**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 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.

Reference for Wiring:

<https://business.tutsplus.com/tutorials/controlling-dc-motors-using-python-with-a-raspberry-pi--cms-20051>

**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.

Code for Algorithm can be found at:

<https://github.com/N03512799/ELSpring2018/blob/master/code/MazeRunner.py>

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.

In our project we used the SunFounder 5 Channel Reflectance Sensor. The operation is similar to the sensor provided in class, the reference sheet for the sensor can be located at, this sensor was much easier to use as it had a git repository dedicated to operating on a Raspberry pi, whereas the one provided in class works optimally with an Arduino

<http://wiki.sunfounder.cc/index.php?title=Line_Follower_Module(5_channel)>

**Vehicle Design:**

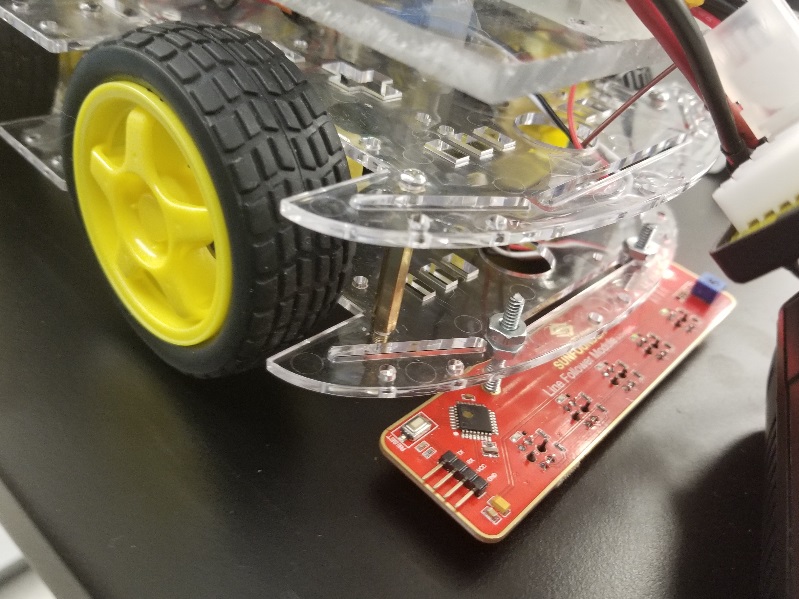


The vehicle comprises of 4 motors mounted to a clear chasis. Separate power was used for the Raspberry Pi and IR Sensor and the 4 motors. The motor needed a much stronger LiPo power supply to operate simultaneously and using the power supply on the Pi would cause it to overload the chip and destroy the unit. Below is an image of the motor battery:



A portable cellphone charging battery was used to allow wireless operation of the unit. To communicate with the RPi we installed VNC viewer on the pi to allow remote access to the unit to allow for ease of debugging code and avoiding interference of components contacting with the wheels.

To connect the reflectance sensor, 2 1 inch #6 machine screws were used to mount it to the chassis, 2 nuts, one above and the other below the mounting deck allowed the sensor to be adjusted.



All non-motor components were mounted to a small sheet of Plexiglass keeping motor components and power supply and chips separate. All components were mounted to the sheet using easily removeable double sided wall mounting tape. A 1 inch spacer and 2 2 inch #6 machine screws were used to offset the Plexiglass from the vehicle chassis Assembled to robot looks like this:

