M-DOT

CSCE 336 Embedded Systems // Project 1 Report

Carston Wiebe

March 31, 2025

Contents

1 Introduction

M-DOT is a Arduino-based maze-navigating robot car programmed using ATmega328P registers and a custom library. It uses two DC motors controlled by a L298 motor driver to maneuver, along with an HC-SR04 ultrasonic sensor mounted to a SG90 servo to "see" its surroundings. The goal of the project is to create a robot car that can successfully navigate a maze-like obstacle course using its onboard sensor, but for this report it will only be following a wall.

2 Preliminary Design

The robot has three main external components that need to be configured: The servo, the sonar (ultrasonic sensor), and the drivetrain (DC motors). They are connected to the Arduino UNO as seen in Figure 1

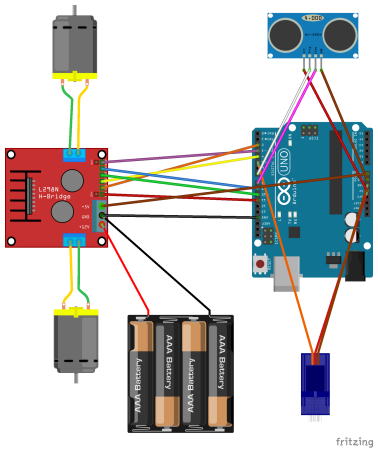

Figure 1: Initial design schematic.

All three timers are used in configuring these components, as seen in Table 1.

Table 1: Timer configurations.

Timer	Purpose	Mode	Output(s)
`timer0`	Servo PWM	Phase correct PWM	`OC0B`
`timer1`	General timer	Normal	None
`timer2`	Drivetrain PWM	Phase correct PWM	`OC2A`, `OC2B`

2.1 Servo Configuration

In order to generate a PWM with the duty cycle and period expected by the SG90 servo, `timer0` was configured to use `OCR0A` as its TOP, the value of which

can be found with equation:

$$\text{TOP} = \text{OCR0A} = \frac{\text{CPU frequency} \times \text{servo period}}{\text{prescaler} \times 2} = 156.25 \approx 156$$

where:

$$\begin{aligned} \text{CPU frequency} &= 16\,[\text{MHz}] \\ \text{servo period} &= 20\,[\text{ms}] \\ \text{prescaler} &= 1024 \end{aligned}$$

Since `OCR0A` is being used as TOP, the PWM is generated on `OC0B`.

2.2 General Timer Configuration

General timer functions — such as those needed by the sonar — are provided by `timer1`. To this end, the timer is configured in normal mode with no major modifications.

Whenever the timer overflows, an interrupt will be triggered that increments a counter, thus enabling it measure larger spans of time.

2.3 Sonar Configuration

The sonar uses two non-PWM pins — one output (trigger) and one input (echo) — along with basic timer functions to measure signal lengths.

A reading begins by sending a ten microsecond pulse to the trigger. The duration of the return signal recieved on echo is then measured and used to calculate the distance, using the equation:

$$\text{distance}\,[\text{cm}] = \text{echo duration}\,[\mu\text{s}] \times \frac{\text{centimeters}}{\text{microsecond}}$$

where:

$$\frac{\text{centimeters}}{\text{microsecond}} = 58$$

Readings should be taken at least 60 milliseconds apart to prevent noise from interfering with the measurement.

2.4 Drivetrain Configuration

Each drivetrain motor is hooked into the L298 motor driver with two pins. The Arduino itself is connected to the L298 using a three-wire interface (three pins per motor):

- One wire is a PWM that connects to the motor enable pin and controls the “power” of the motor.

- The other two (non-PWM) wires control the direction of the motor. When one pin is HIGH and the other LOW, the motor spins one way; swap which pin is HIGH and which is LOW to spin the motor the other way. Set both pins equal to each other to "brake" the motor.

`timer2` is used to generate the PWM signals needed for both motor enable wires. These are generated on pins `OC2A` and `OC2B`.

3 Software Implementation

For the time being, M-DOT is programmed with a very simple path-finding algorithm to follow a wall on its right side, as seen in Algorithm 1.

Algorithm 1: Simple wall-following algorithm for M-DOT.

```
1  function FollowWall is
2      idealDistance ← measure the distance to the wall
3      tolerance ← acceptable margin of error
4      while true do
5          currentDistance ← measure the distance to the wall
6          if currentDistance > idealDistance + tolerance then
7              drive curving towards the wall
8          else if currentDistance < idealDistance − tolerance then
9              drive curving away from the wall
10         else
11             drive straight ahead
```

This algorithm is designed so that there is not a hard-coded "target distance" that M-DOT tries to reach — rather, the robot will measure its initial distance from the wall on start-up and then try and maintain that distance throughout its travels.

4 Hardware Implementation

M-DOT is constructed in accordance with the initial design schematic seen in Figure 1 (which is also the final hardware schematic), with the pin assignments seen in Table 2.

Table 2: Pin assignments.

Component	Arduino Pin	ATmega328P Pin	Special Function
Servo control	5	`PD5`	`OC0B`
Sonar trigger	7	`PD7`	

Component	Arduino Pin	ATmega328P Pin	Special Function
Sonar echo	8	PB0	
Motor A enable	11	PB3	OC2A
Motor A +	2	PD2	
Motor A –	4	PD4	
Motor B enable	3	PD3	OC2B
Motor B +	10	PB2	
Motor B –	9	PB1	
Error report	13	PB5	Built-in LED

The sonar should be angled on its servo so that it is always facing a surface head-on. This is because the wave sent out by the sonar needs to bounce of the surface and return — if the surface is at an angle away from the sonar, the wave can bounce away and never return, as seen in Figure 2.

