Hand Following Mini Robot

Ekin Ercetin, Jingli Kong

1 Introduction

Our project involved designing and building an autonomous robot car capable of following an object, typically a hand, placed in front of it. The robot integrates sensors and control systems to dynamically scan its environment, detect objects, and respond by maintaining a fixed distance from the detected object. This capability is a foundational element of many autonomous systems and allowed our group to explore sensing and responding to a real-time environment.

This robot represents a fundamental implementation of the concept of using sensor input to maintain a desired state, a principle widely applied in various systems. For example, HVAC systems use temperature data to adjust heating or cooling to maintain a set temperature. Similarly, our robot uses distance measurements from two front-facing sensors to decide its actions, whether to turn left, turn right, move forward, or reverse. All to maintain a consistent distance from a detected object.

The project is related to embedded systems because we are using a microcontroller to serve as the brain of our robot processing sensor inputs and controlling motor output to achieve real-time responsiveness. Throughout the development process, we gained valuable skills and knowledge, including using an oscilloscope to analyze PWM output for motor debugging, soldering components, basic electronics, and wiring, designing a chassis while managing weight distribution, and understanding and adapting manufacturer-provided APIs for our microcontroller. Although our final solution doesn't sound very complicated, along the journey of producing it we learned a lot about engineering and debugging electrical, mechanical, and software issues.

1.1 Objectives

1.1.1 Environmental Scanning

The robot continuously scans its surroundings using ultrasonic sensors to detect objects in its path.

1.1.2 Distance Maintenance

Upon detecting an object, the robot adjusts its position to maintain a predefined distance from the object, ensuring smooth and consistent movement.

2 Design and Implementation

2.1 Hardware

For our project, we have designed a hardware setup that integrates several key components. Figure 1 shows the hardware diagram of our system. The microcontroller is the central element that connects and controls the various peripherals. We are using encoders (ENC1 and ENC2), a motor driver, two ultrasonic sensors (ULT1 and ULT2), and additional components such as the two front wheels, one dummy wheel, and a chassis for building the car from scratch. The detailed pin layout for the microcontroller is shown in Figure 2. This setup allows the microcontroller to effectively control and communicate with the motor driver, encoders, and ultrasonic sensors. The wiring of each component is arranged to ensure clear communication paths and minimize interference, although the longer sensor wires may become a concern in future versions due to the potential movement of the wheels.

