Communication and Electronics Department
Microprocessors and Interconnection - EEC 243
Spring 2022 - 2023

Software and Hardware Report

Name	ID
Ahmed Osama Mohamed Afifi	20010038
Mazen Mohamed Hassanen	20011161
Mostafa Mohamed Abdel-Azeem Hassanen	20011950
Mohamed Ashraf Elsayed Mahmoud	20011488
Ahmed Abdel-Hakem Abdel-Salam Ali	20010124

Department:	Communication and Electronics

Group Number: 44.

Hardware Project title:

Zigzag and Circular path Autonomous Moving Robot. (5th Project in the suggestion list).

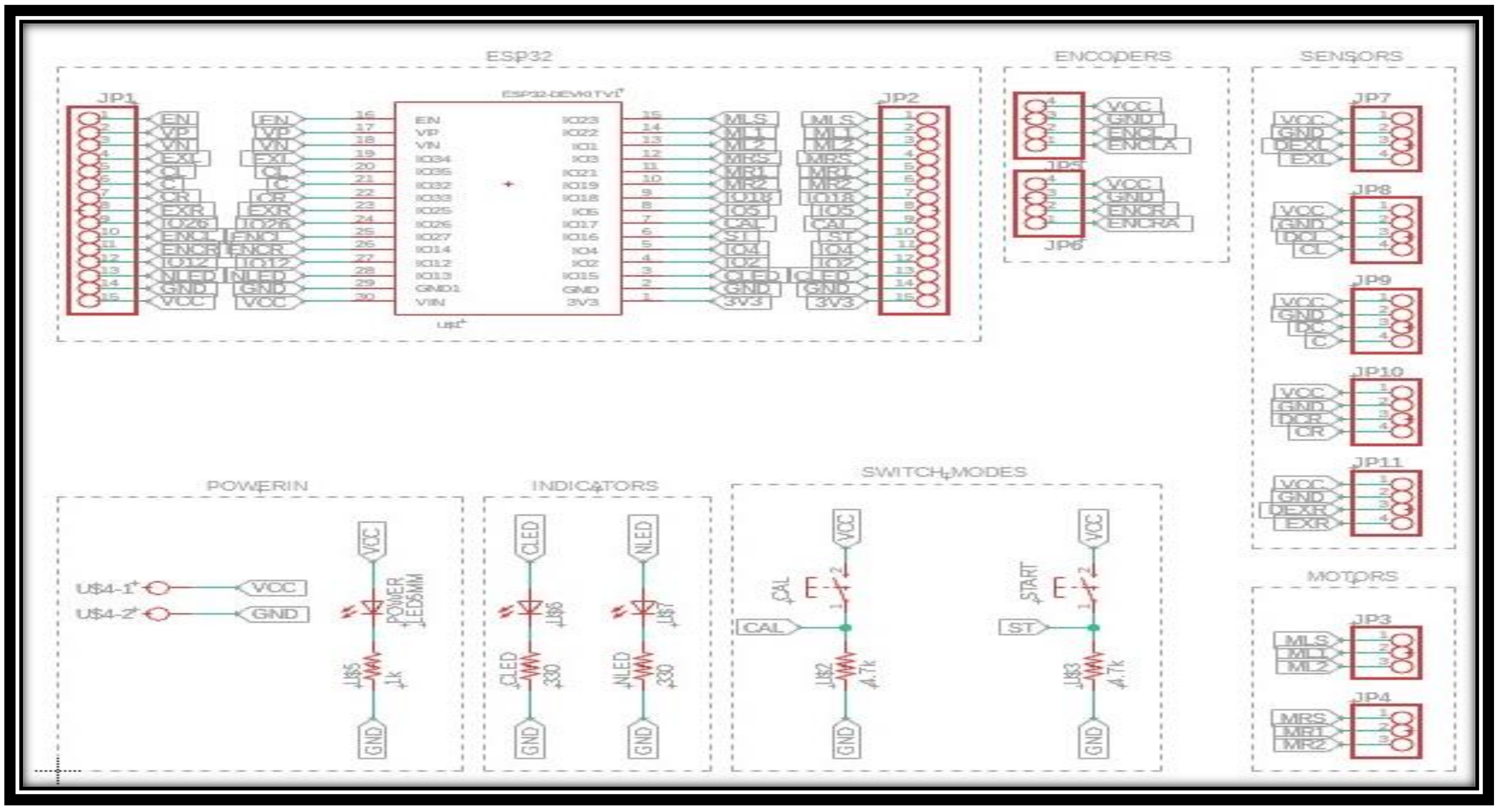
Aim of the Hardware Project:

The aim of creating this autonomous moving robot is to challenge ourselves to design a machine that can autonomously follow a designated path, whether it is zigzag or circular, with high precision by using five sensors and well-organized and uncomplicated code to achieve a reliable and efficient performance. This was complex system as we introduced PID (Proportional-Integral-Derivative controller) to control the motion of the autonomous robot through the feedback to minimize the error and this was a big challenge for us and took much time and effort to achieve the desired behavior.

List of the used components:

1- ESP32 Microcontroller.
2- Five TCRT5000 IR sensors.
3- Two miniQ-Dc motors with gearbox.
4- Two Wheels.
5- A Free Wheel.
6- Chassis of the robot.
7- H- Bridge.
8- Four Lithium batteries.

Schematic of the circuit implemented:



Procedure to use this circuit:

1- Switch on the robot by switching the Power button. This button is present to control the robot from being on all the time and to keep the batteries from running out.

2- Draw the desired path for the robot to follow by black color.

3- Press on the calibration button to make the robot start calibration mode by rotating around itself and take the values from the IR sensor and calculate the threshold, maximum and minimum value for each sensor.

4- After calibration, Put the robot on the designed path.

5- Press on the start button then wait one second and the robot will start to move on the black designed path.

6- The robot will follow the path and change its direction by the five IR sensors and by the help of the PID feedback to fix the errors in tracking the line and motor speed control.

7- Switch the Power button to stop the robot from moving.

Budget of the project:

- **Components budget (You can click on the component to view it on the store) :**

1- ESP32 = 290 L.E

2- IR sensors = 35 * 5= 175 L.E

3- Heat shrink /Jumpers/ Resistors/ Data cables / Data pins / Male pin headers / Female pin headers / Rosetta pluggable / LEDs = 80 L.E

4- Batteries = 4 * 25 = 100 L.E

5-MiniQ Dc with gearbox Motors = 2 * 150 = 300 L.E

6- Chassis of the robot = 145 L.E

8- Double face = 30 L.E

9- Nuts and Bolts = 30 L.E

10- Map = 30 L.E

11- Components to make PCB (Soldering iron / Soldering Wire / Solution / Board / Driller)

= 500 L.E

- **Workspace budget:**

-fees were around 400 to 500 L.E for the four days distributed on the team.

Challenges that the team had and how to overcome them:

1.we encountered some problems regarding the PCB fabrication like:

a) the tracks on the glossy paper didn't stick very well on the copper PCB so we needed to use another one after cleaning the copper PCB with the dish wire.

b) some tracks were cut or scratched by acid, but we managed to put some tin on them and tested them with the Avo-meter and it functions just fine.

2.We had a little difficulty in figuring out the PID constants value so eventually we got to know them by using *The Ziegler-Nichols closed-loop tuning method*.

3. One of the motors had a defect in its gearbox and after replacement the robot moved better. However, it was not giving such a good response, therefore we replaced the two motors with the miniQ motors.

4. A precaution was taken while writing the software regarding motor control by introducing the correction factor for the motor, which solves reverse direction due to reverse connection as if the motor is found to be rotating in the opposite direction, instead of reversing its terminals, simply change the correction factor from 1 to -1.

References:

1-https://www.youtube.com/watch?v=dOVjb2wXI84&ab_channel=LowLevelLearning

2-https://www.youtube.com/playlist?list=PLfgCIULRQavz1evD_oQMN4ytBSf3rSJqi

3- https://www.youtube.com/playlist?list=PL-h0OQrz2Hc9qRzKQPAgRWoHS_QkndCus

4- https://randomnerdtutorials.com/getting-started-with-esp32/

5- https://www.youtube.com/playlist?list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y

6- https://www.youtube.com/watch?v=UR0hOmjaHp0

7- https://www.youtube.com/watch?v=XfAt6hNV8XM

Code:

1-mDriver.h:

```
#ifndef mDriver_h
#define mDriver_h
#include <Arduino.h>

//used in some functions so you don't have to send a speed
#define DEFAULTSPEED 255


class Motor
{
  public:
    // Constructor. Mainly sets up pins and speed.
    Motor(int In1pin, int In2pin, int PWMpin, int correction);

    // Drive in direction given by sign, at speed given by magnitude of the
parameter.
    void drive(int speed);
    // drive(), but with a delay(duration)
    void drive(int speed, int duration);

    // Short brake
    void brake();

    // Stop
    void stop();

  private:
    //variables for the 2 inputs, PWM input, Offset value, and the Standby pin
    int In1, In2, PWM, Correction;

  //private functions that spin the motor CC and CCW
  void fwd(int speed);
  void rev(int speed);
};

/**
 * @brief
 * Takes 2 motors and goes forward, if it does not go forward adjust correction
factor values until it does.
 * These will also take a negative number and go backwards.
 * There is also an optional speed input, if speed is not used, the function will
use the DEFAULTSPEED constant.
 *
 * @param left left motor
 * @param right right motor
 * @param speed motors speed
 */
void forward(Motor left, Motor right, int speed);
void forward(Motor left, Motor right);

/**
 * @brief
 * Similar to forward, will take 2 motors and go backwards.
 * This will take either a positive or negative number and will go backwards
either way.
 * Once again the speed input is optional and will use DEFAULTSPEED if it is not
defined.
 *
 * @param left left motor
 * @param right right motor
 * @param speed motors speed
 */

void back(Motor left, Motor right, int speed);
void back(Motor left, Motor right);

/**
 * @brief
 * Left and right take 2 motors, and it is important the order they are sent.
 * The left motor should be on the left side of the bot.
 * These functions also take a speed value
 *
 * @param left left motor
 * @param right right motor
 * @param speed motors speed
 */
void left(Motor left, Motor right, int speed);
void right(Motor left, Motor right, int speed);

/**
 * @brief
 * These function takes 2 motors and and stops them.
 *
 * @param left left motor
 * @param right right motor
 */
void brake(Motor left, Motor right);
void stop(Motor left, Motor right);
#endif
```

2-mDriver.cpp:

```
#include "mDriver.h"
#include <Arduino.h>

Motor::Motor(int In1pin, int In2pin, int PWMpin, int correction)
{
  In1 = In1pin;
  In2 = In2pin;
  PWM = PWMpin;
  Correction = correction;

  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(PWM, OUTPUT);
}

void Motor::drive(int speed)
{
  speed = speed * Correction;
  if (speed>=0) fwd(speed);
  else rev(-speed);
}
```

```
void Motor::drive(int speed, int duration)
{
  drive(speed);
  delay(duration);
}

void Motor::fwd(int speed)
{
   digitalWrite(In1, HIGH);
   digitalWrite(In2, LOW);
   analogWrite(PWM, speed);

}

void Motor::rev(int speed)
{
   digitalWrite(In1, LOW);
   digitalWrite(In2, HIGH);
   analogWrite(PWM, speed);
}

void Motor::brake()
{
   digitalWrite(In1, HIGH);
   digitalWrite(In2, HIGH);
   analogWrite(PWM,0);
}
void Motor::stop()
{
   digitalWrite(In1, LOW);
   digitalWrite(In2, LOW);
   analogWrite(PWM,0);
}

void forward(Motor left, Motor right, int speed)
{
  left.drive(speed);
  right.drive(speed);
}
void forward(Motor left, Motor right)
{
  left.drive(DEFAULTSPEED);
  right.drive(DEFAULTSPEED);
}
```

```
void back(Motor left, Motor right, int speed)
{
  int temp = abs(speed);
  left.drive(-temp);
  right.drive(-temp);
}
void back(Motor left, Motor right)
{
  left.drive(-DEFAULTSPEED);
  right.drive(-DEFAULTSPEED);
}
void left(Motor left, Motor right, int speed)
{
  int temp = abs(speed)/2;
  left.drive(-temp);
  right.drive(temp);

}
void right(Motor left, Motor right, int speed)
{
  int temp = abs(speed)/2;
  left.drive(temp);
  right.drive(-temp);

}
void brake(Motor left, Motor right)
{
  left.brake();
  right.brake();
}

void stop(Motor left, Motor right)
{
  left.stop();
  right.stop();
}
```

