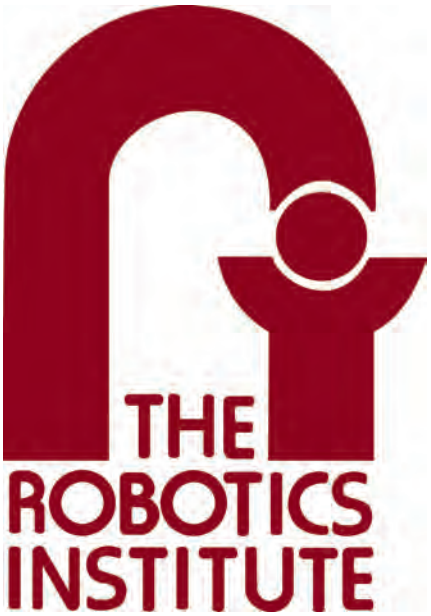

MRSD Project Course

Team I – AIce

Autonomous Zamboni Convoy

Individual Lab Report 1

Team

Rathin Shah

Nick Carcione

Yilin Cai

Jiayi Qiu

Kelvin Shen

Author

Yilin Cai

Feb 6, 2022

Contents

1 Individual Progress **1**
- 1.1 Sensor & Motor Lab . . . 1
  - 1.1.1 Sensor . . . 1
  - 1.1.2 Motor . . . 1
- 1.2 Circuit and Code . . . 2
- 1.3 MRSD Project . . . 3

2 Challenges **3**
- 2.1 Sensor & Motor Lab . . . 3
- 2.2 MRSD Project . . . 4

3 Teamwork **5**
- 3.1 Sensor & Motor Lab . . . 5
- 3.2 MRSD Project . . . 6

4 Plans **7**

5 Sensor & Motor Control Quiz **7**

A Appendix - code **9**

1 Individual Progress

1.1 Sensor & Motor Lab

My responsibilities for this sensor and motor lab were to get the ultrasonic sensor input, use the input to control the stepper motor, develop the average filter and to integrate the electrical circuit.

1.1.1 Sensor

The ultrasonic sensor I used in this lab is the HC-SR04 ultrasonic distance sensor. This economical sensor provides 2 cm to 400 cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm.

The two main pins on the sensor is the Trig (Trigger) and Echo. The Trig pin is used to trigger the ultrasonic sound pulses. The Echo pin produces a pulse when the reflected signal is received. The length of the pulse is proportional to the time it took for the transmitted signal to be detected. The width of the received pulse is then used to calculate the distance to the reflected object. We can calculate the range through the time interval between sending trigger signal and receiving echo signal as: $uS/58 = centimeters$; or: $range = high\ level\ time * velocity\ (340m/s)/2$.

Since the reading from the ultrasonic sensor is noisy, an averaging filter is develop the filter the measured distance. The averaging filter stores the most recent five recorded measurements and returns their average, which helps remove the noise. Here I set a bounding constraint of 0 - 100 cm considering the possible reaching range of the sensor. Any reading outside this range is bounded to the max or minium value.

1.1.2 Motor

Here, the motor to be controlled by the ultrasonic sensor is the SM-42BYG011-25 stepper motor. It is a 2 phase stepper motor with a step angle of 1.8 degree, which means in each step the motor will rotate 1.8 degree. So the steps per revolution is 200. After Arduino receives the ultrasonic sensor reading, assusimg the motor is not being controlled by GUI, commands are given to the stepper motor. The distance (0-100cm) measured by ultrasonic sensor is mapped to the position (0-200 steps) of the stepper motor in one revolution and is saved as desired position. The motor's current position is initialized as 0 and keep updated during the control loop. If the desired position is larger than the current one the stepper motor's direction position is set to high, otherwise it is low to make motor rotate reversely. When the stepper motor is moving from current position to the desired position, a 1000 microseconds delay is set between each step. Once the desired position is reached, the current position will be updated to it.

The stepper motor is driven by a Pololu DRV8825 motor driver. The DRV8825 is a microstepping driver for controlling bipolar stepper motors which have a built-in translator for easy operation. Thus, we can control the stepper motor with just 2 pins from our controller. The DIR pin will control the rotation direction and the STEP pin will control the steps. Before actually using the stepper motor, I tried to limit the maximum amount of current flowing through the stepper coils and prevent it from exceeding the motor's rated current. From the data sheet, the rated current of the stepper motor is 330 mA under 12 V. Here I followed two methods below to set the current limit by adjusting the small trimmer potentiometer on the driver.

