Department of Computer Science

Embedded Linux
CPS 342

Maze Solving Robot

Date: 5/14/18		Semester: Spring 2018
Group Members	**Department**	**Major Contribution**
Seraphim Dmitrieff	**CS**	**Maze Algorithm**
David Schoemer	**CE**	**Line Sensor**
Michael Martinez	**CE**	**Motor Drivers**

Abstract

This project is the design and implementation of a maze solving robot. This robot is designed to slowly solve a maze constructed of dark tape on a light non reflective surface and then return to the start as fast as possible.

Instructor: C. Easwaran

Theory:

The maze solving algorithm follows a simple left-hand rule algorithm. Upon encountering an intersection the robot will always choose the most left-handed path, prioritizing left, then straight then right. If no turns are available the robot will turn around and continue the maze. This algorithm is designed for a walled maze as always following the left most path, the robot would eventually find the end of the maze. In a line maze however this algorithm is impeded in a maze with loops and will only work in cases of open-ended mazes, which will work for our purposes. To be more efficient we would need encoded motors to track distances travelled by measuring the number of turns the motors make. With this information we would be able to more accurately back track the movements and account for loops within a maze.

Motor Design:

In order to get the robot car to go straight, left, right, or backwards, the motor for each wheel of the car had to be coded. The robot car has four motors, one for each wheel. By using two L293D motor drive chip, two motors are connected to each chip. The left two wheels are connected to one motor drive chip and the right two wheels are connected to the other chip. GPIO pins are enable and configure in order to connect the motors to the raspberry pi. The GPIO pins use for the front left wheel are pins 16, 18, 22, and the back left wheel are 19, 21, and 23. The GPIO pins for the front right wheel are pins 36, 38, 40 and the back right are 33, 35, and 37. The three pins are for each motor because each motor has two inputs and one enable input. In order to be able to configure the pins, the program has to setup the pins. So the line code for each motor is GPIO.setup (name of motor, GPIO.out). The inputs make the car be able to turn in any

direction and the enable input turns on the motor. In order to turn on the inputs, the program is writes high and low to turn off the input. If the enable input is off then the car does not move because the enable turns on the motor. For example, in the program code, if the car wants to go straight then input A is high, input B is low, and enable is high, and if the car wants to go left then the input A is low, input B is high, and enable is high. In order to stop the car from moving then the enable input has to be low in order to turn off the motor. The length of time the motor is on depends the number written in the program. The function code to determine the length of time is sleep(# of seconds). Since the car was moving too fast, the program uses pulse width modula to control the speed of the motors. The robot car is able to turn in any direction and turn on and off the right amount time it takes to complete the maze.

Sensor Design and Algorithm:

To traverse the maze a program is implemented using a while loop with a boolean condition to determine if the end of the maze has been reached. The end of the maze is defined by when all the sensors read true after a movement has been completed. While this condition hasn't been met the vehicle is prompted to move forward along the path implementing PWM controls to maintain its course. Movement continues until a turn is detected or a dead end is reached. The movement function returns the sensor reading taken at that point, which is then read by the main program . It then determines what the next path should be executing the appropriate turn function. The main program then tracks the turn, assigning a numeric value to the turn and appends it to a list, named turns, that behaves as a stack. In this program the value 1 is assigned to a left turn, 2 to a straight turn, 3 to a right turn and 4 to a turnaround at a dead end. A separate boolean condition is used to track if a turnaround is made and is implemented to

detect if the robot is backtracking along its current path. The turnaround value of 4 is discarded from the stack then triggers the movement function until the next decision must be made. The program then determines if the turn is backtracked by adding the current decision to the value at the top of the stack. If the sum of these values equals 4 the turn is considered a backtracked path and is discarded from the turns stack. If the sum does not equal 4 the value is appended to the end of the stack and the maze solving algorithm conditions, setting the backtracking condition back to false. An example of this in action is say our vehicle took a straight path (denoted by value 2) and reach a dead end. It turned around and upon reaching the same intersection it previously went straight through it makes a left (denoted by value 1). By adding these values equaling 3 the program views this as a right turn (value 3) comprises the better path travelled and will skip this dead end when returning to the start.

