Robotics Lecture Series

Navigation to the future

Contents

What is this lecture?

Welcome to the first entry of the Robotics Lecture Series. In this series, we will explore particular areas of robotics, look at current research on the topic, and embark on a project that you can make at home.

Our first session is all about collision avoidance and robots navigating in unfamiliar environments. We are going to learn how robots navigate 2-D spaces independently without crashing into one another. Then, we will look at current research on the topic and how it furthers the field of robotics. Last but not least, we will explain how you can build, wire, and program a small car that navigates a maze.

All the information on our next session, the recordings of this session, and extra resources can be found on the KSU Robotics Lecture Series website[1].

Who are we?

We are a small group from the Robotics and Mechatronics Department at Kennesaw State University (KSU). We came here to talk about robotics and how you can learn how to do it at home.

Dr. Tanveer is a professor and researcher at KSU. He received his Ph.D. in Robotics and Autonomous Systems from the University of Genova, Italy. He currently serves as an editorial board member for the Journal of Robotics and a reviewer for several international journals and conferences. His areas of expertise include electronic system designing, robotics, and control systems.

Cecilia McDaniel is the administrative assistant for the Robotics and Mechatronics Department at KSU. She will be your point of contact for questions about the department in general and how you can join us.

Sofia Gomezllata is a robotics and mechatronics major at KSU. She worked on creating the student project, and as such, any questions about the topic can be directed towards her.

Contact Us[2]

Definitions

Robotics:

The branch of technology that deals with autonomous machines.

Autonomous:

An autonomous robot is one that can perform without external influence.

Mechatronics:

Technology that combines electronics and mechanical engineering.

Robotics vs Mechatronics:

While robotics is a part of mechatronics, not all mechatronic systems are robots. For example, a traffic light with a button for pedestrians mixes mechanical components, electronics and code, but cannot operate without human input; making it a mechatronic system but not a robot.

[1] https://engineering.kennesaw.edu/robotics-mechatronics/students/Events.php
[2] Link to page 16, where you can find all our contact information.

What are path planning and obstacle avoidance, and why should I care about them?

Path planning is a procedure that aims to find the optimal way to get from the source to the destination. However, without a healthy dose of obstacle avoidance, path planning can be quite dangerous. For example, imagine your grandmother has a new automatic wheelchair. She could go anywhere by manually driving the wheelchair, or she could input a location and let the path planning algorithms find the best way to get her there. However, without obstacle avoidance, your grandma would end up crashing into you, chairs out of place, and any obstacle that was not part of the path planning map.

Path planning and obstacle avoidance are fundamental for any mobile robot. Self-driving cars, automated drones, and delivery bots are only a few examples of the many applications for path planning and obstacle avoidance. Can you think of another application?

Mobile Robots: Introduction

One of the main objectives of robotics is to create fully autonomous machines. To accomplish this with mobile robotics (robots that have the ability to move), the robot must be able to answer these three questions: where am I, where am I going, and how do I get there.

To answers these questions, the robot must be able to have a model of the environment, perceive and analyze the said environment, find its position/situation within the environment, and plan to execute the movement. This can be accomplished by making sure to include the following components:

- **Locomoting** (the ability to move from one place to another): For our mobile robot to be mobile, we need to give it a way to move independently. This is often accomplished via wheels or legs, but these are no the only solutions. Can you think of other methods to make your robot move?
- **Sensing:** For our robot to be able to avoid the obstacle, it needs to have the ability to sense its surroundings. This is often accomplished via cameras (similar to how some humans use vision to navigate space) or ultrasonic sensors (similar to how bats use sound to navigate space). Can you think of another type of sensor that can be used to navigate space?
- **Reasoning:** For mobile robots, our best option is to implement reactive controllers. Reactive control strategies focus on our robot reactive to individual events. For example, we can use layers in our architecture to achieve obstacle avoidance. On a basic level, our robot wants to move, then on a higher level our robot wants to avoid any obstacles it detects, and on the highest level, our robot wants to get to a specific location. Can you think of how you would write an algorithm for this?
- **Communicating:** While robots need to be autonomous to be considered robots, communication is still an important area of robotics. Remote controls can provide our robot with the initial objective, and the robot can use a text interface to communicate back with us. Can you think of other communication methods our robot could use?

