Report Arduino Madrid 2024

Remote-controlled robot

Introduction

This project is a robot on two wheels that can be controlled over WiFi using WASD controls. The robot also has five LEDs which can light up in five different modes. An Arduino Nano 33 IoT is used to create a server and to send the controls to the motors of the wheels. Using Processing a client can connect to the server to control the robot and change the LED mode.

General Description

We connect the Arduino to the power source integrated in the robot and wait ~10 seconds so the Arduino establishes the connection to the WiFi. Once it is connected, we can control it from any laptop knowing the port and the IP address of the Arduino. We open up Processing which is the interface we chose to control the Arduino with and we input WASD or the led mode we want. Then, the Arduino receives the commands via WiFi and performs accordingly.

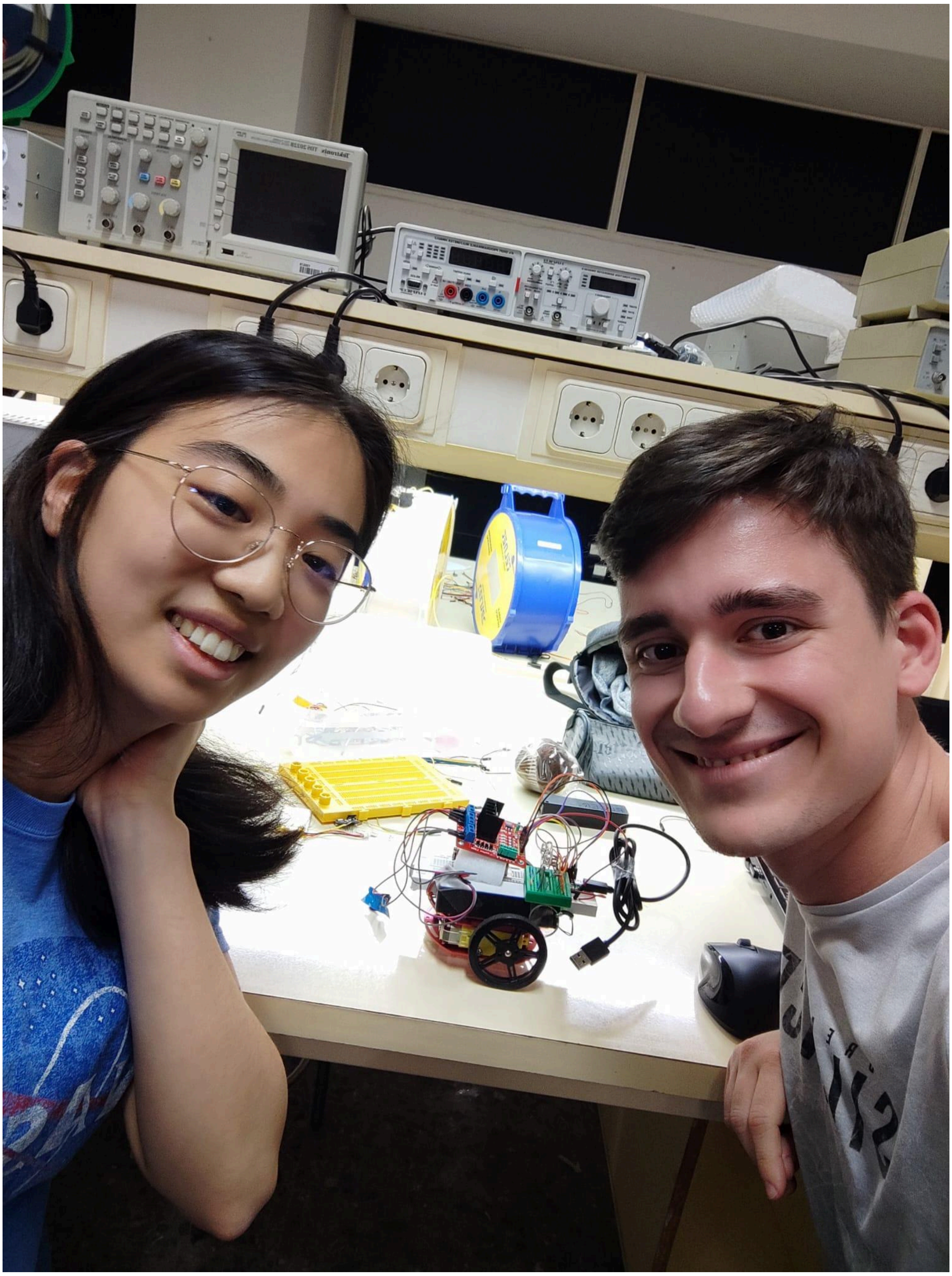
Here are some videos showcasing the functionalities of the robot:

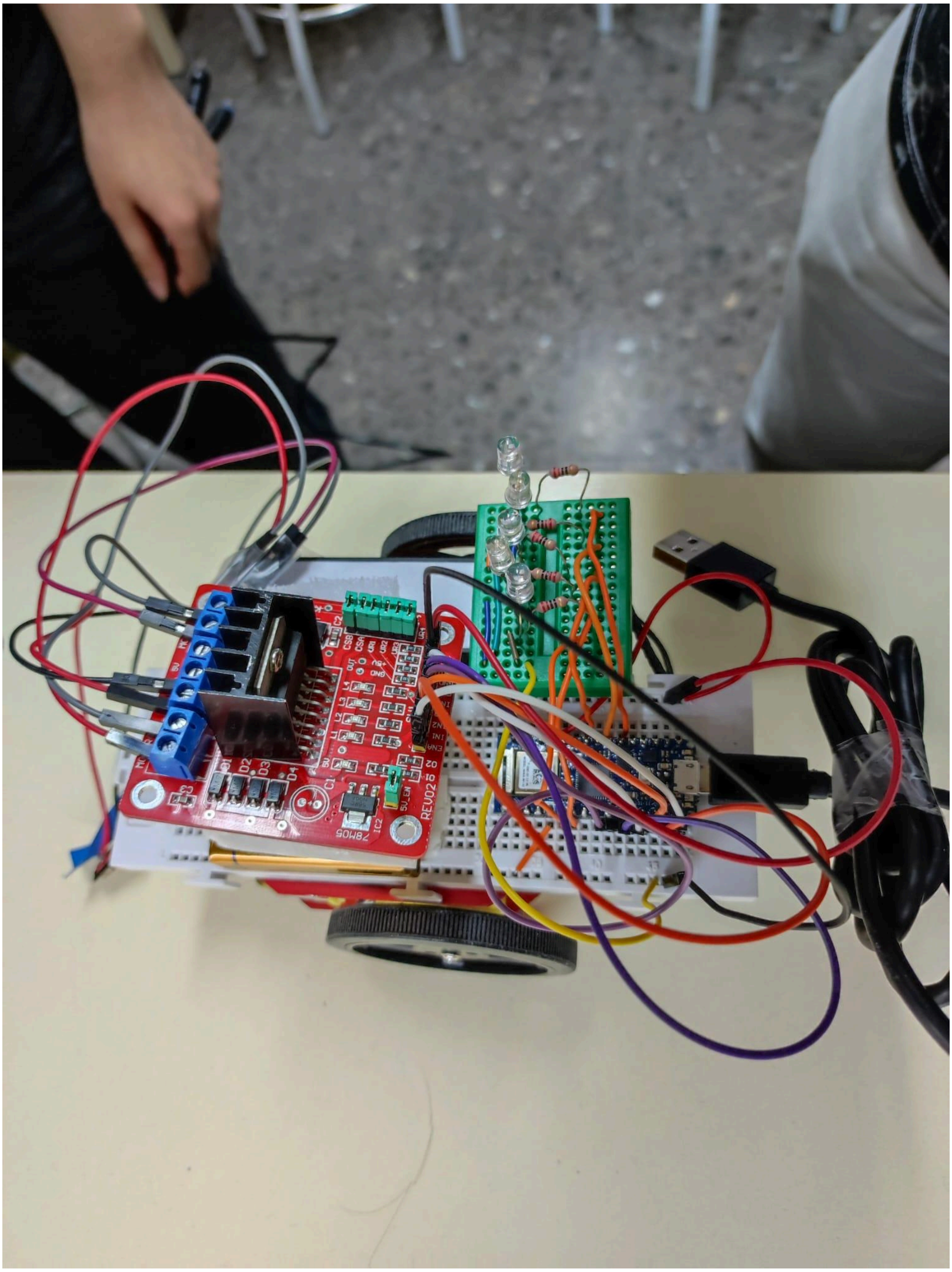
LED Functions Showcase:
https://youtube.com/shorts/bvhirrCKVIs?feature=share

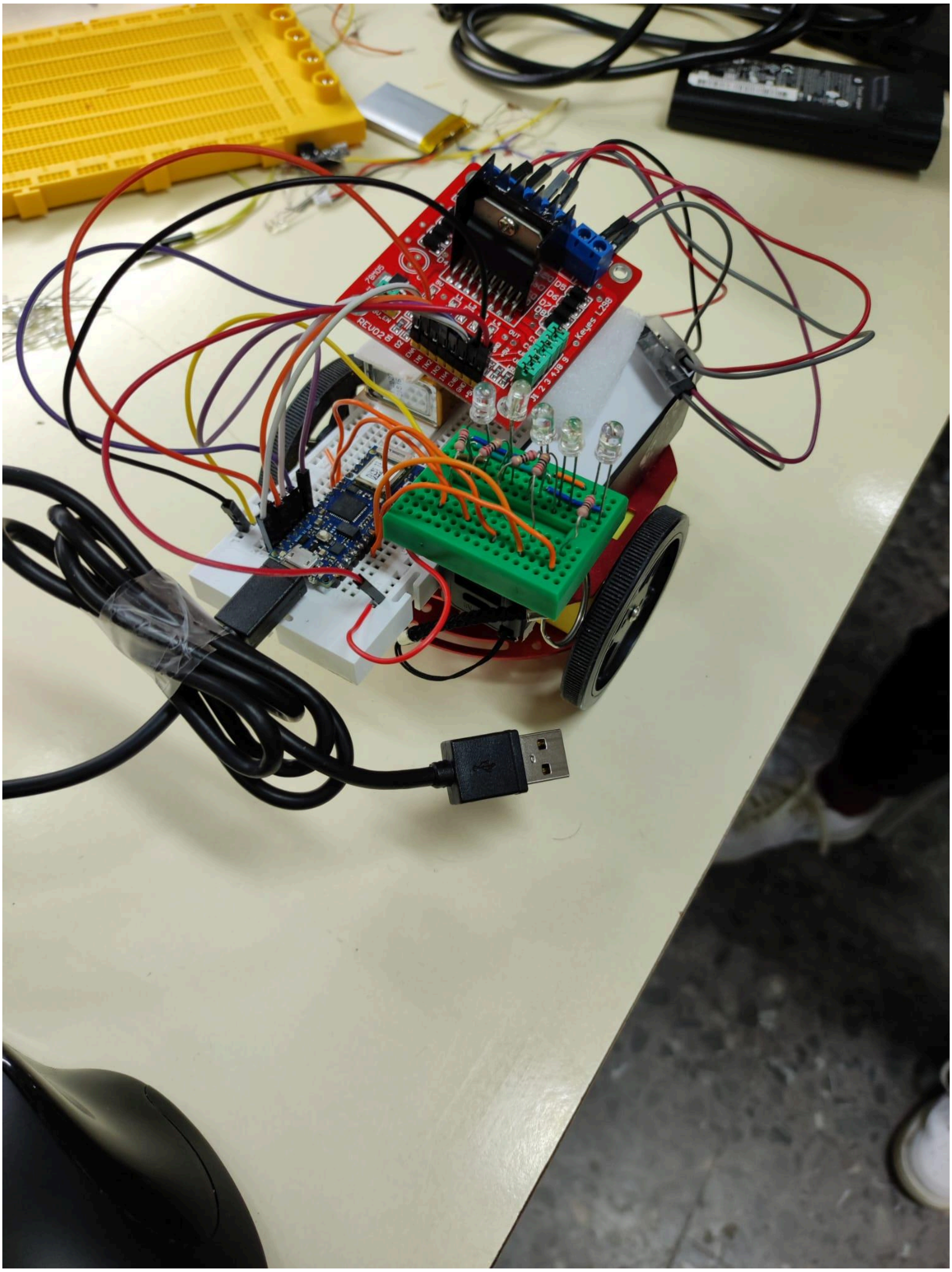
Motor Functions Showcase:
https://youtube.com/shorts/1L7PeVK_Y_4?feature=share

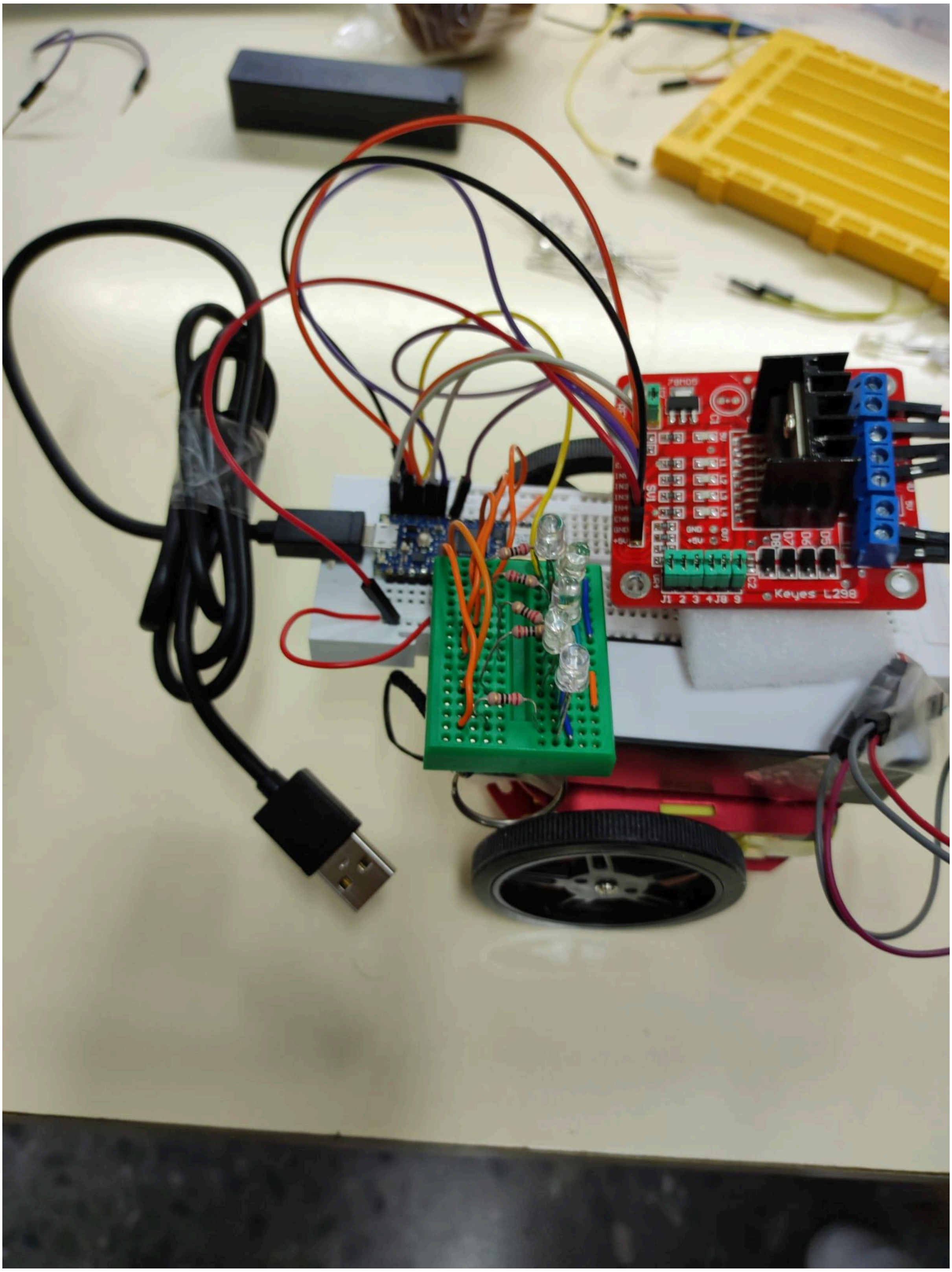
Short Presentation walking around:
https://youtube.com/shorts/KIBCu9JI9Dw?feature=share

