WALL-E ROBOT

Project Description: Wall-E Robot Prototype

Overview: This project is a detailed exploration into robotics and IoT, focusing on creating a fully functional prototype of the beloved character Wall-E. The robot showcases various advanced features including speech capabilities, head and hand movements, and mobile navigation. The control system employs an ESP8266 microcontroller for server functionalities and an Android mobile device for client-side operations, enabling seamless wireless communication.

Features:

1. **Speech Capability:**
    - The Wall-E robot can produce speech, adding an interactive element that enhances user engagement. This is achieved through text-to-speech (TTS) integration, allowing the robot to respond audibly to commands or queries.
2. **Head Movements:**
    - The robot's head can move along two axes, providing lifelike motion. This is controlled via servos connected to the ESP8266, enabling precise and smooth movements.
3. **Hand Movements:**
    - Wall-E's hands can move up and down, simulating gestures that further enhance its interactive capabilities. This is also managed by servos, ensuring accurate positioning and fluid motion.
4. **Four-Wheel Navigation:**
    - The robot is equipped with four wheels for movement, allowing it to navigate its environment efficiently. The wheels are controlled by an L298N motor driver, which is interfaced with the ESP8266 for wireless command execution.
5. **Wireless Communication:**
    - The ESP8266 microcontroller acts as a server, handling all incoming commands and controlling the hardware components. An Android mobile device serves as the client, providing a user-friendly interface to send commands via a custom app.

Technical Details:

- **Microcontroller: ESP8266**
  - Manages all hardware controls including servos and motors.
  - Handles Wi-Fi communication and serves as the central processing unit for the robot.
- **Motor Control:**
  - L298N Motor Driver: Used to control the four wheels, allowing for forward, backward, left, and right movements.
  - Servo Motors: Control head and hand movements with precision.
- **Communication Protocol:**
  - Wi-Fi: Enables seamless communication between the ESP8266 server and the Android mobile client.
- **Android App:**
  - A custom-built Android application interfaces with the ESP8266, sending commands to control Wall-E's movements and actions.
  - The app includes buttons for various actions such as moving forward, backward, turning, and controlling head and hand movements.

Operational Workflow:

1. **Initialization:**
    - Upon powering on, the ESP8266 initializes and sets up a Wi-Fi access point.
    - The Android app connects to this access point to start communication.
2. **Control Commands:**
    - The user interacts with the Android app, pressing buttons to send commands.
    - Commands include movement directions (forward, backward, left, right) and specific actions for head and hand movements.
3. **Command Execution:**
    - The ESP8266 receives the commands and processes them, activating the corresponding motors or servos.
    - Real-time feedback ensures smooth operation and immediate response to user inputs.
4. **Safety Protocol:**
    - A safety feature is implemented to turn off the motors if the Wi-Fi connection is lost, preventing unintended movements.

