VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Thao Tran

REMOTE-CONTROLLED CAMERA DOLLY

Information Technology
2016

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Thao Tran
Title	Remote-Controlled Camera Dolly
Year	2017
Language	English
Pages	71
Name of Supervisor	Santiago Chavez

The motions of cameras play a critical role in cinematography. People have been trying to achieve the steadiest, the most stable camera movement in order to capture the most-qualified scenes and pictures since the old days. This level of preciseness is nearly impossible to be attained by humans, or at least requires a considerable amount of skills and effort. However, the robotics industry has simplified the problem. A robot, generally called “camera dolly”, has gradually replaced humans in controlling the camera movements. This thesis project focused on the electrical and embedded aspects in building a camera dolly.

In this project, two independent camera dollies were built. The first camera dolly had a fixed moving pattern since it has to move on a one-meter curved track. The second camera dolly, on the other hand, can move deliberately on the ground, however, still maintain the balance of the camera regardless of ground topography.

Both camera dolly robots have the same specific requirements. The rolling speed of the wheels has to be precise, so that the camera moves to the right location in time. In addition, to carry a camera and a battery system, the robot must be quite heavy, thus, the motors have to be strong enough to move the whole system. In the mechanical aspect, both robots should be 1.2 meters high, and while one robot moves on a track like a mini tram, the other moves around on the floor as instructed by controller. This thesis did not focus on the mechanical aspects, otherwise, it fulfilled the electrical and programming areas.

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION

When the project was proposed to me, there were some requirements from the customer. Those requirements are shown in Table 1.

Table 1. Specifications

	Dolly train	Dolly robot
Moving Patterns	On a track made from metals, plastic pipes. The track is a slightly curve arc designed by another student.	Controlled by the user. The application to control the robot has forward, backward and turning button, so that the robot can follow the customer's command.
Motor strength	The motor has to be strong enough to carry the robot body and the camera	
Controlling unit	The two robot is controlled by phone or tablet. The operating system is Android.	
Accuracy	The accuracy level of the motors has to be extremely high, so that when the robot moves back and forth on the track for a long time, it will not slip away, and the camera will capture the exact moments.	
Stabilization	The stabilization of the camera is the most important feature of the whole system. No matter how well the robots can be controlled, if the camera is not stable, the pictures captured will be unusable.	

There are three phases in this thesis process: solution approach, implementation and testing. In the solution approach phase, the aim is to gather all the possible methods of building the two systems, then sort out all the pros and cons of each method. From the analysis, the best method would then be chosen.

The implementation process is the most time-consuming phase, when all the components should be carefully chosen, the circuit is fully built and the programs should be neatly written. In this phase, the actual possibility of the chosen solution will also be tested, so that the solution can be improved to meet the requirements.

In the testing phase, first, the test will focus on communication and the function of the motors. The motors have to respond to the controlling unit's commands or the gyroscope's measurements. The motors have to be able to turn in both directions, and the speed must be controlled easily. Then, the motors will be placed in a platform to test the moving patterns, the speeds and the accuracy. This is also the most difficult phase, as it shows the flaws in the system, and it requires a fully built platform to test.

2 BACKGROUND AND PURPOSE

2.1 Background

In this project, there are two robots to build. Both robots are used to carry the camera, and move the camera around, which reduces the work for the camera man. The first robot is relatively simple. It is a train-like robot that moves on a specific track. Its movement is an elementary back and forth pattern, in which the users can control to start, stop or just move the robot to the other end of the track. The track should be a curve arc with the diameter of 4 meters, and the arc angle is 60 degrees, which makes the length of the curve 2 meters. This robot is used to film those events when people sit down and talk, which do not involve much moving. However, the user can have some basic control on this robot such as start, stop and move to the end of the track, then stop.

Because we expected that this robot would only film people sitting down on a couch while being interviewed, we decided that the platform should only be 1.2-meter high, as the higher it gets, the less stabilized the camera will be. Figure 1 shows the prototype of the train robot.

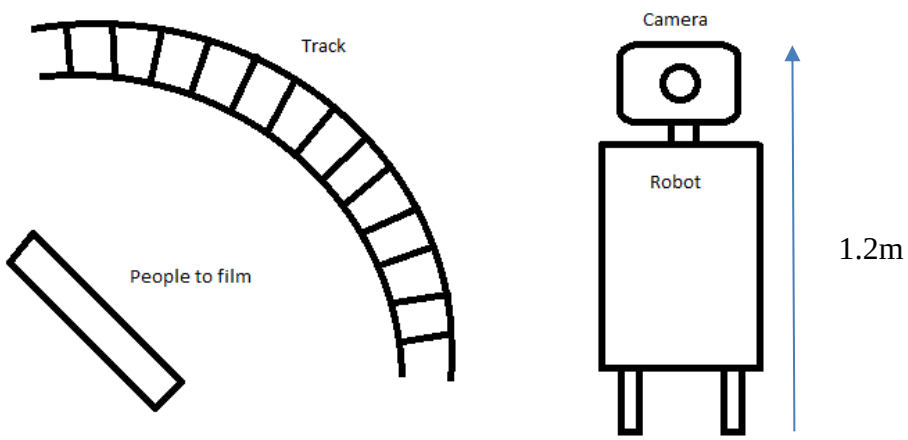


Figure 1: Train robot

The second robot is a remote-controlled robot. Its movement is entirely controlled by the users. This robot requires someone to command, which mean it does not have any

specific pattern to move, and it will not move on its own, without any instruction. Therefore, this robot can move anywhere, but the camera man can not completely rely on it. This robot is used to capture moments in events when people move around, and the space of the events is large.

Because the robot can move freely, its shape does not rely on the track, but relies on the camera pods. The customer is using a tripod to hold the camera, therefore, the platform of this robot should also be triangle, so that it contributes part of the stabilizing effects. Because the height of the tripod is adjustable, the robot can be fixed at 1.2-meter high for a strong, stable platform. Figure 2 shows the prototype of the human-controlled robot.

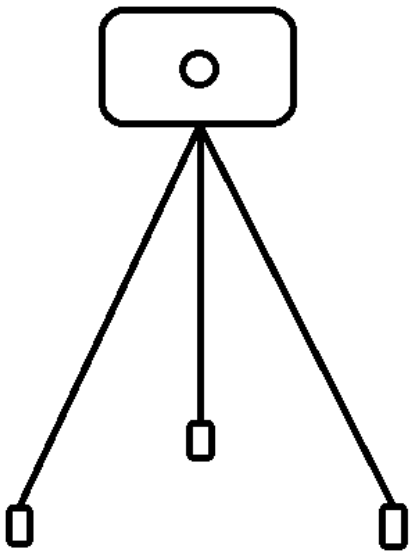


Figure 2: Human-controlled robot

2.2 Purpose

To the customer, this project would be a very helpful assistance. Obviously, a machine has a better accuracy and duration when compared to a human being. Therefore, using robots to carry the camera has been an innovative idea for a long time. Not only this robot will do a better job than a human being will, it also reduces the amount of work humans have to carry out during the filming process.

When working with this project, I also earned knowledge about an actual embedded system in the real life. I had the chance to build the robots from scratch, to choose the components, and to learn about the new programming language that I barely knew before.

3 THEORETICAL BACKGROUND

3.1 Electrical design

To meet the biggest requirements, moving and stabilizing, two systems were designed. The first system would be responsible for the motion of the robots, and the second system would maintain the balance of the cameras.

MOTION

Every moving system needs motors. Whether in a rolling motion of a turbine, or a reciprocating motion of an internal combustion engine, the motor is a critical part. Therefore, the question “Which motors should be used in this system?” always comes up first whenever a design work is starting. There are two main types of motors: AC motors and DC motors. While AC motor has low power demand on start and an adjustable torque limit, DC motor tend to start, stop or reverse more rapidly, thus, they are easier to control. AC motors can be divided into two categories: asynchronous and synchronous AC motors. Meanwhile, there are two main types of DC motors: brushed and brushless. The differences of those types of motors are listed in Table 2.

Table 2. Motors comparison /1/

	Asynchronous AC motors	Synchronous AC motors	DC brushless motors	DC brushed motors
Key feature	Using electromagnetic to produce a current for the rotor.	Rotations is synchronized with the frequency.	Motor controller uses Hall effect sensors to detect motor position.	There are 4 types: series wound, shunt wound, compound wound and permanent magnet, each works differently.

Advantage	They have various load capacity, from smaller for household purposes to larger for industry.	The speed does not change when the motor loads.	Accurately control the motors' speed.	Relatively simple.
Disadvantage	Speed control is difficult.	There is no possibility of speed adjustment.	High initial cost, complicated controller.	Short lifespan.
Applications	Used in industry and household system.	Mostly used in equipment with constant speed.	Used in speed and position controlled applicator.	Used in low-cost system.
Example	Lifts, lathe machines	Used in power station and industry	Fans, pumps, compressors	Automotive, vacuum cleaner

In both robots, the DC brushless motors are chosen to build the motion system.

Next, to control a motor, a controller is needed. The controlling module can be a Raspberry Pi, an Arduino, a self-made board or any kind of programmable board with pins to connect to the motors. A motor driver should also be used to allow some function of the motors, such as reversing, speed controlling.

Other parts, which are mechanically related, such as the wheels, the rails or the platform for the robots will be made by another student.

STABILIZATION

The camera will surely shake when it is being moved. That is inevitable, however, we would try to eliminate the movements as much as we can, because a shaken photo is useless. And by “eliminate the movements”, I want to say that we try to reverse the movement of the camera, for example, if it were shifted to the left by 1 centimeter, we would move it back to the right by 1 centimeter. By that method, the camera mostly stays still, of course, with the condition that the reverse movement happens shortly after the shift. Based on this theory, a stabilizing system can be designed. To maximum the balancing effects, the system should have a 3-dimension movement, which means, at least three motors would be used.

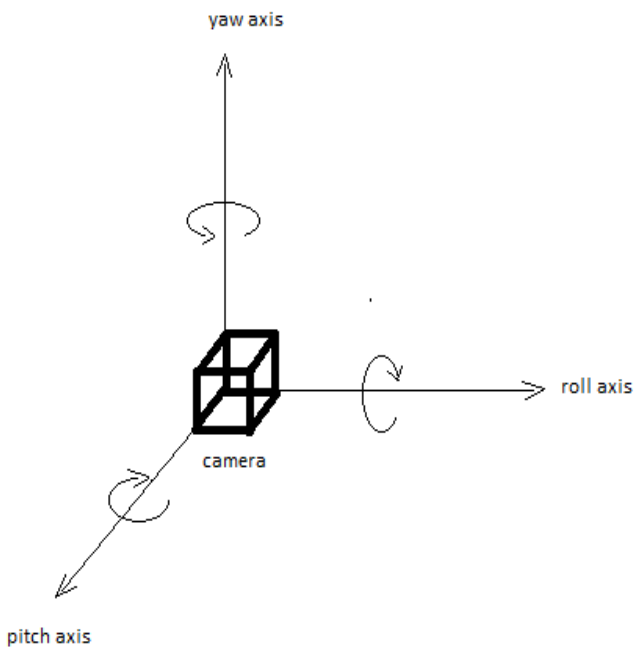


Figure 3: 3-dimension movements to stabilize an object

Normally, a similar stabilizing system is called a gimbal. The motor used to build this system is the gimbal motor. This is a DC brushless motor (BLDC), with a slightly different appearance that can be attached to a platform, so it turns the whole platform, not just

a wheel. However, the composition of the gimbal motor is the same as that of the normal brushless motor.