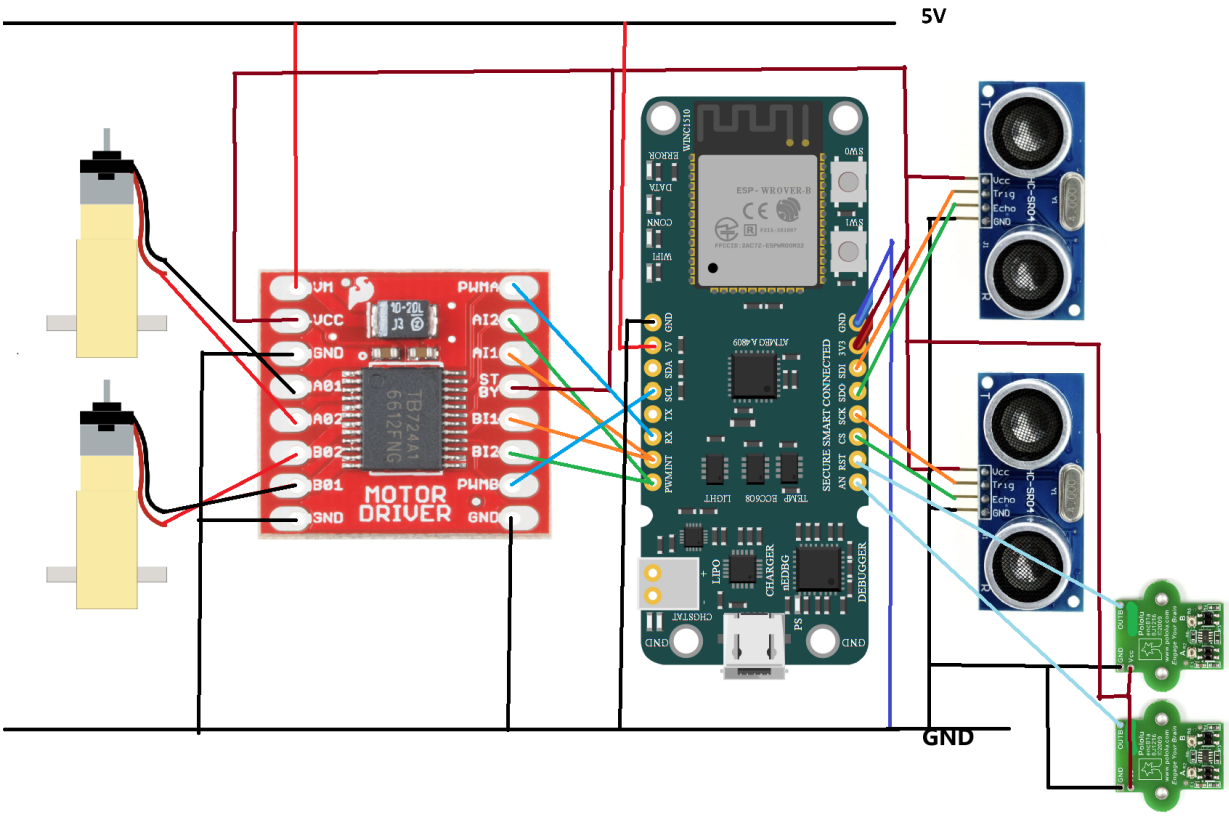

Figure 1: Circuit Diagram

			ENC1	ENC2	MOTOR D	ULT1	ULT2	DISPLAY + IO (not used)
PD7	AN			OUT + INT				
PD5	RST		OUT + INT					
PA7	SPI CS					ECHO		
PA6	SPI SCK					TRIG		
PA5	SPI MISO						ECHO	
PA4	SPI MOSI						TRIG	
3V			VCC	VCC	VCC			VCC
GND			GND	GND	GND	GND	GND	GND
PD1	PWM				AIN2 + BIN2			
PD6	INT				AIN1 + BIN1			
PC1	USART1 RX				M2 PWM			
PC0	USART1 TX				M1 PWM			
PA3	I2C SCL							SCL
PA2	I2C SDA							SDA
5V					VM	VCC	VCC	
GND								

Figure 2: Pin Layouts

2.1.1 Design Decisions

We chose this hardware setup because we lacked soldering skills, which made it difficult to design a system with minimal wiring. As a result, we used longer wires to connect the sensors to the microcontroller. These longer wires could cause problems in the future if the car's wheels catch and damage them. With better soldering skills and more time, we could have created a more compact and reliable design. However other than the mechanical design of the car and the longer wires, we think our system design is efficient and simple to implement and use with the software.

In addition, we did preliminary calculations for Wheel speed/motor performance, acceleration, and energy constraints to make sure our robot could function with our components.

We used the DC Hobby Motors from Sparkfun, 65 mm rubber tires. The DC motor has 200 RPM, and 84.99/600 (oz-in / mNm) torque.

Wheel and Motor Speed Calculations The linear speed of the robot depends on the wheel circumference and the angular velocity of the motors. We were given the diameter and rpm of the motors from the specifications.

For wheels with a diameter $D = 65$ mm $= 2.56$ in, the circumference C is calculated as:

$$C = \pi \times D \approx 8.04 \text{ in/rev}. \tag{1}$$

At the rated motor speed of $\omega = 200$ RPM, we convert to revolutions per second (RPS):

$$\text{RPS} = \frac{200 \text{ rev/min}}{60 \text{ s/min}} \approx 3.33 \text{ rev/s}. \tag{2}$$

The ideal no-slip linear speed v of the robot is then:

$$\begin{aligned} v &= C \times \text{RPS} \\ &\approx 8.04 \text{ in/rev} \times 3.33 \text{ rev/s} \\ &\approx 26.8 \text{ in/s} \approx 2.23 \text{ ft/s}. \end{aligned} \tag{3}$$

Torque and Force Relationship at the Wheels The relationship between motor torque (τ) and force at the wheel periphery (F) is:

$$F = \frac{\tau}{r}, \tag{4}$$

where $r = \frac{D}{2} = 1.28$ in. For the rated torque of $\tau = 85$ oz-in:

$$\begin{aligned} F &= \frac{85 \text{ oz-in}}{1.28 \text{ in}} \\ &\approx 66.4 \text{ ozf} \approx 4.15 \text{ lbf}. \end{aligned} \tag{5}$$

This indicates that each motor can theoretically provide up to 4.15 lbf of linear thrust.