Ardunio C++ CODE

```
#define IN_1  16
#define IN_2  5
#define IN_3  0
#define IN_4  4
#define SP3 14
#define SP4 12
#define SP1 13
#define SP2 15
#define LED 2

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <Servo.h>

Servo myservo1;
Servo myservo2;
Servo myservo3;
Servo myservo4;

String command;
int angle1 = 90;
int angle2 = 90;
int angle3 = 100;
int angle4 = 100;
const char* ssid = "Wall-E";
ESP8266WebServer server(80);

void setup() {
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);
  pinMode(LED, OUTPUT);
  myservo1.attach(SP1);
  myservo1.write(angle1);
  myservo2.attach(SP2);
  myservo2.write(angle2);
  myservo3.attach(SP3);
  myservo3.write(angle3);
  myservo4.attach(SP4);
  myservo4.write(angle4);
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid);
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
```

```
  Serial.println(myIP);
  server.on("/", HTTP_handleRoot);
  server.onNotFound(HTTP_handleRoot);
  server.begin();
}

void moveservo(Servo &servo, int targetAngle) {
  int currentAngle = servo.read();
  if (currentAngle < targetAngle) {
    for (int angle = currentAngle; angle <= targetAngle; angle += 10) {
      servo.write(angle);
    }
    servo.write(targetAngle);
  } else {
    for (int angle = currentAngle; angle >= targetAngle; angle -= 10) {
      servo.write(angle);
    }
    servo.write(targetAngle);
  }
}

void goAhead() {
  digitalWrite(IN_1, LOW);
  digitalWrite(IN_2, HIGH);
  digitalWrite(IN_3, LOW);
  digitalWrite(IN_4, HIGH);
  Serial.print("forward");
}

void goBack() {
  digitalWrite(IN_1, HIGH);
  digitalWrite(IN_2, LOW);
  digitalWrite(IN_3, HIGH);
  digitalWrite(IN_4, LOW);
  Serial.print("backward");
}

void goRight() {
  digitalWrite(IN_1, HIGH);
  digitalWrite(IN_2, LOW);
  digitalWrite(IN_3, LOW);
  digitalWrite(IN_4, HIGH);
  Serial.print("right");
}

void goLeft() {
  digitalWrite(IN_1, LOW);
  digitalWrite(IN_2, HIGH);
  digitalWrite(IN_3, HIGH);
  digitalWrite(IN_4, LOW);
```

```
  Serial.print("left");
}

void stopRobot() {
  digitalWrite(IN_1, LOW);
  digitalWrite(IN_2, LOW);
  digitalWrite(IN_3, LOW);
  digitalWrite(IN_4, LOW);
}

void righthandup() {
  moveservo(myservo1, 180);
}

void righthandmid() {
  myservo1.write(90);
}

void righthanddown() {
  moveservo(myservo1, 30);
}

void lefthandup() {
  moveservo(myservo2, 0);
}

void lefthandmid() {
  moveservo(myservo2, 90);
}

void lefthanddown() {
  moveservo(myservo2, 150);
}

void headup() {
  moveservo(myservo3, 0);
}

void headVmid() {
  moveservo(myservo3, 100);
}

void headdown() {
  moveservo(myservo3, 180);
}

void headleft() {
  moveservo(myservo4, 180);
}
```

```
void headHmid() {
  moveservo(myservo4, 100);
}

void headright() {
  moveservo(myservo4, 0);
}

void sayHi() {
  stopRobot();
  headHmid();
  headVmid();
  delay(200);
  righthandup();
}

void loop() {
  server.handleClient();
  command = server.arg("State");

  if (command == "F") goAhead();
  else if (command == "B") goBack();
  else if (command == "L") goLeft();
  else if (command == "R") goRight();
  else if (command == "RHU") righthandup();
  else if (command == "RHM") righthandmid();
  else if (command == "RHD") righthanddown();
  else if (command == "LHU") lefthandup();
  else if (command == "LHM") lefthandmid();
  else if (command == "LHD") lefthanddown();
  else if (command == "HL") headleft();
  else if (command == "HVM") headVmid();
  else if (command == "HR") headright();
  else if (command == "HU") headup();
  else if (command == "HHM") headHmid();
  else if (command == "HD") headdown();
  else if (command == "GLOW") gloweyes();
  else if (command == "CLOSE") closeeyes();
  else if (command == "SAYHI") sayHi();
  else if (command == "S") stopRobot();
}

void HTTP_handleRoot() {
  if (server.hasArg("State")) {
    Serial.println(server.arg("State"));
  }
  server.send(200, "text/html", "");
  delay(1);
}
```