Inspiration from Nature: Bats

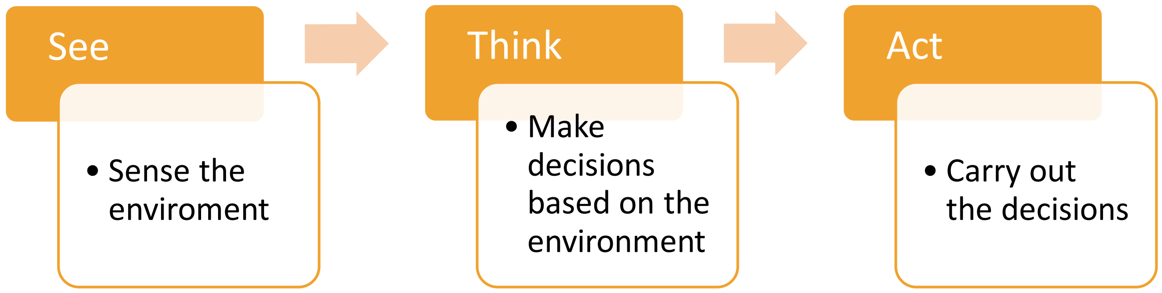
Bats (the animal, not the baseball equipment) use echolocation to navigate and find insects. Bats produce ultrasonic soundwaves—humans are unable to hear at those frequencies—which bounce off objects and return to the bats' ears. Bats' ears are trained to recognize their own unique calls and use the difference between sending the wave and when it came back to create a mental map of where obstacles are.

While the mental map they produce is not as detailed as the one humans get from vision, it is good enough to navigate and is easier to replicate using sensors. Can you think of a reason why photos are not the easiest type of data to work with?

An example of the bat-approach to navigation is sonar-based self-flying drones. Using a mixture of bioinspired sonars, robotics, and machine learning, you get a flying robot that can autonomously navigate three-dimensional spaces.

Mobile Robots: The see-think-act cycle

In mobile robotics, we will want our robots to follow the see-think-act cycle to build our algorithms around them.


See
• Sense the enviroment
Think
• Make decisions based on the environment
Act
• Carry out the decisions

Path Planning

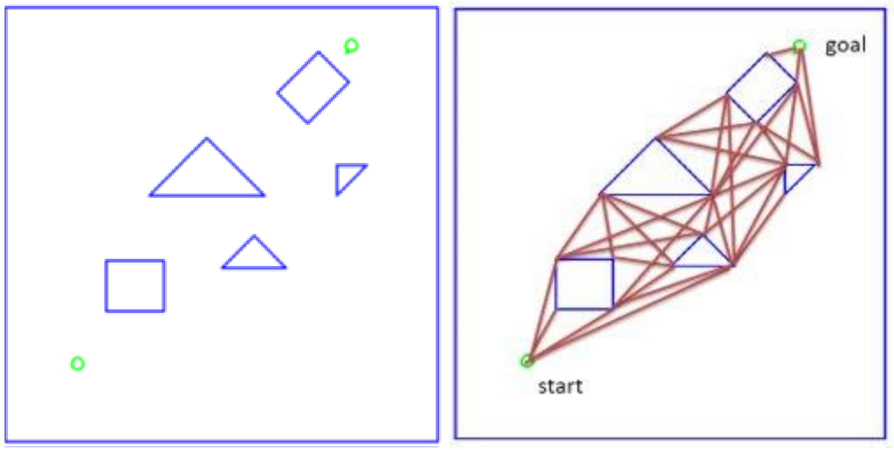
Path planning is a computational problem to find a valid configuration sequence that moves the object from the source to the destination.

The first step of any path-planning system is to transform this possibly continuous environmental model into a discrete map suitable for the chosen path-planning algorithm.

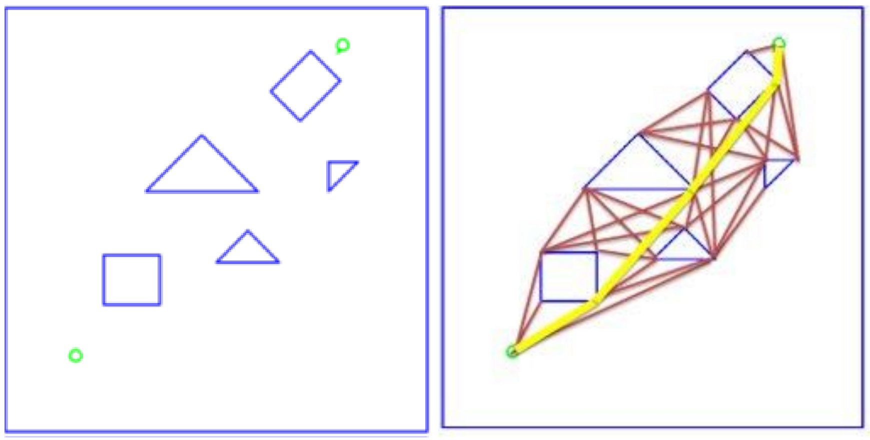
This can be done in multiple ways, but we can focus on visibility graphs and Voronoi graphs.