With this torque, the maximum weight the robot can support is approximately 5.19 lbs. This is calculated by dividing the force generated at the wheels (F) by the coefficient of friction (μ):

$$W_{\max} = \frac{F}{\mu} = \frac{4.15}{0.8} \approx 5.19 \,\text{lbs}.$$

This calculation uses the coefficient of concrete which is close enough to carpet. Additionally, the robot's speed is fast enough to chase down a hand so we felt pretty safe with the design.

For the power side we just made sure to have 6V powering the motors, and ultrasonics sensors.

2.2 Software

Our software implementation consists of two parts: handling sensor readings and handling motor control using the sensor information.

2.2.1 ultrasonic.c

This code is designed to interface with two ultrasonic sensors using our microcontroller.

1. Definitions and Setup
    - We first define the CPU clock frequency for the delay calculations (3333333)
    - We create our sensor global variables to hold configurations for the two sensors. Each sensor has a trigger pin that sends a pulse to initiate ultrasonic measurement, an echo pin that captures the reflected signal to measure distance, and a distance that is used to store the computed distance to be passed to the motor controller.
    - We define a global variable to track the high time of the echo signal for distance calculation.
    - We initialize the Timer B (TCB0) to measure time intervals in microseconds. The timer is able to count up to 19.6ms at the given CPU frequency.

- Lastly, we initialize the ultrasonic sensors by configuring the TRIG pins as outputs and ECHO pins as inputs.

2. Sensor Readings

   - It begins by sending a 10µs pulse from the TRIG PIN of the sensor to initiate the ultrasonic signal. It then waits for the ECHO PIN to go high, indicating the signal has returned after bouncing off an object while monitoring for a timeout to handle cases where no signal is received.
   - Once the ECHO PIN goes high, the timer (TCB0) starts to measure the duration for which the ECHO PIN remains high, representing the time it takes for the ultrasonic wave to travel to the object and back.
   - This pulse duration is converted into distance using the speed of sound formula, scaled to centimeters, and clamped to a range of 0–30 cm to filter out erroneous readings. Finally, the calculated distance is stored in the sensor structure.

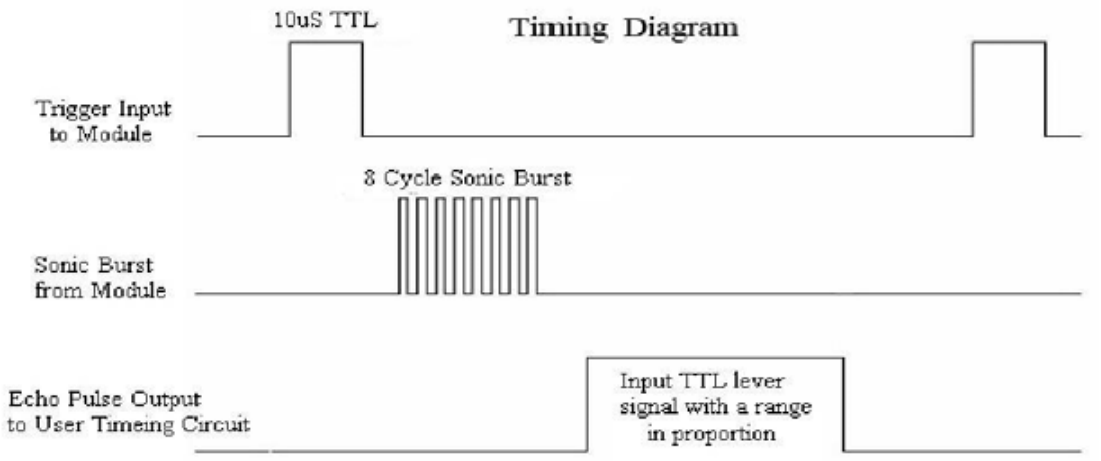

Figure 3: Timing Diagram

2.2.2 move.c

This code implements a control system for our car by using ultrasonic sensor readings and encoder feedback to navigate and adjust its speed and direction dynamically.

1. Definitions and Setup

   - We use various macros for speed, angle, encoder pulse per distance, and timer periods.
   - We additionally use MULTIPLIER, which is set to 100, to help scale integer calculations to avoid floating point arithmetic.

- We use global variables to hold left and right encoder counts, which track pulses from encoders to measure wheel rotations to be able to calculate the actual distance.
- We initialize Timer A (TCA0) for generating periodic interrupts every 10ms and controlling the PWM signals for motor speed.
- We configure encoder input pins and set them to trigger interrupts on rising edges.
- Lastly, we initialize the motors by setting up the motor control pins and assigning them to the timer's PWM outputs.

