CodeRover – An RC Mobile Car

By

Eng. Zeyad Youssef Shaban
Student ID: 221004741

Eng. Waleed Amgad
Student ID: 221005961

1.Objective

This project implements a small RC car platform controlled from a desktop web GUI and an ESP32 firmware stack. The system includes:

- A Unity differential-drive simulation used during algorithm development.
- A web-based GUI with a joystick and a Hoink button.
- Data logging into a local SQLite database of joystick inputs (x, y) for later analysis on how the car travelled.
- ESP32 firmware exposing a BLE GATT service to receive joystick commands and forwarding hoink commands to an Arduino via UART.
- Arduino firmware that actuates the honk on user command.

2.Project Description

Our project uses esp32 to receive signals from the mobile app (x, y coordinates representing the joystick position) through Bluetooth connection (BLE GATT), then the esp32 maps those coords to set the Velocity of left and right motors, difference in velocities will cause difference in the car orientation which allows the car to rotate (Differential Driver mechanism)

Additionally, if the user pressed a “hoink” the esp32 transmits the signal to Arduino through UART and the Arduino will take the action of hoinking.

3. Hardware Component List

- 2x Motors
- 2x Wheels
- 4x 3.7v lithium batteries
- 1x 3.7v Lithium battery holder
- 1x Esp32
- 1x Arduino uno
- 1x H-Bridge L298 Module
- Car chassis
- Wires (lots of them…)

4.Design

4.1 Physics Simulation

In order to have an easier time testing our algorithms we developed a Unity based simulator for a differential driver car.

The mathematics of the simulation was straightforward with reference to the Wikipedia page on differential wheel robot

$$\omega = (v_R - v_L)/b$$
$$R = b/2 \cdot (v_R + v_L)/(v_R - v_L)$$

Using the equation for the angular velocity, the instantaneous velocity V of the point midway between the robot's wheels is given by

$$V = \omega \cdot R = \frac{v_R + v_L}{2}$$

Where b is the base width

```
void FixedUpdate() {
    V = (Vleft + Vright) * 0.5f * Kp;
    W = -(Vright - Vleft) / baseWidth;

    Vector3 dpos = transform.forward * V * Time.fixedDeltaTime;
    rb.MovePosition(rb.position + dpos);

    float deltaYaw = W * Mathf.Rad2Deg * Time.fixedDeltaTime;
    Quaternion deltaRot = Quaternion.Euler(0f, deltaYaw, 0f);
    rb.MoveRotation(rb.rotation * deltaRot);
```

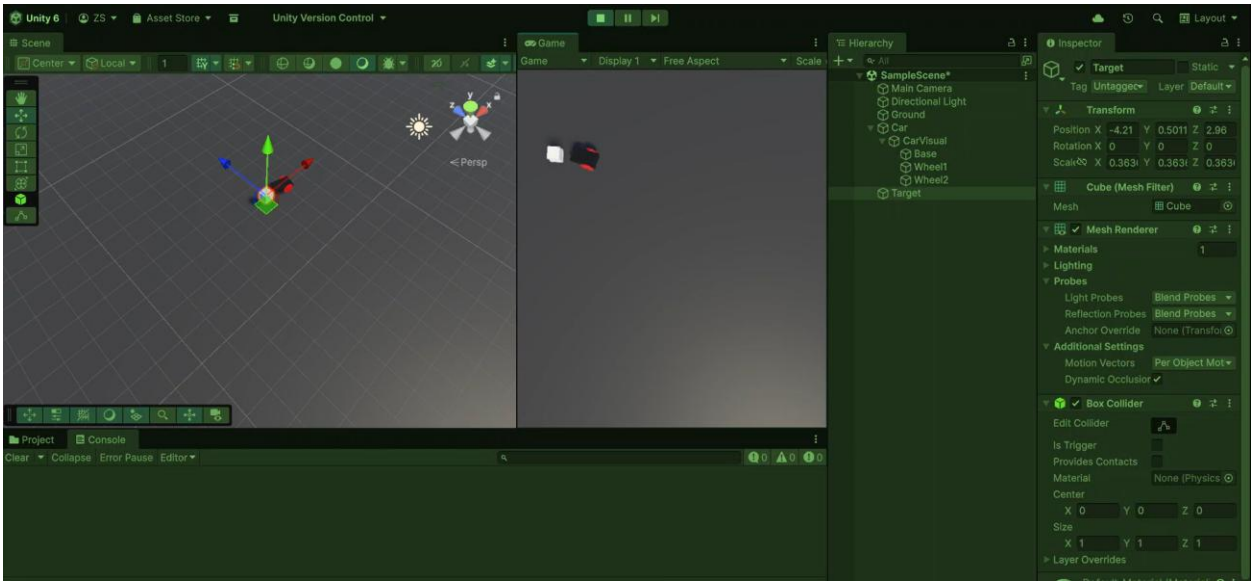
This simulation allowed us to test to test more advanced solutions, for example to make the body go to a specific target point it was just about finding the Vector pointing from our current position to the target and turning this vector into a parallel component and a perpendicular component to the body local axis through Vector decomposition, the parallel component would be facing with in the same car's direction so it determines the velocity directly