Visibility Graph

The start, goal, and vertices are graph notes. The edges are “visible” connections between nodes, including obstacle edges.

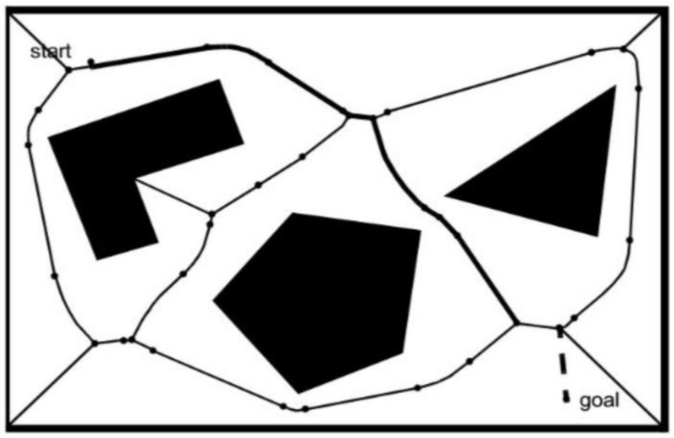


Then, we search for the shortest path via the visible verticles.



Voronoi Diagram

A Voronoi Diagram maximizes the distance between the robot and the objects by considering the mean (average) of the obstacle edges.



Obstacle Avoidance

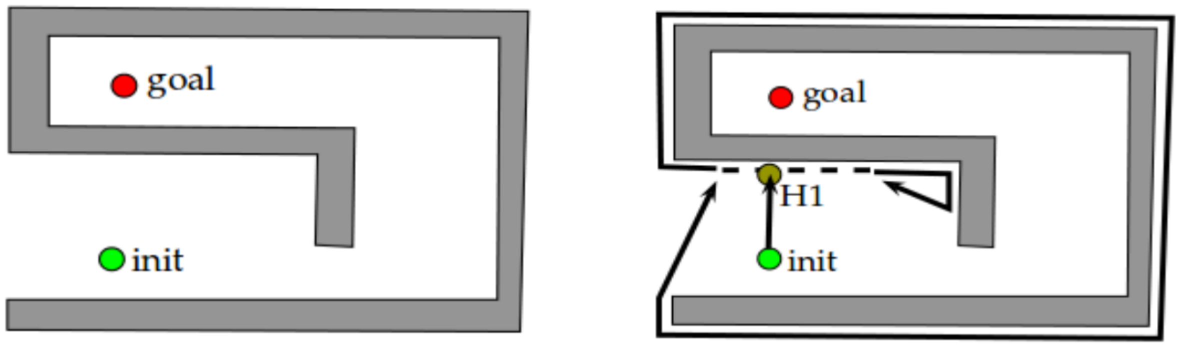
This type of robot gets the information from its surrounding area through mounted sensors. Examples of sensors include ultrasonic sensors, infrared sensors, and cameras.

Ultrasonic sensors are the most suitable for obstacle detection thanks to their low cost and their high ranging capability. Ultrasonic sensors are used from advanced self-driving cars to small student projects such as the one you are going to do.

An algorithm then uses the information gotten from the sensors to avoid obstacles. We are going to look at two different algorithms: the BUG algorithm and the bubble band technique.

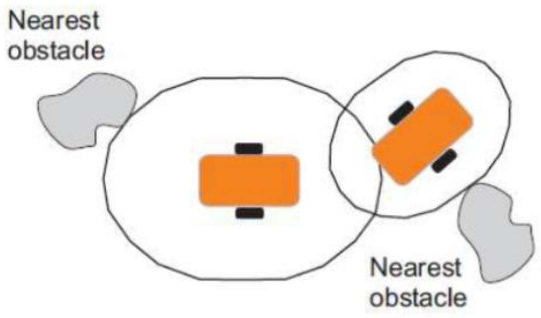
The BUG Algorithms

The simples obstacle avoidance algorithm. When an obstacle is encountered, the robot fully circles the object to find the point with the shortest distance to the goal.



The Bubble Band Technique

With this concept, a band of such bubbles can be used to plan a path between a starting point and a goal. This technique is more a problem of offline path planning than one of obstacle avoidance.



