M-DOT

Embedded Systems // Project Report

C Wiebe

May 9, 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.

Additionally, M-DOT can be controlled remotely using an AX-1838HS IR receiver and Elegoo controller. This functionality is *not* used in maze navigation — that is totally autonomous.

2 Preliminary Design

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

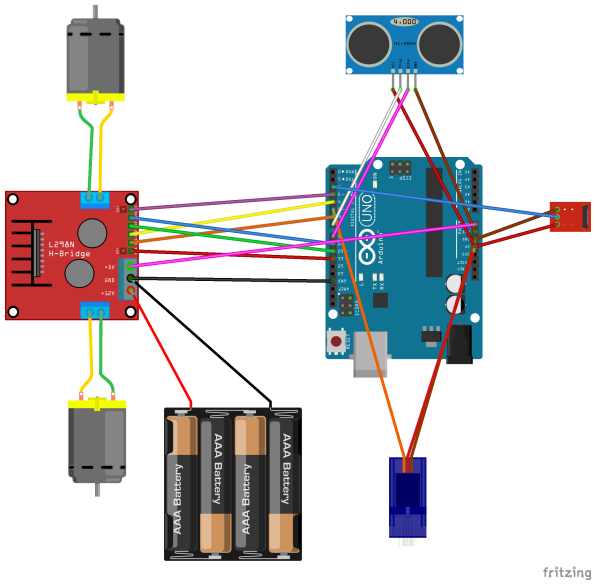

Figure 1: Initial design schematic.

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. The timer does, however, make use of two interrupts:

1. The timer overflow interrupt, `TOV1`, which will be used to increment a counter and track large spans of time.
2. The output compare A interrupt, `OCIE1A`, which will be used by the receiver to determine when a data packet "times out".

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`.

2.5 Receiver Configuration

The IR receiver will be hooked up to an external interrupt (which is idle HIGH) so as to prevent the need for constant polling. It will use `timer1` as a pseudo watchdog timer by enabling and disabling an output compare match interrupt.

The following process is used to receive data, which it stores in a global "data packet":

1. The external interrupt ISR is configured to fire on the falling edge, but that trigger will toggle between falling and rising edges each time it is called.
2. When a falling edge interrupt fires, the current timer count is stored in the data packet and the watchdog timer is disabled, if it isn't already.
3. When the rising edge interrupt fires, the timer count is cleared and the watchdog timer is started. If the watchdog timer terminates, the data packet is marked as complete and the process is reset.

Each data packet sent will begin with a start bit, followed by 32 data bits, and terminating with an end bit. Each bit will consist of a "low half-bit" and a "high half-bit". The low half of the data bits is used to separate the high halves, which will contain the actual data: A long time spent high is a logical 1, and a short time high is a logical 0. The time spent low is funtionally irrevelent. Recorded times for the various half-bits can be seen in Table 2.

Table 2: Recorded half-bit lengths.

Type	Time [ms]
Start low	9.336
Start high	4.456
Data X low	0.644
Data 0 high	0.504
Data 1 high	1.592
Stop low	0.656
Stop high	39.992

In order to properly identify 1's and 0's in the data segement, additional analysis was conducted, as seen in Table 3. The valid range displayed in the table is the range that will correctly identify 99.9999426697% of pulses (plus or minus five standard deviations).

Table 3: Average, standard deviation, and valid ranges for data half-bits.

Type	Avg [ms]	Stdev [μs]	Range [ms]
Data X low	0.661	16.60	0.578 to 0.744
Data 0 high	0.506	2.14	0.495 to 0.517
Data 1 high	1.592	9.90	1.542 to 1.641

With that, various buttons on the Elegoo IR controller were mapped to their corresponding hex codes, and can be seen in Table 4.

Table 4: Recognized buttons.

