# ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

*Харківський національний університет радіоелектроніки*
*Кафедра ЕОМ*

*Кваліфікаційна робота*

*Перший (бакалаврський) рівень*

# МОДЕЛЬ АВТОМОБІЛЯ НА БАЗІ МІКРОКОНТРОЛЕРА СЕРІЇ ESP З ДИСТАНЦІЙНОЮ СИСТЕМОЮ КЕРУВАННЯ

Автор:
Антон Бугрименко,
студ. гр. КІУКІ-21-1

Керівник
Роман Ярошевич
ст. викл. каф. ЕОМ

## Мета і задачі роботи

Мета: Створення функціональної моделі автомобіля на базі мікроконтролера **ESP** з можливістю дистанційного керування через мобільний застосунок.

Завдання:

Аналіз предметної області та вибір компонентів.

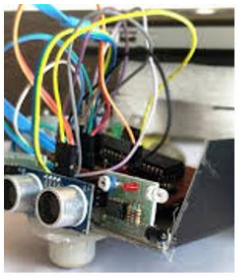
Проєктування та збирання корпусу моделі.

Розробка апаратної частини.

Розробка програмного забезпечення для мікроконтролера (прошивка).

Розробка мобільного застосунку для керування.

Інтеграція та тестування системи.




## Актуальність теми

- Розвиток Інтернету речей (IoT) та робототехніки.
- Потреба в бездротовому керуванні пристроями.
- Навчальний аспект: демонстрація роботи з мікроконтролерами, бездротовими технологіями.
- Практичне застосування: основи для більш складних систем, прототипування.
- Потенціал для розробки наземних дронів для розвідувальних, логістичних чи інших завдань, особливо в сучасних умовах.

## Огляд існуючих рішень та технологій

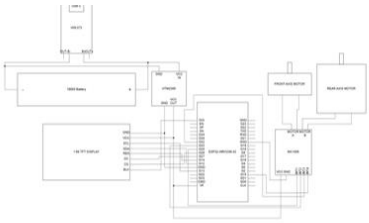
- Традиційні RF-керовані моделі.
- Сучасні рішення на базі мікроконтролерів (Arduino, ESP8266/ESP32).
- Використання Wi-Fi та Bluetooth для керування.

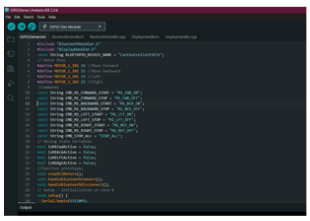



## Розробка апаратної частини та корпусу

- Для розробки корпусу використовувався конструктор Lego (Lego Technic) через гнучкість, можливість модифікації, та доступність.
- Ключові елементи: задня ведуча вісь з редуктором, передня поворотна вісь.
- Апаратна частина складається з мікроконтролера, акумулятору, модулю заряду, підвищуючого перетворювача, драйвера двигунів та 1.69" IPS-дисплею.







## Розробка програмної частини

- Прошивка мікроконтролера у Arduino IDE.
- Ключові функції прошивки:
- Ініціалізація Bluetooth та очікування з'єднання.
- Обробка команд, отриманих по Bluetooth.
- Керування двигунами через драйвер Mx1508.
- Обробка подій підключення/відключення (callbacks).
- Керування виводом інформації на дисплей

- Мобільний застосунок (Android). IDE: - Android Studio, Мова програмування - Kotlin.
- Основні функції застосунку:
- Пошук та підключення до Bluetooth-пристрою (ESP32).
- Інтерфейс користувача для надсилання команд керування (кнопки вперед, назад, вліво, вправо, стоп).
- Відображення статусу з'єднання.

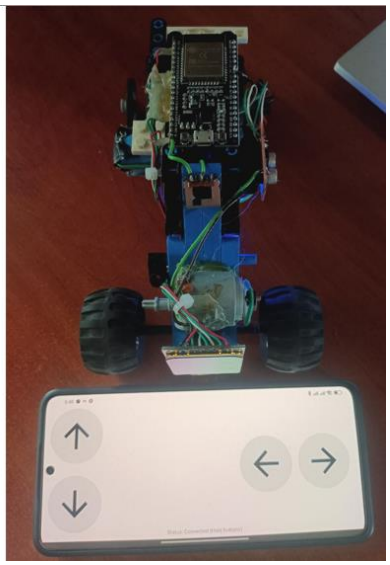
## ДЕМОНСТРАЦІЯ РОБОТИ ПРОЕКТУ

## Результати та тестування

- Тестування з'єднання та виконання основних функцій
- Інші тести:
- Визначення максимальної швидкості: 1.25 м/с (4.5 км/год).
- Тестування надійності Bluetooth-з'єднання: Стабільний зв'язок до 50 метрів. візуальне керування ускладнюється після 10 м.
- Визначення максимального часу автономної роботи: Час роботи – 30-45 хвилин (залежно від активності використання).

## ВИСНОВКИ

- Мета роботи досягнута: створено функціональну керовану модель.
- Успішно інтегровано апаратні та програмні компоненти.
- Отримано практичний досвід у проєктуванні вбудованих систем, роботі з мікроконтролерами ESP32, Bluetooth, розробці мобільних застосунків.



## Перспективи розвитку

- Додавання сенсорів (датчики відстані, лінії).
- Реалізація автономного руху.
- Інтеграція камери (FPV).
- Покращення підвіски та міцності корпуса.
- Покращення інтерфейсу мобільного застосунка, відображення додаткових корисних даних.

# ДОДАТОК Б

## Програмний код

### Б.1 Прошивка мікроконтролеру

Лістинг Б.1.1 – Файл BluetoothHandler.h

```
#ifndef BLUETOOTH_HANDLER_H
#define BLUETOOTH_HANDLER_H
#include <Arduino.h>
typedef void (*BluetoothEventCallback)();
void setupBluetooth(const String& deviceName);
bool loopBluetooth(String &receivedCommand);
void registerBluetoothCallbacks(BluetoothEventCallback
onConnectCallback, BluetoothEventCallback onDisconnectCallback);
#endif
```

Лістинг Б.1.2 – Файл BluetoothHandler.cpp

```
#include "BluetoothHandler.h"
#include <BluetoothSerial.h>
// Bluetooth object
BluetoothSerial SerialBT;
// Type definition for Bluetooth event callbacks
static BluetoothEventCallback connectCallback = nullptr;
static BluetoothEventCallback disconnectCallback = nullptr;
// Internal callback function registered with BluetoothSerial
void btInternalCallback(esp_spp_cb_event_t event,
esp_spp_cb_param_t *param) {
  if (event == ESP_SPP_SRV_OPEN_EVT) { //Connected
    Serial.println("BT_Handler: Client Connected");
    if (connectCallback) {
      connectCallback(); // Call the callback from the main
sketch
    }
  } else if (event == ESP_SPP_CLOSE_EVT) { // Disconnected
    Serial.println("BT_Handler: Client Disconnected");
    if (disconnectCallback) {
      disconnectCallback(); // Call the callback from the main
sketch
    }
  }
}
// Function to setup Bluetooth
```

```
void setupBluetooth(const String& deviceName) {
  if (!SerialBT.begin(deviceName)) {
    Serial.println("!!! BT_Handler: Bluetooth Serial Init Failed
!!!");
    while (true);
  } else {
    Serial.print("BT_Handler: Server started. Device name: '");
    Serial.print(deviceName);
    Serial.println("'");
  }
  SerialBT.register_callback(btInternalCallback); // Register
internal callback
}
// Function to register user-defined callbacks for Bluetooth
events
void registerBluetoothCallbacks(BluetoothEventCallback
onConnectCallback, BluetoothEventCallback onDisconnectCallback)
{
    connectCallback = onConnectCallback;
    disconnectCallback = onDisconnectCallback;
}
//Bluetooth loop function
bool loopBluetooth(String &receivedCommand) {
  static String readBuffer = ""; // Buffer to accumulate
characters
  bool commandComplete = false;
  while (SerialBT.available()) {
    char incomingChar = SerialBT.read();
    if (incomingChar != '\n') {//Command is not complete
      if (readBuffer.length() < 50) {
        readBuffer += incomingChar;
      } else {
        Serial.println("!!! BT_Handler: Receive buffer overflow
!!!");
        readBuffer = "";
      }
    } else { //Command is complete
      receivedCommand = readBuffer;
      readBuffer = "";
      commandComplete = true;
      break;
    }
  }
  return commandComplete; // Return true if successfully read
}
```

Лістинг Б.1.3 – Вміст файлу DisplayHandler.h

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

```
// Display pin definitions
#define TFT_SCLK 13
#define TFT_MOSI 12
#define TFT_RST  14
#define TFT_DC   27
#define TFT_CS   26
#define TFT_WIDTH  240
#define TFT_HEIGHT 280
// Display params
#define ARROW_COLOR ST77XX_BLUE
#define BACKGROUND_COLOR ST77XX_BLACK
#define BLINK_INTERVAL_MS 500
// Constants for arrow drawing actions
#define ACTION_DRAW 1
#define ACTION_ERASE 0
// Extern vars to share with other files
extern volatile char currentDirectionToDisplay;
extern volatile bool displayNeedsUpdate;
extern volatile bool isClientConnected;
extern volatile bool connectionStatusNeedsUpdate;
// Function Prototypes
void setupDisplay();
void startDisplayTask();
void requestDisplayUpdate(char direction);
#endif
```

Лістинг Б.1.4 – Вміст файлу DisplayHandler.cpp

```
#include "DisplayHandler.h"
#include <Adafruit_GFX.h>
#include <Adafruit_ST7789.h>
#include <SPI.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/adc.h"
volatile char currentDirectionToDisplay = ' ';
volatile bool displayNeedsUpdate = true;
volatile bool isClientConnected = false;
volatile bool connectionStatusNeedsUpdate = true;
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_MOSI,
TFT_SCLK, TFT_RST);
TaskHandle_t displayTaskHandle = NULL;
static void displayTaskCode(void *parameter);
static void drawOrEraseArrow(char direction, int action);
static void drawOrEraseArrowHead(char direction, int action);
static void updateConnectionStatusDisplay(int drawAction);
static void displayInitialVoltage(float voltage);
// Setup display function
void setupDisplay() {
  Serial.println("Display_Handler: Initializing Display...");
  tft.init(TFT_WIDTH, TFT_HEIGHT, SPI_MODE2);
```

```
  tft.fillScreen(BACKGROUND_COLOR);
  Serial.println("Display_Handler: Display Initialized.");
}
//xtask
void startDisplayTask() {
  xTaskCreatePinnedToCore(
      displayTaskCode, "DisplayTask", 4096, NULL, 1,
&displayTaskHandle, 1);
  Serial.println("Display_Handler: Display Task created and
pinned to Core 1.");
}
// Request display update
void requestDisplayUpdate(char direction) {
    if (currentDirectionToDisplay != direction) {
        currentDirectionToDisplay = direction;
        displayNeedsUpdate = true;
    }
}
// Draw or erase arrow function
static void drawOrEraseArrow(char direction, int action) {
  if (direction == ' ') return;
  int draw_color = (action == ACTION_DRAW) ? ARROW_COLOR :
BACKGROUND_COLOR;
  if (direction == 'f') tft.setRotation(1);
  else if (direction == 'b') tft.setRotation(3);
  else if (direction == 'l') tft.setRotation(0);
  else if (direction == 'r') tft.setRotation(2);
  int currentWidth = tft.width(); int currentHeight =
tft.height();
  int screen_center_x = currentWidth / 2; int screen_center_y =
currentHeight / 2;
  int arrow_shaft_length = 120; int arrow_shaft_width = 30; int
arrow_head_size = 60;
  int shaft_start_x = screen_center_x - (arrow_shaft_length / 2)
- (arrow_head_size / 2);
  int shaft_y = screen_center_y - arrow_shaft_width / 2;
  int shaft_end_x = shaft_start_x + arrow_shaft_length;
  int tip_x = shaft_end_x + arrow_head_size; int tip_y =
screen_center_y;
  int base_y1 = screen_center_y - arrow_head_size / 2; int
base_y2 = screen_center_y + arrow_head_size / 2;
  tft.fillRect(shaft_start_x, shaft_y, arrow_shaft_length,
arrow_shaft_width, draw_color);
  tft.fillTriangle(tip_x, tip_y, shaft_end_x, base_y1,
shaft_end_x, base_y2, draw_color);
}
// Draw or erase arrow head function (because it needs to blink
separately)
static void drawOrEraseArrowHead(char direction, int action) {
   if (direction == ' ') return;
  int draw_color = (action == ACTION_DRAW) ? ARROW_COLOR :
BACKGROUND_COLOR;
  if (direction == 'f') tft.setRotation(1);
```

```
  else if (direction == 'b') tft.setRotation(3);
  else if (direction == 'l') tft.setRotation(0);
  else if (direction == 'r') tft.setRotation(2);
  int currentWidth = tft.width(); int currentHeight =
tft.height();
  int screen_center_x = currentWidth / 2; int screen_center_y =
currentHeight / 2;
  int arrow_shaft_length = 120; int arrow_head_size = 60;
  int shaft_start_x = screen_center_x - (arrow_shaft_length / 2)
- (arrow_head_size / 2);
  int shaft_end_x = shaft_start_x + arrow_shaft_length;
  int tip_x = shaft_end_x + arrow_head_size; int tip_y =
screen_center_y;
  int base_y1 = screen_center_y - arrow_head_size / 2; int
base_y2 = screen_center_y + arrow_head_size / 2;
  tft.fillTriangle(tip_x, tip_y, shaft_end_x, base_y1,
shaft_end_x, base_y2, draw_color);
}
// Draw or erase connection status text
static void updateConnectionStatusDisplay(int drawAction) {
    tft.setRotation(2);
    int16_t textSize = 2;
    const char* statusText = "READY TO CONNECT";
    int16_t targetY = 15;
    uint16_t draw_color;
    if (drawAction == ACTION_DRAW) {
        draw_color = ST77XX_YELLOW;
    } else {
        draw_color = BACKGROUND_COLOR;
    }
    int16_t textX_ignore, textY_ignore;
    uint16_t textWidth, textHeight;
    tft.setTextSize(textSize);
    tft.getTextBounds(statusText, 0, 0, &textX_ignore,
&textY_ignore, &textWidth, &textHeight);
    int16_t cursorX = (tft.width() - textWidth) / 2;
    int16_t cursorY = targetY + textHeight;
    tft.setCursor(cursorX, cursorY);
    tft.setTextColor(draw_color, BACKGROUND_COLOR);
    tft.print(statusText);
}
// Display task code
static void displayTaskCode(void *parameter) {
  Serial.println("Display_Handler: Task starting loop (Status +
Blinking).");
  static char lastDirectionDrawn = ' ';
  static bool lastConnectionStatusWasConnected = true;
  static bool blinkStateOn = true;
  static unsigned long lastBlinkTime = 0;
  if (!isClientConnected) {
      updateConnectionStatusDisplay(ACTION_DRAW);
      lastConnectionStatusWasConnected = false;
  } else {
```

```
      updateConnectionStatusDisplay(ACTION_ERASE);
      lastConnectionStatusWasConnected = true;
  }
  connectionStatusNeedsUpdate = false;
  for (;;) {
    bool currentConnectionState = isClientConnected;
    char currentDirection = currentDirectionToDisplay;
    bool directionUpdateRequested = displayNeedsUpdate;
    bool statusUpdateRequested = connectionStatusNeedsUpdate;
    if (statusUpdateRequested || (currentConnectionState !=
lastConnectionStatusWasConnected)) {
        connectionStatusNeedsUpdate = false;
        if (currentConnectionState) {
            if (!lastConnectionStatusWasConnected) {
                updateConnectionStatusDisplay(ACTION_ERASE);
            }
            if (currentDirection != ' ' && currentDirection !=
lastDirectionDrawn) {
                displayNeedsUpdate = true;
            }
        } else {
            updateConnectionStatusDisplay(ACTION_DRAW);
            if (lastDirectionDrawn != ' ') {
                 drawOrEraseArrow(lastDirectionDrawn,
ACTION_ERASE);
                 lastDirectionDrawn = ' ';
            }
        }
        lastConnectionStatusWasConnected =
currentConnectionState;
        Serial.print("Display_Handler: Connection Status Text
updated. Connected: ");
        Serial.println(currentConnectionState);
    }
    // Direction update handling
    if (directionUpdateRequested && currentConnectionState) {
      displayNeedsUpdate = false;
      if (currentDirection != lastDirectionDrawn) {
          Serial.print("Display_Handler: Direction changing from
'");
          Serial.print(lastDirectionDrawn);
          Serial.print("' to '");
          Serial.print(currentDirection);
          Serial.println("'");
          if (lastDirectionDrawn != ' ') {
            drawOrEraseArrow(lastDirectionDrawn, ACTION_ERASE);
          }
          if (currentDirection != ' ') {
            drawOrEraseArrow(currentDirection, ACTION_DRAW);
          }
          lastDirectionDrawn = currentDirection;
          blinkStateOn = true;
          lastBlinkTime = millis();
```

```
      }
    }
    // Blinking
    if (currentConnectionState && lastDirectionDrawn != ' ' &&
(millis() - lastBlinkTime >= BLINK_INTERVAL_MS)) {
        blinkStateOn = !blinkStateOn;
        drawOrEraseArrowHead(lastDirectionDrawn, blinkStateOn ?
ACTION_DRAW : ACTION_ERASE);
        lastBlinkTime = millis();
    }
    vTaskDelay(50 / portTICK_PERIOD_MS);
  }
}
```

Лістинг Б.1.5 – Вміст файлу ESP32Server.ino

```
#include "BluetoothHandler.h"
#include "DisplayHandler.h"
const String BLUETOOTH_DEVICE_NAME = "CarControllerESP32";
// Motor Pins
#define MOTOR_1_IN1 32 //Move Forward
#define MOTOR_1_IN2 15 //Move backward
#define MOTOR_2_IN1 33 //Left
#define MOTOR_2_IN2 25 //Right
//Commands
const String CMD_M1_FORWARD_START = "M1_FWD_ON";
const String CMD_M1_FORWARD_STOP = "M1_FWD_OFF";
const String CMD_M1_BACKWARD_START = "M1_BCK_ON";
const String CMD_M1_BACKWARD_STOP = "M1_BCK_OFF";
const String CMD_M2_LEFT_START = "M2_LFT_ON";
const String CMD_M2_LEFT_STOP = "M2_LFT_OFF";
const String CMD_M2_RIGHT_START = "M2_RGT_ON";
const String CMD_M2_RIGHT_STOP = "M2_RGT_OFF";
const String CMD_STOP_ALL = "STOP_ALL";
// Moving State Variables
bool isM1FwdActive = false;
bool isM1BckActive = false;
bool isM2LftActive = false;
bool isM2RgtActive = false;
//function prototypes
void stopAllMotors();
void handleBluetoothConnect();
void handleBluetoothDisconnect();
// Setup - Initialization on Core 0
void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.println("\n====================================");
  Serial.println("ESP32 Car Controller - Main Setup");
  Serial.println("====================================");
  pinMode(MOTOR_1_IN1, OUTPUT); pinMode(MOTOR_1_IN2, OUTPUT);
```

```
  pinMode(MOTOR_2_IN1, OUTPUT); pinMode(MOTOR_2_IN2, OUTPUT);
  stopAllMotors();
  setupDisplay();
  setupBluetooth(BLUETOOTH_DEVICE_NAME);
  registerBluetoothCallbacks(handleBluetoothConnect,
handleBluetoothDisconnect);
  startDisplayTask();
  Serial.println("Main Setup Complete. Waiting for client
connection...");
}
//Main loop
void loop() {
  String receivedCommand = "";
  bool commandReceived = loopBluetooth(receivedCommand);
  if (commandReceived) {
    Serial.print("Main Loop: Process Command [");
Serial.print(receivedCommand); Serial.println("]");
    if (receivedCommand == CMD_M1_FORWARD_START) {
      digitalWrite(MOTOR_1_IN1, HIGH); digitalWrite(MOTOR_1_IN2,
LOW);
      isM1FwdActive = true; isM1BckActive = false;
    } else if (receivedCommand == CMD_M1_FORWARD_STOP) {
      digitalWrite(MOTOR_1_IN1, LOW); digitalWrite(MOTOR_1_IN2,
LOW);
      isM1FwdActive = false;
    } else if (receivedCommand == CMD_M1_BACKWARD_START) {
      digitalWrite(MOTOR_1_IN1, LOW); digitalWrite(MOTOR_1_IN2,
HIGH);
      isM1BckActive = true; isM1FwdActive = false;
    } else if (receivedCommand == CMD_M1_BACKWARD_STOP) {
      digitalWrite(MOTOR_1_IN1, LOW); digitalWrite(MOTOR_1_IN2,
LOW);
      isM1BckActive = false;
    } else if (receivedCommand == CMD_M2_LEFT_START) {
      digitalWrite(MOTOR_2_IN1, HIGH); digitalWrite(MOTOR_2_IN2,
LOW);
      isM2LftActive = true; isM2RgtActive = false;
    } else if (receivedCommand == CMD_M2_LEFT_STOP) {
      digitalWrite(MOTOR_2_IN1, LOW); digitalWrite(MOTOR_2_IN2,
LOW);
      isM2LftActive = false;
    } else if (receivedCommand == CMD_M2_RIGHT_START) {
      digitalWrite(MOTOR_2_IN1, LOW); digitalWrite(MOTOR_2_IN2,
HIGH);
      isM2RgtActive = true; isM2LftActive = false;
    } else if (receivedCommand == CMD_M2_RIGHT_STOP) {
      digitalWrite(MOTOR_2_IN1, LOW); digitalWrite(MOTOR_2_IN2,
LOW);
      isM2RgtActive = false;
    } else if (receivedCommand == CMD_STOP_ALL) {
      stopAllMotors();
    } else {
      Serial.print("Main Loop: Unknown Command: ");
```

```
Serial.println(receivedCommand);
    }
    char prioritizedDirection = ' ';
    if (isM2LftActive) prioritizedDirection = 'l';
    else if (isM2RgtActive) prioritizedDirection = 'r';
    else if (isM1FwdActive) prioritizedDirection = 'f';
    else if (isM1BckActive) prioritizedDirection = 'b';
    requestDisplayUpdate(prioritizedDirection);
  }
}
// Function to stop all motors
void stopAllMotors() {
  digitalWrite(MOTOR_1_IN1, LOW); digitalWrite(MOTOR_1_IN2,
LOW);
  digitalWrite(MOTOR_2_IN1, LOW); digitalWrite(MOTOR_2_IN2,
LOW);
  isM1FwdActive = false; isM1BckActive = false;
  isM2LftActive = false; isM2RgtActive = false;
  Serial.println("Main Loop: All Motors STOPPED & States
Reset.");
}
// Callback handlers for Bluetooth
void handleBluetoothConnect() {
  Serial.println("Main Loop: Bluetooth Client Connected
(Callback).");
  isClientConnected = true;
  connectionStatusNeedsUpdate = true;
}
void handleBluetoothDisconnect() {
  Serial.println("Main Loop: Bluetooth Client Disconnected
(Callback).");
  isClientConnected = false;
  connectionStatusNeedsUpdate = true;
  stopAllMotors();
  requestDisplayUpdate(' ');
}
```