While the perpendicular component is reached through changing the car's angular velocity, the omega

Referencing to the equations in the wiki page we can find the values of Velocity of the right and left motor to satisfy the needed velocity and angular velocity

> One might face a situation where the velocity V and the angular velocity ω are given as inputs, and the angular velocities of the left ω_L and right wheels ω_R are sought as control variables (see figure above). In this case, the already mentioned equation can be easily reformulated. Using the relations $R = V/\omega$ and $\omega_R = v_R/r$ in
>
> $$\omega \cdot (R + b/2) = v_R$$
>
> one obtains the equation for the angular velocity of the right wheel ω_R
>
> $$\omega_R = \frac{V + \omega \cdot b/2}{r}$$
>
> The same procedure can be applied to the calculation of the angular velocity of the left wheel ω_L
>
> $$\omega_L = \frac{V - \omega \cdot b/2}{r}$$

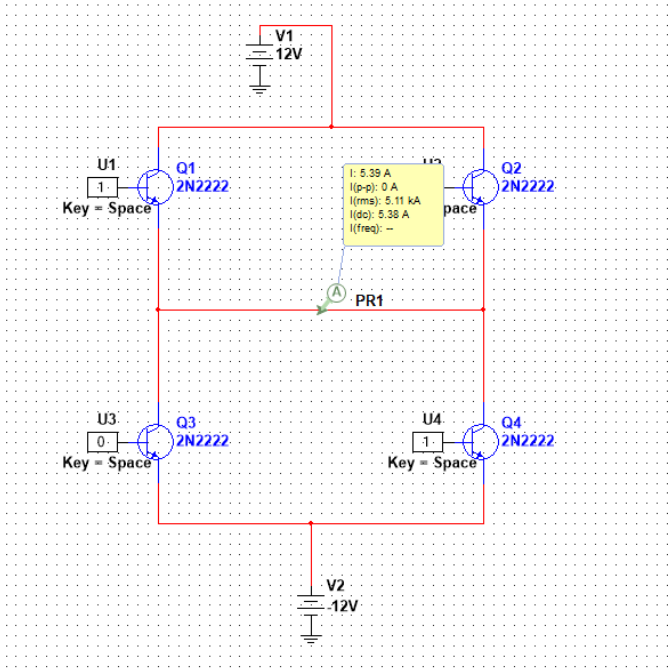
```
Vector2 robotPos = new Vector2(transform.position.x, transform.position.z);
Vector2 targetPos = new Vector2(target.position.x, target.position.z);


float theta = Mathf.Atan2(transform.forward.z, transform.forward.x);
Vector2 targetVec = targetPos - robotPos;
targetVec *= Kp;

float v_parallel = Mathf.Cos(theta) * targetVec.x + Mathf.Sin(theta) * targetVec.y;
float v_perp = -Mathf.Sin(theta) * targetVec.x + Mathf.Cos(theta) * targetVec.y;

float velocity = v_parallel;
float omega = v_perp;

float vRight = (velocity + (omega * baseWidth) / 2) / wheelRadius;
float vLeft = (velocity - (omega * baseWidth) / 2) / wheelRadius;

controller.SetWheelSpeeds(vLeft, vRight);
```



Our only issue implementing this algorithm in real world was due to how hard it was to get the actual car position, so as of now we sticked with a simple joystick controller in real life instead of a full autonomous system although such one is completely possible and was already tested in simulation

4.2 Hardware simulation

We used Multisim in order to understand the behavior of H-Bridge and how it controls the motors



Hbridge is about 4 NPN transistors forming a n H shape, if the top left and bottom right were closed switches and the others were open the electricity will flow through motor from left to right

If the top right and bottom left where closed switches and the others were open the electricity would flow from right to left, basically causing the motor to spin in reversed direction