Once the end of the maze is reached as determined by our large rectangular area that will trigger all the sensors to read true, we prompt the robot to turn around. Another while loop is used to backtrack the best path taken using the same movement functions implemented during the initial traversal of the maze. Instead of using the results from the movement function to determine the next move the next decision is popped of the turns stack we used to track the best path taken. Because we are travelling in the reverse direction the values for left and right turns are swapped with the value 1 representing a right turn instead of a left one and the value 3 representing a left turn instead of a right. This while loop continues until the stack of turns used to track the solution is emptied. Once this stack is cleared the robot will have returned to the start and the program ends.

The IR reflectance sensor detects the reflectance of a surface by using an IR pulse, generated by an IR LED, and measures the amount of IR light that bounces back to a photo-transistor located next to the IR LED. The sensor returns a raw value between 0 and a value set in the program, in our case 3000. The code on the arduino reads this number and based on testing I determined how to manipulate this raw value into a 1 or 0, 1 meaning a line is detected. Because there are 8 sensors this data can be transmitted using a single byte (8 bits). Using i2c this data is transmitted to the pi. The arduino is set as address #08 and is configured as a slave device. Because the arduino is a 5v device and the pi a 3.3v device the pi must be the master as the master sets the voltage for i2c devices. Functions for easily retrieving and manipulating the data on the pi were also written, as well as instructions on how to import them as a library.

Results:

Can the vehicle follow a straight line?

The vehicle can follow a straight line, with forward movement continuing until an intersection is detected or a dead end is reached.

Can the vehicle follow a bent line?

The vehicle uses PWM control on the enable lines on all the motors to adjust for bent paths. There are 5 sensors on our IR sensor, indexed 0-4. The middle sensor, index 2, is used to detect it the robot is currently on a path while the 2 adjacent sensors, indices 1 and 3 detect if the robot is drifting from the path and when a line is detected on either sensor, the movement coding

increases the duty cycles to the opposite side and decreases them on the same side. It maintains this configuration until the sensor no longer detects a line and returns to a normal base line speed

Can the vehicle retrace its path at the end of a line?

The movement function will determine if a dead end is reached. If this condition is detected the vehicle turns left until a line is detected on its middle IR sensor. A separate boolean condition is triggered to determine if the path is backtracked by assigning numeric values to the turn decisions 1,2,3,4 for left, straight, right and turning around respectively. If the path is being backtracked the current decision added to the decision currently at the top of the turns stack equals 4 and both values are discarded. If the sum is less than 4 the new value is appended to the turns stack and recognized as a more optimal path

How is target defined?

The target is defined by a large rectangular area that will read all 5 IR sensors to be true once encountered.

Can the vehicle find its target?

What's the algorithm used to find the target?

A left-hand rule algorithm is implemented to solve the maze. The robot is instructed to continue along a path until an intersection or a dead end is reached. The program uses the line sensor data to make a decision and executes it based on prioritizing left, then straight then right then turning around. One the function for the turn is executes the program loops and prompts the movement function once again. The program continues this loop until the maze end condition is met

Once target is reached, can the vehicle retrace the path next time?

The vehicle uses a stack to track its movements removing any backtracked turns. Upon reaching the end of the maze the vehicle turns around and returns to the start of the Maze using the most optimal Path taken, avoid any dead ends and backtracked paths

Is the vehicle construction clean?

Are the motor control and maze solver codes organized and commented?

Yes

Overall Group Performance

Each person in the group had a part given to them. Michael Martinez part was to build the the 4-wheel chassis robot car, code the motor to be able to turn in any direction, and assemble the cable connection from the battery, and motors to the L293D motor drive chip. David Schoemer part was to code the IR reflectance sensor to be able read the line so it tell the motor when to turn and he had to assemble to line sensor to the robot car by soldered the chip to RPi. Seraphim Dmitrieff part was to implement the algorithm to solve the maze and combine all the code such as the motors, line sensor, and the algorithm. The algorithm had to get the car to solve the maze by getting the car to figure out the fastest route to the main spot and return back. Towards the end of the project design, we all try to add any input to solve all the issues in the code to get the robot car to be able to solve the maze.

Operation:

Utilizing the prebuilt motor chassis and Raspberry Pi the code is downloaded from one of our repositories and the maze_runner.py code run.