Applications

There are hundreds of applications for path planning and obstacle avoidance. One that quite a few people have encountered in their life is autonomous robot vacuum cleaners. Other applications include Care Providing Robot FRIEND (a semi-autonomous wheelchair that is able to provide care for the elderly and the disabled), AISMO (an autonomous humanoid robot), mining robots, and unmanned aerial vehicles.

Why should I study robotics at KSU?

At KSU, we have a robotics and mechatronics major, a minor, and a robotics programming certificate. We are one of the few institutions in the country to offer mechatronics at a bachelor's level. Thanks to our history as a Polytechnique, we have a hands-on approach to learning. Thus, our courses are full of projects and labs to support the theoretical aspects of the lessons.

> *"I love that most of the professors have industry experience. In my engineering graphics class, my professor once brought the schematics for an actual part in an actual plane that the Air Force paid him to design and used it as an example for why the things we were learning were important and how we would apply them. It was great fun and gave me a sense of how what I am learning can be useful for me in the future."*
>
> \- *Sofia Gomezllata, Mechatronics Major*

For example, students in our Mechatronics Fundamentals class have the opportunity to learn how to use the Robotic Operating System (ROS.) One of the projects they perform is similar to our Maze Runner but more complex. Students are asked to use wall-following as a method of path-planning and to code accurate obstacle avoidance algorithms. Then, students use in-depth simulation and real-life testing for feedback.

This project acts as an introduction to ROS, which is an important skill our graduates have noticed is currently key in the robotics field. At KSU, we make sure that our curriculum is up to date with the industry's needs.

For more information about KSU and our robotics' programs, please look for our website[3] or contact us via email[4].

[3] https://engineering.kennesaw.edu/robotics-mechatronics/index.php
[4] mechatronics@kennesaw.edu

Student Project: Maze Runner

Parts

Mechanical: You can use Lego bricks, as in the example, but cardboard is an excellent material to build robots with. As a side project, you should attempt to build the car with recyclable materials you found at home or your school.

Electrical: For this project, you will need one ultrasonic proximity detector, an Arduino Uno (or an Arduino-alternative), a breadboard, and jumper wires.

Depending on what you have access to or how you want to make your robot to be, you will need one of these (or you can use both, as in the sample.)

Motors as Indicators	LEDs as Indicators
2 x DC motors	2 x LEDs
1 x L293D Motor Driver	2 x 220 Ω resistor

Some of you might have these at home or access to them at school, but if you do not, then you can get some basic kits from Elegoo[5].

Programming: Arduino IDE (free download from here[6] or use online[7])

Mechanical

The physical aspect of this project is very fluid. It is strongly encouraged that you create your vehicle as you see fit. Your breadboard and other components will take space, so it is easier to build around a built system than adapt your electrical components to the vehicle.

When using Legos or similar blocks, remember that your breadboard/Arduino might not fit the other blocks' dimensions. Using some tape might help to keep the components from moving and might save you a few headaches. Ensure that it is a removable tape, as you want to be able to use the parts for other projects.

Electrical

Warning: While experimentation is recommended for the Maze Runner's mechanical build, and strongly encouraged in the code's case, circuits should not be experimented with physically. If you want to experiment with circuitry, we encourage you to do so with a simulation. Electronics can be dangerous, and you risk the danger of burning them when used incorrectly. You can find a circuit simulator here[8].

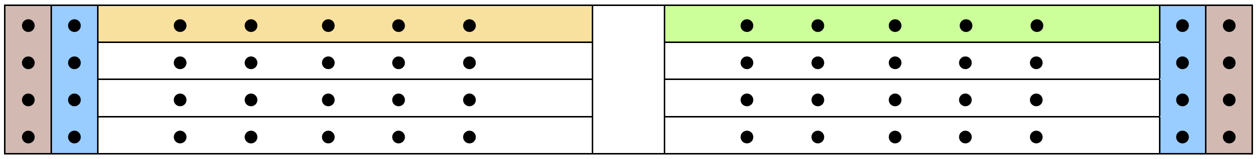
Circuitry is the glue that joins our code with our mechanical aspects. Here, we will focus on three main things: our Arduino, our sensor, our LEDs, and our motors.

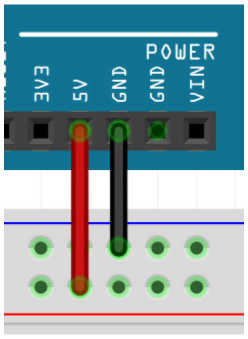
An Arduino is a microcontroller (a small computer) that stores our code and communicates with our electrical components. Every Arduino comes with a set of pins, which have different functions (always double-check that you are using the right pin, especially if your robot is not working and you have already double-checked your code) and are the primary way we are going to communicate with our circuit.