Figure 4: Example of gimbal motor

This gimbal motor needs high torque and high power to turn, therefore, it cannot be used without a driver. In the market, there are many types of gimbal motor controller, which really make the job easier. However, the price of the controller is quite high, and for each controller, we can only control one motor. One substitution is that the controller can be replaced with a self-built driver. 511-L298 H bridge, L293DNE or SN754410NE driver are some possible options.

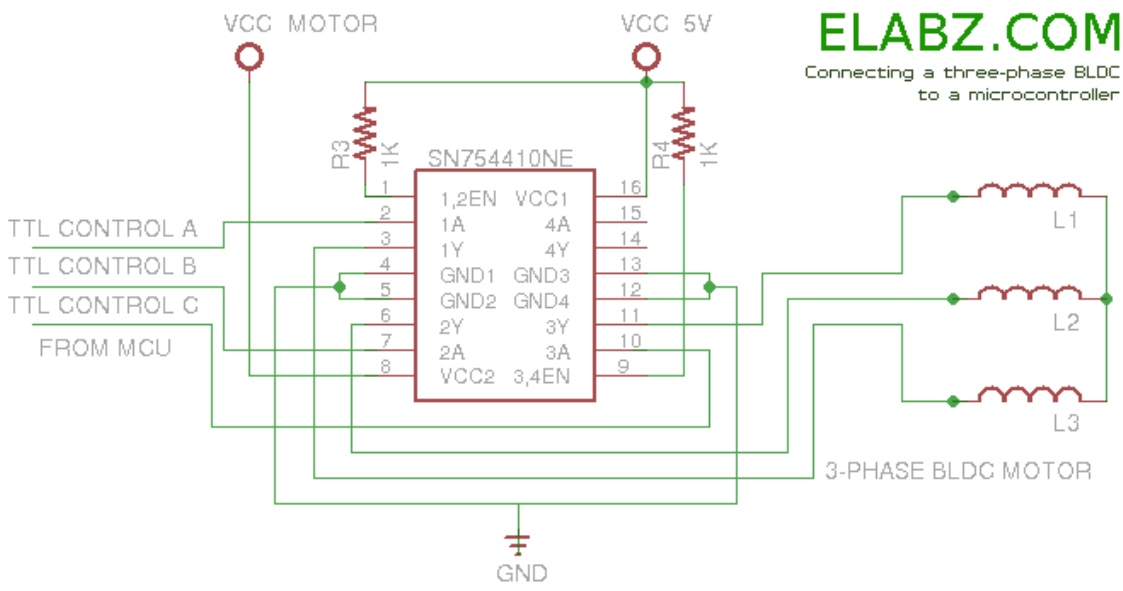


Figure 5: H bridge/3/

The H-bridge is used to control the rotating direction of a brushed DC motor or a bipolar stepper motor. Without this H-bridge, the motor will still operate, but it will not change its rotation direction, which means it can never go in a backward motion.

COMMUNICATION

The train robot is a semi remote-controlled robot, while the other robot will be fully controlled by the users. Therefore, in the controlling application, there are three buttons to run the first robots in different speeds, from SLOW to FAST, 4 buttons to controls the directions of the second robots, and 2 STOP buttons for both robots. There is also a seek bar to control the second robot's speed.

In both robots, to communicate with the controlling device, there should be a Bluetooth module. There are many types of Bluetooth modules. Some of them can be used as a master or slave, however, the other can only be used as a slave. In this project, the HC-06 model, which is a multi-functional Bluetooth module, was used. It can be used as a master or slave, and in this project, it is connected and programmed to be a slave module.

3.2 Software usage

To build the whole project, except for building the circuit, all the Arduino must be programmed and the controlling application should be developed.

To program the Arduino, I used Arduino IDE, which is a development environment for all Arduino board. Except for being an editor, Arduino IDE make it easier to connect the Arduino to the development machine and download the codes to the Arduino. This development tool can be downloaded free of charge from the Arduino website.

The controlling application is a little more complex. It can be a webpage or an Android application. Because this project is designed for a single customer, it would be more convenient to make an Android application that only the customer can access. By doing that, no security wall is needed to prevent the robots from being controlled by unauthorized people.

Android studio is one of the most popular tools for building applications on Android devices. Android studio simplify the developing process, since it automatically produces the XML codes in responding to the Java codes of the programmer. Thank to this software, no XML background is required when building this project.

4 APPROACH AND IMPLEMENTATION

4.1 Solution approach

Both robots, the train robot and the human-controlled robot, can be broken down into three systems: the motion system, the stabilizing system and the communicating system. To simplify the project, I will choose the same controlling board for all systems. The controlling board should be a multi-functional board which has many ports to connect, both digital and analog. Because this will be an affordable product in the end, the ideal price for the board falls between 30e to 80e.

The train robot is a compulsory part of this project. Every requirement of this robot has to be thoroughly fulfilled. The human-controlled robot is an advanced task, which means any solution or idea to build this robot is appreciated.

4.2 Implementation

4.2.1 Choosing components

There are many options for the development boards on the market. Considering the expenses, the Raspberry Pi and the Arduino are the most affordable ones, however, they are still very useful. Besides, they are very easy to find. In this project, I chose Arduino Mega 2560 to be the controller of all systems. In addition of having many ports to connect and a thorough instructive website, the Arduino Mega 2560 is familiar to me.

Arduino Mega 2560 is a large Arduino board with more connecting ports than other Arduino versions such as the Arduino Uno or Arduino MO Pro. For this reason, one Arduino Mega 2560 can control more units. Arduino boards use Atmega8 processor from Atmel.

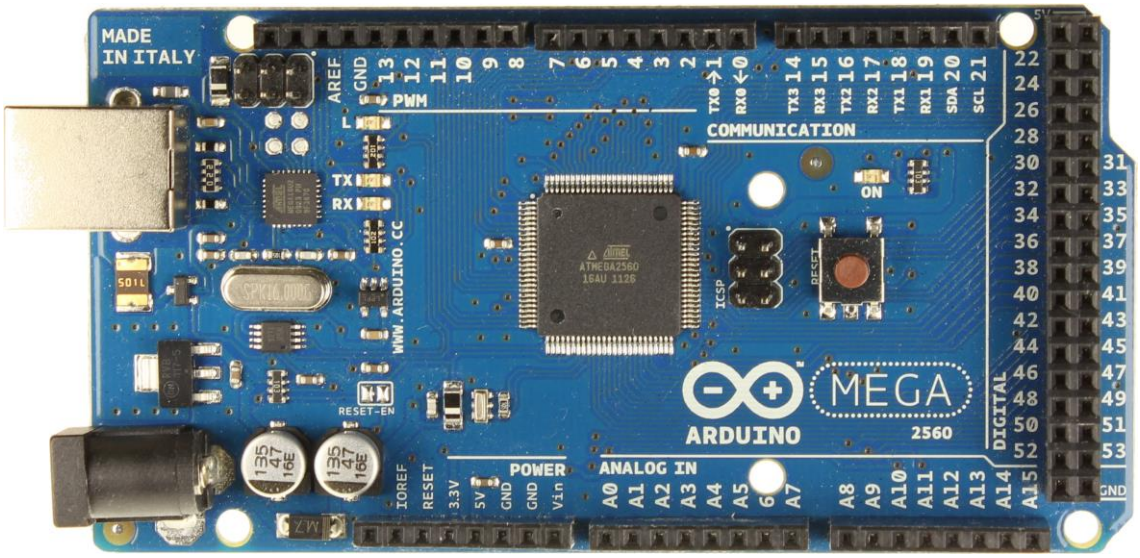


Figure 6: Arduino Mega 2560 /4/

The board comes with a PCB and components such as capacitors. There are 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial port), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller, simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega 2560 board is compatible with most shields designed for the Uno and the former boards Duemilanove or Diecimila. /2/

In an Arduino 2560 board, there are 6 digital pins usable for interrupts, which are 2,3, 18,19, 20 and 21. The channel A and channel B of the motor's encoder must be connected to these pins so that the board can read the encoder. (Figure 13)

MOTION

In addition of the development boards, the system needs the H-bridge and at least two motors to operate. Obviously, I will choose the same types of motors and H bridge for both robots, because they have the exact same functions. For the train robot, I might need 2 to 3 motors. Because in this type of robots, all motors create the same moving direction, the number of motors does not matter. In the human-controlled robot, on the other hand, the movement is unpredictable, therefore, with 3 wheels, it will be so complicated to program the wheels to move based on the user's will. Hence, I will just use 2 motors, each

motor is placed in one side of the robots. To give the balance the robot needs, a ball wheel will be placed as in Figure 7.

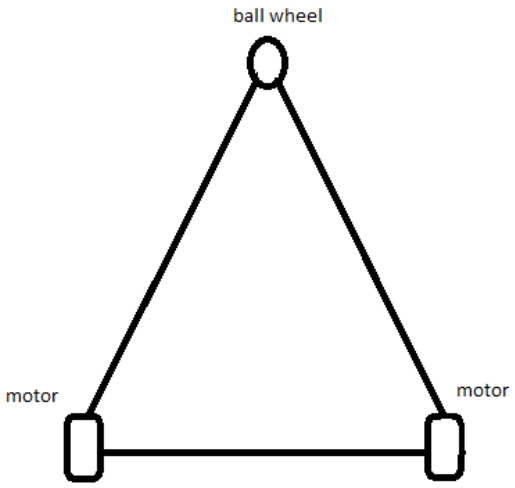


Figure 7: Placement of wheels in human-controlled robot

Any H bridge would do the job of controlling the direction of the motors. I this project, the LMD18201T H bridge is used, simply because it is already available in Technobothnia. This is a 3A H-bridge which can control DC and stepper motors. An advantage of this H bridge is that it can control direction, PWM and brake of the motors. However, one LMD18201T H bridge can only control one motor, therefore, for 2 motors on 2 sides of the robots, 2 LMD18201T are used. Figure 7 shows the pins map of LMD18201T.