(1) I first put the driver into full-step mode by leaving the three microstep selection pins disconnected and hold the motor at a fixed position by not clocking the STEP input (setup as shown in Figure 2). Then I measure the voltage (Vref) on the metal trimmer pot itself while adjusting it. Adjust the Vref voltage using the formula: $Current\ Limit = Vref \times 2$. Since my motor is rated for 330 mA, I adjust the reference voltage to around 0.165V.

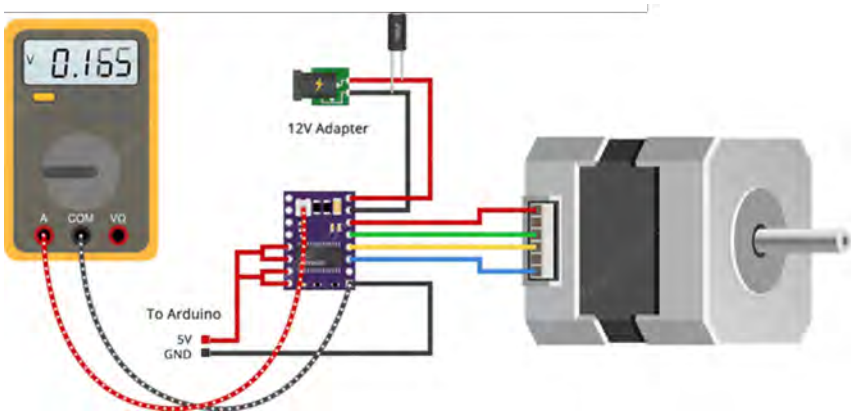

Figure 1: Circuit setup for limiting stepper motor current by adjusting voltage.

(2) Similar to previous method, the driver is put into full-step mode and held at a fixed position. Then the ammeter is placed in series with one of the coils on stepper motor and measure the actual current flowing. By adjusting the limit potentiometer, the current is set to reach the rated value. However, in actual attempt the max current value can only reaches around 280 mA.

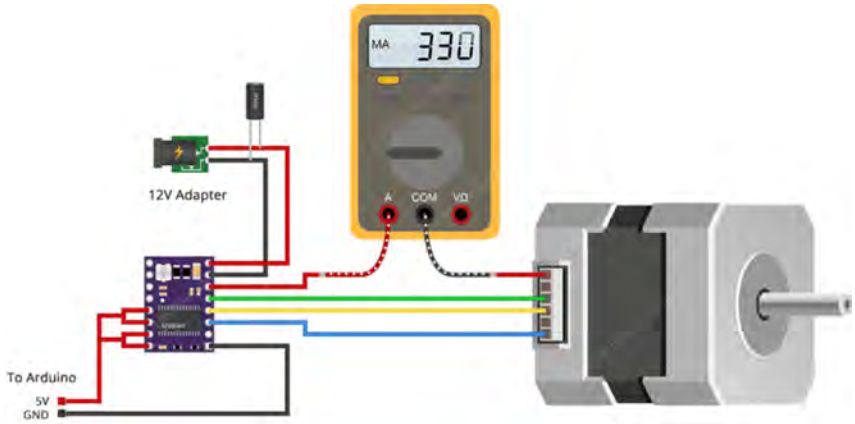

Figure 2: Circuit setup for limiting stepper motor current by adjusting current.

1.2 Circuit and Code

The circuit integrating both the ultrasonic sensor and the stepper motor is shown in figure 3. A switch button is also included considering that the motor need also to be controlled by a force sensor and it needs to be switched between these two. The code for the microcontroller is attached in Appendix A. It contains the code for reading ultrasonic sensor input and writing the mapped degree into the motor. Also, the moving average filter code is implemented in the code as well. The code for controlling stepper motor by Ultrasonic sensor is given in Appendix A.

