University of Texas at Arlington

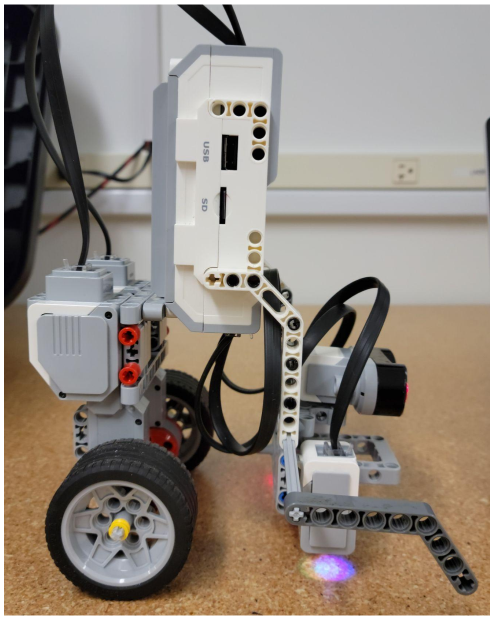
Project #2: Robot Navigation
Project Writeup

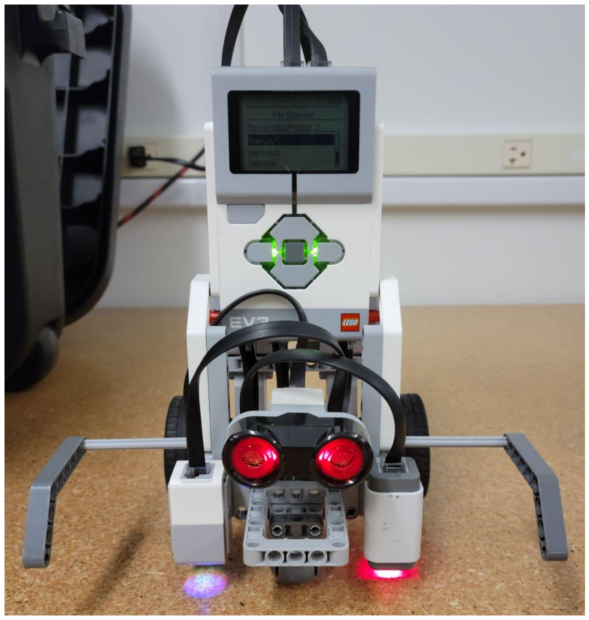
Benjamin Knight
Rogelio Chapa
November 30, 2021

Table of Contents

Hardware Design Choice

Overall Design





The design we went with is a two motor robot with the motors in the rear and 3 sensors in the front. We also added some ‘arms’ that extend out horizontally from the robot. The reason we decided to mount the robots vertically rather than the typical horizontal orientation is to reduce the amount of space the robot occupies. This aids with the precision of our robots navigation when it comes to avoiding going over a wall or to make more precise turns. The downside with this design we found was that when knocking over the goal/can we would sometimes miss it. So to adjust for this we added the horizontal ‘arms’ to help increase the reliability of knocking over the can. To the front we also have mounted an ultrasonic sensor facing forward, a color/light sensor on the right, and a light sensor on the left. These sensors aid with the navigation. The front part of our robot is supported by a steel ball caster. Since this design is a unicycle robot this is the best choice when we perform our turns.

Why a Unicycle Robot?

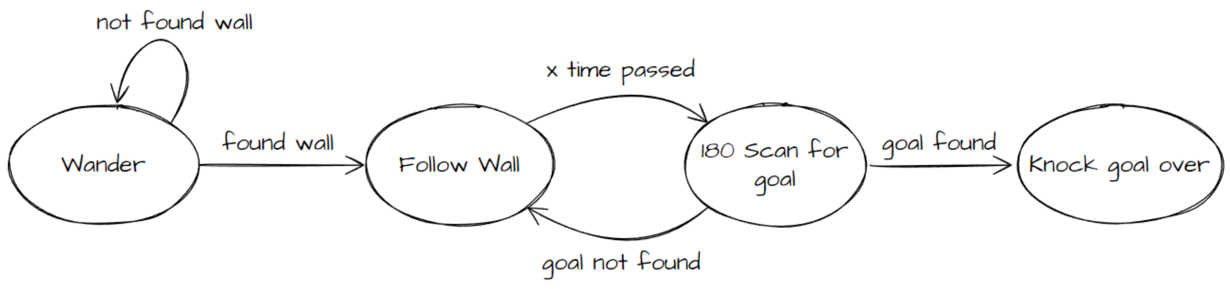
Unicycle type robots can be treated as holonomic robots with one extra step in between movement, turning. Again this allows us to focus on the programming and algorithm itself rather than how to control the robot.