[5] https://www.elegoo.com/collections/arduino-learning-sets
[6] https://www.arduino.cc/en/software
[7] https://create.arduino.cc/
[8] https://phet.colorado.edu/sims/html/circuit-construction-kit-dc/latest/circuit-construction-kit-dc_en.html

General Wiring:

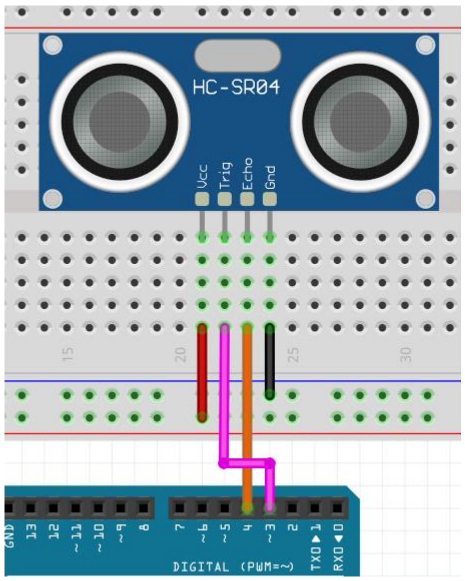
A breadboard (also known as protoboard) is a construction base of electronics. It has a configuration similar to the table below. The sides are joined together vertically, and you should wire them to the ground and to your voltage source (conventionally black is ground and red is voltage, and while it makes no difference in the circuit, it will make your life easier to stick with this rule, so you can avoid confusion.) The middle of the breadboard is made of horizontal pieces of metal that join each row together into the same node.





The points that share a color are joined together, which can be used to create your circuit. Insert components into the same row if you want them to be connected to one another.

For this project, you will have to wire your ground and voltage sources from the Arduino into the breadboard.