KOTLIN APP CODE

```kotlin
package com.example.controller

import android.os.Bundle
import android.os.Vibrator
import android.content.Context
import android.media.MediaPlayer
import android.os.VibrationEffect
import android.speech.tts.TextToSpeech
import android.view.MenuInflater
import android.view.MenuItem
import android.view.MotionEvent
import android.view.View
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import okhttp3.*
import java.io.IOException
import java.util.*

class MainActivity : AppCompatActivity(), TextToSpeech.OnInitListener {

    private val ipAddress = "192.168.4.1"  // Default IP for ESP8266 in AP mode
    private val client = OkHttpClient()
    private var speed = 50  // Initial speed
    private lateinit var vibrator: Vibrator
    private lateinit var tts: TextToSpeech
    private lateinit var textInput: EditText
    private lateinit var speakButton: Button
    private lateinit var mediaPlayer: MediaPlayer
    private var isGlowing = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        vibrator = getSystemService(Context.VIBRATOR_SERVICE) as Vibrator
        tts = TextToSpeech(this, this)
        val ipAddressTextView = findViewById<TextView>(R.id.ipAddressTextView)
        ipAddressTextView.text = "IP Address: $ipAddress"
        val showPopupButton = findViewById<ImageButton>(R.id.emoji)
        showPopupButton.setOnClickListener { view ->
            showPopupMenu(view)
        }
        val rightButton = findViewById<ImageButton>(R.id.buttonRight)
        val forwardButton = findViewById<ImageButton>(R.id.buttonForward)
        val backwardButton = findViewById<ImageButton>(R.id.buttonBackward)
        val leftButton = findViewById<ImageButton>(R.id.buttonLeft)
        val forwardLeftButton = findViewById<ImageButton>(R.id.buttonForwardLeft)
        val forwardRightButton = findViewById<ImageButton>(R.id.buttonForwardRight)
```

```
val stopButton = findViewById<ImageButton>(R.id.buttonStop)
val backwardLeftButton = findViewById<ImageButton>(R.id.buttonBackwardLeft)
val backwardRightButton = findViewById<ImageButton>(R.id.buttonBackwardRight)
val RightHandUp = findViewById<ImageButton>(R.id.RightHandUp)
val RightHandMid = findViewById<ImageButton>(R.id.RightHandMid)
val RightHandDown = findViewById<ImageButton>(R.id.RightHandDown)
val LeftHandUp = findViewById<ImageButton>(R.id.LeftHandUp)
val LeftHandMid =findViewById<ImageButton>(R.id.LeftHandMid)
val LeftHandDown = findViewById<ImageButton>(R.id.LeftHandDown)
val HeadLeft = findViewById<ImageButton>(R.id.HeadLeft)
val HeadHoriMid = findViewById<ImageButton>(R.id.HeadHMid)
val HeadRight = findViewById<ImageButton>(R.id.HeadRight)
val HeadUP = findViewById<ImageButton>(R.id.HeadUp)
val HeadVertMid = findViewById<ImageButton>(R.id.HeadVMid)
val HeadDown = findViewById<ImageButton>(R.id.HeadDown)
val Glow = findViewById<ImageButton>(R.id.gloweyes)
val speedSeekBar = findViewById<SeekBar>(R.id.slider)
val speakButton = findViewById<ImageButton>(R.id.speakButton)
textInput = findViewById(R.id.textInput)


Glow.setOnClickListener {
    vibrate()
        // Toggle the state
        isGlowing = !isGlowing
        // Update the button image based on the new state
        if (isGlowing) {
            Glow.setImageResource(R.drawable.bulb_on)
            // Send command to turn on the LED
            sendCommand("GLOW")
        } else {
            Glow.setImageResource(R.drawable.bulb)
            // Optionally send a command to turn off the LED
            sendCommand("GLOW_OFF")
        }
}

HeadLeft.setOnClickListener{
    vibrate()
    sendCommand("HL")
}
HeadHoriMid.setOnClickListener {
    vibrate()
    sendCommand("HHM")
}
HeadRight.setOnClickListener{
    vibrate()
    sendCommand("HR ")
}
```

```
        HeadUP.setOnClickListener{
            vibrate()
            sendCommand("HU")
        }
        HeadVertMid.setOnClickListener {
            vibrate()
            sendCommand("HVM")
        }
        HeadDown.setOnClickListener{
            vibrate()
            sendCommand("HD")
        }
        speakButton.setOnClickListener {
            vibrate()
            speakOut()
        }
        forwardLeftButton.setOnClickListener {
            vibrate()
            sendCommand("FL")
        }
        forwardRightButton.setOnClickListener {
            vibrate()
            sendCommand("FR")
        }
        stopButton.setOnClickListener {
            vibrate()
            sendCommand("S")
        }
        backwardLeftButton.setOnClickListener {
            vibrate()
            sendCommand("BL")
        }
        backwardRightButton.setOnClickListener {
            vibrate()
            sendCommand("BR")