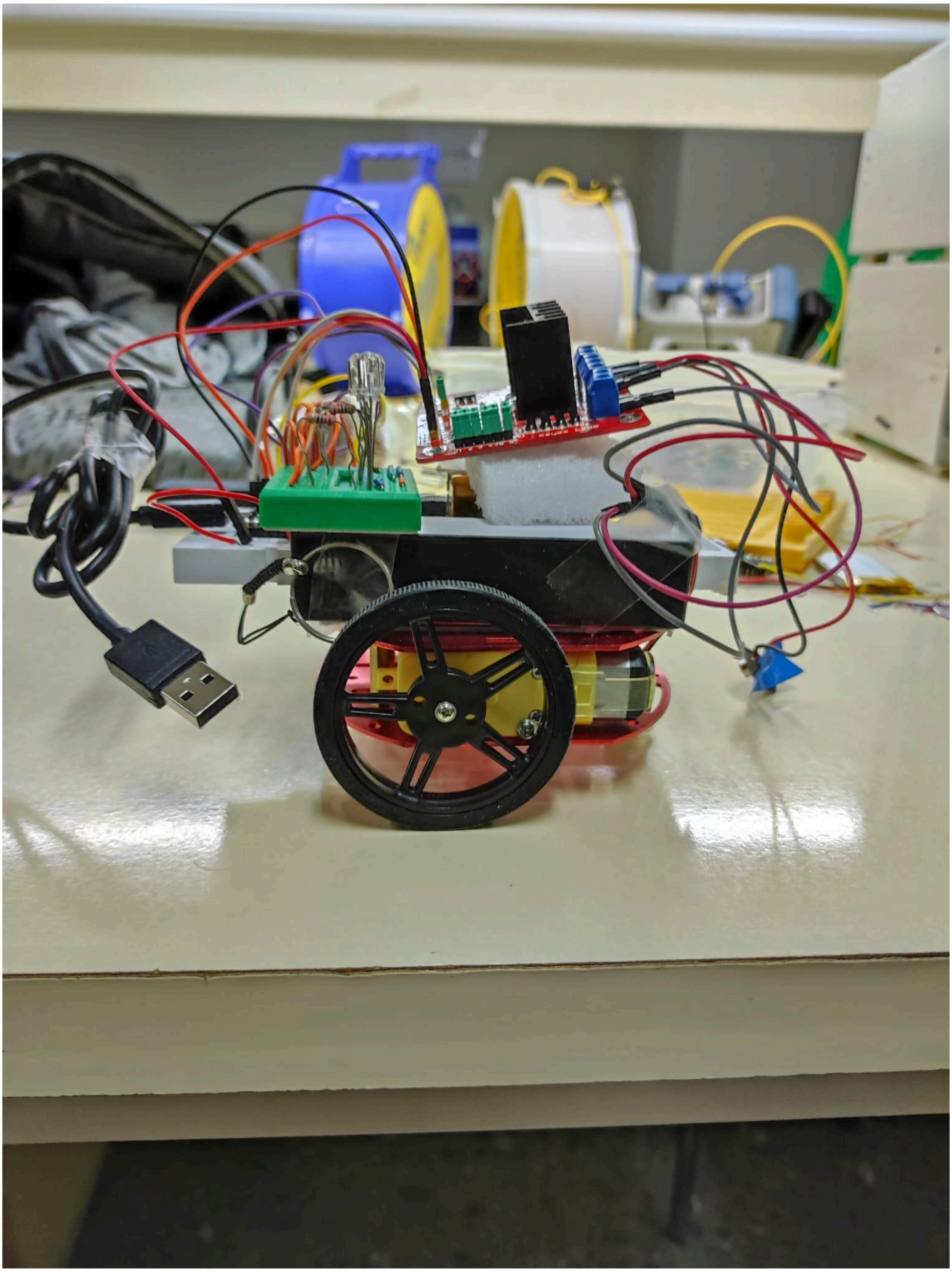
Below are some pictures of the completed project:





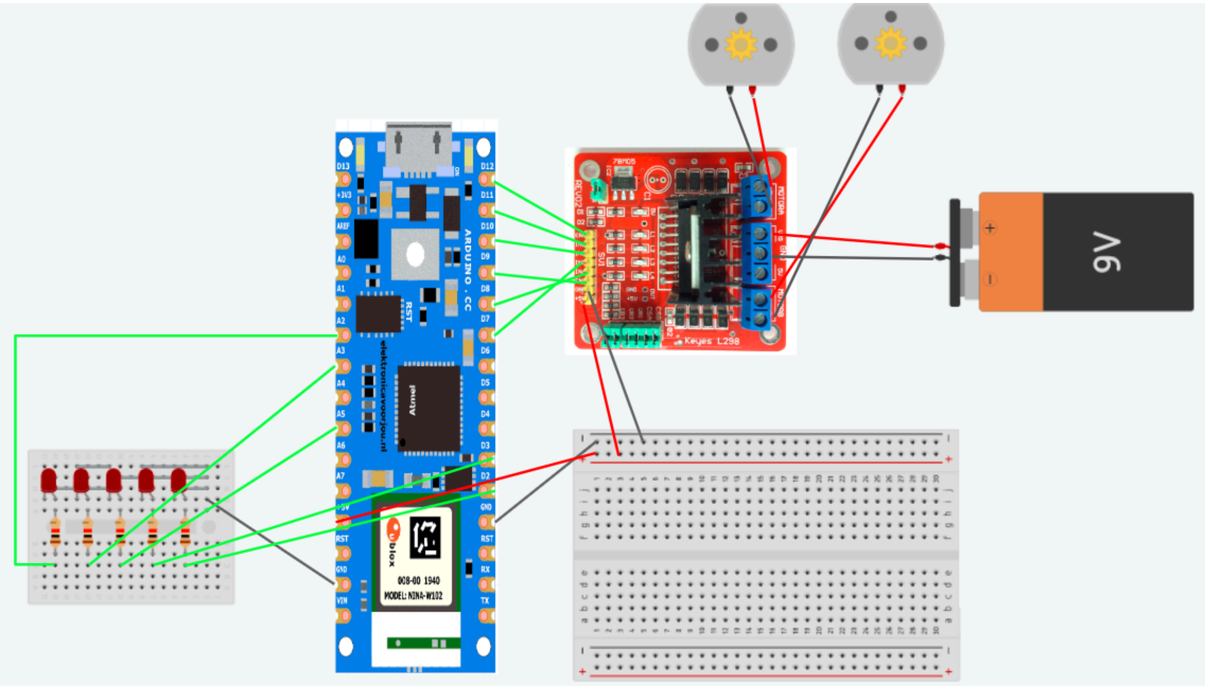

REV02
Keyes L298


78M05
Keyes L298
J1 2 3 4 J8 9
GND
+5V



- # Hardware Design

Hardware schematic



List of components

- Arduino Nano 33 IOT

The Arduino IDE (Integrated Development Environment), a software tool made to make it simple to write, upload, and debug code for the board, can be used to program the Arduino nano 33 IOT, which can be used to control a variety of electronic devices, including sensors, motors, lights, and other components. The Arduino Nano 33 IOT contains an integrated Wifi chip that can be used to easily connect to the Wifi without the need of external devices.