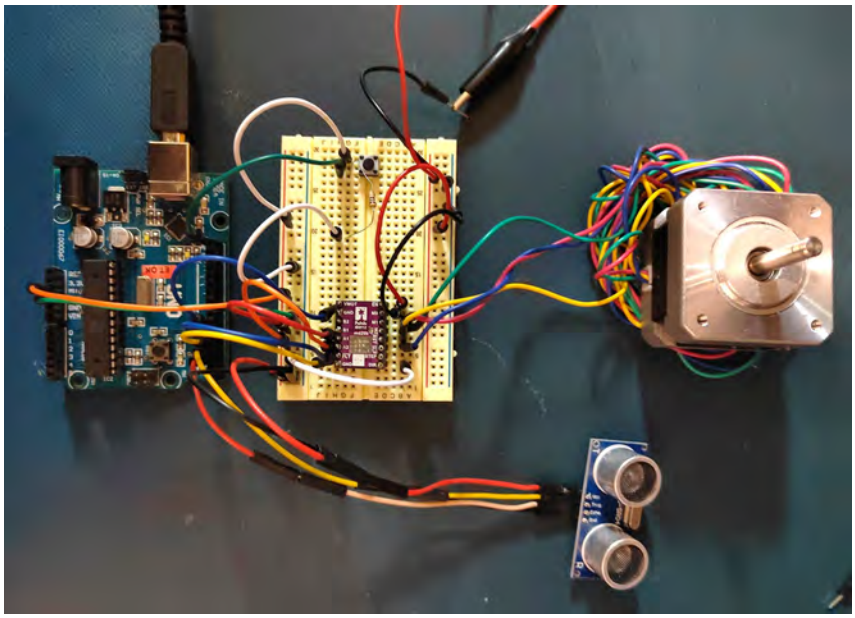

Figure 3: Circuit setup for controlling stepper motor using .

1.3 MRSD Project

Starting from this semester, my responsibility for the Autonomous Zamboni Convoy project mainly focus on setting up and maintenance of the simulation environment. Specifically, I developed the URDF file for the Zamboni vehicle, which had an Ackermann steering mechanism. To better organize the model description, I used XACRO file to build the vehicle model. For the follower Zamboni, in addition to the vehicle model itself, I added IMU, LiDAR and camera sensor to it using Gazebo plugins. For better visualization, I built a detailed mesh file for the Zamboni, which was exported from the Solidworks CAD model provided by the sponsor. However, the CAD model includes very detailed component of the vehicle, thus the mesh file was extremely complex. I simplified the mesh model in MeshLab and add color to the surface of the mesh file to make it closer to the actual vehicle appearance. Two Zamboni vehicles were set up in the ice rink environment.

Moreover, I set up the controller interface in gazebo, including the rear wheel velocity and front wheel steering angle controller. This controller is actually within the frame of gazebo sending the controlling command. I realized a keyboard control for both the leader and follower Zamboni to make them run individually in the simulation environment. I also set the ROS tf information and visualize the vehicles in RVIZ. I also wrote a odometer based on encoder on wheels (wheel velocity published by gazebo). Figure 4 show the ROS node map of the current simulation setup.

2 Challenges

2.1 Sensor & Motor Lab

The major challenge for the sensor and motor lab lies in the current limit setting for the stepper motor. At first, I tried to set the current limit by only reading the current going from the DRV8825 drive to the motor according to method (2) in section 1.1.2. However, it took me a lot of time to find that our multimeter was unable to get the current reading, specifically, we found it was the fuse who broken.

