MEGN540 Project Final Report

Sebastian Negrete-Alamillo, Keenan Buckley, Chris Larson

1. Problem Statement

Many people are unable or unwilling to get up, walk to, and grab an item they need or desire around their home. Examples include elderly persons who need to take medication at a specific time daily but may be forgetful, and college students who are thirsty but too incapacitated to get their next beverage themselves.

We designed and built a robot that can deliver a necessary item to those people when they need it.

2. Design Concept

We built a two-platform tank-drive mobile delivery robot. The lower platform secured our electronics, and the upper platform carried the delivery payload. Upon activation, our robot identified the people nearest it, drove to those people, and stopped within their arms' reach to deliver the payload. The initial requirements we set for the robot were as follows:

1. Listen for and react to an activation signal over serial.
2. Identify persons in its FOV and target the nearest person to it (if any).
3. Orient itself toward the person and drive to them in a straight line on a flat, carpeted surface.
4. Carry a payload of at least 16oz.
5. Stop within arm's reach of the target person to deliver the payload.

We successfully met all of our requirements. The final build appears in Figure 1.

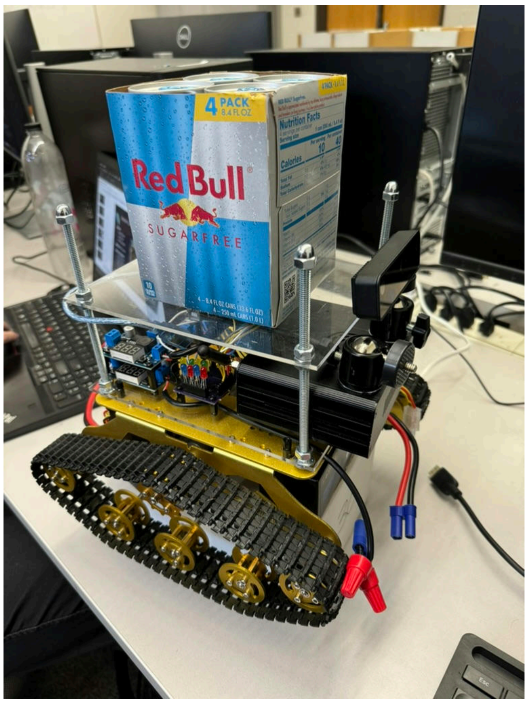


Figure 1: Final mobile delivery robot build.

2.1. Sensors

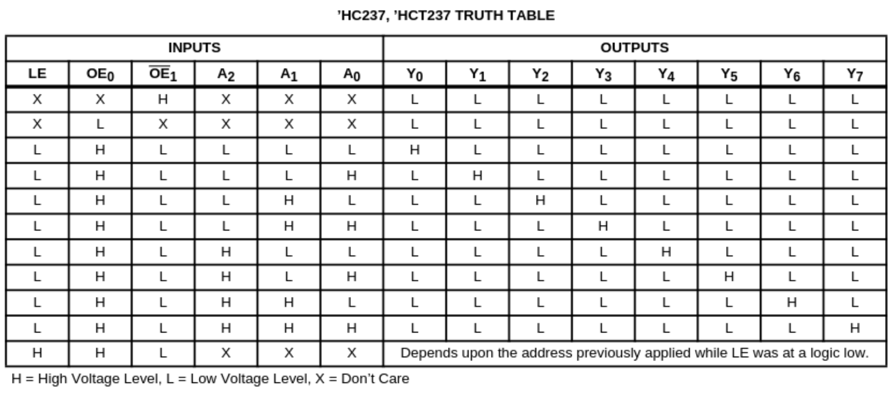
- *Stereo camera:* Luxonis Oak-D Lite for object detection and depth estimation.
- *Wheel encoder (x2):* Hall encoders for motion control feedback.

2.2. Actuators

- *DC motor (x2):* 12V, 150rpm DC motors power the robot's drivetrain.