- DC motors (2x)

DC motors are widely used in various applications, from robotics to industrial machinery, due to their simplicity and ease of control. A DC motor, or a direct current motor, is an electromechanical device that converts electrical energy into mechanical energy. It operates based on the principles of electromagnetism and involves the interaction between magnetic fields and current-carrying conductors.

- Keyes L298 motor driver

The Keyes L298 motor driver is a chip used for controlling two DC motors separately using H-bridge circuits. There are six pins for controlling the two motors, motor A and motor B. The pins ENA, IN1 and IN2 are used to enable motor A and make it turn forward or backward. The pins ENB, IN3 and IN4 are used in the same fashion for motor B. The motor driver is powered with a 5V logic supply while the motors are powered with an extra 5V-35V battery.

- Structure consisting of two plates and a plastic knob

This structure holds the motors in place and raises them so the wheels can be attached. The Arduino, the breadboards and the batteries are fixed on the top of the structure.

- Breadboard (2x)
- LEDs (5x)
- 5V power bank
- 9V alkaline battery
- USB cable
- 220 ohm resistors (5x)
- Wires
- Rubber wheels (2x)

Software Design

We made this project in Arduino IDE and Processing.
Extra libraries:

- For Arduino: WifiNINA
- For Processing: g4p_controls; processing.net

Sources 3-rd party:

- We also used Tinkercad to test certain hardware aspects of this project.

Arduino code

I start off the software by defining all the variables, including the library and setting up the components. In the setup section of the code, we initialize all the variables and then set up the server using wifi. We print useful information to Serial like the IP address of the Arduino. The code is referenced in the Appendix, under section A1.

In the loop section, we test whether the client is connected and read the inputs that he sends. The variable `event_listener` listens for 1 character at a time and tests a series of commands which set the `event_boolean` to a certain digit code to be used later. If 2 of the same inputs are sent, the `event_boolean` is set to 0, later sending the message to stop the motors. The “w” input will set the `event_boolean` to start the motors forward, “s” to start them backwards, “a” to rotate left, “d” to rotate right, “q” will set a higher speed, “e”

will set a lower speed, “r” will set the normal speed again. The inputs “q”, ”e” and “r” will set the event_speed variable to “0”, “1”, “2” and “3” instead of the event_boolean. There are another set of inputs that will set the led_mode variable to a specific mode. The input “0” stops the LEDs, the input “1” will set them to “rainbow mode”, input “2” will set them to “police siren mode”, input “3” to “random” mode, input “4” to “white blink mode”, and input “5” to “all blink mode”. Sending 2 of the same input will set the led_mode variable to “0”. Going forward, we have a switch case that chooses an event depending on the event_boolean: “0” stops the motors, “1” starts them forward, “2” starts them backwards, “3” turns right, “4” turns left. The default is there as a debugging option. The next switch tests for event_speed and does nothing in case of “0”, speeds it up for “1”, slows it down for “2” and sets a normal speed for “3”. The third and final switch will test for led_mode. Case “0” will reset all LEDs to LOW. Case “1” will set a max_counter variable to 1200 and calls the led_mode_rainbow() function. This variable is later used for counting the milliseconds. Likewise, in the next cases we set it to the preferred time we want a loop to run for using max_counter and set the “police siren”, “random”, “white blink” and “all blink” modes. We then proceed to the code that defines the max_counter to have the measuring unit 1 millisecond, such that 1200 will set it to 1200 milliseconds. If the led_counter reaches max_counter then we reset all LEDs and set the led_counter to 0. Otherwise, we increase the led_counter by 1.
The code is referenced in the Appendix, under section A2.

Now we will talk about the functions used.
We used a MotorA_Direction1 function to define the speed of the motor A and start them forward by setting IN2 to HIGH and IN1 to LOW. Likewise, we set the motor backwards with MotorA_Direction2 and brake with MotorA_Brake . MotorB_Direction1 MotorB_Direction2 and MotorB_Brake implements the same functionalities but for the Motor B. Turn_Left and Turn_Right functions will turn the robot with a set speed of 120 to the left/right using the previous functions. Speed_Up Slow_Down and Speed_Normal functions will set the speed of the motors to 200, 100 or 150 respectively.
Led_mode_rainbow will turn on the LEDs in a 500 millisecond interval and then turn it off for the rest of the loop. The LEDs turn on one after the other, some being active at the same time, allowing for a smooth transition between them.
led_mode_police_siren will turn on only the red and blue LEDs alternating every 300 milliseconds.
reset_all_leds turns off all LEDs.
pick_random_led picks a random LED between the 5 options.
led_mode_random uses reset_all_leds and pick_random_led functions to turn on random LEDs.
led_mode_white_blink blinks the white LED.
led_mode_all_blink blinks all LEDs.

Processing code

The code can be found in Appendix B.1 to B.4.

We first define the variables for the location of the buttons, their color, the LED mode options and the information needed to establish the connection to the Arduino (Appendix B.1). We also define two enums `Direction` and `Axis` for better readability.

The variables and the window information are set up in the `setup()` section (Appendix B.2). We also connect to the Arduino as a client there using the Network library.

We add listeners in the `keyTyped()` function (Appendix B.3) for the keys controlling the robot and send these controls to the Arduino. The robot drives forward when “w” is typed, to the right with “a”, backwards with “s”, left with “d”, faster with “q”, slower with “e” and normal with “r”. We store the pressed key in a variable to be sure that only one message is sent to the Arduino for the key press.
When the key is released, we send another message to the Arduino to stop its motion in the `keyReleased()` function.

In the GUI there are also buttons that can be used to control the direction of the robot. When the button is clicked or its corresponding key typed, the button changes its color (Appendix B.4).