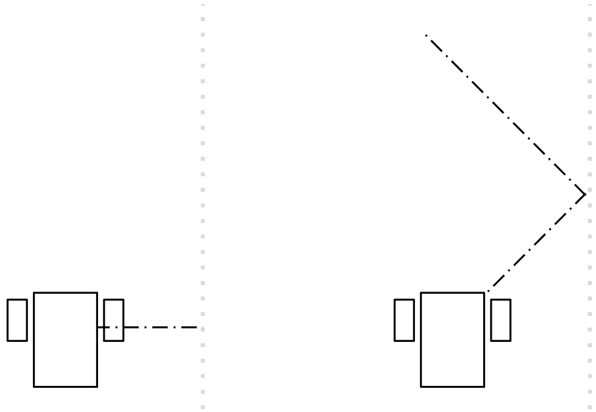

Figure 2: If the sonar is at too large an angle with the surface it is facing, the return signal could bounce away.

However, if the sonar is measuring along M-DOT's center of rotation then it may not detect the robot drifting off-course quick enough to prevent that angle from growing dangerous, as seen in Figure 3.

To this end, the sonar is kept at a 30 degree offset from the wall, as seen in Figure 4 — large enough to detect alterations to M-DOT's drive path early, but small enough to still receive a return wave that can measured with confidence.

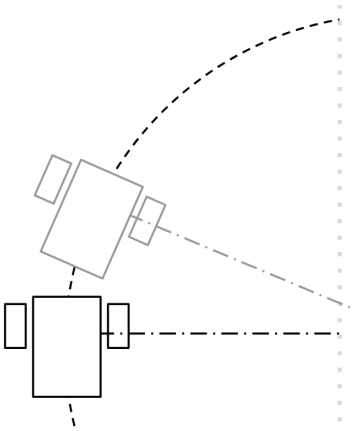

Figure 3: The measured distance does not differ much between the two points when measuring along the center of rotation.

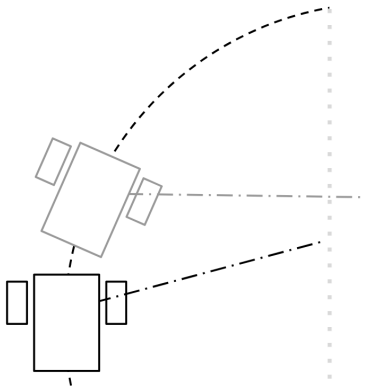

Figure 4: There is a much more noticeable difference between the measured distances when measuring at an offset.

5 Testing

5.1 Debugging

There were a few issues encountered during the testing process that had to be debugged, all of which concerned the sonar. First and foremost among them was the fact that the sonar could not get a reading from an angled surface, which was not considered during the preliminary design and had to be adjusted for when implementing and testing the project.

The other major hiccup was that the `getSonarDistance()` function (initially) did not force the 60 millisecond measurement cycle itself, so it was easy to forget and try to take back-to-back measurements.

Both of these issues where debugged by simply taking measurements with the sonar and printing them to the serial monitor in a loop while observing the output.

5.2 Methodology

Testing of M-DOT happened in multiple stages:

1. Each component was tested individually to confirm basic functionality, e.g. checking to make sure the servo spins and the sonar returns an accurate distance.
2. The drivetrain was tested for drift by driving it straight forward at max speed and observing its path.

3. The range of angles at which the sonar can get a valid reading was measured by taking readings at increasing angles until the output became unreliable.
4. The full program was then tested in various environments (carpeted floors vs. smooth floors, varying initial distances from the wall).

5.3 Results

Each component was able to perform its basic functionality without too much trouble, though the drivetrain as a whole did drift to the right by a not insignificant amount. This was corrected for in the code by reducing the power sent to the left motor by five percent at all times.

The sonar was able to take accurate measurements up to around a 45 degree offset from the opposite surface, at which point the measurements became unusable. This test is what determined the chosen offset of 30 degrees that is used in the final implementation.

M-DOT then proved its merit by successfully following walls in multiple environments, such as the smooth floors of Kiewit Hall and the carpeted floors of my home. One such demonstration can be seen in this video[1].

6 Q&A

6.1 Motor Driver

How are the motors wired up? Is it a two or three wire interface? The motors themselves are hooked into the L298 motor driver with two pins each. The Arduino is connected to the L298 using a three-wire interface (three pins per motor): one PWM pin specifying speed, and two pins controlling direction.

What will be done with the motor enable pins? The motor enable pins are connected to PWM signals and used to control the speed of the motors.

What pins/timer will create the PWM signals? The PWM signals will be generated on pins `OC2A` (Pin 11) and `OC2B` (Pin 3) using `timer2`.

How will the motors change direction? The direction of the motors will be controlled by the non-PWM pins in the three-wire interface — when one pin is HIGH and the other LOW, the motors spin forward; swap which pin is HIGH and which is LOW to spin backwards.

7 Conclusion

The drivetrain, servo, and sonar were all successfully configured to maneuver M-DOT and take accurate distance measurements — all without using the Arduino library. With these components the robot was able to demonstrate wall-following

[1] https://www.youtube.com/watch?v=LuJOR8e18cY

capabilities, which is a good first step to eventually navigating a more maze-like environment. M-DOT could likely fulfill its purpose with nothing but a more complicated path-finding algorithm, no hardware modifications required.

8 Documentation

No collaboration.

Resources used:

- Various datasheets for the ATmega328P, HC-SR04, L298, and SG90.
- Schematic for the Arduino UNO.
- Fritzing[2] for wiring diagrams.
- CSCE 336 resources like the slides and recorded videos.

[2] https://fritzing.org/