3-PID.h:

```
#ifndef PID_h
#define PID_h
#include "mDriver.h"
#include <Arduino.h>


class PID
{
  public:
    PID(float kP, float kI, float kD);
    int calculate(int mv);
    void setConstants(float kP, float kI, float kD);
    void setSetpoint(int setPoint);
    void setSpeeds(int v);
    void setConstrains(int lower_bound, int upper_bound);
    void calibrate(Motor leftMotor, Motor rightMotor, int minValues[], int
maxValues[], int threshold[], int sensors[]);
    void linefollow(Motor leftMotor, Motor rightMotor, int lsp, int rsp);
  private:
    float kp=0,ki=0,kd=0;
    int sp = 0;
    int error = 0;
    int lfspeed = 0;
    int lower = -255, upper = 255;
    int P=0, D=0, I=0, previousError=0, PIDvalue=0;
};
#endif
```

4-PID.cpp:

```
#include "PID.h"


PID::PID(float kP, float kI, float kD)
{
  kp = kP;
  ki = kI;
  kd = kD;
}

void PID::setSpeeds(int v)
{
  lfspeed = v;
}
```

```
void PID::setConstants(float kP, float kI, float kD)
{
  kp = kP;
  ki = kI;
  kd = kD;
}

void PID::setSetpoint(int setPoint)
{
  sp = setPoint;
}

void PID::setConstrains(int lower_bound, int upper_bound)
{
  lower = lower_bound;
  upper = upper_bound;
}

int PID::calculate(int mv)
{
  P = mv;
  I = I + mv;
  D = mv - previousError;

  PIDvalue = (kp * P) + (ki * I) + (kd * D);

  return PIDvalue;
}

void PID::linefollow(Motor leftMotor, Motor rightMotor, int lsp, int rsp)
{
  error = sp;

  PIDvalue = calculate(error);

  previousError = error;



  lsp = lfspeed - PIDvalue;
  rsp = lfspeed + PIDvalue;


  if (lsp > upper) {
    lsp = upper;
  }
  if (lsp < lower) {
    lsp = lower;
  }
  if (rsp > upper) {
    rsp = upper;
  }
  if (rsp < lower) {
    rsp = lower;
  }
  leftMotor.drive(lsp);
  rightMotor.drive(rsp);
}
void PID::calibrate(Motor leftMotor, Motor rightMotor, int minValues[], int
maxValues[], int threshold[], int sensors[])
{
  for ( int i = 0; i < 5; i++)
  {
    minValues[i] = analogRead(sensors[i]);
    maxValues[i] = analogRead(sensors[i]);
  }
  for (int i = 0; i < 5000; i++)
  {
    leftMotor.drive(60);
    rightMotor.drive(-60);

    for ( int i = 0; i < 5; i++)
    {
      if (analogRead(sensors[i]) < minValues[i])
      {
        minValues[i] = analogRead(sensors[i]);     //minValues for white regions
      }
      if (analogRead(sensors[i]) > maxValues[i])
      {
        maxValues[i] = analogRead(sensors[i]);     //maxValues for dark regions
      }
    }
  }
  for ( int i = 0; i < 5; i++)
  {
    threshold[i] = (minValues[i] + maxValues[i]) / 2;
    //Serial.print(threshold[i]);
    //Serial.print("   ");
  }
  //Serial.println();
  stop(leftMotor, rightMotor);
}
```