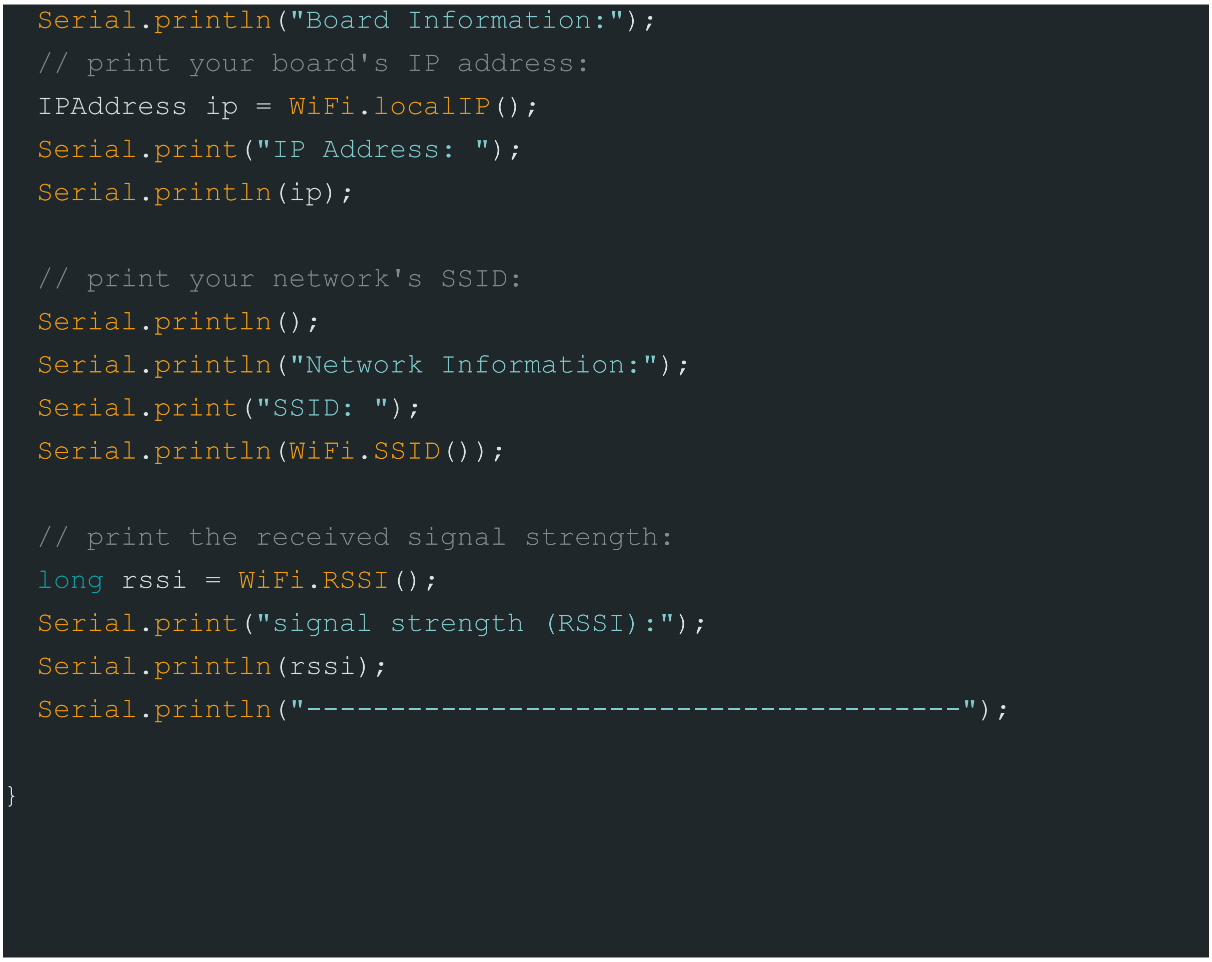
On the left side of the GUI there are options that control how the LEDs light up. The options are made using the G4P library. When an option is selected, a callback function is called which sends the option number to the Arduino (Appendix B.5).

Appendix

A1.

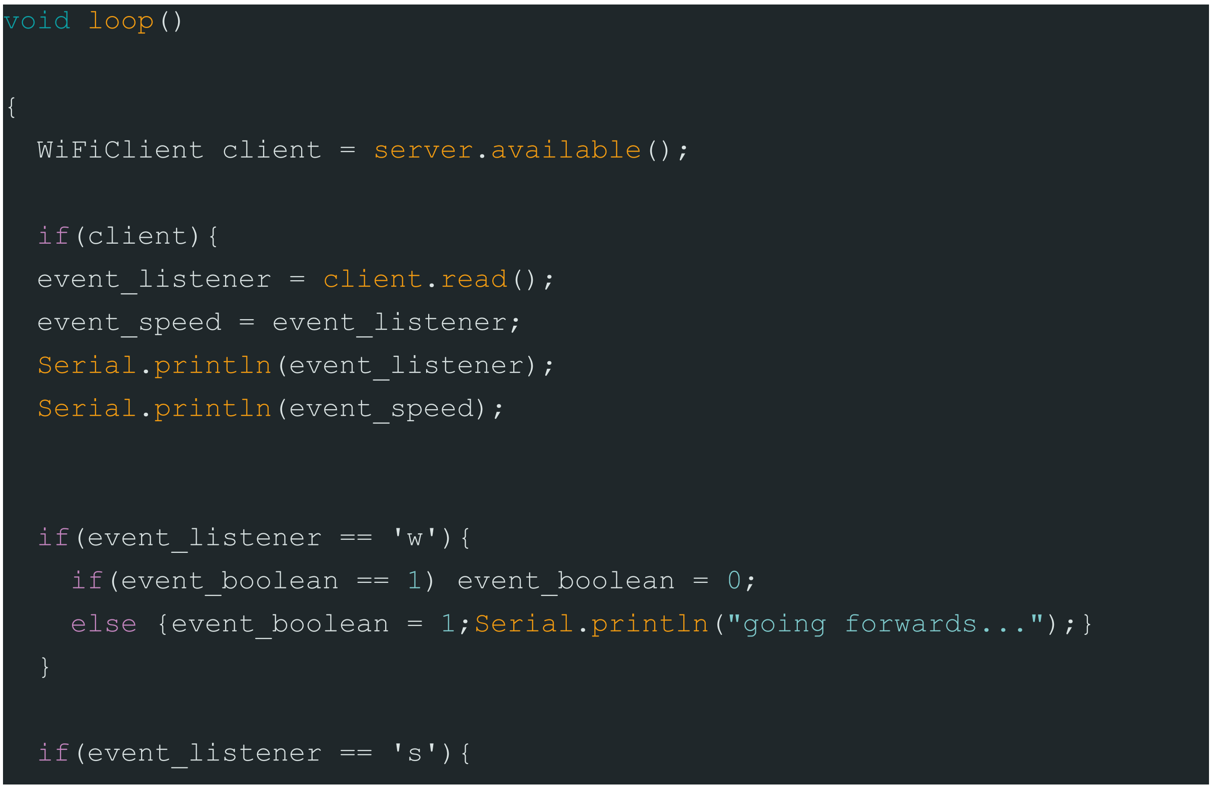
```
#include <WiFiNINA.h>

#define ENA 12 // connect ENA to pin 12
#define ENB 9 // connect ENB to pin 9

#define IN1 11 // connect IN1 to pin 11
#define IN2 10 // connect IN2 to pin 10

#define IN3 8
#define IN4 7

#define led1 16
#define led2 17
#define led3 19 //originally 16
#define led4 3
```

```
#define led5 2

int motorSpeed = 200;

char event_listener;

char event_speed;

int event_boolean; // 0 for braking, 1 for forward, 2 for backward

int led_mode = 0;
int led_counter = 0;
int max_counter = 0;
unsigned long currentMillis = millis();
unsigned long previousMillis = 0;

char ssid[] = "Athens2016";                // your network SSID (name)
char pass[] = "Arduino2016";                // your network password
(use for WPA, or use as key for WEP)
int status = WL_IDLE_STATUS;             // the Wi-Fi radio's status

WiFiServer server(80);

void setup()

{
  Serial.begin(9600);
 pinMode(ENA, OUTPUT);

 pinMode(IN1, OUTPUT);

 pinMode(IN2, OUTPUT);

 pinMode(ENB, OUTPUT);

 pinMode(IN3, OUTPUT);

 pinMode(IN4, OUTPUT);

 digitalWrite(ENA, LOW);

 digitalWrite(IN1, LOW);
 digitalWrite(IN2, LOW);

 digitalWrite(ENB, LOW);

 digitalWrite(IN3, LOW);

 digitalWrite(IN4, LOW);

 //LED initialization

 pinMode(led1, OUTPUT);
 digitalWrite(led1, LOW);
  pinMode(led2, OUTPUT);
 digitalWrite(led2, LOW);
  pinMode(led3, OUTPUT);
 digitalWrite(led3, LOW);
  pinMode(led4, OUTPUT);
 digitalWrite(led4, LOW);
  pinMode(led5, OUTPUT);
 digitalWrite(led5, LOW);

 setupServer();
}


void setupServer() {
  // attempt to connect to Wi-Fi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to network: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }

  // you're connected now, so print out the data:
  Serial.println("You're connected to the network");
  Serial.println("----------------------------------------");

  server.begin();
```



```
  Serial.println("Board Information:");
  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print your network's SSID:
  Serial.println();
  Serial.println("Network Information:");
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.println(rssi);
  Serial.println("---------------------------------------");

}
```

A2.