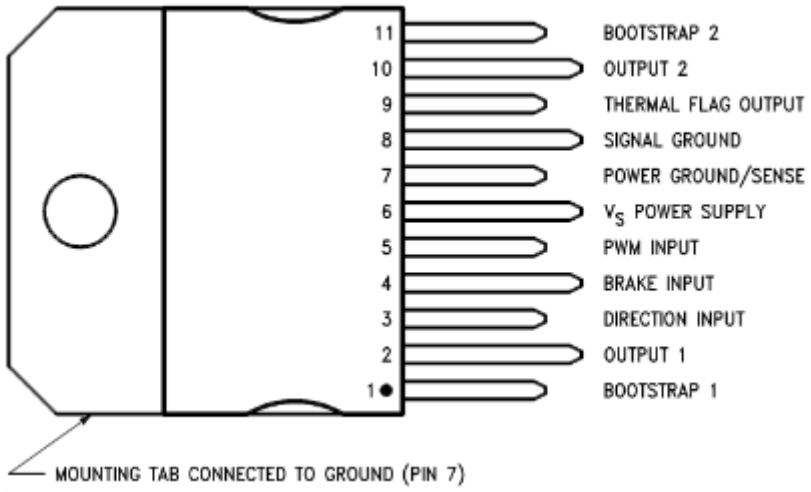


Figure 8: LMD18201T pins

STABILIZATION

The stabilization system also requires H-bridge for controlling the motors. The gimbal motors used in this system are actually 3-phase brushless motors, therefore, a L293 H-bridge will be the most suitable to control this type. Other components such as resistors, capacitors are chosen based on L293 datasheet.

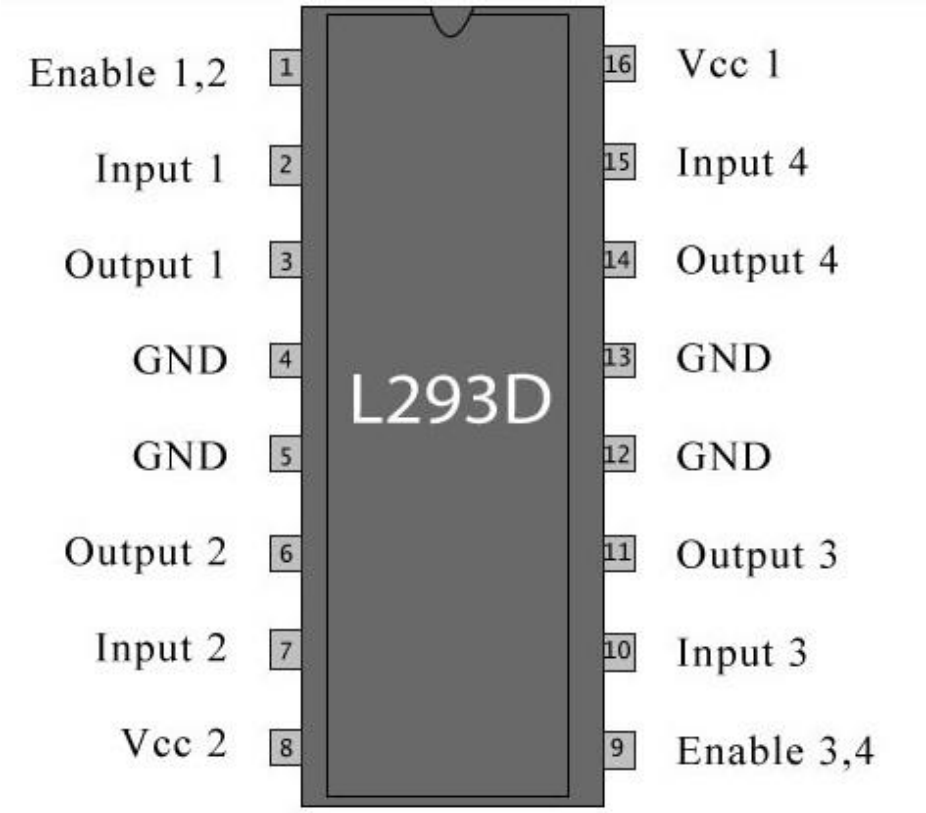


Figure 9: L293D

MCU6050 is a popular gyroscope to use with the Arduino. Hence, it is used in this project to detect the coordinate of the camera object. Three gimbal motors will turn and try to balance the camera according to the gyroscope values.

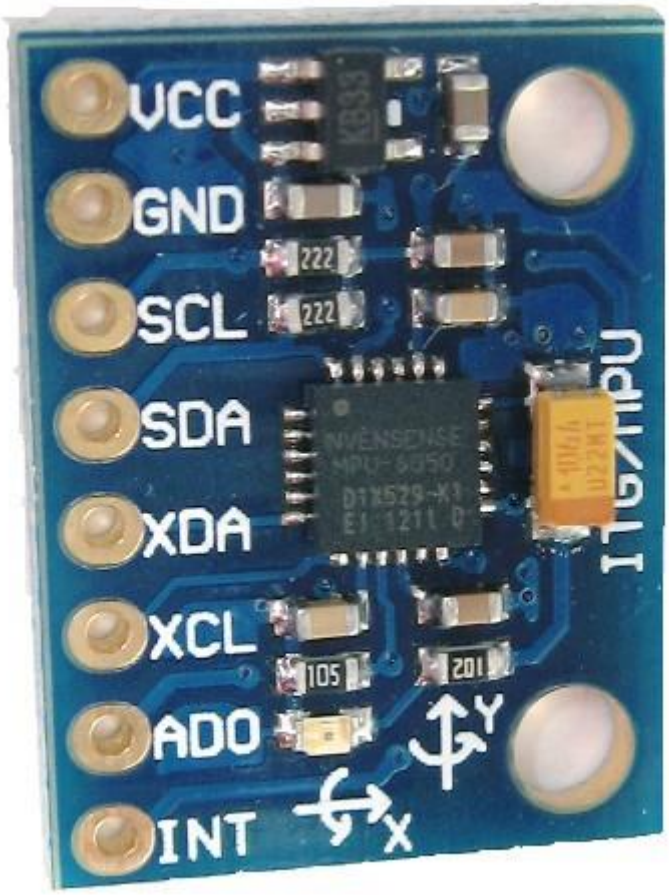


Figure 10: MCU6050 /5/

COMMUNICATION

HC-05 was chosen to be the Bluetooth module of both robots. This Bluetooth module can be programmed to be a master or a slave. The Bluetooth module is inexpensive and easy to find, and its circuit is very simple, therefore, it can also be replaced when broken.

Figure 11:HC-05 /6/

4.2.2 Preparing boards

When programming the Arduino board, the Arduino board s connected to the computer or laptop via a USB type B cable. Then the codes are written in Arduino IDE and downloaded to the board using the method displayed in Figure 10, Figure 11 and Figure 12.