Button	Hex code
POWER	0x00FFA25D
VOL+	0x00FF629D
FUNC/STOP	0x00FFE21D
FAST BACK	0x00FF22DD
PAUSE	0x00FF02FD
FAST FORWARD	0x00FFC23D
DOWN	0x00FFE01F
VOL-	0x00FFA857
UP	0x00FF906F
EQ	0x00FF9867

3 Software Implementation

M-DOT is programmed with a conceptually simple algorithm — Algorithm 1 — which navigates by taking sonar measurements while stationary and moving in discrete bursts based on those measurements. The four "cases" for how it moves are:

1. *If an object is in the immediate drive path,* pivot 90 degrees away from the wall.
2. *Else if the wall is too close,* curve away from it.
3. *Else if the wall is very far (twice the ideal distance),* curve sharply towards the wall.
4. *Else,* curve towards the wall.

The algorithm is designed so that there is not a hard-coded "target distance" from the wall 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.

CheckWall and CheckFront (Algorithms 2 and 3) are two near-identical processes that simply return the difference between the ideal distance and the current following distance, as measured from both the side and the front.

Algorithm 1: M-DOT's primary process.

```
1  function Navigate is
2      idealDistance ← measure distance to wall
3      while true do
4          wallMargin ← CheckWall(idealDistance)
5          frontMargin ← CheckFront(idealDistance)
6          if frontMargin < 0 then
7              pivot 90 degrees from wall
8          else if wallMargin < 0 then
9              curve away from wall
10         else if wallMargin > idealDistance then
11             curve sharply towards wall
12         else
13             curve towards wall
14         brake the robot
```

Algorithm 2: Measures the distance to the wall.

```
1  function CheckWall(idealDistance) is
2      currentDistance ← measure distance to wall
3      return currentDistance − idealDistance
```

Algorithm 3: Measures the distance to the nearest obstacle.

```
1  function CheckFront(idealDistance) is
2      currentDistance ← measure distance to front
3      return currentDistance − idealDistance
```

M-DOT also features a much faster wall-following algorithm, Algorithm 4, that is an alternative to Navigate as a program "entry point". This method does not account for obstacles in the drive path, but neither does it constantly start and stop.

