor to 45 deg all the time
  Servo::TrunDeg(45);
};
```

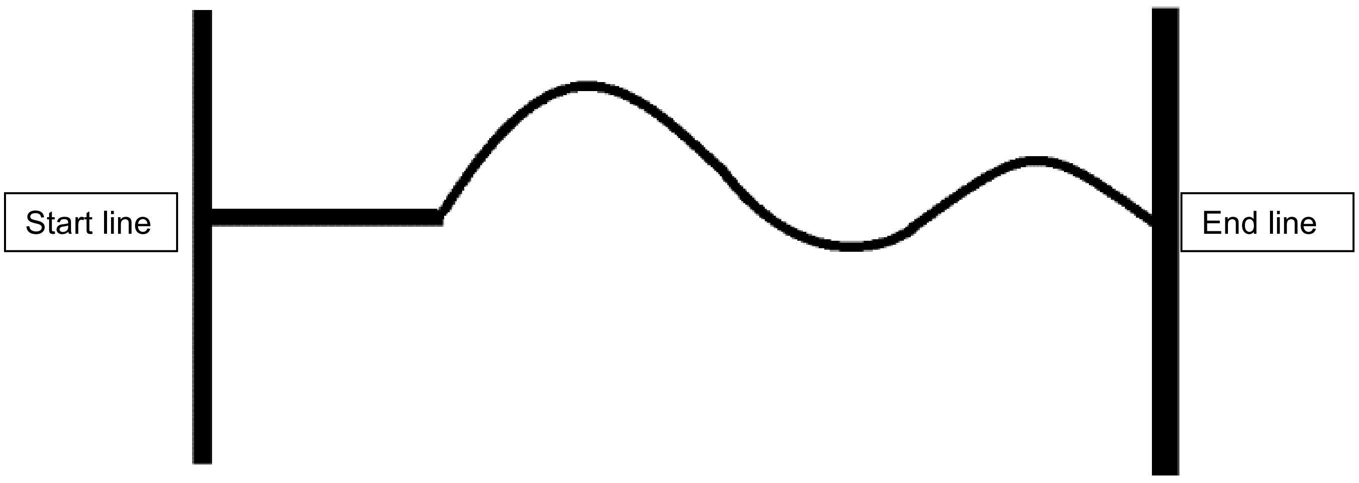
Check point 4:

1) Test the movement of the car with the following sequences:
   (i) Move forward: 2 – 3 seconds.
   (ii) Turn right: 90 degrees.
   (iii) Turn left: 90 degrees.
   (iv) Move backward: 1 – 2 seconds.
   (v) Stop the car.

   Show the results to the TA.

Task 5 – Basic Line Tracking

The line tracking map is shown below:



Procedures:

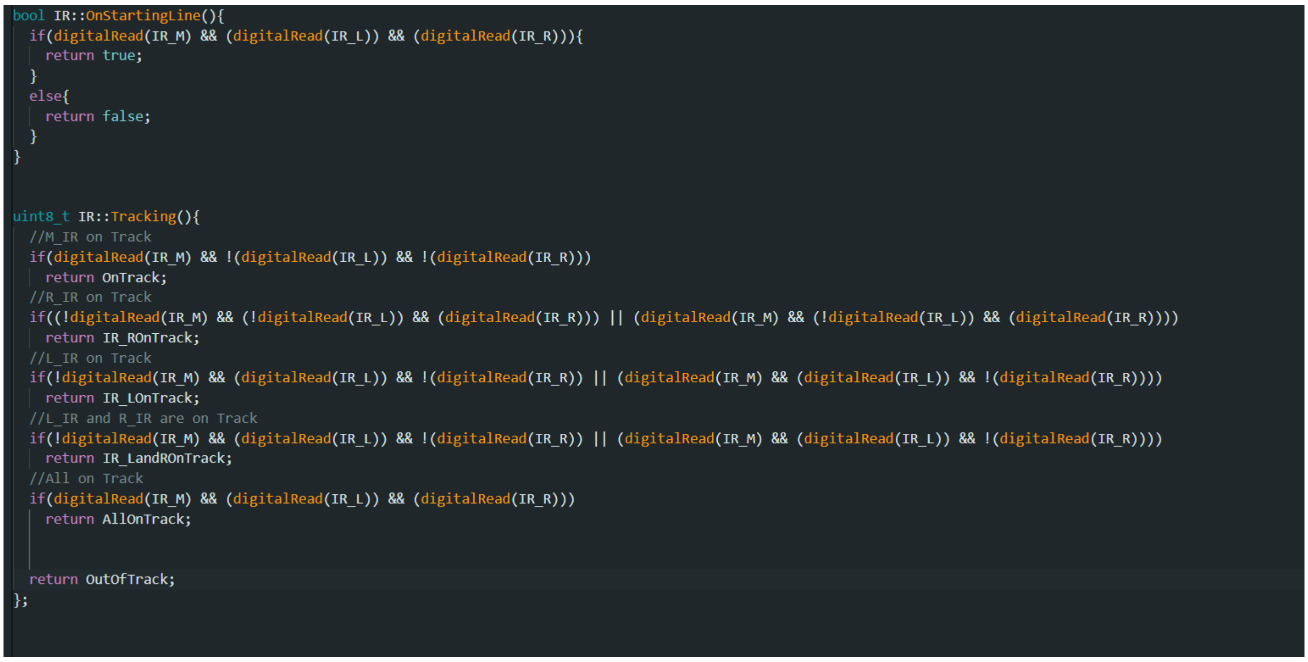
1. Put the car at the start line.
2. Power on the car.
3. After 1 – 2 seconds, the car starts to move.
4. Follow the track and stop at the end line.

Tip 1: The car has three IR sensors, and the output of the sensor will be “**1**” and “**0**” when the sensor is above the “**dark**” and “**light**” surfaces, respectively. We can use three sensors to track a line with the following logic.
If the **middle sensor** is on the dark line, keep moving.
If the **left sensor** is on the dark line, turn left.
If the **right sensor** is on the dark line, turn right.
You may design a **truth table** to complete this task. A truth table defines how you want the MCU to control the motors for given inputs from the sensors.

Left Sensor (IR_L)	Middle Sensor (IR_M)	Right Sensor (IR_R)	Action
0	0	0	
0	0	1	

Code provided:
In “**IRSensing.cpp**”

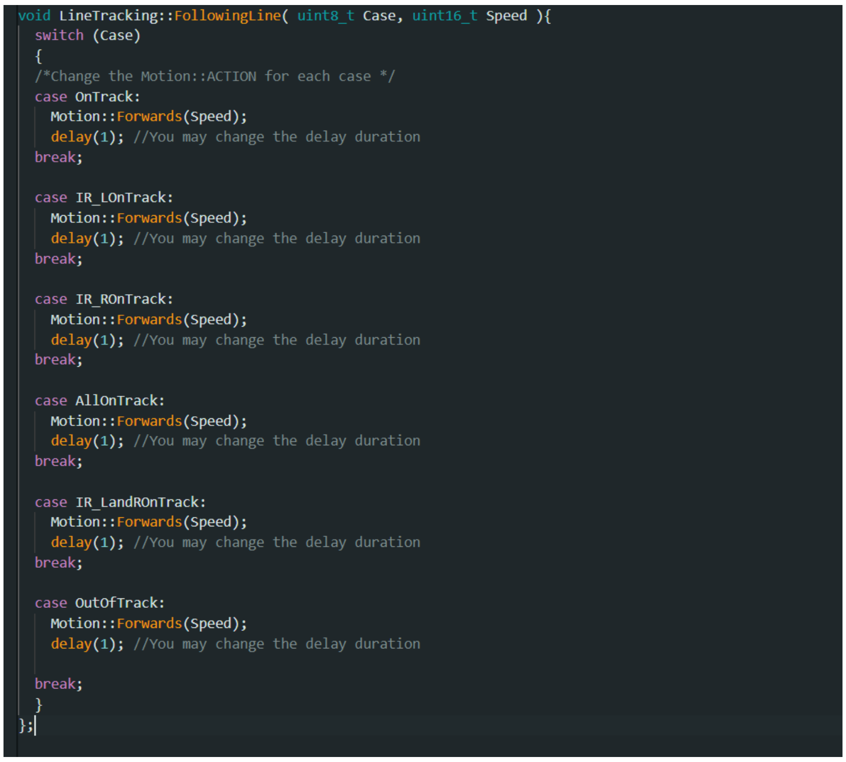
```
bool IR::OnStartingLine(): For the car to stop at the start line.
uint8_t IR::Tracking()    : Based on the conditions of 3 IR Sensors, output different
                             status for further decision making.
```