2. Interrupts

   - Encoder ISR tracks the edges of encoder signals on specific pins and increments left and right encoder counts accordingly.
   - Timer ISR fires every 10ms to update the control logic. After a fixed interval (CYCLE), which we currently set as 5, it calls the `set_motor_speed` function to adjust the motor's speed and angle.

3. Motor Speed Control

   - This function dynamically adjusts the power delivered to the motor to achieve a desired speed and direction. The system calculates the average speed of the car based on encoder feedback from the left and right tires which ensures precise measurement of actual speed. It then compares the current speed and direction angle to target values, incrementally adjusting them to minimize error. Speed adjustments are determined using a proportional approach, where the PWM signals for each motor are computed based on the desired speed and turning angle. The angle affects the differential speed between the left and right motors, which enables turning.
   - We also include bounds to prevent exceeding maximum speed and angle limits.
   - Direction control is managed by setting the motor driver pins to specify forward or backward motion based on the sign of the speed. Setting AIN1/BIN1 high, and AIN2/BIN2 low moves the car forward, and setting AIN1/BIN1 low, and AIN2/BIN2 high moves the car backward. Figure 4 shows all possible configurations.
   - Lastly, one special feature we included was logarithmic scaling for smoother transitions, both for turning and increasing speed. Figure 5 shows the lines of code used to implement this feature.

4. Main Loop

   - The main loop is responsible for sensor processing, where the system reads distances from ultrasonic sensors and applies a smoothing filter for stability. The

system then updates sensor speed and sensor total angle variables. If the object is far, the car moves forward with increasing speed. If the object is close, the car moves backward. And if the object is at the deadzone, which is set to around 15cm, then the car doesn't move. Finally, the system adjusts the turning angle based on the difference between the two sensor readings.

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Figure 4: Motor Driver Control Logic

```
//Logaritmic scale : 0 -> 25%
if (pwm0 > 0) pwm0 = pwm0 + (TIMER_PERIOD - pwm0) / 3;
if (pwm1 > 0) pwm1 = pwm1 + (TIMER_PERIOD - pwm1) / 3;
```

Figure 5: Logarithmic scaling

2.2.3 FreeRTOS

Additionally, we have included a working FreeRTOS version of this project; however, after implementing our final version without using FreeRTOS, we concluded that using FreeRTOS was not a needed feature. This is mainly due to the number of concurrent tasks we do without using interrupts. The motor control is done using 10ms interrupts, which offloads critical timing-sensitive operations from the processor without requiring task scheduling. As a result, the remaining tasks, such as sensor readings, do not need the multitasking capabilities provided by FreeRTOS. Also, avoiding FreeRTOS simplifies the overall system architecture, reduces resource overhead, and minimizes debugging complexity, which makes it a more suitable choice for our specific application.

2.2.4 Challenges

One of the main technical challenges we encountered in our implementation was making sense of the sensor readings. Specifically, the ultrasonic sensors we initially chose for the project proved to be not well suited for our application. These sensors struggled to provide accurate and consistent distance measurements, especially in environments with irregular surfaces. In retrospect, a different type of sensor, like LIDAR or infrared, could have been a more suitable choice for getting precise and dependable distance information for our specific use case.

2.3 Reflection

One of the initial improvements we would make is to redesign the chassis, which currently relies on a rear support wheel. This wheel often interferes with the robot's ability to turn precisely, as it tends to obstruct smooth maneuvering during turns. Aside from that our chassis balanced weight fairly well and is able to detect sensor inputs and react to them.

In addition, I would slightly change the algorithm to maintain a fixed distance from an object. After understanding more about differential drive kinematics I now understand there's a much simpler way to implement the movement of our robot.

So right now the robot is equipped with two front-facing distance sensors:

- One is positioned toward the left side (measuring distance d2), and the other is toward the right side (measuring distance d2).
- Our goal is to maintain a set distance say 5 cm from the object in front of the car
- Then we can define the errors:
    1. e1 = d1- 5, e2 = d2 - 5
    2. These errors represent how far or too close the robot is relative to the desired distance on each side.

Then we can control the two wheels' velocities v1 and v2 as follows:

- v1 = ke2
- v2 = ke1

Where k is just some proportional gain constant, we can experimentally choose it. The left wheel velocity depends on the error from the right-hand sensor, and the right wheel velocity depends on the error from the left-hand sensor.