        }
        RightHandUp.setOnClickListener {
            vibrate()
            sendCommand("RHU")
        }
        RightHandMid.setOnClickListener {
            vibrate()
            sendCommand("RHM")
        }
        RightHandDown.setOnClickListener {
            vibrate()
            sendCommand("RHD")
        }
```

```
        LeftHandUp.setOnClickListener {
            vibrate()
            sendCommand("LHU")
        }
        LeftHandMid.setOnClickListener {
            vibrate()
            sendCommand("LHM")
        }
        LeftHandDown.setOnClickListener {
            vibrate()
            sendCommand("LHD")
        }
        rightButton.setOnTouchListener { v, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    // Send "R" command
                    sendCommand("R")
                }
                MotionEvent.ACTION_UP -> {
                    // Send "S" command
                    sendCommand("S")
                    v.performClick() // Ensure accessibility and proper click handling
                }
            }
            true
        }
        forwardButton.setOnTouchListener { v, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    // Send "F" command
                    sendCommand("F")
                }
                MotionEvent.ACTION_UP -> {
                    // Send "S" command
                    sendCommand("S")
                    v.performClick() // Ensure accessibility and proper click handling
                }
            }
            true
        }
        backwardButton.setOnTouchListener { v, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    // Send "B" command
                    sendCommand("B")
                }
                MotionEvent.ACTION_UP -> {
                    // Send "S" command
                    sendCommand("S")
                    v.performClick() // Ensure accessibility and proper click handling
```

```
                }
            }
            true
        }

        leftButton.setOnTouchListener { v, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    // Send "L" command
                    sendCommand("L")
                }
                MotionEvent.ACTION_UP -> {
                    // Send "S" command
                    sendCommand("S")
                    v.performClick() // Ensure accessibility and proper click handling
                }
            }
            true
        }

        speedSeekBar.setOnSeekBarChangeListener(object : SeekBar.OnSeekBarChangeListener {
            override fun onProgressChanged(seekBar: SeekBar, progress: Int, fromUser:
Boolean) {
                val commandValue = when (progress) {
                    in 1..9 -> "0"  // Send stop command for 0% to 9%
                    in 10..19 -> "1"
                    in 20..29 -> "2"
                    in 30..39 -> "3"
                    in 40..49 -> "4"
                    in 50..59 -> "5"
                    in 60..69 -> "6"
                    in 70..79 -> "7"
                    in 80..89 -> "8"
                    in 90..100 -> "9"
                    else -> "S"  // Default to stop if out of range
                }
                sendCommand(commandValue)
            }
            override fun onStartTrackingTouch(seekBar: SeekBar) {}

            override fun onStopTrackingTouch(seekBar: SeekBar) {}
        })
    }
    override fun onInit(status: Int) {
        if (status == TextToSpeech.SUCCESS) {
            val result = tts.setLanguage(Locale.US)
            if (result == TextToSpeech.LANG_MISSING_DATA || result ==
TextToSpeech.LANG_NOT_SUPPORTED) {
                Toast.makeText(this, "Language not supported", Toast.LENGTH_SHORT).show()
            }
```

```
            tts.setPitch(2.3f)  // High pitch for a robotic sound
            tts.setSpeechRate(0.9f)  // Slow speech rate for deliberate, mechanical delivery
            val voices = tts.voices
            if (voices != null) {
                for (voice in voices) {
                    if (voice.name.contains("en-us-x-sfg#male")) {
                        tts.voice = voice
                        break
                    }
                }
            }
        } else {
            Toast.makeText(this, "Initialization failed", Toast.LENGTH_SHORT).show()
        }
    }
    private fun vibrate() {
        if (vibrator.hasVibrator()) {
            if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
                vibrator.vibrate(VibrationEffect.createOneShot(200,
      VibrationEffect.DEFAULT_AMPLITUDE))
            } else {
                vibrator.vibrate(200)
            }
        }
    }
    private fun speakOut() {