```
bool IR::OnStartingLine(){
  if(digitalRead(IR_M) && (digitalRead(IR_L)) && (digitalRead(IR_R))){
    return true;
  }
  else{
    return false;
  }
}


uint8_t IR::Tracking(){
  //M_IR on Track
  if(digitalRead(IR_M) && !(digitalRead(IR_L)) && !(digitalRead(IR_R)))
    return OnTrack;
  //R_IR on Track
  if((!digitalRead(IR_M) && (!digitalRead(IR_L)) && (digitalRead(IR_R))) || (digitalRead(IR_M) && (!digitalRead(IR_L)) && (digitalRead(IR_R))))
    return IR_ROnTrack;
  //L_IR on Track
  if(!digitalRead(IR_M) && (digitalRead(IR_L)) && !(digitalRead(IR_R)) || (digitalRead(IR_M) && (digitalRead(IR_L)) && !(digitalRead(IR_R))))
    return IR_LOnTrack;
  //L_IR and R_IR are on Track
  if(!digitalRead(IR_M) && (digitalRead(IR_L)) && !(digitalRead(IR_R)) || (digitalRead(IR_M) && (digitalRead(IR_L)) && !(digitalRead(IR_R))))
    return IR_LandROnTrack;
  //All on Track
  if(digitalRead(IR_M) && (digitalRead(IR_L)) && (digitalRead(IR_R)))
    return AllOnTrack;


  return OutOfTrack;
};
```

In “**LineTracking.cpp**”

```
void LineTracking::FollowingLine( uint8_t Case, uint16_t Speed ):
Based on the feedback of IR::Tracking(), this function makes the motors to move.
```



```
void LineTracking::FollowingLine( uint8_t Case, uint16_t Speed ){
  switch (Case)
  {
  /*Change the Motion::ACTION for each case */
  case OnTrack:
    Motion::Forwards(Speed);
    delay(1); //You may change the delay duration
  break;

  case IR_LOnTrack:
    Motion::Forwards(Speed);
    delay(1); //You may change the delay duration
  break;

  case IR_ROnTrack:
    Motion::Forwards(Speed);
    delay(1); //You may change the delay duration
  break;

  case AllOnTrack:
    Motion::Forwards(Speed);
    delay(1); //You may change the delay duration
  break;

  case IR_LandROnTrack:
    Motion::Forwards(Speed);
    delay(1); //You may change the delay duration
  break;

  case OutOfTrack:
    Motion::Forwards(Speed);
    delay(1); //You may change the delay duration

  break;
  }
};
```

Change the movement function inside the switch case to fulfill the functions defined in the truth table.

Check point 5:

1) **Show the results to the TA.**

Fill the answers in the blanks for Part I.
Commit the revised codes of Part II to GitHub
Show your result to the TA.

---------------------------------End----------------------------------