## Б.2 Файли мобільного застосунка

Лістинг Б.2.1 – Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.antonio.carcontroller">
    <uses-permission android:name="android.permission.BLUETOOTH"
android:maxSdkVersion="30" />
    <uses-permission
```

```
android:name="android.permission.BLUETOOTH_ADMIN"
android:maxSdkVersion="30" />
    <uses-permission
android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission
android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-feature android:name="android.hardware.bluetooth"
android:required="true"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.CarController"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:screenOrientation="landscape">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ControlActivity"
            android:exported="false"
            android:screenOrientation="landscape"/>
    </application>
</manifest>
```

Лістинг Б.2.2 – Файл activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/buttonConnect"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Connect to CarControllerESP32"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/textViewStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Status: Idle"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/buttonConnect"
/>

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:visibility="gone"
        app:layout_constraintBottom_toTopOf="@+id/buttonConnect"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        tools:visibility="visible"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Лістинг Б.2.3 – Файл activity_control.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".ControlActivity">
    <ImageButton
        android:id="@+id/buttonForward"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:background="@drawable/button_background_circle"
        android:contentDescription="Move Forward"
        android:padding="20dp"
        android:scaleType="fitCenter"
        android:src="@drawable/ic_arrow_up"

app:layout_constraintBottom_toTopOf="@+id/buttonBackward"
        app:layout_constraintEnd_toEndOf="@+id/buttonBackward"
```

```
app:layout_constraintStart_toStartOf="@+id/buttonBackward"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_chainStyle="packed"
        tools:ignore="HardcodedText" />
    <ImageButton
        android:id="@+id/buttonBackward"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_marginTop="32dp"
        android:background="@drawable/button_background_circle"
        android:contentDescription="Move Backward"
        android:padding="20dp"
        android:scaleType="fitCenter"
        android:src="@drawable/ic_arrow_down"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/buttonForward"
        tools:ignore="HardcodedText" />
    <ImageButton
        android:id="@+id/buttonLeft"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_marginEnd="28dp"
        android:layout_marginBottom="84dp"
        android:background="@drawable/button_background_circle"
        android:contentDescription="Turn Left"
        android:padding="20dp"
        android:scaleType="fitCenter"
        android:src="@drawable/ic_arrow_left"

app:layout_constraintBottom_toTopOf="@+id/textViewControlStatus"
        app:layout_constraintEnd_toStartOf="@+id/buttonRight"
        app:layout_constraintVertical_chainStyle="packed"
        tools:ignore="HardcodedText" />
    <ImageButton
        android:id="@+id/buttonRight"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_marginTop="133dp"
        android:layout_marginBottom="112dp"
        android:background="@drawable/button_background_circle"
        android:contentDescription="Turn Right"
        android:padding="20dp"
        android:scaleType="fitCenter"
        android:src="@drawable/ic_arrow_right"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="HardcodedText" />
    <TextView
        android:id="@+id/textViewControlStatus"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Status: Connected"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        tools:ignore="HardcodedText" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Лістинг Б.2.4 – Файл BluetoothConnectionManager.kt