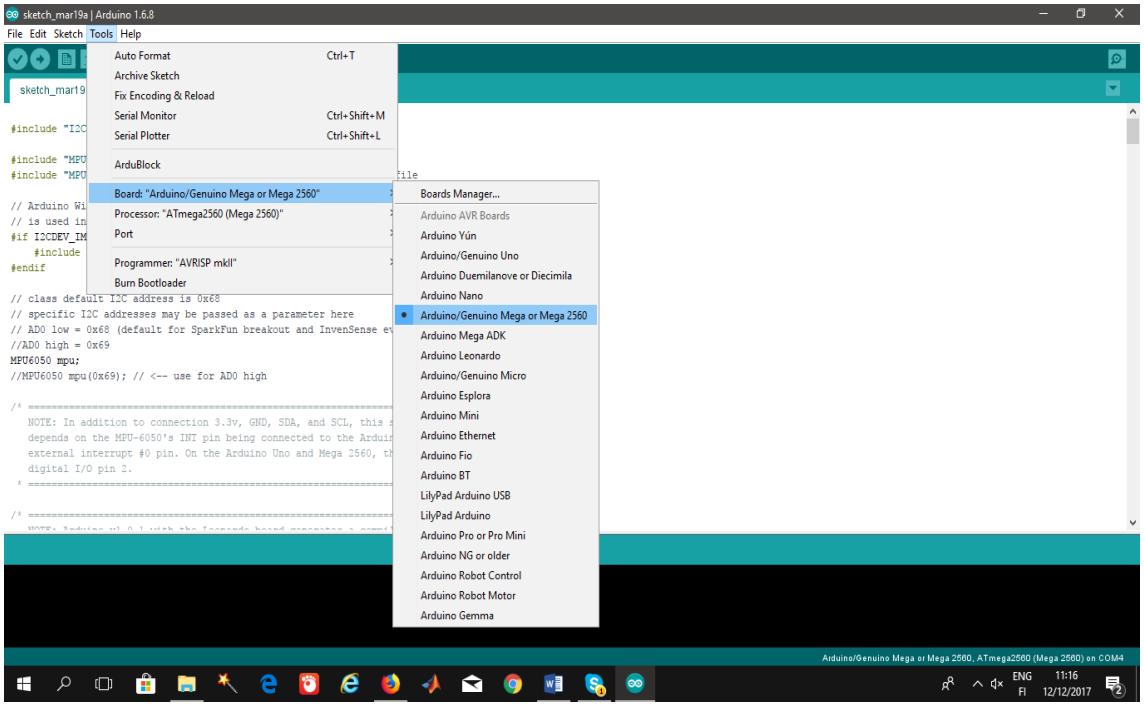


Figure 12: Choosing the type of Arduino

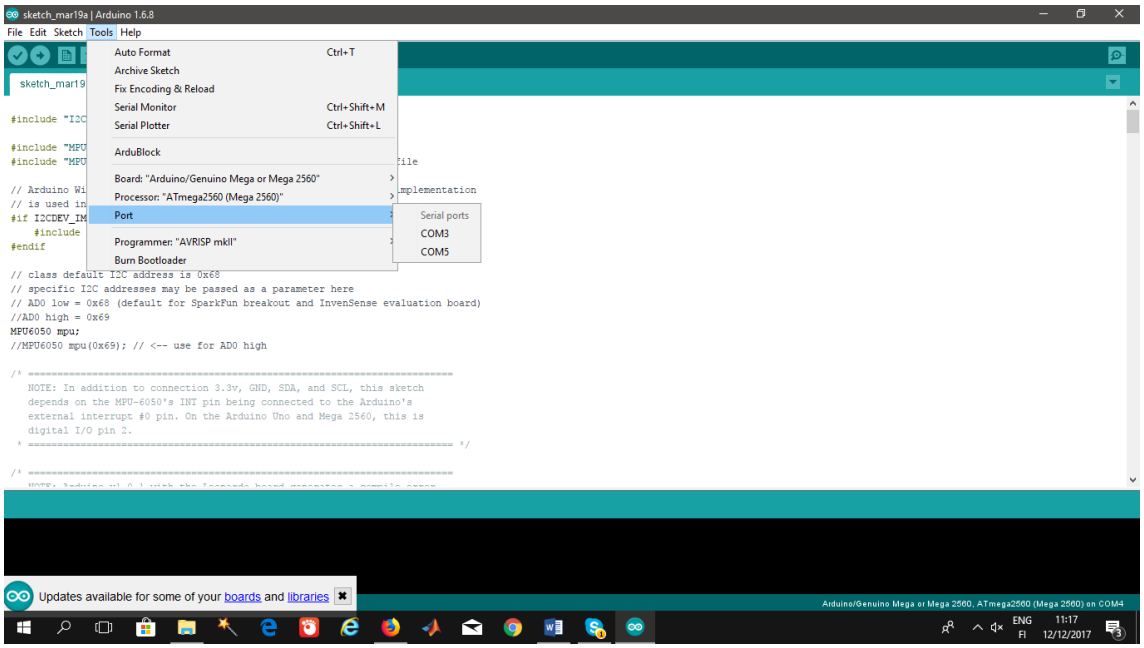


Figure 13: Choosing the port which Arduino is connected to

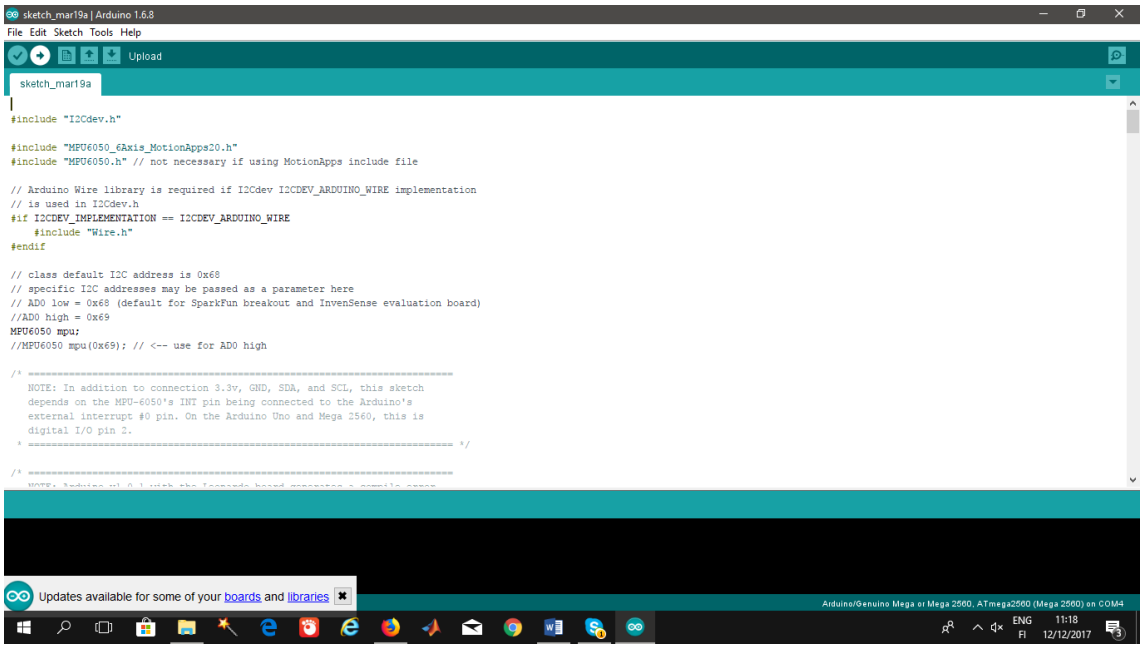


Figure 14: Compile and Upload the codes

4.2.3 Programs and circuits

To make use of the Arduino board, a program should be written and downloaded to the board. In this report, all the codes needed are included for the sake of those who would like to re-create the project, however, only the critical parts will be explained.

Those Arduino programs are created using the open source Arduino Software IDE. The software is used to write and download the code to all Arduino board versions in an easy way.

MOTION

❖ **Circuit**

Using one Arduino board to connect this circuit.

The LMD18201T H bridge (figure 8) has pin 3 and pin 5 as the direction and PWM input pins. These two pins will be connected to two output pins of the Arduino, so that when the Arduino pins change their value, the direction and the speed of the motors also changes.

The connection is made based on LMD18201T datasheet.

The encoder of the motor has six pins: channel A, channel B, motor +, motor-, VCC and GND. Motor+ and motor- pins are connected to the driver as in the datasheet, while channel A and channel B are connected to two interrupt pins of the Arduino. In the first model, I used 5V output of the Arduino to power the motor, however, it was not enough. Therefore, in the later model, VCC was connected to a 9V battery.

A 9V battery is connected to pin 6 of the driver to power up the motor.

The Bluetooth Module HC-05 unit is connected to the RXD and TXD pins of the Arduino, and also powered by a 9V Battery.

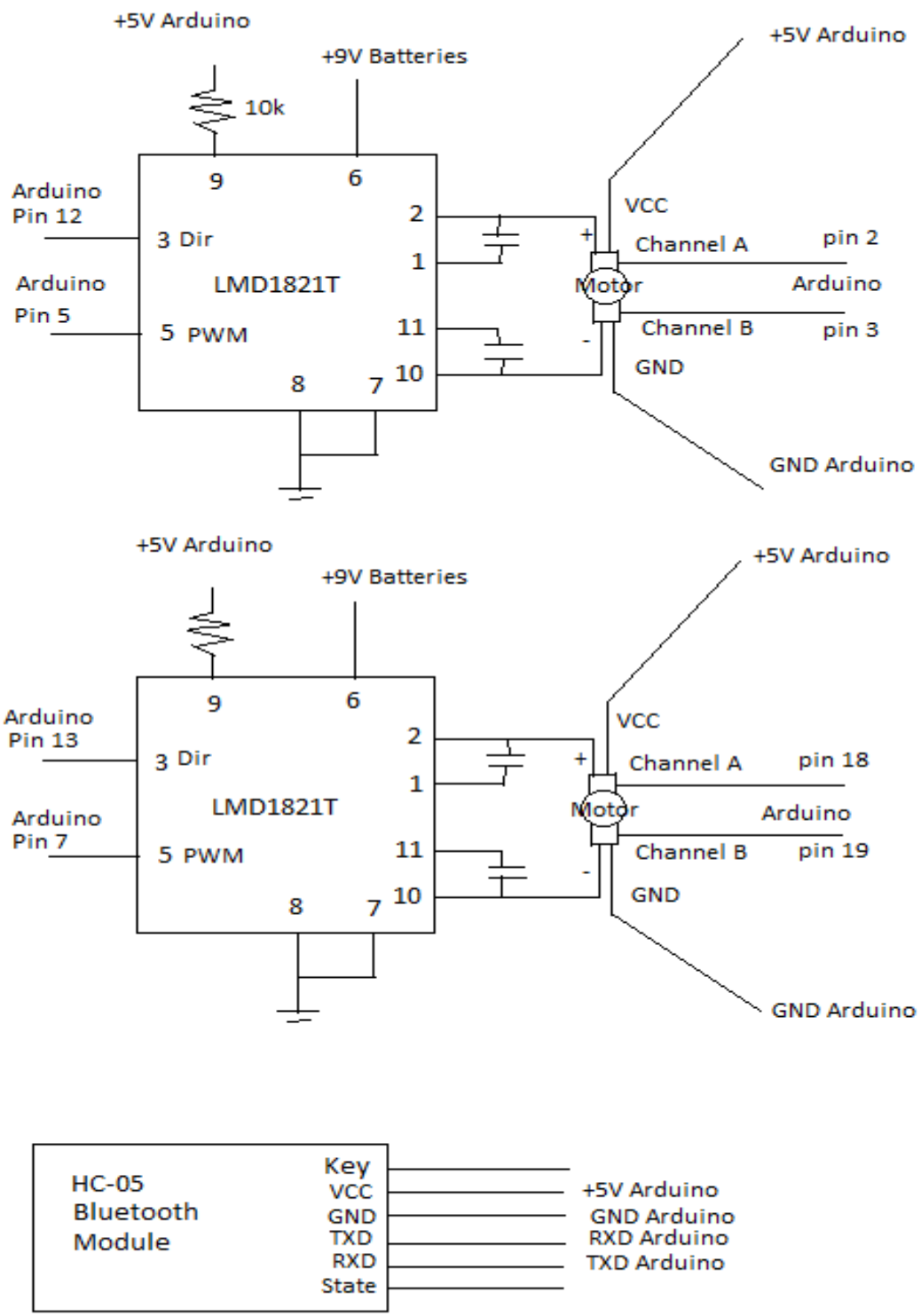
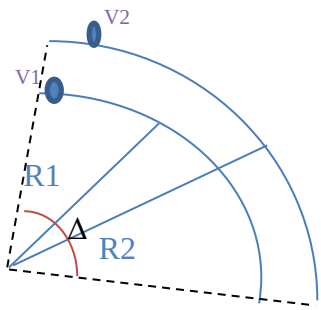


Figure 15: Motion circuit

❖ Program

When the user presses a button on the controlling application, a string of command will be sent to the Arduino via Bluetooth. The program in this Arduino will then capture the command and save as a variable called “String”. The String variable has different values. Each value will trigger a function. For example, if String is T1, under the condition that the train robot is being connected to the controlling device, the train will move at a medium speed. Because the moving pattern is a curve, two wheels of the train will move at different speeds.



For the train robot, assuming that, the speed of the inner wheel is v1 and the speed of the outer wheel is v2. The curve tracks that the wheels move on are the arc of the circle which has the same center and radius alternatively is R1 and R2. The length of the inner curve is S1 and the length of the outer curve is S2.

$$S1 = 2*R1*\pi \quad (1)$$

$$S2 = 2*R2*\pi \quad (2)$$

From (1) and (2):

$$\frac{S1}{S2} = \frac{R1}{R2} \quad (3)$$

Because both wheels have to complete its curve in the same amount of time:

$$T1 = T2 \quad (4)$$

From (3) and (4):

$$\frac{S1}{V1} = \frac{S2}{V2}$$

⇨ $\frac{V1}{V2} = \frac{S1}{S2}$

- ⇨ By this, we can define the speed of both wheels so that the moving pattern remain the same.

Figure 16 is a demonstration of the flow chart of the program that is downloaded to the Arduino controlling the motion system.