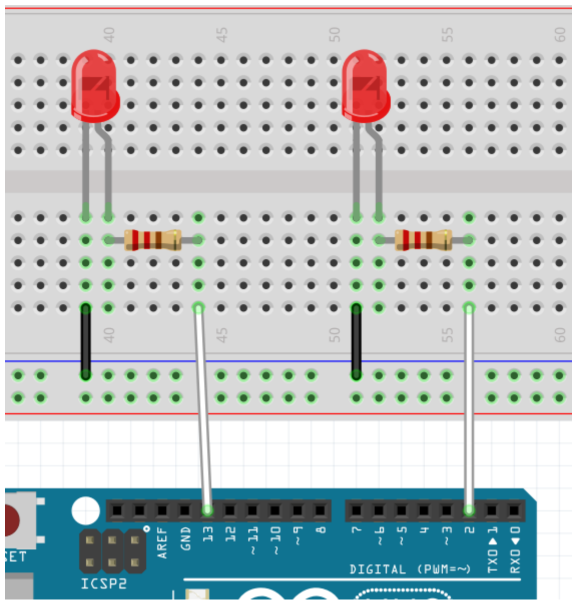


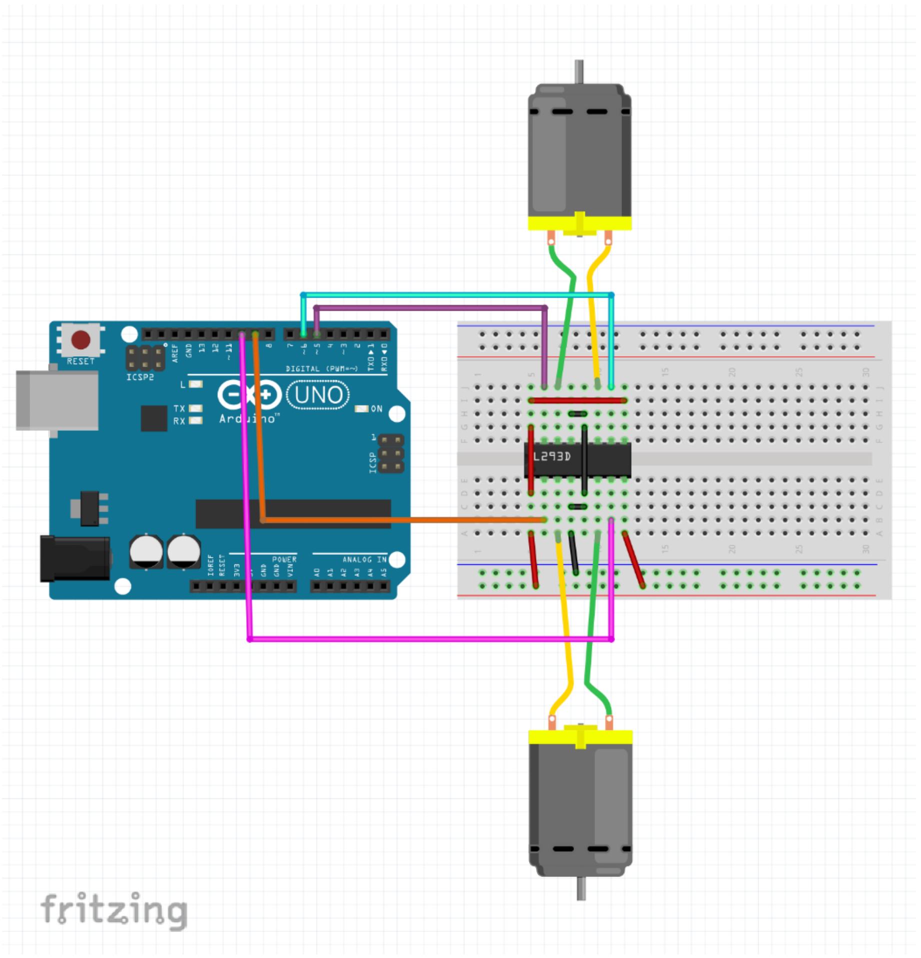
Ultrasonic Sensor:

An ultrasonic sensor has four legs, and each leg will connect to a different node. The outside legs are going to be connected to the ground (0V) and our voltage source (5V). The other two legs will be the actual sensor. One of the legs will send a signal to send a burst of sound. The other leg will receive the burst of sound and indicate how long it takes between the initial burst and when it came back. Our code will then transform the time that it takes for the echo to hit our sensor into a distance that we can read.

LEDs:

The LEDs are not necessary for your vehicle's functioning, but they are a good way to troubleshoot and can be used as an alternative to seeing the output of your work if you do not have motors. Always remember to use a resistor when connecting your LEDs, as otherwise, they will burn. The resistor's actual value will depend on your LED, but 330Ω and 220Ω resistors usually work well enough for these types of projects.