```
package com.antonio.carcontroller
import android.annotation.SuppressLint
import android.bluetooth.BluetoothSocket
import android.util.Log
import java.io.IOException
import java.io.OutputStream
object BluetoothConnectionManager {
    private var btSocket: BluetoothSocket? = null
    private var outputStream: OutputStream? = null
    private const val TAG = "BluetoothConnectionMgr"
    fun setSocket(socket: BluetoothSocket) {
        btSocket = socket
        try {
            outputStream = socket.outputStream
            Log.d(TAG, "Output stream acquired.")
        } catch (e: IOException) {
            Log.e(TAG, "Error getting output stream", e)
            outputStream = null
            closeConnection()
        }
    }
    fun isConnected(): Boolean {
        return btSocket?.isConnected == true && outputStream !=
null
    }
    fun sendMessage(message: String): Boolean {
        if (!isConnected()) {
            Log.e(TAG, "Cannot send message, not connected.")
            return false
        }
        val msgBuffer = (message + "\n").toByteArray()
        return try {
            outputStream?.write(msgBuffer)
            outputStream?.flush()
            Log.d(TAG, "Message sent: $message")
            true
        } catch (e: IOException) {
            Log.e(TAG, "Error sending message: $message", e)
            closeConnection()
            false
```

```
        }
    }
    @SuppressLint("MissingPermission")
    fun closeConnection() {
        try {
            outputStream?.close()
            btSocket?.close()
            Log.d(TAG, "Bluetooth connection closed.")
        } catch (e: IOException) {
            Log.e(TAG, "Error closing connection", e)
        } finally {
            outputStream = null
            btSocket = null
        }
    }
}
```

Лістинг Б.2.5 – Файл MainActivity.kt

```
package com.antonio.carcontroller
import android.Manifest
import android.annotation.SuppressLint
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothManager
import android.bluetooth.BluetoothSocket
import android.content.Intent
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Button
import android.widget.ProgressBar
import android.widget.TextView
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import kotlinx.coroutines.*
import java.io.IOException
import java.util.*
class MainActivity : AppCompatActivity() {
    private lateinit var bluetoothManager: BluetoothManager
    private var bluetoothAdapter: BluetoothAdapter? = null
    private val targetDeviceName = "CarControllerESP32"
    private val esp32DeviceUUID: UUID =
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")
    private lateinit var connectButton: Button
    private lateinit var statusTextView: TextView
```

```
    private lateinit var progressBar: ProgressBar
    private val TAG = "MainActivityBluetooth"
    private var connectionJob: Job? = null
    private val requestMultiplePermissions =

registerForActivityResult(ActivityResultContracts.RequestMultipl
ePermissions()) { permissions ->
            var allGranted = true
            permissions.entries.forEach {
                if (!it.value) {
                    allGranted = false
                    Log.w(TAG, "Permission ${it.key} not
granted.")
                }
            }
            if (allGranted) {
                Log.d(TAG, "All required Bluetooth permissions
granted.")
                checkBluetoothStateAndConnect()
            } else {
                Log.e(TAG, "One or more Bluetooth permissions
were denied.")
                updateStatus("Bluetooth permissions denied.
Cannot function.", isError = true)
                Toast.makeText(this, "Bluetooth permissions are
required", Toast.LENGTH_LONG).show()
            }
        }
    private val enableBluetoothLauncher =

registerForActivityResult(ActivityResultContracts.StartActivityF
orResult()) { result ->
            if (result.resultCode == RESULT_OK) {
                Log.d(TAG, "Bluetooth enabled by user.")
                startConnectionProcess()
            } else {
                Log.e(TAG, "Bluetooth enabling was cancelled or
failed.")
                updateStatus("Bluetooth not enabled. Cannot
connect.", isError = true)
                Toast.makeText(this, "Bluetooth must be enabled
to connect", Toast.LENGTH_SHORT).show()
            }
        }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        connectButton = findViewById(R.id.buttonConnect)
        statusTextView = findViewById(R.id.textViewStatus)
        progressBar = findViewById(R.id.progressBar)
        bluetoothManager =
getSystemService(BluetoothManager::class.java)
        bluetoothAdapter = bluetoothManager.adapter
```