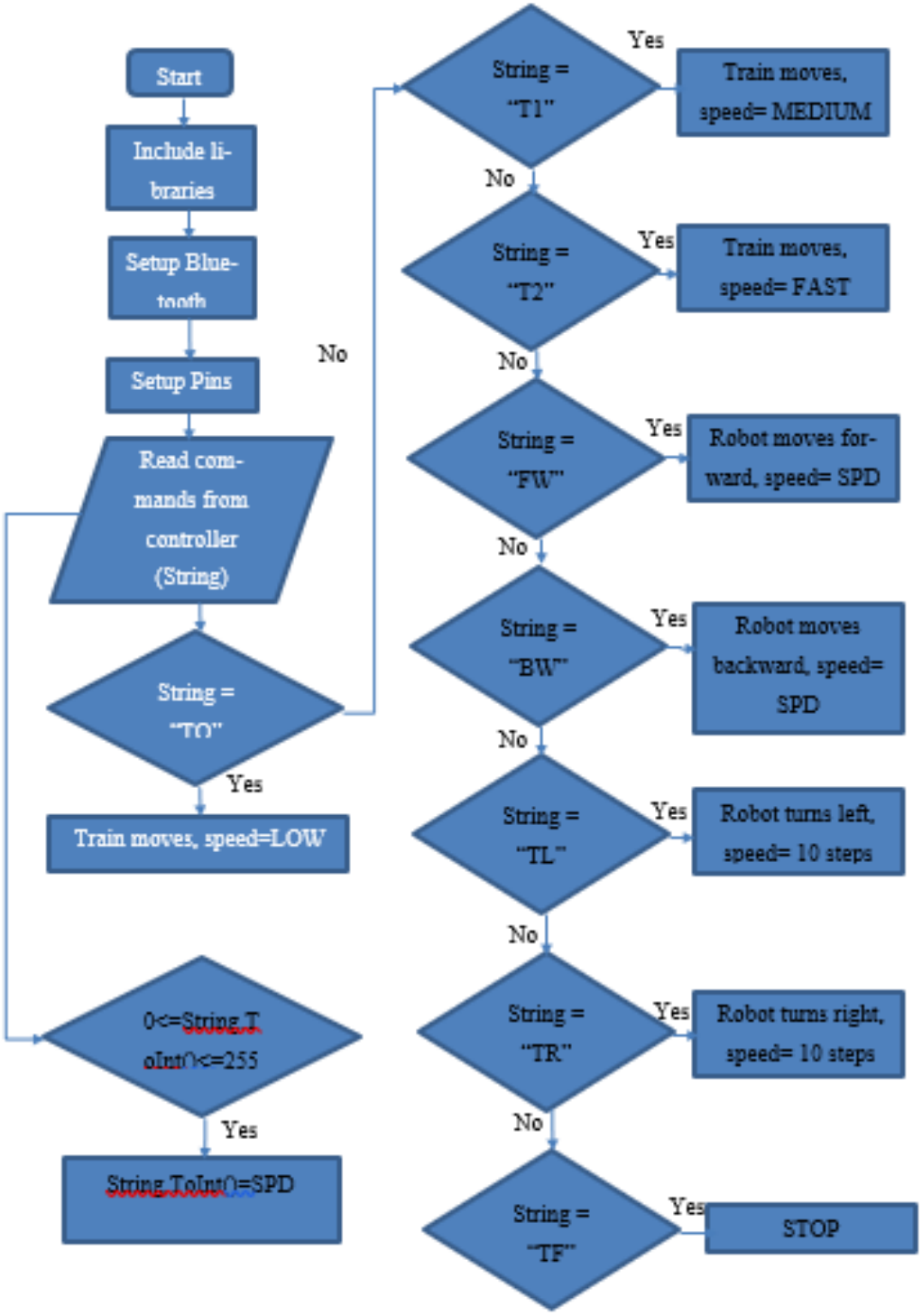


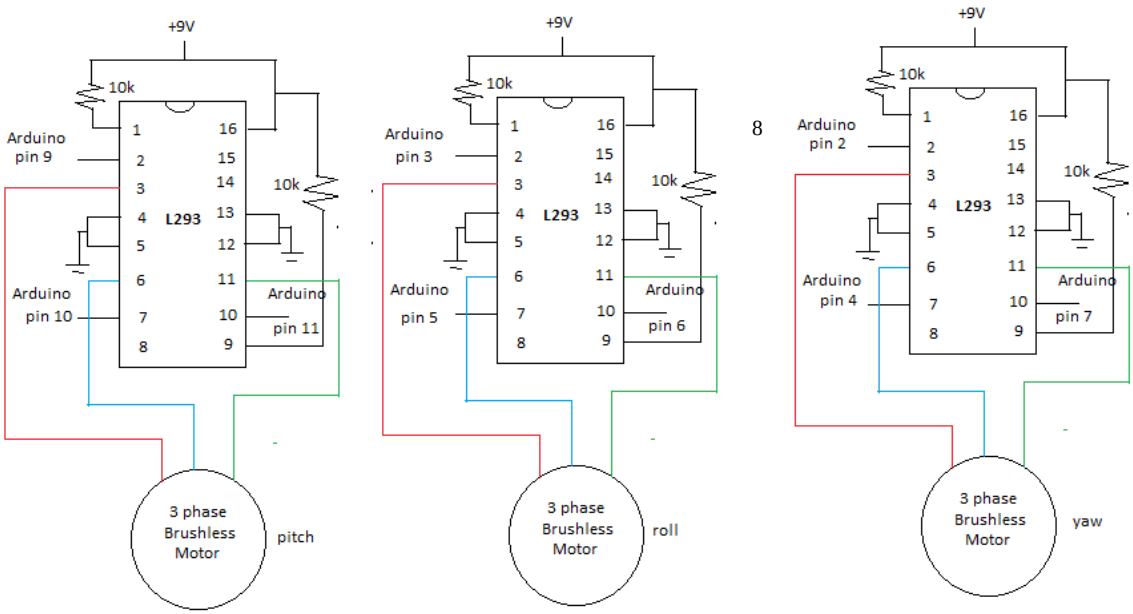
Figure 16: Flow chart for motion system

STABILIZATION

- **Circuit**

  L293D H bridge has 4 input pins and 4 output pins. Each motor is connected to one L293D driver. Three Arduino PWM pins are connected to three input pins and three wires of the motor are connected to three correspondent output pins. By doing that, when adjust the value of the PWM pins in the Arduino, we can adjust the motion of the motor.

  The MCU6050 is connected to the Arduino as instructed in the datasheet.



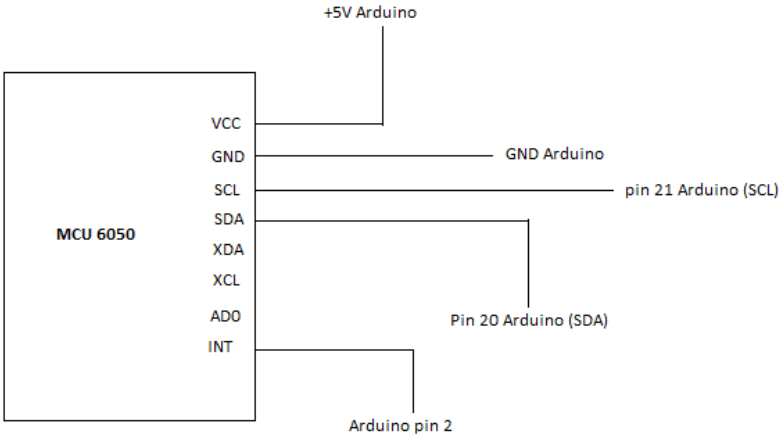


Figure 17: 3 -axis motors circuit

❖ **Program**

MCU6050 acts as both accelerometer and gyroscope. In this project, MCU6050 was used as a gyroscope, which means it is attached to the camera so that it can send the orientation of the camera to the controlling board. A gyroscope is a device which uses the earth's gravity to measure orientation of an object. Orientation of an object is a

coordinate system with the center placed in the center of the object. Every object has its own orientation. Figure 3 shows an object with its orientation system.

The overall idea of the program is simple. The program receives the orientation of the camera. Then, it commands the three motors to turn to maintain the initial orientation of the camera. This program works as long as the robot is still connected to the controlling unit. Figure 19 indicates the logic behind the program.

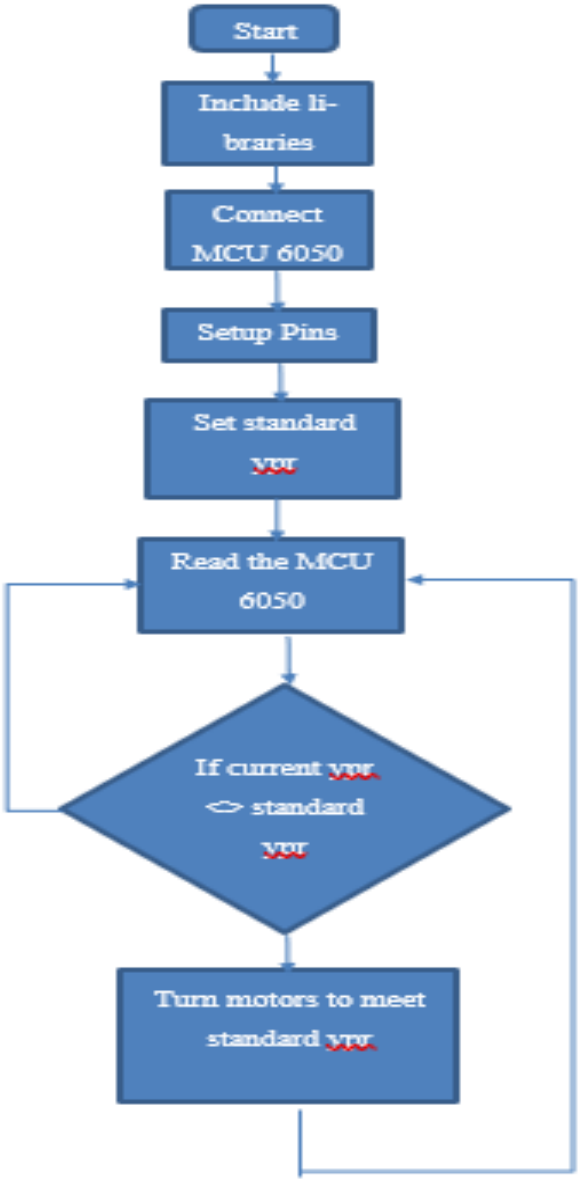


Figure 18: Flowchart of stabilization system

In Figure 18, ypr is an abbreviation of yawn pitch roll, which are the orientation axis of the camera.

COMMUNICATION

The controlling device that the customer wanted to use has an Android operating system. To create the controlling application for that device, Android Studio is used. When the programmer first opens Android studio, Android studio will automatically create a project package which contain XML and Java files, so that the programmer can create his application in a simple way. In short, the XML files are responsible for organizing the appearance of the application, while the Java files decide how the application will work.

In this report, I will only include the Java codes needed to build the controlling application, because the appearance of the application will mainly be based on personal preference. Moreover, Android studio has one superior feature that allows the programmer to design his/her application only by dragging and dropping components on a graphical screen representing the real device's screen. The XML codes will be automatically generated based on the programmer's design. Thanks to this feature, half of the work has been reduced.

Figure 15 represents the graphical screen being arrange. This is also the interface of this project's application.