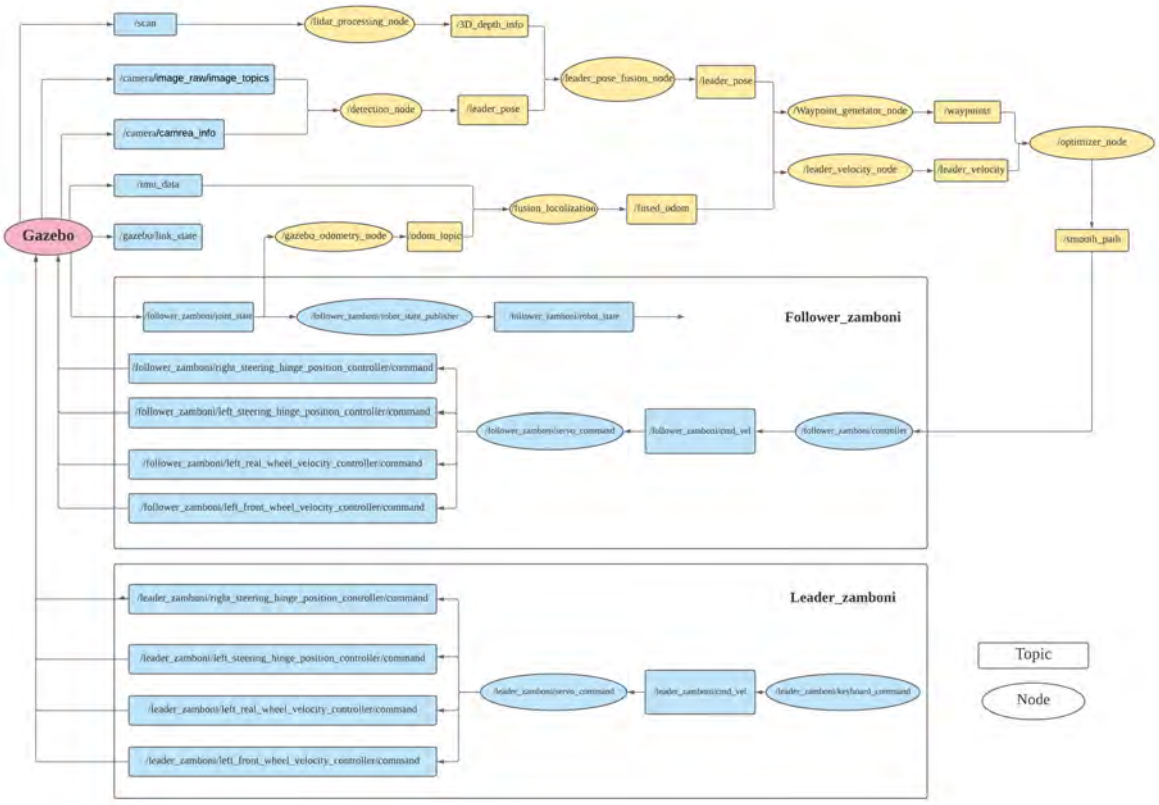

Figure 4: ROS node map for the autonomous Zamboni simulation.

After fixing the multimeter, we measured the current going through the stepper when it was set to full step mode. We found the default current can only reach around 280 mA and by trying to adjusting the limit potentiometer, the current won't exceed 280 mA and won't reach the rated value. Since I turned the potentiometer on the driver too much, I believe it somehow got broken due to the current overload and not drive the motor anymore. After getting a new driver, I found without any adjustment on current limit, the motor would get very hot after being powered for a certain time. So I follow the method (1) in section 1.1.2 to set the current limit by measuring the Vref and then make sure the actual current going through the motor is within the rated range. Following this step, the motor's performance returned to normal.

2.2 MRSD Project

The first major challenge I faced for the Autonomous Zamboni Convoy project is building the URDF file for the Zamboni vehicle. The sponsor provided a very detailed CAD model with over 1,000 parts describing all the details of a car. However, I combined all the parts into only five: the four wheels and a car body, and exported them as mesh files. Due to the complexity of the original mesh file, gazebo will get stuck when loading the vehicle. Thus I try to use MeshLab to simplify the faces and vectors on of the mesh file, deleting unnecessary detail while keeping the major outer appearance. Also for better visualisation, I learned to use Blender to change the scale, center position, and more importantly, the color of each face. It was very time consuming to preserve the details of appearance while keeping the model as simple as possible. Also in URDF file, to align the frame of them also requires much time to try and modify to finally make

the vehicle looks more realistic.

The second major challenge is about spawning multiple robots in the gazebo simulation. In our current setup, each follower and leader vehicle should have separate URDF description, controller and configurations. To distinguish them, I had to set separate namespace within which the ROS topics for different vehicles can be published or subscribed separately. So it took me much time to setting up the namespace so that both the tf tree is organized and the configuration of one vehicle wouldn't influence another.

3 Teamwork

3.1 Sensor & Motor Lab

Kelvin Shen is in charge of GUI design. His contribution includes:

- Created a ROS publisher and subscriber node in Arduino to send or receive data.
- Created a serial interface between ROS and the Arduino using rosserial.
- Created a URDF for motor that can control the motor outputs through $joint\_states$ publisher in GUI.
- Created a visualization in RVIZ to show the GUI motor control output.
- Created a RQT GUI that contained RVIZ and plots that displayed all the sensor outputs in real time.

Rathin Shah is in charge of the Potentionmeter sensor and the servo motor. His contribution includes:

- Wired potentiometer and servo motor to Arduino.
- Mapped potentiometer sensor analog output to servo motor input to control its position.
- Added a button to switch between GUI control and sensor control.
- Helped combine code together and debugged electrical issues.