        val text = textInput.text.toString()
        val pitch = 4.5f // Set the pitch value to modify the voice (1.0 is default)
        tts.setPitch(pitch) // Set the pitch for helium-like voice
        tts.setSpeechRate(0.6f)
        tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, null)
        tts.setPitch(1.0f) // Reset the pitch back to normal after speaking
    }
    private fun sendCommand(command: String) {
        val url = "http://$ipAddress/?State=$command"
        val request = Request.Builder().url(url).build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                runOnUiThread {
                    Toast.makeText(applicationContext, "Failed to send command:
${e.message}", Toast.LENGTH_SHORT).show()
                }
            }
            override fun onResponse(call: Call, response: Response) {
                runOnUiThread {
                    Toast.makeText(applicationContext, "Command sent successfully",
Toast.LENGTH_SHORT).show()
                }
```

```
            }
        })
    }

    override fun onDestroy() {
        if (tts != null) {
            tts.stop()
            tts.shutdown()
        }

        if (this::mediaPlayer.isInitialized) {
            mediaPlayer.release()
        }
        super.onDestroy()
    }

    private fun sendSpeed(speed: Int) {
        val url = "http://$ipAddress/?Speed=$speed"
        val request = Request.Builder().url(url).build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                runOnUiThread {
                    Toast.makeText(applicationContext, "Failed to send speed: ${e.message}",
Toast.LENGTH_SHORT).show()
                }
            }

            override fun onResponse(call: Call, response: Response) {
                runOnUiThread {
                    Toast.makeText(applicationContext, "Speed set successfully",
Toast.LENGTH_SHORT).show()
                }
            }
        })
    }
    private fun showPopupMenu(view: View) {
        val popup = PopupMenu(this, view)
        val inflater: MenuInflater = popup.menuInflater
        inflater.inflate(R.menu.emotes, popup.menu)
        popup.setOnMenuItemClickListener { menuItem ->
            handleMenuItemClick(menuItem)
        }
        popup.show()
    }

    private fun handleMenuItemClick(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.hello -> {
                Toast.makeText(this, "Action One clicked", Toast.LENGTH_SHORT).show()
```

```
            sendCommand("SAYHI")
            mediaPlayer = MediaPlayer.create(this, R.raw.walle)
            mediaPlayer.start()
            true
        }
        R.id.no -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("SAYNO")
            mediaPlayer = MediaPlayer.create(this, R.raw.no)
            mediaPlayer.start()
            true
        }
        R.id.Yes -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("SAYYES")
            mediaPlayer = MediaPlayer.create(this, R.raw.yes)
            mediaPlayer.start()
            true
        }
        R.id.eva -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("EVA")
            mediaPlayer = MediaPlayer.create(this, R.raw.eve)
            mediaPlayer.start()
            true
        }
        R.id.welcome -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("WEL")
            mediaPlayer = MediaPlayer.create(this, R.raw.welcome)
            mediaPlayer.start()
            true
        }
        R.id.gm -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("GM")
            mediaPlayer = MediaPlayer.create(this, R.raw.gm)
            mediaPlayer.start()
            true
        }
        R.id.ga -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("GA")
            mediaPlayer = MediaPlayer.create(this, R.raw.ga)
            mediaPlayer.start()
            true
        }
        R.id.ge -> {
            Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
            sendCommand("GE")
```

```
                mediaPlayer = MediaPlayer.create(this, R.raw.ge)
                mediaPlayer.start()
                true
            }
            R.id.exm -> {
                Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
                sendCommand("EXM")
                mediaPlayer = MediaPlayer.create(this, R.raw.excuseme)
                mediaPlayer.start()
                true
            }
            R.id.help -> {
                Toast.makeText(this, "Action Two clicked", Toast.LENGTH_SHORT).show()
                sendCommand("EXM")
                mediaPlayer = MediaPlayer.create(this, R.raw.excuseme)
                mediaPlayer.start()
                true
            }
            else -> false
        }
    }
}
```