Putting digital high on bottom right and bottom left (or top right and top left, but not both) would cause voltage difference to be zero and thus stopping the motor immediately

Lowering how “closed” the electronic switch is will cause less voltage difference, less voltage difference will cause motors to spin slower which allows us to control speed

The ESP32 was connected directly to the HBridge 4 inputs to control there speeds, and was connected to the 5V pin to get its own power supply, an external 3.7v x 4 external battery was supplied to the motors through the HBridge (and powering up the ESP32 and the arduino) and all sharing a common connected ground

4.3 Real world implementation

Regarding real world on we decided to use a joystick to control the car movement for simplicity, a Joystick have both x and y axis constrained between -1 and 1 where the origin is at <0,0>, the joystick itself was simple CSS/HTML/Javascript design, it searches for the esp32 bluetooth through its characteristic UUID, once the connection is establishes we send the joystick data to the esp

```
async function ConnectBT() {
    const device = await navigator.bluetooth.requestDevice({
        filters: [{ services: [SERVICE_UUID] }]
    });

    const server = await device.gatt.connect();
    const service = await server.getPrimaryService(SERVICE_UUID);
    bleCharacteristic = await service.getCharacteristic(CHAR_UUID);

    alert("Bluetooth connected!");
}
```

```
function sendControl(honk = 0, force = false) {
    if (bleCharacteristic) {
        const dataString = `${x},${y},${honk}`;
        const encoder = new TextEncoder();
        bleCharacteristic.writeValue(encoder.encode(dataString))
            .catch(error => console.log("BLE send error", error));
    }

    sendToServer(force, honk);
}

honkBtn.addEventListener("click", () => {
    sendControl(1);
});
```

We set the ESP32 pin responsible for motor controls as PWM pin and mapped the joystick readings accordingly to match the speed, we calculate a vector representing the joystick position and calculating its magnitude to determine velocity, while for the difference of motors velocity we (1-x) * full velocity to reduce the motor speed on the side we want to turn in, if we want to turn left we reduce the left motor's speed, if we want to turn right we reduce the speed of the right motor

```
void updateMotors(float x, float y)
{
  x = constrain(x, -1, 1);
  y = constrain(y, -1, 1);

  float vecMag = sqrt(x * x + y * y);

  float vLeft = constrain(vecMag * 255, 0, 255);
  float vRight = vLeft;

  if (x >= 0)
    vRight = vRight * (1 - x);
  else
    vLeft = vLeft * (1 - abs(x));

  if (y >= 0)
    moveForward(vLeft, vRight);
  else
    moveBackward(vLeft, vRight);
}
```

Finally we save the joystick readings into a local database, the javascript client calls our backend server written in Python FastAPI