3. PCB Design

- *LED Demux:* Our printed circuit board's primary purpose is to enable us to indicate our robot's status (Error, Ready, Linear Closed-loop, PWM, Find Targets Velocity, Angular Closed-loop, and Waiting) visually while occupying only four output pins on our microcontroller. Our PCB incorporated a 3 to 8 demuxer, a resistor, and ports for 8 LEDs to achieve this. The final PCB directed four inputs from the Arduino to A0, A1, A3, and OE0 on the demux. LE and OE1 were also required to be grounded to logical low for the demux to work correctly.



'HC237, 'HCT237 TRUTH TABLE

INPUTS						OUTPUTS							
LE	OE_0	$\overline{OE}_1$	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
X	X	H	X	X	X	L	L	L	L	L	L	L	L
X	L	X	X	X	X	L	L	L	L	L	L	L	L
L	H	L	L	L	L	H	L	L	L	L	L	L	L
L	H	L	L	L	H	L	H	L	L	L	L	L	L
L	H	L	L	H	L	L	L	H	L	L	L	L	L
L	H	L	L	H	H	L	L	L	H	L	L	L	L
L	H	L	H	L	L	L	L	L	L	H	L	L	L
L	H	L	H	L	H	L	L	L	L	L	H	L	L
L	H	L	H	H	L	L	L	L	L	L	L	H	L
L	H	L	H	H	H	L	L	L	L	L	L	L	H
H	H	L	X	X	X	Depends upon the address previously applied while LE was at a logic low.							

H = High Voltage Level, L = Low Voltage Level, X = Don't Care

Figure 2: Truth table for the demux on our PCB [1]. Inputs from A0-A2 set, which output Y0-Y7 to set.

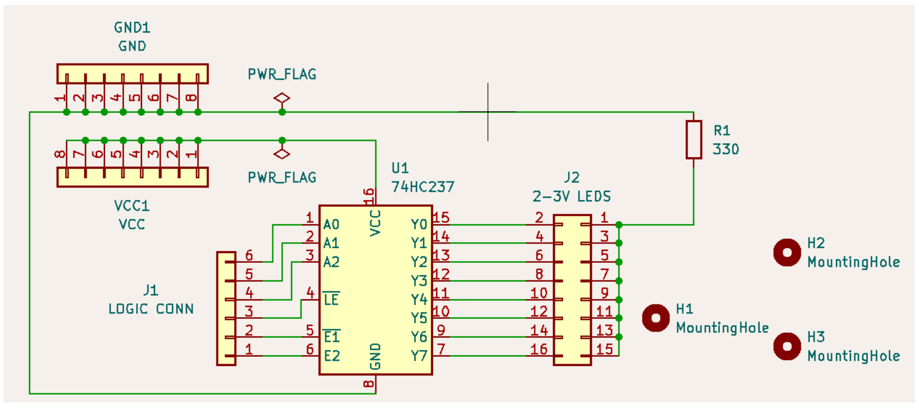


Figure 3: PCB schematic from KiCad.

- *VCC and GND Rails:* The PCB also cleaned up our robot's wiring by supplying 5V power rails for our Arduino and motor encoders. These power rails were eight-pin headers connected by 17 mil traces. We computed our trace width of 17 mil using an online calculator [2]. We set the width so that a 1.4 mil thick copper trace would have a 10 degrees Fahrenheit increase in temperature when conducting 1 Amp at 70 degrees Fahrenheit. 1 Amp is sufficient to handle the Arduino's max current of 1 Amp and the comparatively small current requirements of the demux and encoders.