5-CarNavigation.ino:

```
#include "PID.h"
#include "mDriver.h"

#include <BluetoothSerial.h>

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;
String message = "";

// Motor pins
#define AIN1 22
#define BIN1 21
#define AIN2 1
#define BIN2 19
#define PWMA 23
#define PWMB 3


// IR pins
#define extremeLeft 34
#define centerLeft 35
#define center 32
#define centerRight 33
#define extremeRight 25

// Calibration and normal modes' pins
#define calibration 17
#define normal 16

// Indicators
#define calibrationLED 15
#define normalLED 13

// Correction factors for easily modifying motor configuration
// line up with function names like forward.  Value can be 1 or -1
const int correctionA = -1;
const int correctionB = -1;


// Initializing motors.  The library will allow you to initialize as many motors
as you have memory for.
Motor leftMotor = Motor(AIN1, AIN2, PWMA, correctionA);
Motor rightMotor = Motor(BIN1, BIN2, PWMB, correctionB);
```

```
// Motor speeds
int lsp, rsp;
int lfspeed = 255; // standard speed can be modified later

// PID constants
float Kp = 0;
float Kd = 0;
float Ki = 0;
// set-point variable
int sp = 0;
PID carPID(Kp, Ki, Kd); // PID object

// color map values
int minValues[5], maxValues[5], threshold[5], sensors[5];

void setup()
{
  Serial.begin(115200);
  SerialBT.begin("Hossam Hassan");
  Serial.println("Car Started! Ready to pair...");

  carPID.setSpeeds(lfspeed);
  carPID.setConstrains(0, 255);

  Kp = 5;
  Kd = 1;
  Ki = 0.001;
  carPID.setConstants(Kp, Ki, Kd);

  pinMode(calibration, INPUT);
  pinMode(normal, INPUT);
  pinMode(calibrationLED, OUTPUT);
  pinMode(normalLED, OUTPUT);
  digitalWrite(normalLED, LOW);

  sensors[0] = extremeLeft;
  sensors[1] = centerLeft;
  sensors[2] = center;
  sensors[3] = centerRight;
  sensors[4] = extremeRight;

  for(int i = 0; i < 5; i++)
  {
    pinMode(sensors[i], INPUT);
  }
}
```

```
void loop()
{
  // waiting for calibration
  while ((!digitalRead(calibration)) && (message != "0")) {
      if (SerialBT.available()){
        char incomingChar = SerialBT.read();
        if (incomingChar != '\n'){
          message += String(incomingChar);
        }
        else{
          message = "";
        }
      }
    }

  SerialBT.println("Calibrating...");
  Serial.println("Calibrating...");
  digitalWrite(calibrationLED, HIGH);
  digitalWrite(normalLED, LOW);
  delay(1000);
  carPID.calibrate(leftMotor, rightMotor, minValues, maxValues, threshold,
sensors); // calibration mode
  digitalWrite(calibrationLED, LOW);

  SerialBT.print("Thresholds\n");
  SerialBT.print("Extreme left: ");
  SerialBT.print(threshold[0]);
  SerialBT.print("\n");
  SerialBT.print("Center left: ");
  SerialBT.print(threshold[1]);
  SerialBT.print("\n");
  SerialBT.print("Center: ");
  SerialBT.print(threshold[2]);
  SerialBT.print("\n");
  SerialBT.print("Center right: ");
  SerialBT.print(threshold[3]);
  SerialBT.print("\n");
  SerialBT.print("Extreme right: ");
  SerialBT.print(threshold[4]);
  SerialBT.println("\n");
  // waiting to be set to normal mode
  while ((!digitalRead(normal)) && (message != "1")) {
    if (SerialBT.available()){
        char incomingChar = SerialBT.read();
        if (incomingChar != '\n'){
          message += String(incomingChar);
        }
        else{
          message = "";
        }
      }
    }
  SerialBT.println("Running...brrrr\n");
  Serial.println("Running...brrrr\n");
  digitalWrite(calibrationLED, LOW);
  digitalWrite(normalLED, HIGH);
  delay(1000);

  // Normal mode in action
  while (1)
  {
    if (SerialBT.available()){
    char incomingChar = SerialBT.read();
    if (incomingChar != '\n'){
      message += String(incomingChar);
    }
    else{
      message = "";
    }
  }
    if(digitalRead(normal) || message == "1")
    {
      SerialBT.println("Stopping...");
      stop(leftMotor, rightMotor);
      digitalWrite(calibrationLED, LOW);
      digitalWrite(normalLED, LOW);
      break;
    }
//    for ( int i = 0; i < 5; i++)
//    {
//      Serial.print(analogRead(sensors[i]));
//      Serial.print("   ");
//    }
//  Serial.println();
```

```
    // Extreme left turn when extremeLeft sensor detects dark region while
extremeRight sensor detects white region
    if (analogRead(extremeLeft) > threshold[0] && analogRead(extremeRight) <
threshold[4] )
    {
      //Serial.println("left");
      lsp = 0; rsp = lfspeed;
      leftMotor.drive(0);
      rightMotor.drive(rsp);
      //SerialBT.println("left");
      //left(leftMotor, rightMotor, lfspeed);
    }

    // Extreme right turn when extremeRight sensor detects dark region while
extremeLeft sensor detects white region
    else if (analogRead(extremeRight) > threshold[4] && analogRead(extremeLeft) <
threshold[0])
    {
      //Serial.println("right");
      lsp = lfspeed; rsp = 0;
      leftMotor.drive(lsp);
      rightMotor.drive(0);
      //SerialBT.println("right");
      //right(leftMotor, rightMotor, lfspeed);
    }
    else if (analogRead(center) > threshold[2])
    {
      // arbitrary PID constans will be tuned later
      // forward(leftMotor, rightMotor);
      // Kp = 5;
      // Kd = 1;
      // Ki = 0.001;
      // Serial.print("center\t");
      // Serial.println(Kp);
      // carPID.setConstants(Kp, Ki, Kd);
      sp = (analogRead(centerLeft) - analogRead(centerRight));
      //Serial.print("error\t");
      //Serial.println(sp);
      carPID.setSetpoint(sp);
      carPID.linefollow(leftMotor, rightMotor, lsp, rsp);
    }
  }
}
```