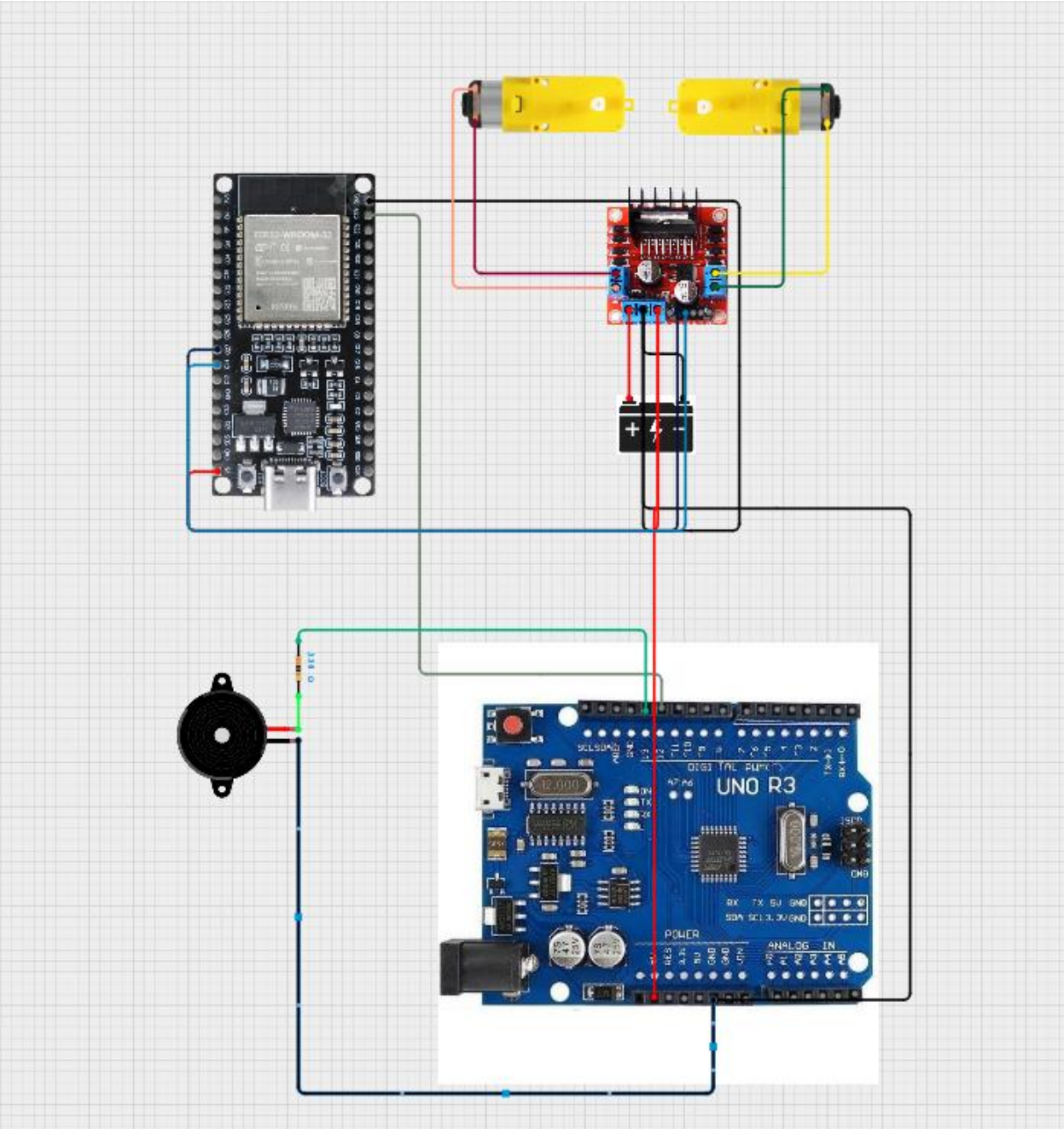
```
function sendToServer(force = false, honk = 0) {
    const now = Date.now();

    if (!force && (now - lastSend < SEND_INTERVAL)) return;
    lastSend = now;

    fetch("/control", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ x: x, y: y, honk: honk })
    }).catch(err => console.error("Server error:", err));
}
```

We can view the logs we have saved by fetching the fastAPI

```
async function openModal() {
    modal.style.display = "block";
    logContainer.innerHTML = "Fetching...";

    try {
        const response = await fetch("/logs");
        const data = await response.json();

        logContainer.innerHTML = data.map(entry => `
            <div class="log-entry">
                [${entry.time}] X: ${entry.x} | Y: ${entry.y}
            </div>
        `).join('') || "No logs found.";
    } catch (err) {
        logContainer.innerHTML = "Error loading logs.";
    }
}
```

```
@app.get("/logs")
def get_logs() -> list[dict[str, Any]]:
    db = SessionLocal()
    # Get last 50 entries
    logs = db.query(Control).order_by(Control.timestamp.desc()).limit(50).all()
    db.close()
    return [{"x": l.x, "y": l.y, "time": l.timestamp.strftime("%H:%M:%S")} for l in logs]

app.mount("/", StaticFiles(directory="./"), name="static")
```

5.Circuit Diagram:



6. Procedure:

First assemble the car model, attach esp32, Arduino, hbridge and the system battery, attach motors and wheels at the back

Connect +ve battery pin to the hbridge +12v, and -ve to gnd

Connect esp32 5V to +5V in hbridge and ESP GND to H-Bridge GND

Connect the right motor to Output1, Output2 on the H-Bridge

Connect esp32 GPIO27 -> In1, GPIO 14 -> in2

Connect esp32 GPIO 12 -> In3, GPIO 13 -> in4

Connect Arduino Gnd to H-Bridge GND (all of Arduino, esp, and battery should have common GND)

Connect Arduino +5v to H-Bridge +5v

Connect external buzzer to a resistance 330 ohm in series to arduino’s pin 13

Connect arduinos pin 12 to Esp32 GPIO 23

Open the server script and run uvicorn server:app then open localhost:8000

Press the Connect button and select “ESP” (note if it wasn’t showing press the En button on esp32 to reset the Bluetooth connection)

Start moving the Joystick to control the car

7.GUI:

We used FastAPI to set up a server to receive the <x,y> coordinates and save them to a based database

Each table entry contains the timestamp, the <x,y> coordinates and a unique identifier

```python
class Control(Base):
    __tablename__ = "controls"
    id = Column(Integer, primary_key=True)
    x = Column(Float)
    y = Column(Float)
    timestamp = Column(DateTime, default=datetime.utcnow)
```

When the server receives a request from the client to save a “Control” instance we create a new entry and save it..