```
        if (bluetoothAdapter == null) {
            updateStatus("Device does not support Bluetooth",
isError = true)
            connectButton.isEnabled = false
            return
        }
        connectButton.setOnClickListener {
            checkPermissionsAndConnect()
        }
        updateStatus("Ready to connect")
    }
    override fun onDestroy() {
        super.onDestroy()
        connectionJob?.cancel()
        if (BluetoothConnectionManager.isConnected()) {
            BluetoothConnectionManager.closeConnection()
        }
    }
    private fun checkPermissionsAndConnect() {
        Log.d(TAG, "Checking permissions...")
        val requiredPermissions = if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.S) {
            arrayOf(
                Manifest.permission.BLUETOOTH_SCAN,
                Manifest.permission.BLUETOOTH_CONNECT
            )
        } else {
            arrayOf(
                Manifest.permission.BLUETOOTH,
                Manifest.permission.BLUETOOTH_ADMIN
            )
        }
        val missingPermissions = requiredPermissions.filter {
            ContextCompat.checkSelfPermission(this, it) !=
PackageManager.PERMISSION_GRANTED
        }
        if (missingPermissions.isNotEmpty()) {
            Log.d(TAG, "Requesting missing permissions:
${missingPermissions.joinToString()}")

requestMultiplePermissions.launch(missingPermissions.toTypedArra
y())
        } else {
            Log.d(TAG, "All permissions already granted.")
            checkBluetoothStateAndConnect()
        }
    }
    @SuppressLint("MissingPermission")
    private fun checkBluetoothStateAndConnect() {
        if (!bluetoothAdapter!!.isEnabled) {
            Log.d(TAG, "Bluetooth is disabled. Requesting user
to enable.")
            updateStatus("Bluetooth disabled. Please enable.")
```

```
            val enableBtIntent =
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
            enableBluetoothLauncher.launch(enableBtIntent)
        } else {
            Log.d(TAG, "Bluetooth is enabled.")
            startConnectionProcess()
        }
    }
    @SuppressLint("MissingPermission")
    private fun startConnectionProcess() {
        Log.d(TAG, "Starting connection process...")
        updateStatus("Searching for $targetDeviceName...",
showProgress = true)
        connectButton.isEnabled = false
        connectionJob?.cancel()
        connectionJob = CoroutineScope(Dispatchers.IO).launch {
            try {
                val pairedDevices: Set<BluetoothDevice>? =
bluetoothAdapter?.bondedDevices
                val targetDevice = pairedDevices?.find { it.name
== targetDeviceName }
                if (targetDevice == null) {
                    Log.w(TAG, "$targetDeviceName not found in
paired devices.")
                    withContext(Dispatchers.Main) {
                        updateStatus("Device '$targetDeviceName'
not paired. Please pair it first.", isError = true)
                    }
                    return@launch
                }
                Log.d(TAG, "Found device: ${targetDevice.name}
[${targetDevice.address}]")
                withContext(Dispatchers.Main) {
                    updateStatus("Found $targetDeviceName.
Connecting...")
                }
                var socket: BluetoothSocket? = null
                try {
                    socket =
targetDevice.createRfcommSocketToServiceRecord(esp32DeviceUUID)
                    Log.d(TAG, "Bluetooth socket created.")
                    socket.connect()
                    Log.d(TAG, "Bluetooth connection
established.")
                    BluetoothConnectionManager.setSocket(socket)
                    withContext(Dispatchers.Main) {
                        updateStatus("Connected!")
                        Toast.makeText(this@MainActivity,
"Connected to $targetDeviceName", Toast.LENGTH_SHORT).show()
                        val intent = Intent(this@MainActivity,
ControlActivity::class.java)
                        startActivity(intent)
                    }
```

```
                } catch (e: IOException) {
                    Log.e(TAG, "Socket connection failed", e)
                    try {
                        socket?.close()
                    } catch (closeException: IOException) {
                        Log.e(TAG, "Could not close the client
socket", closeException)
                    }
                    withContext(Dispatchers.Main) {
                        updateStatus("Connection failed:
${e.message}", isError = true)
                    }
                } catch (e: SecurityException) {
                    Log.e(TAG, "Socket connection failed due to
security exception (permissions?)", e)
                    withContext(Dispatchers.Main) {
                        updateStatus("Connection failed:
Permissions missing?", isError = true)
                    }
                }
            } catch (e: SecurityException) {
                Log.e(TAG, "Security exception during
bonding/connection check (permissions?)", e)
                withContext(Dispatchers.Main) {
                    updateStatus("Connection failed: Permissions
missing?", isError = true)
                }
            } finally {
                withContext(Dispatchers.Main) {
                    if
(!BluetoothConnectionManager.isConnected()) {
                        progressBar.visibility = View.GONE
                        connectButton.isEnabled = true
                    } else {
                        progressBar.visibility = View.GONE
                    }
                }
            }
        }
    }
    private fun updateStatus(message: String, isError: Boolean =
false, showProgress: Boolean = false) {
        runOnUiThread {
            statusTextView.text = "Status: $message"
            statusTextView.setTextColor(
                ContextCompat.getColor(
                    this,
                    if (isError) android.R.color.holo_red_dark
else android.R.color.black
                )
            )
            progressBar.visibility = if (showProgress)
View.VISIBLE else View.GONE
```