Nick Carcione is in charge of the IR sensor and the DC motor. His contribution includes:

- Wired Sharp IR Proximity Sensor to Arduino.
- Developed a transfer function that mapped sensor output voltage to measured distance.
- Implemented moving average filter to reduce input noise.
- Wired DC motor and H-bridge to Arduino. Developed PID controls for velocity and position control of the motor.
- Wrote button debouncing code to switch between position and velocity control.
- Interfaced sensor with DC motor (velocity control)

Yilin Cai is in charge of the Ultrasonic sensor and the stepper motor. His contribution includes:

- Soldered driver board for stepper motor, wired Ultrasonic sensor and wired the stepper motor to Arduino.
- Setting current limit for stepper motor by adjusting the driver.
- Developed code for Ultrasonic sensor data reading and stepper motor position control based on the distance sensed by the sensor.
- Integrated and wired all sensors and motors together.

Jiayi Qiu is in charge of the FlexiForce sensor and the stepper motor. Her contribution includes:

- Wired FlexiForce sensor with Arduino.
- Mapped FlexiForce sensor outputs to stepper motor steps.
- Integrated code for FlexiForce sensor, Ultrasonic sensor and stepper motor.
- Wrote button code with debouncing to switch between these two sensors to control the stepper motor.
- Wrote a moving average filter to reduce input noise.

3.2 MRSD Project

Kelvin Shen is in charge of perception and recognition. His contribution includes:

- Learned cv_bridge that communicates between ROS camera topic and OpenCV.
- Generated a mesh file of a board of ArUco markers with appropriate size.
- Tested the ArUco wall with Zamboni model inside the Gazebo environment.
- Apply OpenCV libraries of ArUco marker detection to get leader pose.

Rathin Shah is in charge of the controller development. His contribution includes:

- Developed the Pure Pursuit Controller on Simulink for Ackermann Geometry.
- Developed package for fusing IMU + Wheel Odometry for zamboni using ROS.
- Developed the SIMULINK-ROS Interface.

Nick Carcione is in charge of the DBW hardware and follower localization. His contribution includes:

- Researched methods and packages for fusing wheel encoder and IMU data to obtain accurate velocity estimation of follower.
- Looked into DBW conversion hardware
- Identified hardware necessary for conversion of Zamboni to DBW.

Yilin Cai is in charge of the simulation setup. His contribution includes:

- Set up and maintenance of the simulation environment URDF (XACRO) file development the Ackermann steering Zamboni with sensor and controller plugins.
- Simplified and colored mesh file for better visualization in URDF.
- Command vehicle motion command with keyboard teleoperation.
- Spawned multi-robot in Gazebo and individually control with keyboard teleoperation.
- Setup tf_tree odometer and frame, and visualized vehicles in Rviz.

Jiayi Qiu is in charge of the simulation environment setup and leader's estimation. Her contribution includes:

- Investigated vehicle simulation in Gazebo.
- Helped to simulate the Zamboni.
- Built the ice rink simulation environment.

4 Plans

The next step of my MRSD project works will still focus on the simulation part. First, I will work on the camera and LiDAR data visualization in Rviz. In addition, I will continue refinement of simulation details, like physical parameter in URDF file and gazebo controller plugins. I will also modify the mesh file to equip the leader Zamboni with a ArTag figure at the back of it, which will create the interface for detection. Moreover, I will clarify the frame definition, including map and odometer frame, which will enable the localization.

5 Sensor & Motor Control Quiz

ADXL335 accelerometer

- The range is ±3 g (minimum), ±3.6 g (typical).
- The dynamic range is 6g (minimum), 7.2g (typical).
- The purpose of the capacitor is to reduce the input voltage noise and maintain the voltage at 3V. It can filter out high-frequency noises from the power source. The capacitor will perform the role as a battery and discharge the circuit when the voltage drops below 3V. I then gets charged when the voltage increases to make the voltage stable.
- $V_out = 1.5V + \frac{0.3V}{g} \times a$ where a is the acceleration with unit in gs.
- $0.3\% \times 7.2g = 0.0216g$
- Bandwidths is from 0.5 Hz to 1600 Hz for the X and Y axes.
- $RMS = Noise\ Density \times \sqrt{BW \times 1.6} = 150\frac{\mu g}{\sqrt{Hz}} \times \sqrt{25Hz \times 1.6} = 948.68\mu g$
- We can fist palce the accelerometer on the static surface and measure the output voltage over a long time duration. Then, calculating the root mean square for the reading will help determine the noise.