Software Design Choices

Why Python?

Python was a no brainer to us after having major problems with Eclipse not only on Windows but the Virtual Machine/Linux. It increased our productivity exponentially allowing us to code and test our algorithms rather than trying to get the development environment up and running. We did not use any libraries other than the ones required to move our robot. Theoretically you could copy our program's logic almost line by line into C and it would still work.

General Algorithm

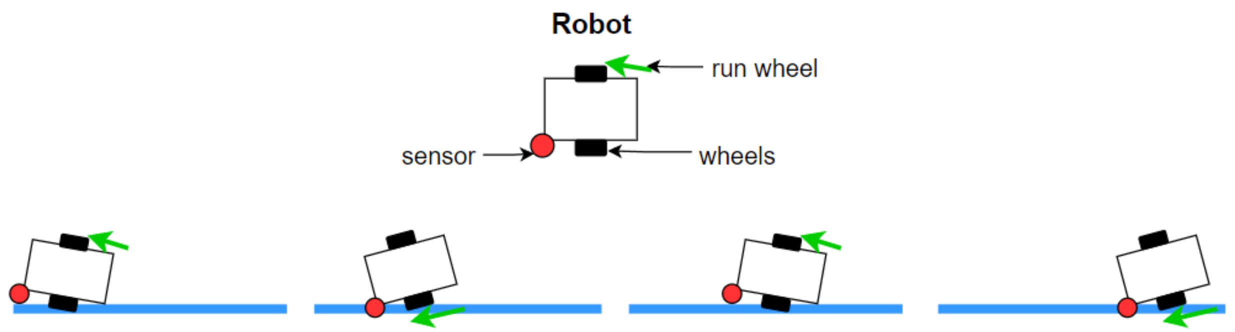


The general idea we followed is to initially start with the wander function. Once the wander function finds the wall we proceed to follow the wall. After a set amount of time spent following the wall function we will perform a 180 scan using the ultrasonic sensor. The sensor will check if the can/goal is close by. If the goal is close by we then proceed to knock it over. If the goal is not close by we then go back to following the wall. This process repeats until the goal is found.

Program Flow

All of our functions are running inside of a while loop under the condition of the goal being found. Once the goal is found we then exit the while loop and our program has completed. If the goal is not found we will keep running the while loop.

Wall Following Function



Our "*follow_wall()*" uses a function to check if there is a wall detected by our light sensor and we alternate between moving the left and right motors. If a wall is detected (we assume the wall is always on the left side of the robot) then the left motor runs until a wall is no longer detected. If a wall is not detected then we run the right motor until a wall is detected. Essentially this gives our robot a sort of "crawling" way to navigate along the wall and a steady pace. If we happen to continue to be on the wall on the left after already having run the left motor then we will run the right motor in reverse to prevent having the sensor go over the wall to the other side. This is a case where we have hit a corner and need to prevent the sensor from going over the wall to the other side.

Wander Function

Our "*wander()"* function is what is initially run at the start of our program (assuming we are not already on a wall). This is simply a function that runs the left and right motors but we have our right motor running at a slower pace than our left motor. This causes our robot to navigate in a circular manner. Overtime we increase the speed of the right motor and effectively increase the radius of the circle we are navigating until we reach a point where a wall is detected. Once the wall is hit we then call the "*follow_wall()"* function.

Scanning

We implemented a scanning method called *“check_for_goal()”* that gets called repeated every x amount of seconds. During the wall following function this function will do a 180 degree “scan” of the area to attempt to find the goal. Using the ultrasonic sensor we give it a minimum distance to detect a goal. This will return true if the goal is found. If a goal is not found then we just proceed back to the wall following function.

Goal Detected

Once a goal is found we have a few functions to complete our task. Initially the robot will stop and then restart the scanning process, but this time it is trying to find the center of the object. We do this by using the ultrasonic sensor to find the shortest distance as it scans the entirety of the object and around it. Once it finds a minimum distance we once again scan until we find that minimum distance again and then stop. Afterwards, we then proceed to move forward and knock the can off of the goal