Then based on these control laws we would have our Forward/Backward movement and turning behavior.

2.3.1 Forward/Backward Movement

If both sensors measure a distance larger than the desired distance (i.e., both e1 ¿ 0 and e2 ¿ 0), then both v1 and v2 will be positive. This means both wheels will turn forward, pushing the robot closer to the target. If both sensors measure a distance less than the desired distance (e1 ¡ 0 and e2 ¡ 0), both wheel velocities will become negative, causing the robot to move backward and increase the distance again. Thus, the system naturally handles whether to move forward or backward based on the sign of the errors; no special handling or angle calculation is required.

2.3.2 Turning Behavior

If one side is closer than desired and the other side is farther than desired, the wheel velocities become unbalanced. For example, if d1 ¡ 5 and d2 ¿ 5 then: v1 = ke2 ¿ 0 and v2 = ke1 ¡ 0

This means one wheel spins forward and the other spins backward, causing the robot to pivot in place, effectively turning it towards the side that is farther from the target. Over time, this corrects the orientation so that the robot faces the object more directly. The difference in wheel velocities induces a rotation without the need to explicitly compute any angles.

2.4 Final Thoughts

Using this system of thought the control of the robot would be simpler. We wouldn't have to worry about calculating angles and the motor speeds would be more accurate. We can control the direction of the robots using a motor driver and then use PWM to change the velocity of the motors on the sides of the robot.

The primary improvement would involve replacing the ultrasonic sensors with IR range finders to enhance our project. This change would increase accuracy and minimize noise caused by the ultrasonic sensors, which can sometimes detect unintended objects due to their wider field of detection. But looking further the end goal would be to stream video from some camera instead of distance sensors and be able to map and navigate based on that data.

3 Final Product

Figure 6 shows our final product from different angles.

The main project goal that we met was building a small robot that could read distance information from two ultrasonic sensors in front of the car and follow a hand in front of the car. We incorporated FreeRTOS into the project and were able to use a motor driver to control the movement of the robot.

The main project goal that we failed to meet was using an oled display to display some data from the devices. The OLED display that we received was faulty. We tried using an

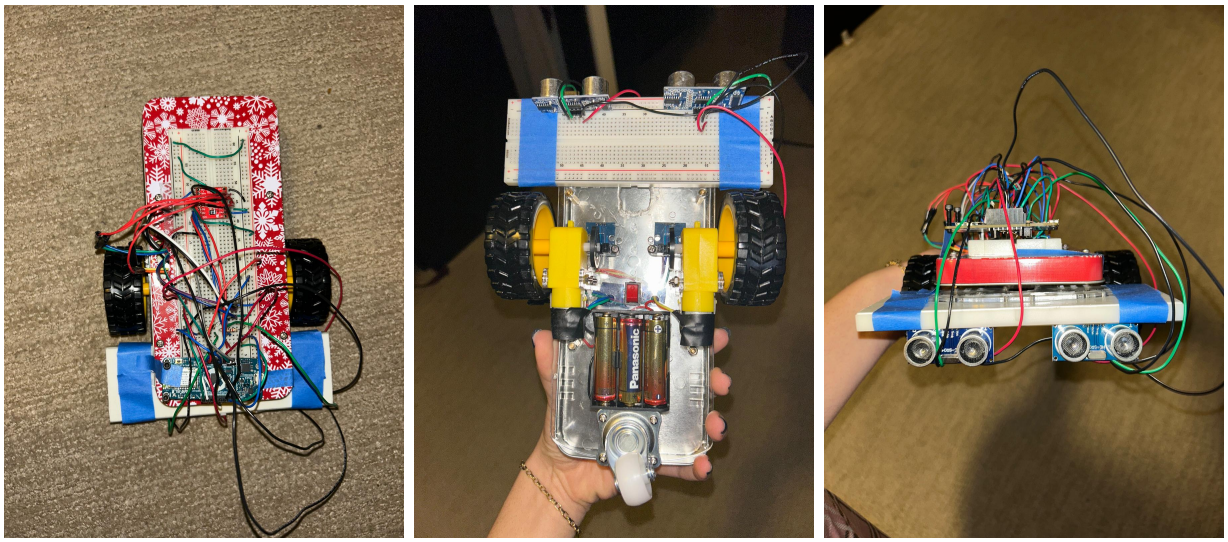

Figure 6: Final product

arduino with the display and that didn't work so we decided to forgo adding the OLED display to the project.