Mobile Application:

1-main.dart:

```dart
import 'package:flutter/material.dart';
import 'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';
import 'package:AutonomousCar/connection.dart';
import 'package:AutonomousCar/car.dart';

void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: FutureBuilder(
        future: FlutterBluetoothSerial.instance.requestEnable(),
        builder: (context, future) {
          if (future.connectionState == ConnectionState.waiting) {
            return Scaffold(
              body: Container(
                height: double.infinity,
                child: Center(
                  child: Icon(
                    Icons.bluetooth_disabled,
                    size: 200.0,
                    color: Colors.blue,
                  ),
                ),
              ),
            );
          } else if (future.connectionState == ConnectionState.done) {
            return Home();
          } else {
            return Home();
          }
        },
      ),
    );
  }
}
```

```
class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return SafeArea(
        child: Scaffold(
      appBar: AppBar(
        title: Text('Connection'),
      ),
      body: SelectBondedDevicePage(
        onCahtPage: (device1) {
          BluetoothDevice device = device1;
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) {
                return ChatPage(server: device);
              },
            ),
          );
        },
      ),
    ));
  }
}
```

2-device.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';
class BluetoothDeviceListEntry extends StatelessWidget {
  final Function onTap;
  final BluetoothDevice device;
  BluetoothDeviceListEntry({this.onTap, @required this.device});
  @override
  Widget build(BuildContext context) {
    return ListTile(
      onTap: onTap,
      leading: Icon(Icons.devices),
      title: Text(device.name ?? "Unknown device"),
      subtitle: Text(device.address.toString()),
      trailing: TextButton(
        onPressed: onTap,
        child: Text('Connect',
        style: TextStyle(
      color: Colors.green,
    ),),),)
    );
  }
}
```

3-connection.dart:

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';
import 'package:AutonomousCar/device.dart';

class SelectBondedDevicePage extends StatefulWidget {
  /// If true, on page start there is performed discovery upon the bonded
devices.
  /// Then, if they are not avaliable, they would be disabled from the selection.
  final bool checkAvailability;
  final Function onCahtPage;

  const SelectBondedDevicePage(
      {this.checkAvailability = true, @required this.onCahtPage});

  @override
  _SelectBondedDevicePage createState() => new _SelectBondedDevicePage();
}

enum _DeviceAvailability {
  no,
  maybe,
  yes,
}

class _DeviceWithAvailability extends BluetoothDevice {
  BluetoothDevice device;
  _DeviceAvailability availability;
  int rssi;

  _DeviceWithAvailability(this.device, this.availability, [this.rssi]);
}

class _SelectBondedDevicePage extends State<SelectBondedDevicePage> {
  // ignore: deprecated_member_use
  List<_DeviceWithAvailability> devices = List<_DeviceWithAvailability>();

  // Availability
  StreamSubscription<BluetoothDiscoveryResult> _discoveryStreamSubscription;
  bool _isDiscovering;

  _SelectBondedDevicePage();

  @override
  void initState() {
    super.initState();
    _isDiscovering = widget.checkAvailability;

    if (_isDiscovering) {
      _startDiscovery();
    }
    // Setup a list of the bonded devices
    FlutterBluetoothSerial.instance
        .getBondedDevices()
        .then((List<BluetoothDevice> bondedDevices) {
      setState(() {
        devices = bondedDevices
            .map(
              (device) => _DeviceWithAvailability(
                device,
                widget.checkAvailability
                    ? _DeviceAvailability.maybe
                    : _DeviceAvailability.yes,
              ),
            )
            .toList();
      });
    });
  }

  void _startDiscovery() {
    _discoveryStreamSubscription =
        FlutterBluetoothSerial.instance.startDiscovery().listen((r) {
      setState(() {
        Iterator i = devices.iterator;
        while (i.moveNext()) {
          var _device = i.current;
          if (_device.device == r.device) {
            _device.availability = _DeviceAvailability.yes;
            _device.rssi = r.rssi;
          }
        }
      });
    });
    _discoveryStreamSubscription.onDone(() {
      setState(() {
        _isDiscovering = false;
      });
    });
  }

  @override
```

```
  void dispose() {
    _discoveryStreamSubscription?.cancel();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    List<BluetoothDeviceListEntry> list = devices
        .map(
          (_device) => BluetoothDeviceListEntry(
            device: _device.device,
            onTap: () {
              widget.onCahtPage(_device.device);
            },
          ),
        )
        .toList();
    return ListView(
      children: list,
    );
  }
}
```