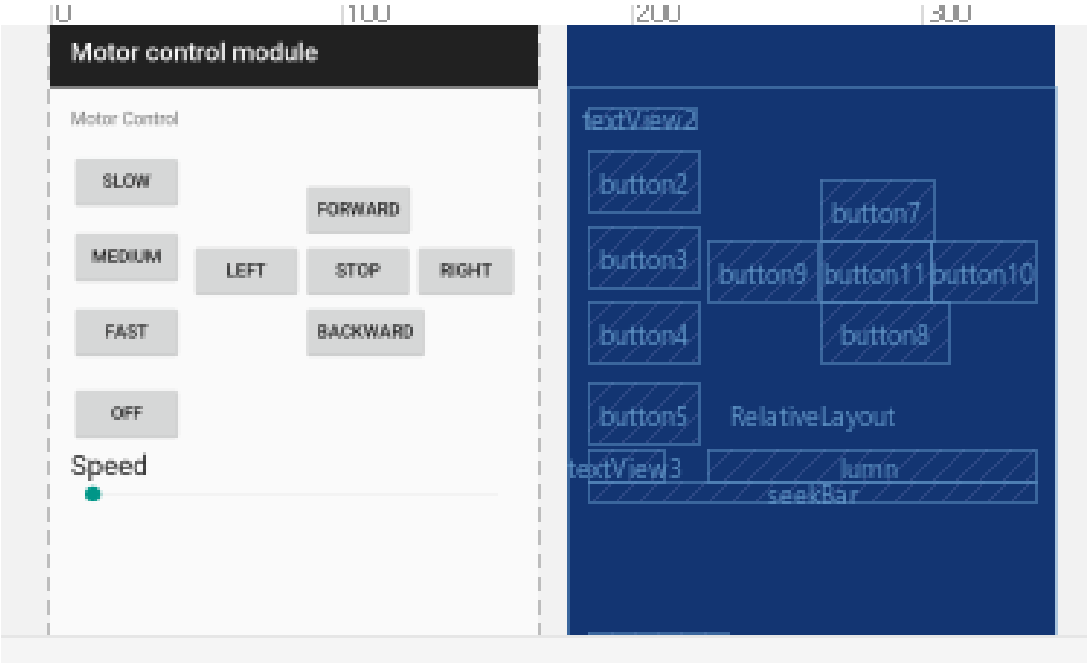


Figure 19:Graphic interface

In Figure 15, it is shown that there are buttons that allow the users to control the motion speed or direction of the two robots. Two robots will share the same controlling interface, however, the buttons on the left including SLOW, MEDIUM, FAST and OFF are enable only when the train robot is communicating with the controlling device. Vice versa, only the buttons LEFT, RIGHT, FORWARD, BACKWARD, STOP and the SPEED seek bar are enable when the controlling device is connected to the freely moving robot.

The Java codes below shows the Bluetooth connection screen. In this screen, the user can see the device lists which devices' Bluetooths are ON around the controlling device, and choose the Bluetooth's connection.

4.3 Testing and debugging

4.3.1 Testing

There are two parts of the testing process. In the first part, all the circuits and programs are tested so that the circuits and programs work perfectly independently and together. In the second part, the systems are attached to the mechanical parts of the robots and tested

if the movements of the robots cause any problem to the robots' platforms, or if the moving pattern of the train robot synchronizes with the curve track. Because the scope of the thesis does not include the mechanical parts, the second testing phase was dismissed.

The first testing phase was successfully completed. The motors run in responding to the controlling unit's command, and the gimbal motors turn when the gyroscope values change. Also, a screen shot of the gyroscope value is displayed on the serial monitor of the Arduino IDE.

4.3.2 Debugging

In this project, the most challenging part is not the programming, but the circuit building. Therefore, in the debugging process, I mainly fixed the problem with the circuit, for example, the problem of missing components or choosing the wrong components. In this section, I will mention an example of my debugging process.

In the first version of the motion system, I connected the Arduino to the motor through a L7912CV chip, which is only a voltage regulator. Therefore, it provided enough power for the motor to turn, which resulted in the motor turning when I ran my program. However, in the program, regardless of the direction I set, the motor only turned in one direction. In the end, I decided to use the H-bridge to control the direction of the motor, and the motor can turn in both direction in the next version.

5 OUTCOME

The motors for the mobilization system can be controlled by an Android phone. When the app is opened on the phone, the user will see a list of Bluetooth devices in the range. The user has two options: to connect with the train robot or to connect with the remote-controlled robot. The program will not do anything if the controlling unit is connected to a different device.

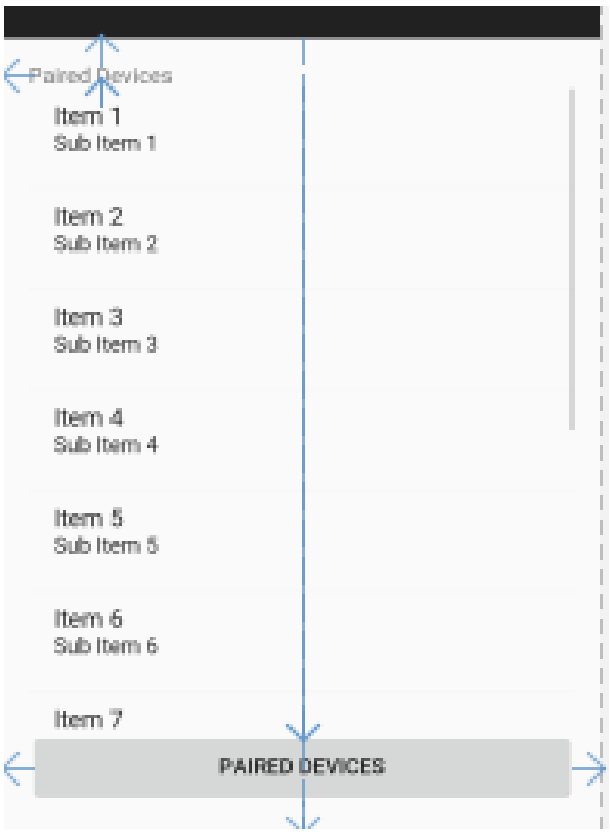


Figure 20: Bluetooth device list to connect

After setting up the connection, the user is taken to the controlling page (Figure 21). On this page, the user can control the train robot to run in 3 different speeds which is SLOW, MEDIUM and FAST, and to stop, or control the remote-controlled robot's motion by using the buttons FORWARD, BACKWARD, LEFT, RIGHT, STOP and its speed by using the seek bar. When finished using, the user can also disconnect the controlling unit from the robot by using the button DISCONNECT. All of these function work well with

little delay. If the user uses the wrong buttons for the robots, the robot will not reply to these commands to avoid any damage for the system and ensure safety.

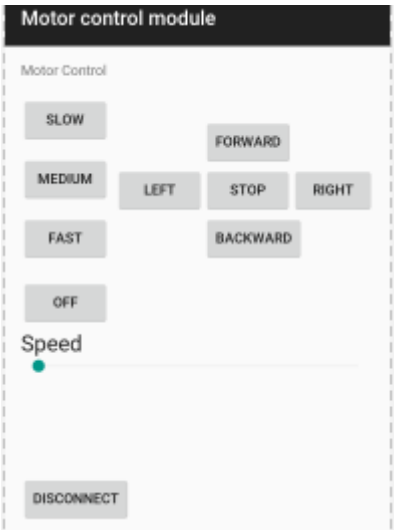


Figure 21: Controlling page

Figure 22 shows the mobilization system running.

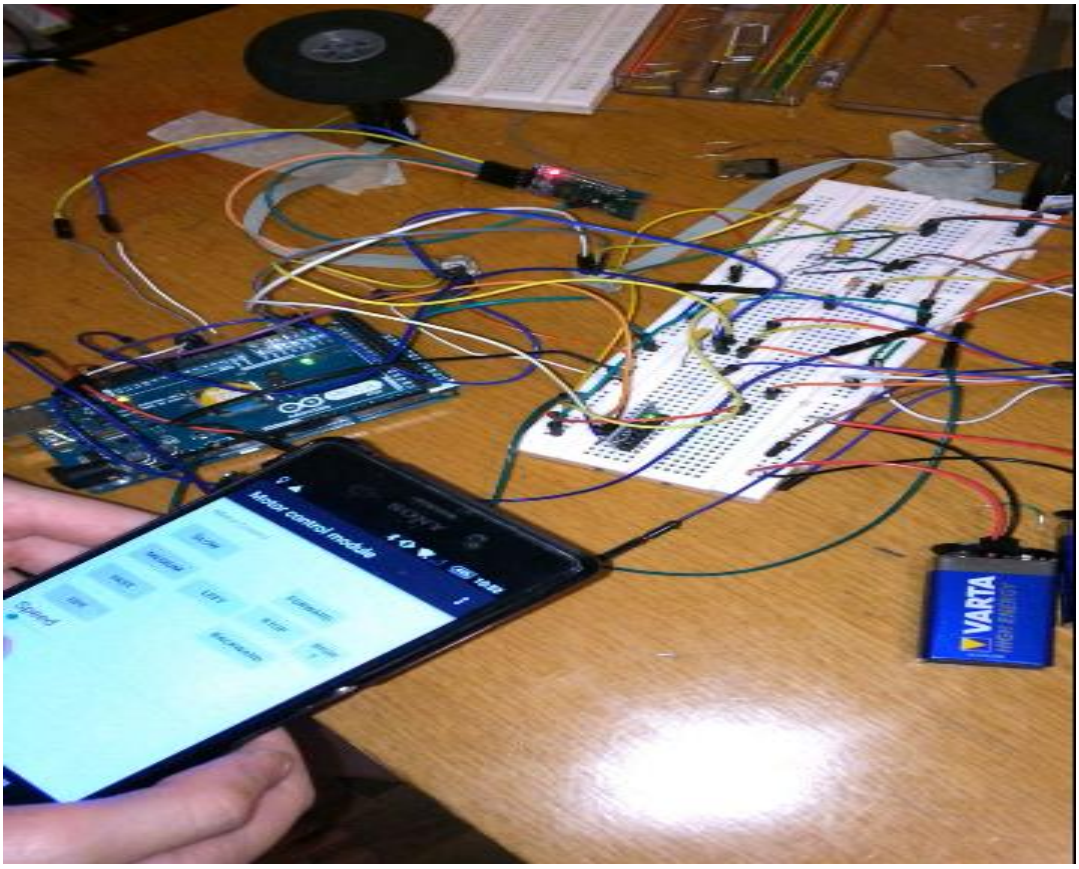


Figure 22: Mobilization motors controlled by phone app

The gimbal motors react to the change of the roll, pitch and roll coordinates that the gyroscope detects. Figure 23 indicates the orientation of the camera captured by the gyroscope.