```
            if (!showProgress &&
!BluetoothConnectionManager.isConnected()) {
                connectButton.isEnabled = true
            } else if (showProgress) {
                connectButton.isEnabled = false
            }
        }
    }
}
```

Лістинг Б.2.6 – Файл ControlActivity.kt

```
package com.antonio.carcontroller
import android.annotation.SuppressLint
import android.os.Bundle
import android.util.Log
import android.view.MotionEvent
import android.view.View
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import android.widget.ImageButton
import androidx.appcompat.app.AppCompatActivity
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
class ControlActivity : AppCompatActivity() {
    private val CMD_M1_FORWARD_START = "M1_FWD_ON"
    private val CMD_M1_FORWARD_STOP = "M1_FWD_OFF"
    private val CMD_M1_BACKWARD_START = "M1_BCK_ON"
    private val CMD_M1_BACKWARD_STOP = "M1_BCK_OFF"
    private val CMD_M2_LEFT_START = "M2_LFT_ON"
    private val CMD_M2_LEFT_STOP = "M2_LFT_OFF"
    private val CMD_M2_RIGHT_START = "M2_RGT_ON"
    private val CMD_M2_RIGHT_STOP = "M2_RGT_OFF"
    private val CMD_STOP_ALL = "STOP_ALL"
    private lateinit var buttonForward: ImageButton
    private lateinit var buttonBackward: ImageButton
    private lateinit var buttonLeft: ImageButton
    private lateinit var buttonRight: ImageButton
    private lateinit var statusTextView: TextView
    private val TAG = "ControlActivity"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_control)
        buttonForward = findViewById(R.id.buttonForward)
        buttonBackward = findViewById(R.id.buttonBackward)
        buttonLeft = findViewById(R.id.buttonLeft)
        buttonRight = findViewById(R.id.buttonRight)
        statusTextView =
```

```
findViewById(R.id.textViewControlStatus)
        if (!BluetoothConnectionManager.isConnected()) {
            Log.e(TAG, "ControlActivity started but not
connected!")
            Toast.makeText(this, "Connection lost!",
Toast.LENGTH_SHORT).show()
            finish()
            return
        }
        Log.d(TAG, "ControlActivity created, connection valid.")
        statusTextView.text = "Status: Connected (Hold buttons)"
        setupTouchListener(buttonForward, CMD_M1_FORWARD_START,
CMD_M1_FORWARD_STOP)
        setupTouchListener(buttonBackward,
CMD_M1_BACKWARD_START, CMD_M1_BACKWARD_STOP)
        setupTouchListener(buttonLeft, CMD_M2_LEFT_START,
CMD_M2_LEFT_STOP)
        setupTouchListener(buttonRight, CMD_M2_RIGHT_START,
CMD_M2_RIGHT_STOP)
    }
    @SuppressLint("ClickableViewAccessibility")
    private fun setupTouchListener(button: View, commandStart:
String, commandStop: String) {
        button.setOnTouchListener { _, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    sendCommand(commandStart)
                    true
                }
                MotionEvent.ACTION_UP -> {
                    sendCommand(commandStop)
                    true
                }
                MotionEvent.ACTION_CANCEL -> {
                    sendCommand(commandStop)
                    true
                }
                else -> false
            }
        }
    }
    private fun sendCommand(command: String) {
        if (!BluetoothConnectionManager.isConnected()) {
            if (command != CMD_M1_FORWARD_STOP && command !=
CMD_M1_BACKWARD_STOP &&
                command != CMD_M2_LEFT_STOP && command !=
CMD_M2_RIGHT_STOP && command != CMD_STOP_ALL) {
                Log.e(TAG, "Cannot send command '$command', not
connected.")
                Toast.makeText(this, "Connection lost!",
Toast.LENGTH_SHORT).show()
                finish()
            } else {
```

```
                Log.w(TAG, "Attempting to send STOP command
($command) despite potential connection issue.")
                CoroutineScope(Dispatchers.IO).launch {
BluetoothConnectionManager.sendMessage(command) }
            }
            return
        }
        CoroutineScope(Dispatchers.IO).launch {
            Log.d(TAG,"Sending command: $command")
            val success =
BluetoothConnectionManager.sendMessage(command)
            if (!success && command != CMD_STOP_ALL) {
                withContext(Dispatchers.Main) {
                    Log.e(TAG,"Failed to send command
'$command'. Connection likely lost.")
                    Toast.makeText(this@ControlActivity, "Send
failed. Connection lost?", Toast.LENGTH_LONG).show()
                    finish()
                }
            } else if (!success) {
                Log.e(TAG,"Failed to send command '$command'.")
            }
        }
    }
    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "ControlActivity destroyed.")
        if (BluetoothConnectionManager.isConnected()) {
            Log.d(TAG, "Sending STOP_ALL before closing
connection.")
            sendCommand(CMD_STOP_ALL)
            try { Thread.sleep(100) } catch (e:
InterruptedException) {}
            BluetoothConnectionManager.closeConnection()
            Log.d(TAG, "Bluetooth connection closed on
ControlActivity destroy.")
        } else {
            BluetoothConnectionManager.closeConnection()
        }
    }
}
```