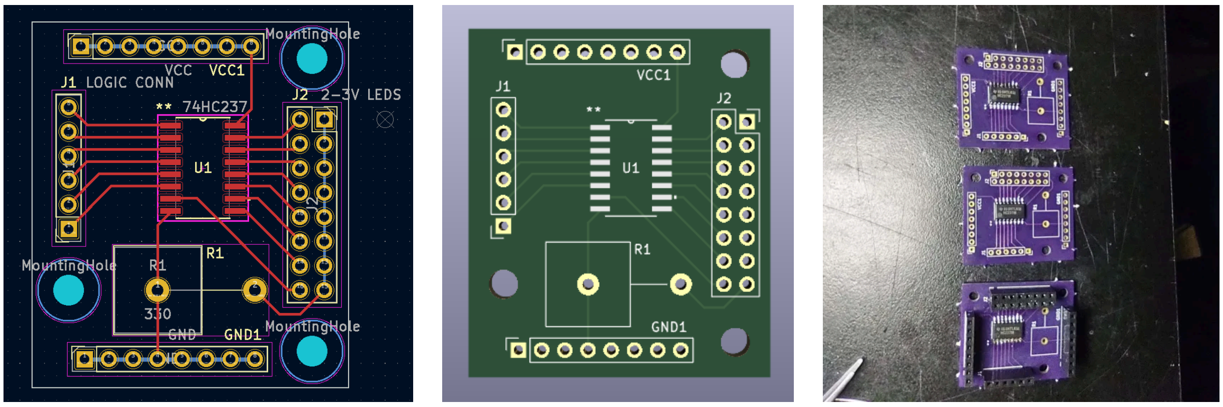


Figure 4: PCB design, render, and final result.

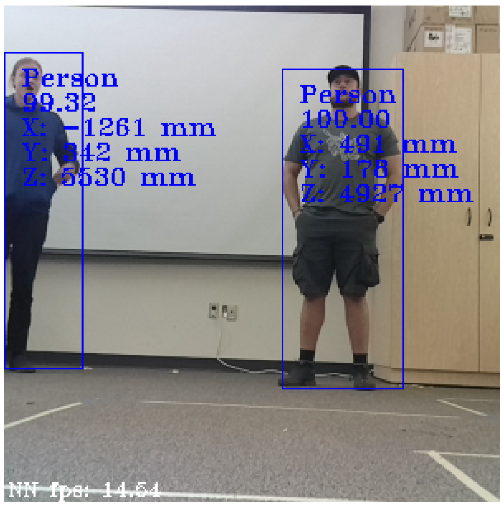


Figure 5: Person detection results with two people in field of view.

4. Software Implementation

4.1. Person Detection & Waypoint Algorithm

The person detection algorithm we implemented identifies people in the camera's FOV and tracks them (using a Euclidean distance heuristic) frame-to-frame. It considers targets "locked" if detected in the FOV for 30 consecutive frames. Upon identifying a set of locked targets, the world coordinates of the targets get turned into waypoints for the robot to travel to along a multi-segment trajectory.

Person Detection & Waypoint Algorithm Pseudocode:

```
Let tracked_persons be an empty list. Let targets_locked equal false.

While targets_locked is false:

• Get the depth frame from the camera

• Get the RGB frame from the camera

• Run MobileNet image classifier on RGB frame:
    ‣ Filter to person objects with confidence > 0.7
    ‣ Denote bounding boxes around all persons detected

• If persons detected in frame:
    ‣ Merge depth frame and person bounding boxes
    ‣ Compute R^3 world coordinates of bounding boxes using depth + camera intrinsics

• Check if detected persons are tracked persons:
    ‣ For tracked_person in tracked_persons:
    ‣ Is the Euclidean distance of tracked_person within 0.2m of the detected person?
        – If yes, increment tracked_person's match count
            • Recursively update tracked_persons position in the world using moving average position
        – If no, add the person to tracked persons

• Check: do all tracked_persons have match_count > 30?
    ‣ If yes, targets are locked. Set targets_locked = True and break.
    ‣ If no, the targets are not locked. Continue.
```

4.2. Translate Waypoints to Path

After determining the waypoints, we converted them to linear displacement in millimeters and angular displacement in radians to plan our robot's path. We did this by looping over the waypoints and computing the next stop's relative displacement and heading.

Our camera often gave us a depth estimate, or z-coordinate, greater than the actual depth of our target. We reduced our initially computed displacements by 20%, which corrected the camera depth errors adequately enough to meet our "arm's length" criteria.