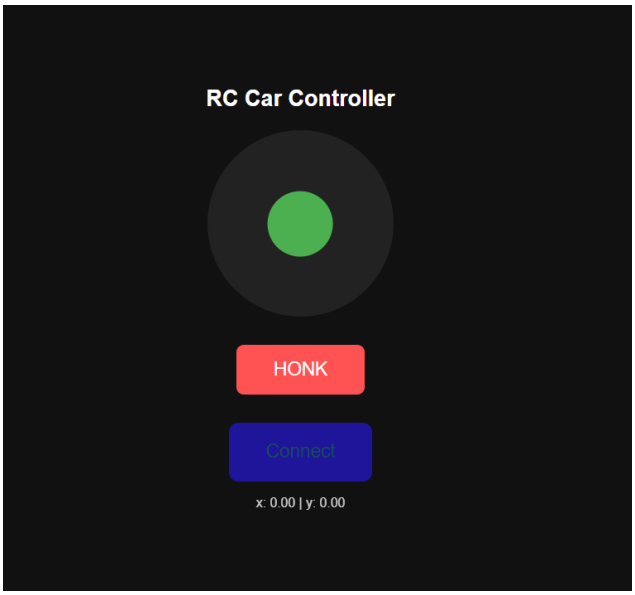
```python
@app.post("/control")
def receive_control(data: ControlData) -> dict[str, str]:
    db = SessionLocal()
    entry = Control(x=data.x, y=data.y)
    db.add(entry)
    db.commit()
    db.close()
    return {"status": "ok"}
```

We handle an API for the client to request the log data

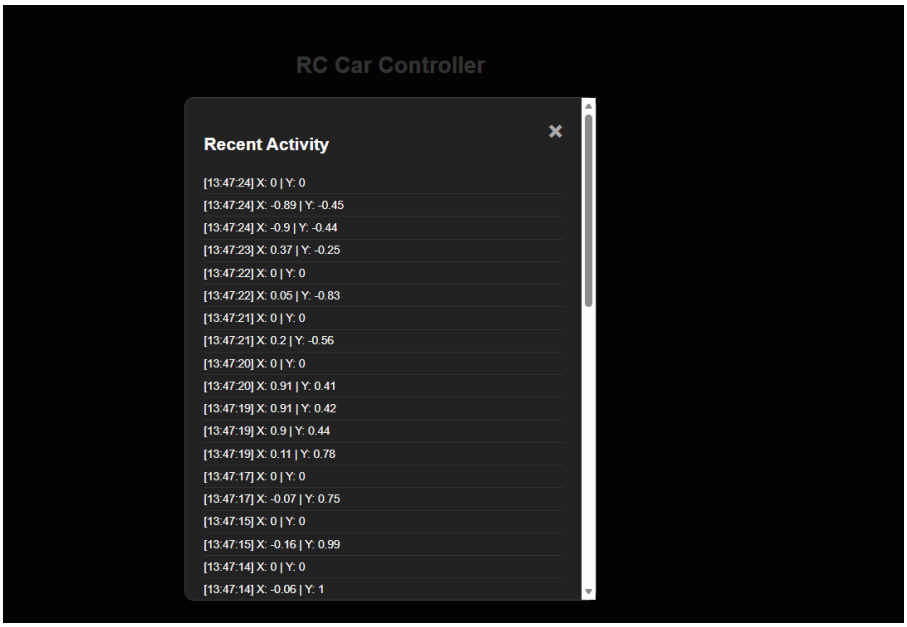
```python
@app.get("/logs")
def get_logs() -> list[dict[str, Any]]:
    db = SessionLocal()
    # Get last 50 entries
    logs = db.query(Control).order_by(Control.timestamp.desc()).limit(50).all()
    db.close()
    return [{"x": l.x, "y": l.y, "time": l.timestamp.strftime("%H:%M:%S")} for l in logs]

app.mount("/", StaticFiles(directory="./"), name="static")
```

Note that all of the data sent and received are in JSON format which is standard for APIs

8.GUI:



Virtual Joystick



Previous logs for the car movements in terms of joystick control <x,y>

9.Resources:

Differential wheeled robot - Wikipedia