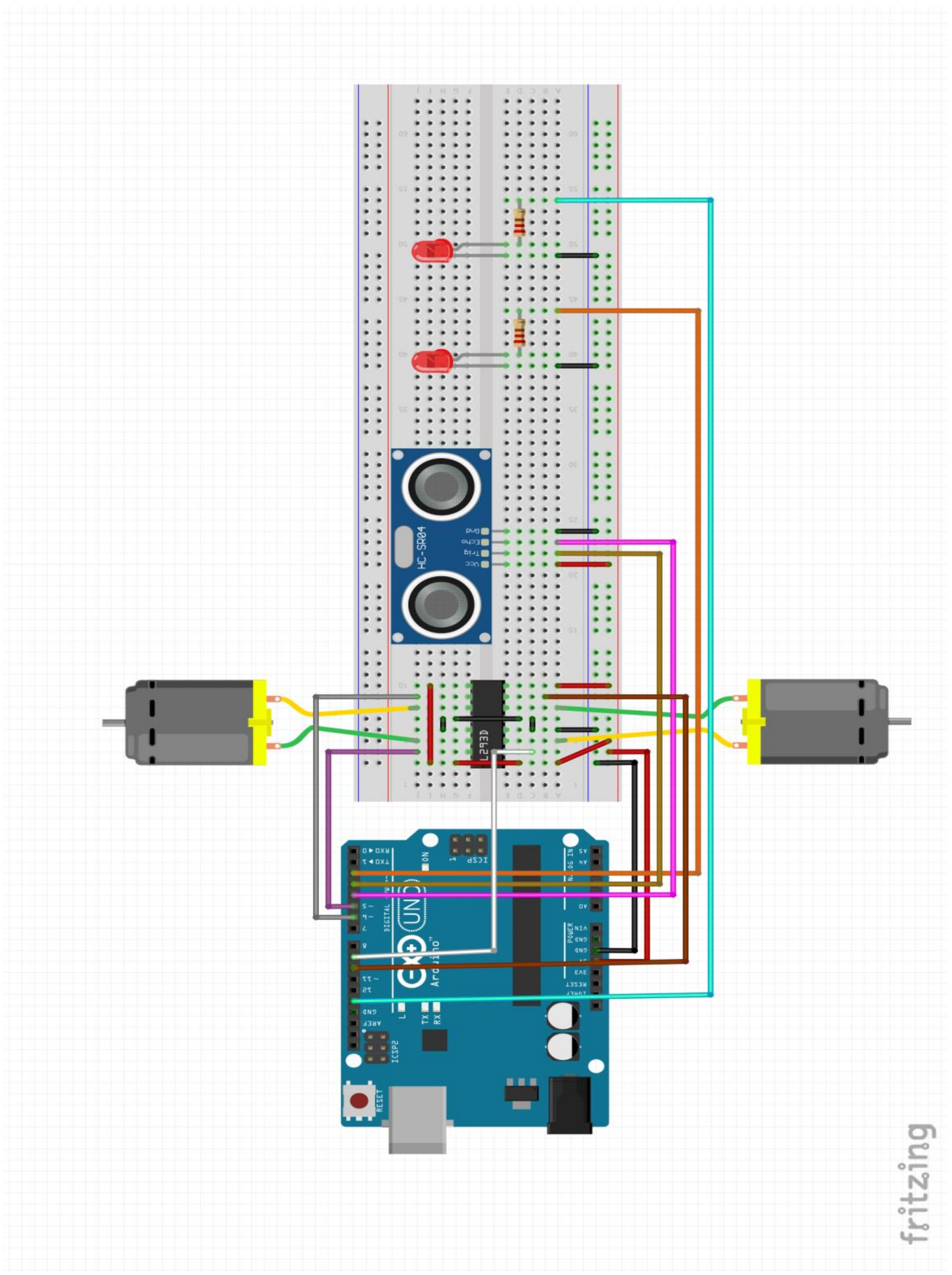


DC Motors:

This step will depend on which electronics you have. I will use an L293D Motor Driver for this example, but if you have different parts, just search online for a tutorial for your particular kit. REMEMBER: Do not experiment; follow a tutorial until you know what you are doing, and even then, please use a simulation rather than wire your own circuit.

While the other schematics have the freedom of which pins you use (as long as they are digital), you will need to use PWM pins for the motors (those with a ~ sign next to their number.)

The complete circuit looks like this(a .png can be found on the .zip folder):



Programming

For this project, we will be using the Arduino IDE (free download from here[9] or use online[10]) as our programming platform. The sample code can be accessed here[11] and has been thoroughly commented on. We strongly encourage you to write your own code and design your own algorithms rather than use the sample code provided.

Let's take a look at what everything means:

```
//Motor A
const int motorAPin1  = 6;  // Pin 14 of L293
const int motorAPin2  = 5;  // Pin 10 of L293
//Motor B
const int motorBPin1  = 9; // Pin  7 of L293
const int motorBPin2  = 10;  // Pin  2 of L293

//Ultrasonic Sensor
const int pingPin = 3; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 4; // Echo Pin of Ultrasonic Sensor

//LED
const int rightLed = 13; //Pin for the Right Led
const int leftLed = 2; // Pin for the Left Led
```

First of all, we are going to set up our pins. These are the same pins numbers that we used for the circuit, but remember that you can change them (just make sure to change both the code and the physical circuit.)

```
int turnDirection = 0; //Variable that indicates the direction of the turn
```

Then, I include a variable for the turn direction with 0 and 1 being Right and 2 and 3 being Left. Starting from zero is a programming convention, and while it is not the most intuitive at first, I strongly encourage you to start getting used to it.

[9] https://www.arduino.cc/en/software
[10] https://create.arduino.cc/
[11] https://engineering.kennesaw.edu/robotics-mechatronics/students/Events.php

```
void setup() {
  // put your setup code here, to run once:
  pinMode(motorAPin1, OUTPUT);
  pinMode(motorAPin2, OUTPUT);
  pinMode(motorBPin1, OUTPUT);
  pinMode(motorBPin2, OUTPUT);

  //Ultrasonic Sensor:
  pinMode(pingPin, OUTPUT);
  pinMode(echoPin, INPUT);
  //LEDs:
  pinMode(rightLed, OUTPUT);
  pinMode(leftLed, OUTPUT);

}
```

Then, we are writing a setup for our pins. We are telling each pin what they will be doing; OUTPUT means that they will be sending a bit of current into the circuit, while INPUT means that they will be receiving the current. This code will only run once, so remember not to write here things you want in your while loop.

```
void loop() {
  // put your main code here, to run repeatedly:
  long myDistance = ultraSonicDistance();
  if(myDistance <3){
    turn();
  }
  else{
    goForwards();
  }
}
```