4-car.dart:

```
import 'dart:async';
import 'dart:convert';
import 'dart:typed_data';
import 'package:flutter/material.dart';
import 'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';

class ChatPage extends StatefulWidget {
  final BluetoothDevice server;

  const ChatPage({this.server});

  @override
  _ChatPage createState() => new _ChatPage();
}

class _Message {
  int whom;
  String text;

  _Message(this.whom, this.text);
}
```

```
class _ChatPage extends State<ChatPage> {
  static final clientID = 0;
  BluetoothConnection connection;

  List<_Message> messages = List<_Message>();
  String _messageBuffer = '';

  final TextEditingController textEditingController =
      new TextEditingController();
  final ScrollController listScrollController = new ScrollController();

  bool isConnecting = true;
  bool get isConnected => connection != null && connection.isConnected;

  bool isDisconnecting = false;

  @override
  void initState() {
    super.initState();

    BluetoothConnection.toAddress(widget.server.address).then((_connection) {
      print('Connected to the device');
      connection = _connection;
      setState(() {
        isConnecting = false;
        isDisconnecting = false;
      });

      connection.input.listen(_onDataReceived).onDone(() {
        // Example: Detect which side closed the connection
        // There should be `isDisconnecting` flag to show are we are (locally)
        // in middle of disconnecting process, should be set before calling
        // `dispose`, `finish` or `close`, which all causes to disconnect.
        // If we except the disconnection, `onDone` should be fired as result.
        // If we didn't except this (no flag set), it means closing by remote.
        if (isDisconnecting) {
          print('Disconnecting locally!');
        } else {
          print('Disconnected remotely!');
        }
        if (this.mounted) {
          setState(() {});
        }
      });
    }).catchError((error) {
      print('Cannot connect, exception occured');
      print(error);
```

```dart
  });
}
@override
void dispose() {
  // Avoid memory leak (`setState` after dispose) and disconnect
  if (isConnected) {
    isDisconnecting = true;
    connection.dispose();
    connection = null;
  }
  super.dispose();
}
@override
Widget build(BuildContext context) {
  final List<Row> list = messages.map((_message) {
    return Row(
      children: <Widget>[
        Container(
          child: Text(
              (text) {
                return text == '/shrug' ? '¯\\_(ツ)_/¯' : text;
              }(_message.text.trim()),
              style: TextStyle(color: Colors.white)),
          padding: EdgeInsets.all(12.0),
          margin: EdgeInsets.only(bottom: 8.0, left: 8.0, right: 8.0),
          width: 222.0,
          decoration: BoxDecoration(
              color:
                  _message.whom == clientID ? Colors.blueAccent : Colors.grey,
              borderRadius: BorderRadius.circular(7.0)),
        ),
      ],
      mainAxisAlignment: _message.whom == clientID
          ? MainAxisAlignment.end
          : MainAxisAlignment.start,
    );
  }).toList();
  return Scaffold(
    appBar: AppBar(
        title: (isConnecting
            ? Text('Connecting chat to ' + widget.server.name + '...')
            : isConnected
                ? Text('Live chat with ' + widget.server.name)
                : Text('Chat log with ' + widget.server.name))),
    body: SafeArea(
      child: Column(
        children: <Widget>[
          Container(
            padding: const EdgeInsets.all(5),
            width: double.infinity,
            child: FittedBox(
              child: Row(
                children: [
                  TextButton(
                    onPressed: isConnected ? () => _sendMessage('1') : null,
                    child: ClipOval(child: Text('Start')),
                  ),
                  TextButton(
                    onPressed: isConnected ? () => _sendMessage('0') : null,