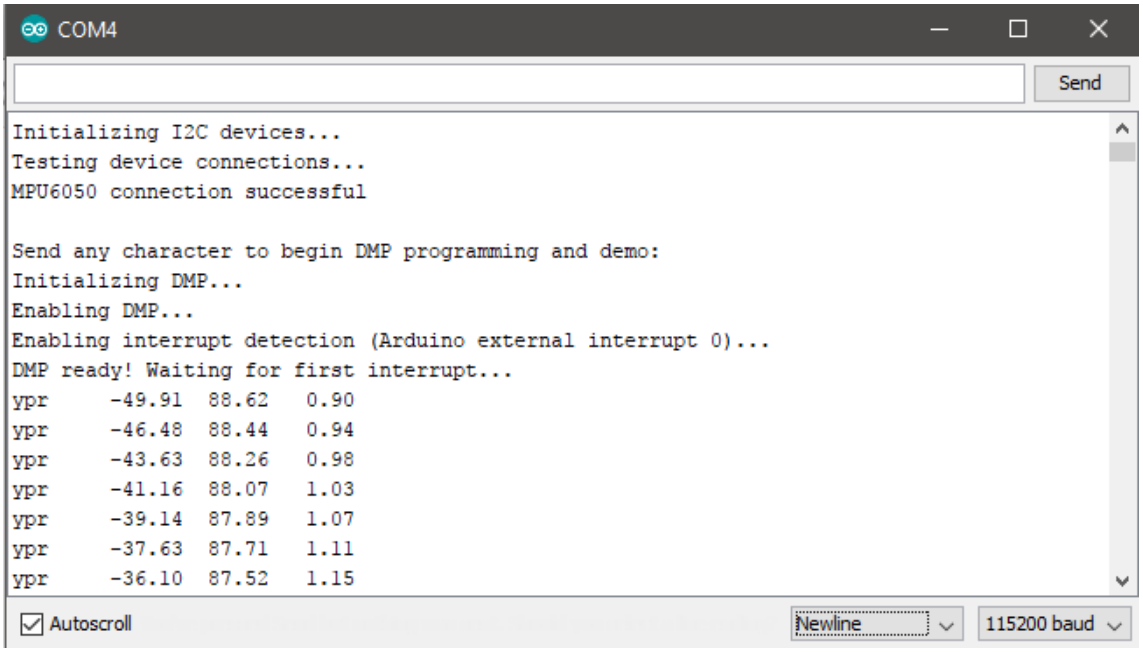


Figure 23: The position of the camera detected by MPU6050

6 CONCLUSION AND DISCUSSION

In conclusion, the functional requirements of this project are already met. The motors can be control from a controlling unit, and the gimbal motors turn when the coordinates of the gyroscope change. However, this system is not tested with the real mechanical components, therefore, it is not the finalized version of the system.

This system can be developed further by adding a mechanical platform and track to the robots. Then, the speed of the train robot's wheel is genuinely adjusted so that the it has the exact same pattern with the track, preventing the train from shifting out of the track after a long time running. PID tuning method can also be used to achieve the right speed with minimum delay. From the official website of Matlab and Simulink, the instruction on PID tuning can be found. /7/

REFERENCES

/1/ https://www.rs-online.com/designspark/different-types-of-motors-and-their-use

/2/ https://store.arduino.cc/arduino-mega-2560-rev3

/3/ Elabz.com

/4/https://www.arduino.cc/en/Guide/ArduinoMega2560

/5/https://playground.arduino.cc/Main/MPU-6050

/6/https://arduino-info.wikispaces.com/BlueTooth-HC05-HC06-Modules-How-To

/7/ http://ctms.engin.umich.edu/CTMS/index.php?aux=Activities_DCmotorB

CODES

Program to run the robots' wheels

```
#include <PID_v1.h>
#include <SoftwareSerial.h>
#include <Encoder.h>
#include <I2Cdev.h>
#include <MPU6050.h>

#include "Wire.h"
#define InA1          2          // INA motor pin
#define InB1          3          // INB motor pin
#define PWM1          5          // PWM motor pin
#define InA2          18         // INA motor pin
#define InB2          19         // INB motor pin
#define PWM2          7          // PWM motor pin
#define DIR1          13         // direction motor 1
#define DIR2          12         // direction motor 1

char data=0;
int duration;
boolean Direction;
unsigned long lastMilli = 0;
unsigned long lastMilliPrint = 0;
SoftwareSerial bluetooth(0, 1);
String string;
long tickNumber =0;
long count = 0;
long countInit;
boolean run = false;
boolean on= false;
```

```
void setupBluetooth() {
  bluetooth.begin(115200);
  bluetooth.print("$$$");
  delay(100);
  bluetooth.println("U,9600,N");
  bluetooth.begin(9600);
}

void setup() {
 pinMode(InA1, INPUT);
 pinMode(InB1, INPUT);
 pinMode(PWM1, OUTPUT);
 pinMode(InA2, INPUT);
 pinMode(InB2, INPUT);
 pinMode(PWM2, OUTPUT);
 pinMode(DIR1, OUTPUT);
 pinMode(DIR2, OUTPUT);
 pinMode(0, INPUT);
 pinMode(1, OUTPUT);
 //attachInterrupt(digitalPinToInterrupt(2),rencoder,FALLING);
 //attachInterrupt(digitalPinToInterrupt(18),rencoder,FALLING);
 Wire.begin();
 Serial.begin(9600);
// Serial.println("Initializing I2C devices...");
 //Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful" : "MPU6050 connection failed");

 //accelgyro.initialize();
 setupBluetooth();
}

void motorForward(int PWM_val) {
```

```
 digitalWrite(DIR1, HIGH);
 analogWrite(PWM1, PWM_val);
 digitalWrite(InB1, HIGH);
 digitalWrite(InA1, LOW);
 digitalWrite(InA1, HIGH);
}

void motorBackward(int PWM_val)  {
 digitalWrite(DIR1, LOW);
 analogWrite(PWM1, PWM_val);
 digitalWrite(InB1, LOW);
 digitalWrite(InA1, LOW);
 digitalWrite(InA1, HIGH);

}

void motorForward2(int PWM_val)  {
 digitalWrite(DIR2, HIGH);
 analogWrite(PWM2, PWM_val);
 digitalWrite(InB2, HIGH);
 digitalWrite(InA2, LOW);
}

void motorBackward2(int PWM_val)  {
 digitalWrite(DIR2, LOW);
 analogWrite(PWM2, PWM_val);
 digitalWrite(InA2, HIGH);
 digitalWrite(InB2, HIGH);
}

void motorStop()  {
 analogWrite(PWM1, 0);
```

```
 analogWrite(PWM2, 0);
}

void loop() {
  Serial.println(analogRead(0));
  delay(1000);
  //Serial.print("Pulse:");
  //Serial.println(duration);
  //duration = 0;
  if (Serial.available() > 0)
    {string = "";}

  while(Serial.available()>0){
    data=(byte)Serial.read();
    Serial.print(data);
     if(data == ':')
      {
        break;
      }

      else
      {
        string += data;
      }

      delay(1);
    }

    if(string=="TO"){
       motorBackward(15);
       motorBackward2(15);
       delay(2000);
```

```
    motorStop();
    delay(400);
    motorForward(15);
    motorForward2(15);
    delay(2000);
    motorStop();
    delay(400);
  }
  else if(string=="T1"){
    motorBackward(50);
    motorBackward2(50);
    delay(2000);
    motorStop();
    delay(400);
    motorForward(50);
    motorForward2(50);
    delay(2000);
    motorStop();
    delay(400);
  }
  else if(string=="T2"){
    motorBackward(100);
    motorBackward2(100);
    delay(2000);
    motorStop();
    delay(400);
    motorForward(100);
    motorForward2(100);
    delay(2000);
    motorStop();
    delay(400);
  }
```

```
else if(string=="TF"){
  motorStop();
  delay(2000);
}
else if(string=="FW"){
   on=true;
   motorForward(50);
   motorForward2(50);
}
else if(string=="BW"){
   on=true;
   motorBackward(50);
   motorBackward2(50);
}
else if(string=="TL"){
    motorForward(10);
    delay(100);
    motorStop();
}
else if(string=="TR"){
    motorForward2(10);
    delay(100);
    motorStop();
}
else if(string=="ST"){
    on= false;
    motorStop();
}
  if ((string.toInt()>=0)&&(string.toInt()<=255))
{
  if (on==true)
  {
```

```
    analogWrite(PWM1,string.toInt());
    analogWrite(PWM2,string.toInt());
    delay(10);
    }
  }
}
```

Program for the stabilization system

```
#include "I2Cdev.h"



#include "MPU6050_6Axis_MotionApps20.h"

//#include "MPU6050.h"

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE im-
plementation

// is used in I2Cdev.h

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

   #include "Wire.h"

#endif



// AD0 high = 0x69

MPU6050 mpu;

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

```
#define OUTPUT_READABLE_YAWPITCHROLL


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


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 measure-
ments

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


// packet structure for InvenSense teapot demo

uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };
```

```
// Pins to control pitch motor
const int pitchMotor1 =9;
const int pitchMotor2 =10;
const int pitchMotor3 =11;


// Pins to control roll motor
const int rollMotor1 =3;
const int rollMotor2 =5;
const int rollMotor3 =6;


// Variable to let us move motor in desired direction
int increment = 0;


// Controls speed of motor updates
const int motorDelay = 7; // together with pot controls the RPM
long lastMotorDelayTime = 0;




// Variable to intialize orientation of axes
```

```
const int pitchOrient = 25;

const int rollOrient = -20;



const boolean pitch = true;

const boolean roll = false;

const int pwmSin[] = {127, 135, 144, 152, 160, 168, 176, 183, 190, 198, 204, 211,
217,

            222, 228, 233, 237, 241, 244, 247, 250, 252, 253, 254, 254, 254,

            253, 252, 250, 247, 244, 241, 237, 233, 228, 222, 217, 211, 204,

            198, 190, 183, 176, 168, 160, 152, 144, 135, 127, 119, 110, 102,

            94, 86, 78, 71, 64, 56, 50, 43, 37, 32, 26, 21, 17, 13, 10, 7,

            4, 2, 1, 0, 0, 0, 1, 2, 4, 7, 10, 13, 17, 21, 26, 32, 37, 43,

            50, 56, 64, 71, 78, 86, 94, 102, 110, 119

           };

int pitchStepA = 0;

int pitchStepB = 32;

int pitchStepC = 64;

int rollStepA = 0;

int rollStepB = 32;

int rollStepC = 64;

volatile bool mpuInterrupt = false;
```

```
void dmpDataReady() {

   mpuInterrupt = true;

}

void move(int &currentStepA, int &currentStepB, int &currentStepC,

         int motorPin1, int motorPin2, int motorPin3, boolean axis) {



  if(  axis == true ) {

   if( (ypr[1] * 180/M_PI) <= pitchOrient ) {

     increment = -1;

   } else {

     increment = 1;

   }

  } else if( axis == false) {

   if( (ypr[2] * 180/M_PI) <= rollOrient ) {

     increment = 1;

   } else {

     increment = -1;