This while loop will be our main bit of code. This loop will be running again and again while our robot is ON. Personally, I find it easier to keep my while loop short and readable and have quite a few functions (self-contained bits of code that accomplish one specific task.) This is my preference and a good habit with longer pieces of code, but you can code everything on your main loop.

```
//Find the distance between the robot and the closest obstacle, in inches, using the ultraSonic Sensor
long ultraSonicDistance(){
   long duration;
   //send ping
   digitalWrite(pingPin, LOW);
   delayMicroseconds(2);
   digitalWrite(pingPin, HIGH);
   delayMicroseconds(10);
   digitalWrite(pingPin, LOW);
   //receive ping
   duration = pulseIn(echoPin, HIGH);
   return duration / 74 / 2; // A transformation that returns inches
}
```

Our first function finds the distance between the sensor and the walls. Notice the digitalWrite function; it is the function we use to tell a pin what to do. You will be using digitalWrite (and analogWrite, the version for the analog pins) frequently when using Arduino, so remember to always start with the pin you want and then what you want it to do.

```
void turn(){
  if (turnDirection < 2){
    turnRight();
  }
  else{
    turnLeft();
  }
  turnDirection = (turnDirection + 1) % 4;
  //we add the % 4 at the end so we cycle through 0 1 2 3 back to 0
  //it is called a modulo (mod) operator and is essential for coding
}
```

Our main algorithm is found on the Turn function. We encourage you to find a better way to write this function.

Mod operators find the remain that you get in a division. For example, imagine we are on turn 3; when you divide it by 4 (3/4), we get 0.75 which is 0 for the computer. However, when we use the mod 3%4, we get 3 (that .75 the computer didn’t take into account). When you do 4%4, then you get 0, as the division gives you a perfect 1. Therefore, you can have a nice loop of 0,1,2,3 back to 0 by just continuously adding numbers.

```
void turnRight(){
  stopMotors();
  rightBlinker();
  goBack();
  delay(500); // wait .5 second, the numbers inside are miliseconds. Adjust the time depending on the size of the maze
  goRight();
  goForwards();
}
```

In this function, it is important to notice the delay function. Remember that 1 second is 1000 milliseconds, so write your numbers accordingly.

```
void rightBlinker(){
    digitalWrite(rightLed, HIGH); //turn Led on
    delay(500);
    digitalWrite(rightLed, LOW); //turn Led off
    delay(500);
    digitalWrite(rightLed, HIGH);
    delay(500);
    digitalWrite(rightLed, LOW);
}
```

Blinkers are quite fun to program. Can you create a light show using different LEDs?

```
void stopMotors(){
    digitalWrite(motorAPin1, LOW);
    digitalWrite(motorAPin2, LOW);
    digitalWrite(motorBPin1, LOW);
    digitalWrite(motorBPin2, LOW);
}
```

Even if you do not want your robot to stop, it is convenient to have a Stop Motors function. When you are testing things around, it is easier to call this function in your main loop than having to hold the robot up, so it doesn't escape.

```
void goForwards(){
    digitalWrite(motorAPin1, HIGH);
    digitalWrite(motorAPin2, LOW);
    digitalWrite(motorBPin1, HIGH);
    digitalWrite(motorBPin2, LOW);
}
```

Which motor pin should be HIGH and which one LOW will most likely require trial and error. You change the code here, you can switch the pin numbers when we first initialized them, or you could physically change the pins in your circuit. Switching the pin numbers around is usually the most painless correction to make.

The complete code can be accessed here[12].

[12] https://engineering.kennesaw.edu/robotics-mechatronics/students/Events.php

Contact Us

Muhammad Hassan Tanveer, Ph.D.
Assistant Professor of Robotics and Mechatronics Engineering

Phone: (470) 578-5612
Email: mtanveer@kennesaw.edu
Location: Q 316A

Cecilia McDaniel
Office Administrator

Phone: (470) 578-7234
Email: cmcdan47@kennesaw.edu
Location: Q 321

Kristin Barnhill
Robotics and Mechatronics Advisor

Phone: 470-578-3817
Email: kpianka@kennesaw.edu
Location: Q 143

Keyani Grier
Admissions Counselor

Phone: (470) 578-4636
Email: kgrier8@kennesaw.edu
Location: B 108

Sofia Gomezllata-Marmolejo
Student Assistant

Email: sgomezll@students.kennesaw.edu