                    child: ClipOval(child: Text('Calibrate')),
                  ),
                ],
              ),
            ),
          ),
          Flexible(
            child: ListView(
                padding: const EdgeInsets.all(12.0),
                controller: listScrollController,
                children: list),
          ),
          Row(
            children: <Widget>[
              Flexible(
                child: Container(
                  margin: const EdgeInsets.only(left: 16.0),
                  child: TextField(
                    style: const TextStyle(fontSize: 15.0),
                    controller: textEditingController,
                    decoration: InputDecoration.collapsed(
                      hintText: isConnecting
                          ? 'Wait until connected...'
                          : isConnected
                              ? 'Type your message...'
                              : 'Chat got disconnected',
                      hintStyle: const TextStyle(color: Colors.grey),
                    ),
                    enabled: isConnected,
                  ),
                ),
              ),
              Container(
                margin: const EdgeInsets.all(8.0),
                child: IconButton(
                    icon: const Icon(Icons.send),
                    onPressed: isConnected
                        ? () => _sendMessage(textEditingController.text)
                        : null),
              ),
            ],
          )
        ],
      ),
    ),
  );
}
void _onDataReceived(Uint8List data) {
  // Allocate buffer for parsed data
  int backspacesCounter = 0;
  data.forEach((byte) {
    if (byte == 8 || byte == 127) {
      backspacesCounter++;
    }
  });
  Uint8List buffer = Uint8List(data.length - backspacesCounter);
  int bufferIndex = buffer.length;

  // Apply backspace control character
  backspacesCounter = 0;
  for (int i = data.length - 1; i >= 0; i--) {
    if (data[i] == 8 || data[i] == 127) {
      backspacesCounter++;
    } else {
      if (backspacesCounter > 0) {
        backspacesCounter--;
      } else {
        buffer[--bufferIndex] = data[i];
      }
    }
  }
  // Create message if there is new line character
  String dataString = String.fromCharCodes(buffer);
  int index = buffer.indexOf(13);
  if (~index != 0) {
    setState(() {
      messages.add(
        _Message(
          1,
          backspacesCounter > 0
              ? _messageBuffer.substring(
                  0, _messageBuffer.length - backspacesCounter)
              : _messageBuffer + dataString.substring(0, index),
        ),
      );
      _messageBuffer = dataString.substring(index);
    });
  } else {
    _messageBuffer = (backspacesCounter > 0
        ? _messageBuffer.substring(
            0, _messageBuffer.length - backspacesCounter)
        : _messageBuffer + dataString);
  }
}
void _sendMessage(String text) async {
  text = text.trim();
  textEditingController.clear();

  if (text.length > 0) {
    try {
      connection.output.add(utf8.encode(text + "\r\n"));
      await connection.output.allSent;
      setState(() {
        if(text == "1")
        {
          messages.add(_Message(clientID, "start command"));
        }
        else if(text == "0")
        {
          messages.add(_Message(clientID, "calibration command"));
        }
        else
        {
          messages.add(_Message(clientID, text));
        }
      });
      Future.delayed(Duration(milliseconds: 333)).then((_) {
        listScrollController.animateTo(
            listScrollController.position.maxScrollExtent,
            duration: Duration(milliseconds: 333),
            curve: Curves.easeOut);
      });
    } catch (e) {
      // Ignore error, but notify state
      setState(() {});
    }
  }
}
}
```

- **<u>For further information, </u>our GitHub repository link: https://github.com/ahmedosama07/Autonomous-Car**