   }

  }
```

```
  currentStepA = currentStepA + increment;

  if(currentStepA > 95) currentStepA = 0;

  if(currentStepA < 0) currentStepA = 95;


  currentStepB = currentStepB + increment;

  if(currentStepB > 95) currentStepB = 0;

  if(currentStepB < 0) currentStepB = 95;


  currentStepC = currentStepC + increment;

  if(currentStepC > 95) currentStepC = 0;

  if(currentStepC < 0) currentStepC = 95;


  analogWrite(motorPin1, pwmSin[currentStepA]);

  analogWrite(motorPin2, pwmSin[currentStepB]);

  analogWrite(motorPin3, pwmSin[currentStepC]);


  lastMotorDelayTime = millis();


}

void setup() {
```

```
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
#endif
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();
```

```
// verify connection

Serial.println(F("Testing device connections..."));

Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));



// wait for ready

Serial.println(F("\nSend any character to begin DMP programming and demo: "));

while (Serial.available() && Serial.read()); // empty buffer

while (!Serial.available());                 // wait for data

while (Serial.available() && Serial.read()); // empty buffer again



// 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.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        attachInterrupt(0, 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(")"));
  }

  // Declares the motor pins as output pins
  pinMode(pitchMotor1, OUTPUT);
  pinMode(pitchMotor2, OUTPUT);
  pinMode(pitchMotor3, OUTPUT);
  pinMode(rollMotor1, OUTPUT);
  pinMode(rollMotor2, OUTPUT);
  pinMode(rollMotor3, OUTPUT);

}
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) {



   // Move pitch motor

   if((millis() - lastMotorDelayTime) >  motorDelay) {



       move(pitchStepA, pitchStepB, pitchStepC,

               pitchMotor1, pitchMotor2, pitchMotor3, pitch);



       move(rollStepA, rollStepB, rollStepC,

               rollMotor1, rollMotor2, rollMotor3, roll);



  }



   if( mpuInterrupt ) {

     break;

   }
```

```
  }

  // reset interrupt flag and get INT_STATUS byte

  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

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

  // Serial.println("checking overflow");
  // check for overflow (this should never happen unless our code is too inefficient)
  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {

    // reset so we can continue cleanly
      mpu.resetFIFO();
      Serial.println(F("FIFO overflow!"));

  // otherwise, check for DMP data ready interrupt (this should happen frequently)
```

```
} else if (mpuIntStatus & 0x02) {
    // 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_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        Serial.print("quat\t");
        Serial.print(q.w);
        Serial.print("\t");
        Serial.print(q.x);
        Serial.print("\t");
        Serial.print(q.y);
```

```
    Serial.print("\t");
    Serial.println(q.z);
#endif

#ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#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");

    Serial.println(ypr[2] * 180/M_PI);

#endif


#ifdef OUTPUT_READABLE_REALACCEL

    // display real acceleration, adjusted to remove gravity

    mpu.dmpGetQuaternion(&q, fifoBuffer);

    mpu.dmpGetAccel(&aa, fifoBuffer);

    mpu.dmpGetGravity(&gravity, &q);

    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);

    Serial.print("areal\t");

    Serial.print(aaReal.x);

    Serial.print("\t");

    Serial.print(aaReal.y);

    Serial.print("\t");
```

```
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t");
    Serial.print(aaWorld.x);
    Serial.print("\t");
    Serial.print(aaWorld.y);
    Serial.print("\t");
    Serial.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
```

```
        // display quaternion values in InvenSense Teapot demo format:

        teapotPacket[2] = fifoBuffer[0];

        teapotPacket[3] = fifoBuffer[1];

        teapotPacket[4] = fifoBuffer[4];

        teapotPacket[5] = fifoBuffer[5];

        teapotPacket[6] = fifoBuffer[8];

        teapotPacket[7] = fifoBuffer[9];

        teapotPacket[8] = fifoBuffer[12];

        teapotPacket[9] = fifoBuffer[13];

        Serial.write(teapotPacket, 14);

        teapotPacket[11]++; // packetCount, loops at 0xFF on purpose

      #endif



  }

}
```

Program for the control application

```
package com.RB.RB;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.bluetooth.BluetoothAdapter;
```

```java
import android.bluetooth.BluetoothDevice;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;


public class DeviceList extends ActionBarActivity
{
    //widgets
    Button btnPaired;
    ListView devicelist;
    //Bluetooth
    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    public static String EXTRA_ADDRESS = "device_address";

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device_list);

        //Calling widgets
        btnPaired = (Button)findViewById(R.id.button);
        devicelist = (ListView)findViewById(R.id.listView);

        //if the device has bluetooth
        myBluetooth = BluetoothAdapter.getDefaultAdapter();

        if(myBluetooth == null)
        {
            //Show a mensag. that the device has no bluetooth adapter
            Toast.makeText(getApplicationContext(), "Bluetooth Device Not
Available", Toast.LENGTH_LONG).show();

            //finish apk
            finish();
        }
        else if(!myBluetooth.isEnabled())
        {
                //Ask to the user turn the bluetooth on
                Intent turnBTon = new Intent(BluetoothAdapter.ACTION_RE-
QUEST_ENABLE);
                startActivityForResult(turnBTon,1);
        }

        btnPaired.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                pairedDevicesList();
            }
        });

    }

    private void pairedDevicesList()
    {
        pairedDevices = myBluetooth.getBondedDevices();
        ArrayList list = new ArrayList();

        if (pairedDevices.size()>0)
        {
```

```
            for(BluetoothDevice bt : pairedDevices)
            {
                list.add(bt.getName() + "\n" + bt.getAddress()); //Get
the device's name and the address
            }
        }
        else
        {
            Toast.makeText(getApplicationContext(), "No Paired Bluetooth
Devices Found.", Toast.LENGTH_LONG).show();
        }

        final ArrayAdapter adapter = new ArrayAdapter(this,android.R.lay-
out.simple_list_item_1, list);
        devicelist.setAdapter(adapter);
        devicelist.setOnItemClickListener(myListClickListener); //Method
called when the device from the list is clicked

    }

    private AdapterView.OnItemClickListener myListClickListener = new
AdapterView.OnItemClickListener()
    {
        public void onItemClick (AdapterView<?> av, View v, int arg2,
long arg3)
        {
            // Get the device MAC address, the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Make an intent to start next activity.
            Intent i = new Intent(DeviceList.this, ledControl.class);

            //Change the activity.
            i.putExtra(EXTRA_ADDRESS, address); //this will be received
at ledControl (class) Activity
            startActivity(i);
        }
    };


    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_device_list, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

```java
package com.RB.RB;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.AsyncTask;

import java.io.IOException;
import java.util.UUID;


public class ledControl extends ActionBarActivity {

    Button btnOn,btnOn1,btnOn2, btnOff,btFW,btBW,btL,btR, btStop, btnDis;
    SeekBar speed;
    TextView lumn;
    String address = null;
    private ProgressDialog progress;
    BluetoothAdapter myBluetooth = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;
    //SPP UUID. Look for it
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        Intent newint = getIntent();
        address = newint.getStringExtra(DeviceList.EXTRA_ADDRESS); //re-
ceive the address of the bluetooth device

        //view of the ledControl
        setContentView(R.layout.activity_led_control);

        //call the widgtes
        btnOn = (Button)findViewById(R.id.button2);
        btnOn1 = (Button)findViewById(R.id.button3);
        btnOn2 = (Button)findViewById(R.id.button4);
        btnOff = (Button)findViewById(R.id.button5);
        btnDis = (Button)findViewById(R.id.button6);
        btFW = (Button)findViewById(R.id.button7);
        btBW = (Button)findViewById(R.id.button8);
        btL = (Button)findViewById(R.id.button9);
        btR = (Button)findViewById(R.id.button10);
        btStop = (Button)findViewById(R.id.button11);
        speed = (SeekBar)findViewById(R.id.seekBar);
        lumn = (TextView)findViewById(R.id.lumn);
```

```
new ConnectBT().execute(); //Call the class to connect

//commands to be sent to bluetooth
btnOn.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        turnOnLed();
    }
});

btnOn1.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        turnOnLed1();
    }
});

btnOn2.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        turnOnLed2();
    }
});

btnOff.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        turnOffLed();
    }
});

btnDis.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        Disconnect(); //close connection
    }
});

btFW.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        Forward();
    }
});

btBW.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        Backward();
    }
});

btL.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View v)
            {
                Turnleft();
            }
        });

        btR.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                Turnright();
            }
        });

        btStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                Stop();
            }
        });

        speed.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeLis-
tener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
                if (fromUser==true)
                {
                    lumn.setText(String.valueOf(progress));
                    try
                    {
                        btSocket.getOutputStream().write(String.val-
ueOf(progress).getBytes());
                    }
                    catch (IOException e)
                    {

                    }
                }
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {

            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {

            }
        });
    }

    private void Disconnect()
    {
        if (btSocket!=null) //If the btSocket is busy
        {
            try
            {
                btSocket.close(); //close connection
            }
            catch (IOException e)
            { msg("Error");}
```

```
        }
        finish(); //return to the first layout

    }

    private void turnOffLed()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("TF".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void turnOnLed()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("TO".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void turnOnLed1()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("T1".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }
    private void turnOnLed2()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("T2".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
```

```
        }
    }

    private void Forward()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("FW".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void Backward()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("BW".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void Turnleft()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("TL".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }

    private void Turnright()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("TR".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
```

```
    }

    private void Stop()
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("ST".toString().get-
Bytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }
    // fast way to call Toast
    private void msg(String s)
    {
        Toast.makeText(getApplicationCon-
text(),s,Toast.LENGTH_LONG).show();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_led_control, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    private class ConnectBT extends AsyncTask<Void, Void, Void>  // UI
thread
    {
        private boolean ConnectSuccess = true; //if it's here, it's al-
most connected

        @Override
        protected void onPreExecute()
        {
            progress = ProgressDialog.show(ledControl.this, "Connect-
ing...", "Please wait!!!");  //show a progress dialog
        }

        @Override
        protected Void doInBackground(Void... devices) //while the pro-
gress dialog is shown, the connection is done in background
        {
            try
```

```
            {
                if (btSocket == null || !isBtConnected)
                {
                 myBluetooth = BluetoothAdapter.getDefaultAdapter();//get
the mobile bluetooth device
                 BluetoothDevice dispositivo = myBlue-
tooth.getRemoteDevice(address);//connects to the device's address and
checks if it's available
                 btSocket = dispositivo.createInsecureRfcommSock-
etToServiceRecord(myUUID);//create a RFCOMM (SPP) connection
                 BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                 btSocket.connect();//start connection
                }
            }
            catch (IOException e)
            {
                ConnectSuccess = false;//if the try failed, you can check
the exception here
            }
            return null;
        }
        @Override
        protected void onPostExecute(Void result) //after the doInBack-
ground, it checks if everything went fine
        {
            super.onPostExecute(result);

            if (!ConnectSuccess)
            {
                msg("Connection Failed. Is it a SPP Bluetooth? Try
again.");
                finish();
            }
            else
            {
                msg("Connected.");
                isBtConnected = true;
            }
            progress.dismiss();
        }
    }
}
```