To determine the turn radius, we used the equations below to center our turns so that the origin was straight ahead at the robot's $\frac{\pi}{2}$ axis and corrected the angles by subtracting the previous waypoint's heading. We show an example result in Figure 6.

$$\theta_{\text{Calc}} = \operatorname{atan2}(\text{Diff\_z}, \text{Diff\_x}) \tag{1}$$

$$\theta_{\text{Centered}} = \theta_{\text{Calc}} - \frac{\pi}{2} \tag{2}$$

$$\theta_{\text{Corrected}} = \theta_{\text{Centered}} - \theta_{\text{Prev}} \tag{3}$$

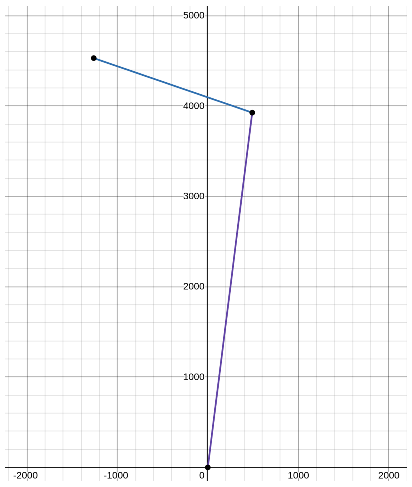


Figure 6: Resulting path from waypoint conversion for two people detected.

4.3. Motion Control Algorithm

We sent our path generated using waypoints to the Arduino over serial. We implemented a trapezoidal trajectory between waypoints to ensure smooth motion and to avoid spilling any payload contents.

Motion Control Pseudocode

```
displacement_to_decel = target_displacement - (velocity_current)**2 / 2*max_acceleration;
if (current_displacement < displacement_to_decel) {
  target_velocity = current_velocity + (direction * max_acceleration * dt);
  target_velocity = Saturate(target_velocity, max_velocity);
} else {
  target_velocity = current_velocity - (direction * max_acceleration * dt);
}
```

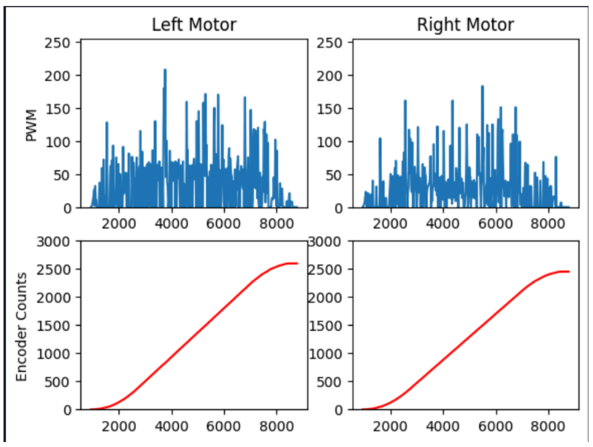


Figure 7: Resulting trapezoidal trajectory and PWM signals to motors.

4.4. Serial Communication

We designed and implemented our own two-way serial communication protocol using Python's struct library as a basis. With struct, we defined 11 message types where each message would begin with a byte that specified which message type was being transmitted and end with the newline character. Python and Arduino's serial libraries already handled buffering for us, so we would read a line of bytes from serial and use Python's struct and C++'s packed structs to decode the sequence of bytes into bytes, ints, and floats. Then, we defined specific behavior for each message on a switch statement for the first byte in each message.

5. Conclusion and Future Work

In the end, we succeeded in creating a minimally viable mobile delivery robot. It can accurately detect people within a 5m radius. It can accurately generate a trapezoidal trajectory that ensures no spillage and can execute the delivery in a straight line, stopping at arm's reach. Since this is a minimally viable product, it cannot actively scan its environment and search for a person. Moreover, the robot cannot detect and avoid any obstacles in its path, so its success is limited to unobstructed paths to a person within 5m of the robot.

In the future, we would like to implement an environment scanning routine to have the robot turn or traverse in a circle to actively search for a person and lock onto their position for dynamic deliveries. Moreover, we would modify the trajectory generation to detect and avoid obstacles to increase the capabilities of the robot.

6. Works Cited

[1] Texas Instruments, "CD74HC137, CD74HCT137, CD54HC237, CD74HC237, CD74HCT237 datasheet (Rev. F)," SCHS146F, October 2003.

[2] "Printed Circuit Board Trace Width Tool | Advanced Circuits," www.4pcb.com. https://www.4pcb.com/trace-width-calculator.html