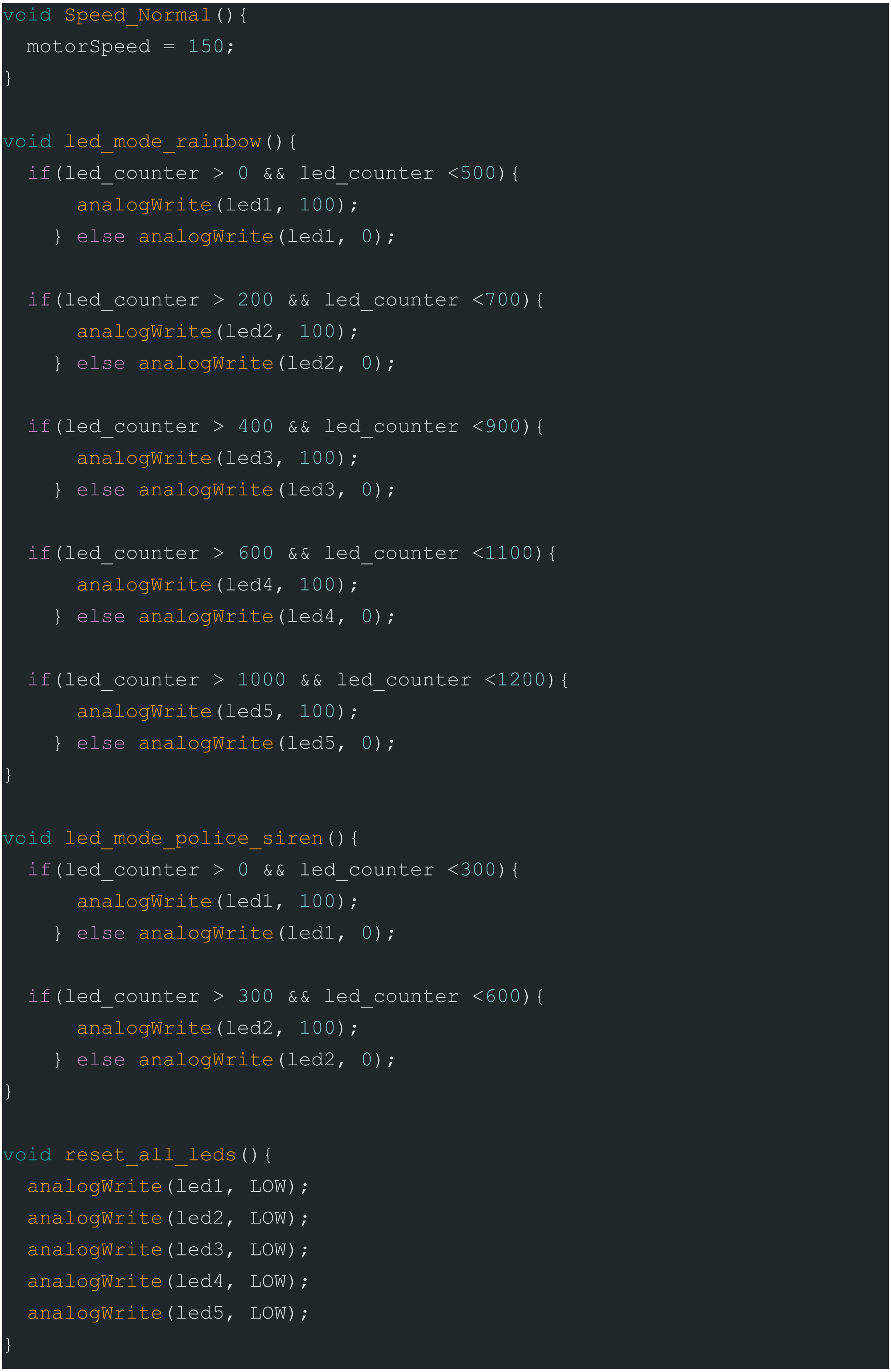
```
void loop()

{
  WiFiClient client = server.available();

  if(client){
  event_listener = client.read();
  event_speed = event_listener;
  Serial.println(event_listener);
  Serial.println(event_speed);


  if(event_listener == 'w'){
    if(event_boolean == 1) event_boolean = 0;
    else {event_boolean = 1;Serial.println("going forwards...");}
  }

  if(event_listener == 's'){
    if(event_boolean == 2) event_boolean = 0;
    else {event_boolean = 2;;Serial.println("going backwards...");}
  }

  if(event_listener == 'a'){
    if(event_boolean == 3) event_boolean = 0;
    else {event_boolean = 3;Serial.println("turning left...");}
  }

  if(event_listener == 'd'){
    if(event_boolean == 4) event_boolean = 0;
    else {event_boolean = 4;Serial.println("turning right...");}
  }

  if(event_listener == 'q'){
    if(event_speed == 1) event_speed = 0;
    else {event_speed = 1;Serial.println("speeding up...");}
  }

  if(event_listener == 'e'){
    if(event_speed == 2) event_speed = 0;
    else {event_speed = 2;Serial.println("slowing down...");}
  }

  if(event_listener == 'r'){
    if(event_speed == 3) event_speed = 0;
    else {event_speed = 3;Serial.println("normal speed...");}
  }

  if(event_listener == '0'){
    if(led_mode != 0) {led_mode = 0; reset_all_leds();}
  }

  if(event_listener == '1'){
    if(led_mode == 1) {led_mode = 0; reset_all_leds();}
    else {led_mode = 1;led_counter = 0;}
  }

  if(event_listener == '2'){
    if(led_mode == 2) {led_mode = 0;  reset_all_leds();}
    else {led_mode = 2;led_counter = 0;}
  }
```

```
if(event_listener == '3'){
  if(led_mode == 3) {led_mode = 0; reset_all_leds();}
  else {led_mode = 3;led_counter = 0;}
}

if(event_listener == '4'){
  if(led_mode == 4) {led_mode = 0; reset_all_leds();}
  else {led_mode = 4;led_counter = 0;}
}

if(event_listener == '5'){
  if(led_mode == 5) {led_mode = 0; reset_all_leds();}
  else {led_mode = 5;led_counter = 0;}
}

if(event_boolean == 0){Serial.println("braking...");}

switch(event_boolean){
  case 0:
    MotorA_Brake();
    MotorB_Brake();
    break;
  case 1:
    MotorA_Direction1(motorSpeed);
    MotorB_Direction1(motorSpeed);
    break;
  case 2:
    MotorA_Direction2(motorSpeed);
    MotorB_Direction2(motorSpeed);
    break;
  case 3:
    Turn_Left(motorSpeed);
    break;
  case 4:
    Turn_Right(motorSpeed);
    break;
  default:
    Serial.println("Smth's wrong");
}

  switch(event_speed){
  case 0:
    break;
    case 1:
      Speed_Up();
      break;
    case 2:
      Slow_Down();
      break;
    case 3:
      Speed_Normal();
    default:
      break;
    }

}
if(led_mode != 0){
switch(led_mode){
  case 0: reset_all_leds(); break;
  case 1: max_counter = 1200; led_mode_rainbow(); break;
  case 2: max_counter = 600; led_mode_police_siren(); break;
  case 3: max_counter = 400; led_mode_random(); break;
  case 4: max_counter = 1700; led_mode_white_blink(); break;
  case 5: max_counter = 1700; led_mode_all_blink(); break;
  default: Serial.println("it shouldn't have gotten here"); break;
  }


currentMillis = millis();
if (currentMillis - previousMillis >= 1) {
    // Save the current time for the next iteration
    previousMillis = currentMillis;
    if(led_counter < max_counter) led_counter++;
    if(led_counter == max_counter) {reset_all_leds(); led_counter = 0;}

  }
}

}
```