Algorithm 4: A 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
```

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 5.

Table 5: Pin assignments.

Component	Arduino Pin	ATmega328P Pin	Special Function
Servo control	5	`PD5`	`OC0B`
Sonar trigger	7	`PD7`	
Sonar echo	8	`PB0`	
Motor A enable	11	`PB3`	`OC2A`
Motor A +	6	`PD6`	
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
IR signal	2	`PD2`	`INT0`

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.

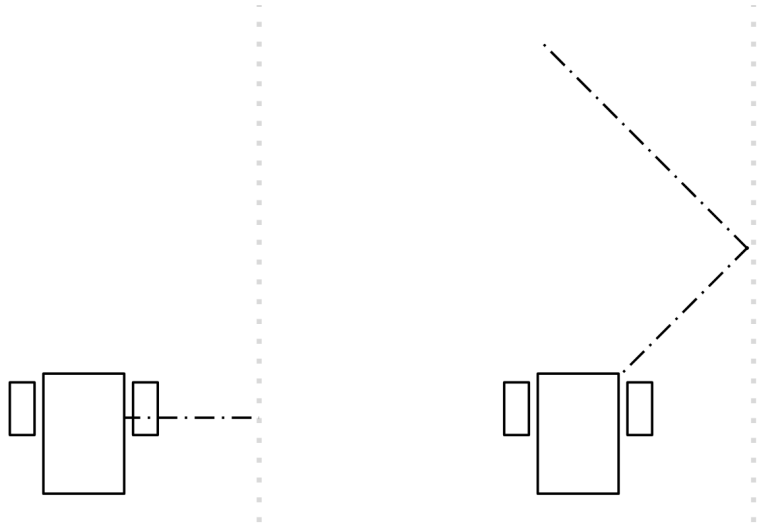

Figure 2: If the sonar is at too large an angle with the surface it is facing, the return signal could bounce away.

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.

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.

Other design challenges that were encountered (but that are not considered "bugs" per se) include:

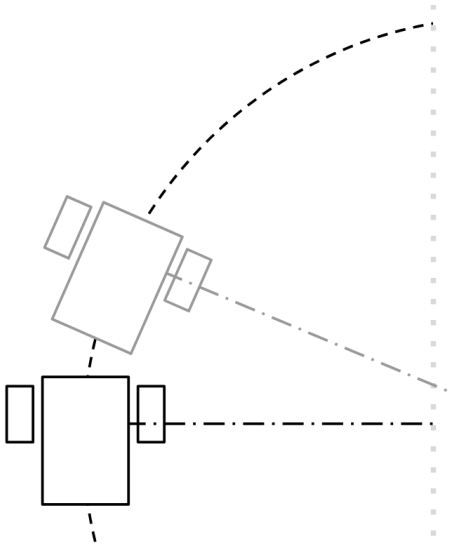

Figure 3: The measured distance does not differ much between the two points when measuring along the center of rotation.

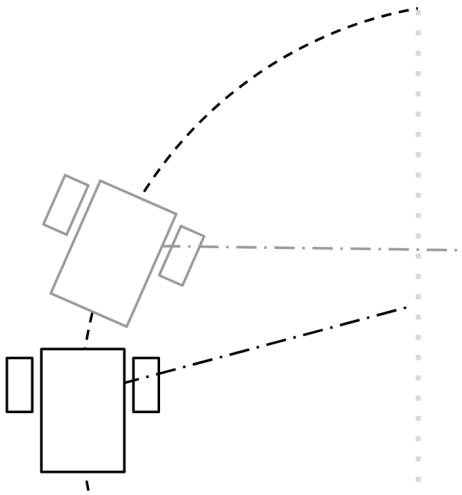

Figure 4: There is a much more noticeable difference between the measured distances when measuring at an offset.

- A relatively slow angular velocity on the servo, which forced ~500 millisecond measurement cycles when watching both the front and the side — 200ms per turn and 60ms per sonar reading. This made moving while measuring dangerous at any appreciable speed in the full navigation algorithm.
- Inaccurate sonar measurements while moving, which further encouraged using the sonar while stationary.
- Drivetrain drift that varied based on how hot the motors are, which discouraged using precise powers to drive the motors.

5.2 Testing 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, etc.).

5.3 Results

Each component was able to perform its basic functionality without 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 found to be able to take accurate measurements up to around a 30 degree offset from the opposite surface, at which point the measurements became unusable.

M-DOT then completed several tests verifying the functionality of its navigation algorithms, such as following a wall[1] with the simplified algorithm and navigating a small course[2] with the full algorithm.

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.

6.2 IR Receiver

How long does `timer1` take to roll over? Overflows take 262.14 milliseconds in my implementation.

How long does a timer count last with a prescaler of 64? Given the CPU frequency of 16 megahertz on the Arduino UNO, each count will last four microseconds.

How long are the pulses generated by the IR remote? Pulse lengths can be seen in Table 2.

What is the average/standard deviation of each pulse type? Statistical analysis on pulse lengths can be seen in Table 3.

What are the hex codes for the buttons on the IR remote? Button codes can be seen in Table 4.

[1] https://www.youtube.com/watch?v=LuJOR8e18cY
[2] https://www.youtube.com/watch?v=sb-tt6APl8w

7 Conclusion

The drivetrain, servo, sonar, and receiver were all successfully configured to maneuver M-DOT and take accurate distance measurements — without using the Arduino library. With these components, the robot was able to demonstrate wall-following and maze-navigation capabilities, culminating in two runs demonstrating full navigation: One with smoother, safer, and slower defaults[3], and one with a little more speed and a little more sway[4].

8 Documentation

No collaboration.

Resources used:

- Various datasheets for the ATmega328P, HC-SR04, L298, SG90, and AX-1883HS.
- Schematic for the Arduino UNO.
- Fritzing[5] for wiring diagrams.
- Class resources such as slides and recorded videos.

[3] https://www.youtube.com/watch?v=uleEuvvNTLQ
[4] https://www.youtube.com/watch?v=FaIQ-QEpM34
[5] https://fritzing.org