

Group Project – Rover Robot Course

Gio M, Simon M

Department of Engineering, Florida Gulf Coast University

CEN 4930 Embedded Programming

Prof. Paul Allen

12/04/2025

Contents

Overall Approach / Abstract	3
States	4
Main States:	4
Internal States:.....	5
Turning Method	6
Arduino Features Used	7
Stepper Motors:	7
Ultrasonic Sensor:.....	7
Servo:	7
Arduino Uno R3:	7
GitHub	8
Link:	8
Purpose:	8
Our Challenges	8

Overall Approach / Abstract

Our overall approach was to implement everything we learned throughout the year into our rover so that it would pass the course, but in an unconventional way. For example, instead of using the servo to turn the ultrasonic sensor towards the left or right wall, we would turn the whole rover to measure the distance between walls. Additionally, we used the servo to simulate a head shake, similar to the motion of the ultrasonic sensor, which would indicate that we had hit a wall. After three head shakes, the rover would then go into its turning state. While it may seem counterintuitive, we wanted our rover to stand out and complete the course. In fact, our rover almost completed the course; however, due to debris on the floor, the rover kept sliding to one side, hitting a wall, and getting stuck. Nevertheless, this document will cover the significant aspects of our project, including states, methods, logic, and challenges.

States

Main States:

`!Moving`: This is where the majority of the code lives. We took Prof Allen's example code and reused it for our own purposes. `!Moving` is where the logic of moving forward, turning, and reading ultrasonic lives. As the name implies, it is when the rover is not moving. We are not driving the motors, and we are reading the ultrasonic sensor to see the distance between walls. Based on the sensor's readings, our logic determines whether to proceed forward, turn left, or turn right. Once the decision is made, we set the target for the stepper motors and then set `moving = true`, which then puts us into the moving state.

`Moving`: this is where we move the rover. Once `moving` state is true, we do nothing other than run the motors until we reach the target we set in `!Moving`. Once we reach the target we set `moving` back to false, and we got back to our main logic to see what to do next.

`FORWARD`: This state is one of the main states that moves the rover forward. We start in forward state when in `!Moving`. All this state does is read the ultrasonic sensor. We measure how far we are from the front wall and keep moving forward until we are within 30 cm of the front wall. Once we get within 30cm of the front wall, we then do 3 servo head shakes to indicate that we truly are stopped (it's not some error), then we go into turning state.

`TURNING`: This is one of the main states that determines what direction we need to turn in by comparing the distance between the left and right walls, then proceeding in the direction where the wall is the farthest. Nevertheless, we enter this state after the 3 head shakes. Once in the state we turn left 90 degrees and read the left wall distance. We then make a 180 degree turn right and read the right wall. We then make a 90 degree left turn to face the rover back in the

original direction. After that, we compare the distances between the left and right walls.

Whichever has the greatest distance, then we know we need to go in that direction, so we turn in that direction and return to the forward class. This will complete the logic loop, because once back in forward, the rover will then go to the left wall. Once it gets within 30 cm of the left wall, it will go back to our turning logic to see where to turn again next.

Internal States:

readLeftWall & readRightWall: these states tell us whether or not we've got the distance between the left and right wall.

movedLeft, movedRight, & movedBack: these states tell us whether we moved left, right, and back center when we were measuring the distance between the walls.

Turning Method

Our turning method is a simple function that we created based off prof. Allen's Motor class methods. To turn left, we call turnLeft(); A function we created, which asks for a speed and degree you want to turn. In motor, we saw that the Motor::forward and Motor::Reverse needed a speed and scale. We know that 1.09 scale is a full turn, so we know that that's a 360 turn if one stepper motor goes forward and the other reverse. Then for 180, we just divide the scale in half. Nevertheless, our function takes in a speed and degree turn (which is a scale) below. So, if you want to turn left at 90 degrees at regular speed you would do: turnLeft(regSpeed, _90);

float _360 = 1.09f; float _180 = 0.545f; float _90 = 0.27f; float _45 = 0.13625;

```
void turnLeft(float turnSpeed, floatdeg) {  
    leftMotor.reverse(turnSpeed,deg);  
    rightMotor.forward(turnSpeed,deg);  
}
```

```
void turnRight(float turnSpeed, float deg){
```

```
    leftMotor.forward(turnSpeed, deg);  
    rightMotor.reverse(turnSpeed,deg);  
}
```

Arduino Features Used

Stepper Motors:

Two motors in the front of the rover are used to propel it forward

Ultrasonic Sensor:

Used to determine the distance away from the walls of the track

Servo:

Used to turn the sensor a few times showing the trigger to the turn state

Arduino Uno R3:

We used an Arduino Uno R3 for this project

GitHub

Link:

The link to our repo is here: <https://github.com/GioMonci/Rover>

Purpose:

The purpose of the repo is so that we were able to work together, go back to old versions if we messed up, and to have something nice to put on our resumes.

Our Challenges

Since the ultrasonic sensor requires a nearly perfectly perpendicular wall to receive a signal back, we had issues when the rover was slightly off angle. We encountered problems when the rover was slightly off-angle. Another challenge we faced was when ultrasonic sensor relies on an almost perfectly perpendicular wall to get a signal back, we had issues when the rover was slightly off-angle trying to determine why our code would cause the rover to go into the “FORWARD” state but fail going into the “TURN” state even though the trigger was correct. The solution was to add curly brackets around each case in the switch statement that handled state switching. Since we were declaring variables in the “TURN” case, we needed to use curly brackets when we wouldn’t need to if we weren’t declaring variables inside the case.