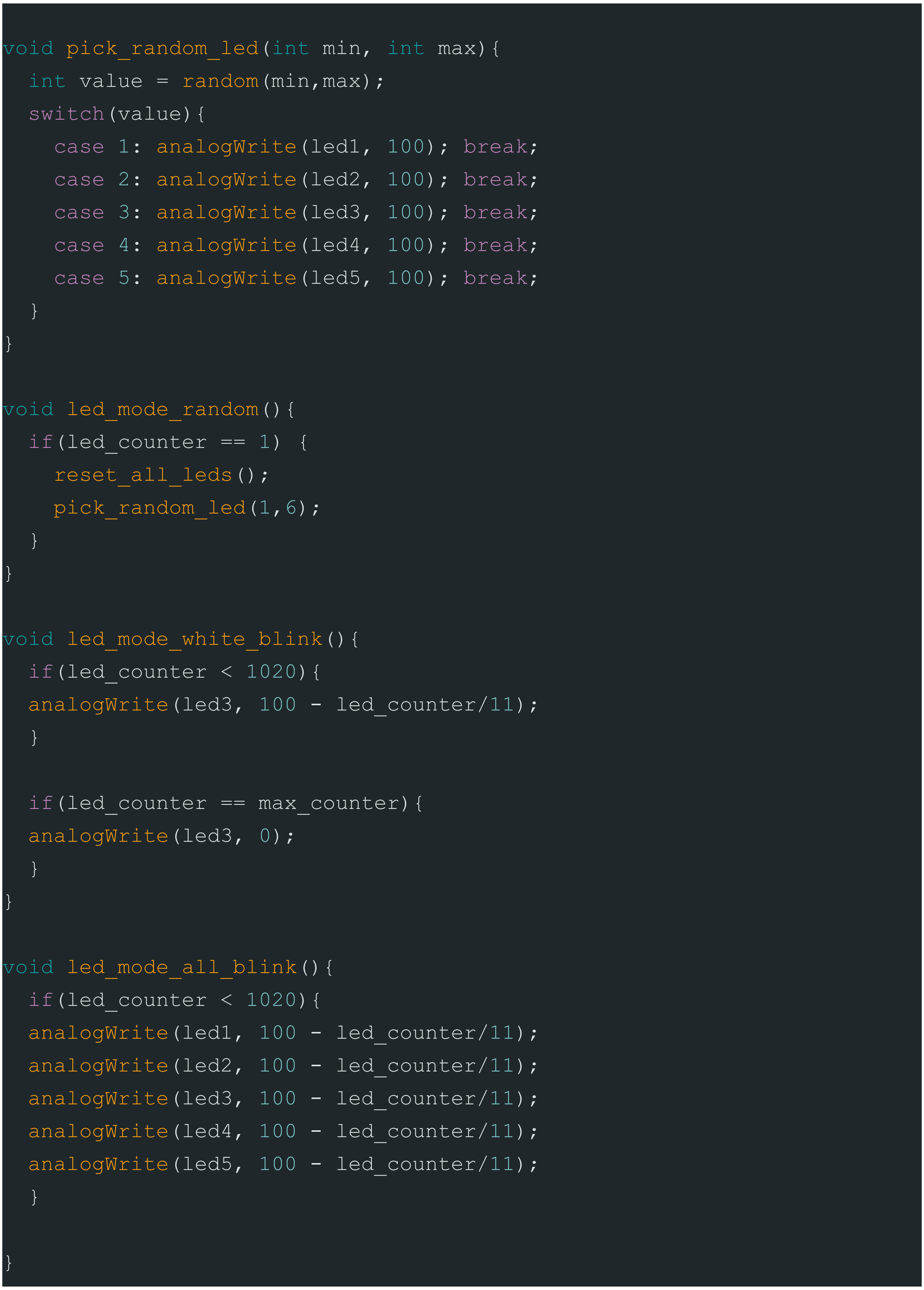
A3.

```
void MotorA_Direction1(int motorSpeed)

{ //forward
 analogWrite(ENA, motorSpeed);
 digitalWrite(IN1, LOW);

 digitalWrite(IN2, HIGH);
}



void MotorA_Direction2(int motorSpeed)

{ //backwards
 analogWrite(ENA, motorSpeed);

 digitalWrite(IN1, HIGH);

 digitalWrite(IN2, LOW);
}



void MotorA_Brake()

{
 digitalWrite(ENA, LOW);

 digitalWrite(IN1, LOW);

 digitalWrite(IN2, LOW);
}


void MotorB_Direction1(int motorSpeed)

{ //forward
 analogWrite(ENB, motorSpeed);

 digitalWrite(IN3, LOW);

 digitalWrite(IN4, HIGH);
}
```

```
void MotorB_Direction2(int motorSpeed)

{ //backwards
 analogWrite(ENB, motorSpeed);

 digitalWrite(IN3, HIGH);

 digitalWrite(IN4, LOW);
}




void MotorB_Brake()

{
 digitalWrite(ENB, LOW);

 digitalWrite(IN3, LOW);

 digitalWrite(IN4, LOW);
}

void Turn_Left(int motorSpeed)
{
  MotorA_Direction2(120);
  MotorB_Direction1(120);

}

void Turn_Right(int motorSpeed)
{
  MotorA_Direction1(120);
  MotorB_Direction2(120);
}

void Speed_Up(){
  motorSpeed = 200;
}

void Slow_Down(){
  motorSpeed = 100;
}
```



```
void Speed_Normal(){
  motorSpeed = 150;
}

void led_mode_rainbow(){
  if(led_counter > 0 && led_counter <500){
      analogWrite(led1, 100);
    } else analogWrite(led1, 0);

  if(led_counter > 200 && led_counter <700){
      analogWrite(led2, 100);
    } else analogWrite(led2, 0);

  if(led_counter > 400 && led_counter <900){
      analogWrite(led3, 100);
    } else analogWrite(led3, 0);

  if(led_counter > 600 && led_counter <1100){
      analogWrite(led4, 100);
    } else analogWrite(led4, 0);

  if(led_counter > 1000 && led_counter <1200){
      analogWrite(led5, 100);
    } else analogWrite(led5, 0);
}

void led_mode_police_siren(){
  if(led_counter > 0 && led_counter <300){
      analogWrite(led1, 100);
    } else analogWrite(led1, 0);

  if(led_counter > 300 && led_counter <600){
      analogWrite(led2, 100);
    } else analogWrite(led2, 0);
}

void reset_all_leds(){
  analogWrite(led1, LOW);
  analogWrite(led2, LOW);
  analogWrite(led3, LOW);
  analogWrite(led4, LOW);
  analogWrite(led5, LOW);
}
```



```
void pick_random_led(int min, int max){
  int value = random(min,max);
  switch(value){
    case 1: analogWrite(led1, 100); break;
    case 2: analogWrite(led2, 100); break;
    case 3: analogWrite(led3, 100); break;
    case 4: analogWrite(led4, 100); break;
    case 5: analogWrite(led5, 100); break;
  }
}

void led_mode_random(){
  if(led_counter == 1)  {
    reset_all_leds();
    pick_random_led(1,6);
  }
}

void led_mode_white_blink(){
  if(led_counter < 1020){
  analogWrite(led3, 100 - led_counter/11);
  }

  if(led_counter == max_counter){
  analogWrite(led3, 0);
  }
}

void led_mode_all_blink(){
  if(led_counter < 1020){
  analogWrite(led1, 100 - led_counter/11);
  analogWrite(led2, 100 - led_counter/11);
  analogWrite(led3, 100 - led_counter/11);
  analogWrite(led4, 100 - led_counter/11);
  analogWrite(led5, 100 - led_counter/11);
  }


}
```

B.1

```
import processing.net.*;
import g4p_controls.*;

enum Axis {
 X(0),
 Y(1);

 public int index;
 private Axis(int index) {
   this.index = index;
 }
}

enum Direction {
 W(0, 'w'),
 A(1, 'a'),
 S(2, 's'),
 D(3, 'd');

 public int index;
 public char character;
 private Direction(int index, char c) {
   this.index = index;
   this.character = c;
 }

 public static Direction getDirFromChar(char c) {
    switch(c) {
     case 'w':
     case 'W':
       return Direction.W;
     case 'a':
     case 'A':
       return Direction.A;
     case 's':
     case 'S':
       return Direction.S;
     case 'd':
     case 'D':
       return Direction.D;
      default:
        return null;
    }
 }
};

Client client;

int[][] buttonLocation = new int[4][2];
boolean[] buttonClicked = new boolean[4];
int rectSize;
color rectColor, rectHighlight;

char currentDir = '\0';
char currentSpeed = '\0';

String ip = "192.168.0.106";
int port = 80;

GToggleGroup ledOptions;
GOption ledOff, ledRainbow, ledSiren, ledRandom, ledWhiteBlink,
ledAllBlink;
```