Signal conditioning
Filtering

- First, if the moving average filter has a large window size, the output could have a significant lag. Second, a moving average filter cannot effectively filter the individual occasional errors (like a occasional huge number).
- First, the median filter is computationally expensive. Second, sustained increases or decreases in the signal will delay the response.

Opamps

- Your uncalibrated sensor has a range of -1.5 to 1.0V
  - V_2 is the input voltage and V_1 is the reference voltage.

–

$$V_{out} = (V_2 - V_1\frac{R_f}{R_i}) + V)2$$

$$0 = -1.5 \times (1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i}$$

$$5 = 1 \times (1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i}$$

$$\frac{R_f}{R_i} = 1, V_{ref} = -3V$$

- Your uncalibrated sensor has a range of -2.5 to 2.5V
  - It is impossible to find a solution.
  - If V_1 is the input voltage and V_2 is the reference voltage.

$$0 = 2.5 \times (1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i}$$

$$5 = 2.5 \times (1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i}$$

    However, there is no solution for this situation.
    If V_2 is the input voltage and V_1 is the reference voltage.

$$0 = -2.5 \times (1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i}$$

$$5 = 2.5 \times (1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i}$$

$$\frac{R_f}{R_i} = 0$$

    However, this situation is impossible to calibrate.

Control

- Form a digital input for each of the PID
  - The P term is the difference between the desired position and the current position.
  - The I term can be calculated by keeping a running sum of the P terms at each time step.
  - The D term is adding current position error with the previous position error, then divided by the timestep.
- If the system is sluggish, I would want turn up the P terms. They should reduce rise time and make the system react faster.
- If the system then has steady-state error, I will turn up the I term because it compensate for the accumulated errors.
- If the system then has overshoot, the D term should be increased to reduces overshoot.

A Appendix - code

```
#define window_size 5

// Ultrasonic Sensor variables
const int TrigPin = 7;
const int EchoPin = 13;
float distanceCm;
int duration;

// Stepper Motor variables
const int stepperEnable = 6;
const int stepperStep = 5;
const int stepperDir = 4;
const int stepsPerRevolution =200;
int currentStep;

// Filter function
int sensorReadings[window_size];
int sum = 0;
int index = 0;
int RunningAverage_filter(int reading){
  sum -= sensorReadings[index];
  sensorReadings[index] = reading;
  sum += sensorReadings[index];
  index = (index+1) % window_size;
  return sum/window_size;
}

// Ultrasonic Sensor function
float Ultrasonic(){
  digitalWrite(TrigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);

  duration = pulseIn(EchoPin, HIGH);
  int filtered_duration = RunningAverage_filter(duration);
  distanceCm = filtered_duration / 58.0;
  distanceCm = (int(distanceCm * 100.0)) / 100.0;

  if(distanceCm > 100.0){//bounds check
    distanceCm = 100.0;
  }
  if(distanceCm < 0){
    distanceCm = 0;
  }
```

```
  Serial.print("Distance:");
  Serial.print(distanceCm);
  Serial.print("cm");
  Serial.println();
  delay(10);
  return distanceCm;
}

// Stepper Motor function
void stepper(int desiredStep){
  if(desiredStep > currentStep) digitalWrite(stepperDir,HIGH);
  else digitalWrite(stepperDir,LOW);

  for(int x = 0; x < abs(desiredStep-currentStep); x++){
  digitalWrite(stepperStep, HIGH);
  delayMicroseconds(1000);
  digitalWrite(stepperStep, LOW);
  delayMicroseconds(1000);
  }
  currentStep = desiredStep;
//return;
}


void setup(){
  Serial.begin(9600);
  pinMode(TrigPin, OUTPUT);
  pinMode(EchoPin, INPUT);

  pinMode(stepperEnable,OUTPUT); // Enable
  pinMode(stepperStep,OUTPUT); // Step
  pinMode(stepperDir,OUTPUT); // Dir
  digitalWrite(stepperEnable,LOW); // Set Enable low
}

void loop(){

  // Ultrasonic Sensor
  float distanceCm = Ultrasonic();
  int desiredStep = round(map(distanceCm, 0, 100, 0,
     stepsPerRevolution));
  stepper(desiredStep);
  Serial.print("CurrentStep:");
  Serial.println(currentStep);
  Serial.print("desiredStep:");
  Serial.println(desiredStep);
  delay(100);
}
```