B.2

```
void setup() {
  size(640, 360);
  background(255);

  client = new Client(this, ip, port);

  setupButtons();
  setupLedOptions();
}

void setupButtons() {
  rectColor = color(51);
  rectHighlight = color(0, 0, 190);
  rectSize = min(width/5, height/5, 90);

  buttonLocation[Direction.W.index][Axis.X.index] = buttonX(0, 0);
  buttonLocation[Direction.W.index][Axis.Y.index] = buttonY(10, -1);

  buttonLocation[Direction.D.index][Axis.X.index] = buttonX(10, 1);
  buttonLocation[Direction.D.index][Axis.Y.index] = buttonY(0, 0);

  buttonLocation[Direction.S.index][Axis.X.index] = buttonX(0, 0);
  buttonLocation[Direction.S.index][Axis.Y.index] = buttonY(10, 1);

  buttonLocation[Direction.A.index][Axis.X.index] = buttonX(10, -1);
  buttonLocation[Direction.A.index][Axis.Y.index] = buttonY(0, 0);
}

void setupLedOptions() {
  ledOptions = new GToggleGroup();
  int optionWidth = width/5;
  int optionHeight = height/10;

  ledOff = new GOption(this, 0, height/5 - optionHeight, optionWidth,
optionHeight);
  ledOff.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
  ledOff.setText("Off");
  ledOff.addEventHandler(this, "handleOff");

  ledRainbow = new GOption(this, 0, height/5, optionWidth,
optionHeight);
  ledRainbow.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
  ledRainbow.setText("Rainbow");
  ledRainbow.addEventHandler(this, "handleRainbow");

  ledSiren = new GOption(this, 0, height/5+optionHeight, optionWidth,
optionHeight);
  ledSiren.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
  ledSiren.setText("Police Siren");
  ledSiren.addEventHandler(this, "handleSiren");

  ledRandom = new GOption(this, 0, height/5+optionHeight*2,
optionWidth, optionHeight);
  ledRandom.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
  ledRandom.setText("Disco Pogo dingelingeling");
  ledRandom.addEventHandler(this, "handleRandom");

  ledWhiteBlink = new GOption(this, 0, height/5+optionHeight*3,
optionWidth, optionHeight);
  ledWhiteBlink.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
  ledWhiteBlink.setText("White Blink");
  ledWhiteBlink.addEventHandler(this, "handleWhiteBlink");
  ledAllBlink = new GOption(this, 0, height/5+optionHeight*4, optionWidth, optionHeight);
  ledAllBlink.setTextAlign(GAlign.LEFT, GAlign.MIDDLE);
  ledAllBlink.setText("All Blink");
  ledAllBlink.addEventHandler(this, "handleAllBlink");

  ledOptions.addControl(ledOff);
  ledOptions.addControl(ledRainbow);
  ledOptions.addControl(ledSiren);
  ledOptions.addControl(ledRandom);
  ledOptions.addControl(ledWhiteBlink);
  ledOptions.addControl(ledAllBlink);
  ledRainbow.setSelected(true);
  client.write('1');
}

int buttonY(int offset, int dir) {
  return height/2 + dir * rectSize + dir * offset - rectSize/2;
}

int buttonX(int offset, int dir) {
  return width/2 + dir * rectSize + dir * offset - rectSize/2;
}
```

B.3

```
void keyTyped() {
  switch(key) {
  case 'w':
  case 's':
  case 'a':
  case 'd':
    Direction dir = Direction.getDirFromChar(key);
    buttonClicked[dir.index] = true;
    handleDir(key);
    break;

  case 'q':
  case 'e':
  case 'r':
    handleSpeed(key);
  case 'W':
  case 'S':
  case 'A':
  case 'D':
    Direction dir2 = Direction.getDirFromChar(toLowerCase(key));
    buttonClicked[dir2.index] = true;
    handleDir(toLowerCase(key));
    break;

  case 'Q':
  case 'E':
  case 'R':
    handleSpeed(toLowerCase(key));
    break;

   default:
     return;
  }
}

void keyReleased() {
  switch(key) {
    case 'w':
    case 's':
    case 'a':
    case 'd':
      Direction dir = Direction.getDirFromChar(key);
      buttonClicked[dir.index] = false;
      currentDir = '\0';
      client.write(key);
      break;

    case 'e':
    case 'q':
    case 'r':
      currentSpeed = '\0';
      client.write(key);
      break;

    case 'W':
    case 'S':
    case 'A':
    case 'D':
      Direction dir2 = Direction.getDirFromChar(toLowerCase(key));
      buttonClicked[dir2.index] = false;
      currentDir = '\0';
      client.write(toLowerCase(key));
      break;

    case 'Q':
    case 'E':
    case 'R':
      currentSpeed = '\0';
      client.write(toLowerCase(key));
      break;
  }
}

char toLowerCase(char c) {
  return (char) (c - 'A' +'a');
}

void handleDir(char keyChar) {
  if (currentDir != keyChar) {
      client.write(keyChar);
      println(keyChar);
      currentDir = keyChar;
    }
}

void handleSpeed(char keyChar) {
   if (currentSpeed != keyChar) {
      client.write(keyChar);
      println(keyChar);
      currentSpeed = keyChar;
   }
}
```

B.4

```
void draw() {
  background(255);

  for(Direction dir: Direction.values()) {
    if (buttonClicked[dir.index]) {
      fill(rectHighlight);
    } else {
      fill(rectColor);
    }
    stroke(255);
    rect(buttonLocation[dir.index][Axis.X.index],
buttonLocation[dir.index][Axis.Y.index], rectSize, rectSize);
  }
}

void mousePressed() {
  for (Direction dir: Direction.values()) {
    if(overRect(buttonLocation[dir.index][Axis.X.index],
buttonLocation[dir.index][Axis.Y.index], rectSize, rectSize)) {
      boolean prevVal = buttonClicked[dir.index];
      resetButtons();
      buttonClicked[dir.index] = !prevVal;
      print(dir.toString());
      println(buttonClicked[dir.index]);
      client.write(dir.character);
    }
  }
}

boolean overRect(int x, int y, int width, int height)  {
  if (mouseX >= x && mouseX <= x+width &&
      mouseY >= y && mouseY <= y+height) {
    return true;
  } else {
    return false;
  }
}

void resetButtons() {
  for(Direction dir: Direction.values()) {
    if (buttonClicked[dir.index]) {
      client.write(dir.character);
    }
    buttonClicked[dir.index] = false;
  }
}
```

B.5

```
public void handleOff(GOption option, GEvent event) {
  client.write('0');
  println("0");
}

public void handleRainbow(GOption option, GEvent event) {
  client.write('1');
  println("1");
}

public void handleSiren(GOption option, GEvent event) {
  client.write('2');
  println("2");
}

public void handleRandom(GOption option, GEvent event) {
  client.write('3');
  println("3");
}

public void handleWhiteBlink(GOption option, GEvent event) {
  client.write('4');
  println("4");
}

public void handleAllBlink(GOption option, GEvent event) {
  client.write('5');
  println("5");
}
```

Obtained Results:

After completing this project, we managed to create a robot that can be fully controlled remotely from the laptop and has multiple functionalities such as going forwards or backwards, rotating and turning on LEDs.

Team:

This project has been created by:

- Stan-Soponaru Ioan-Alexandru
- Lin Selina