Preliminary Design Review

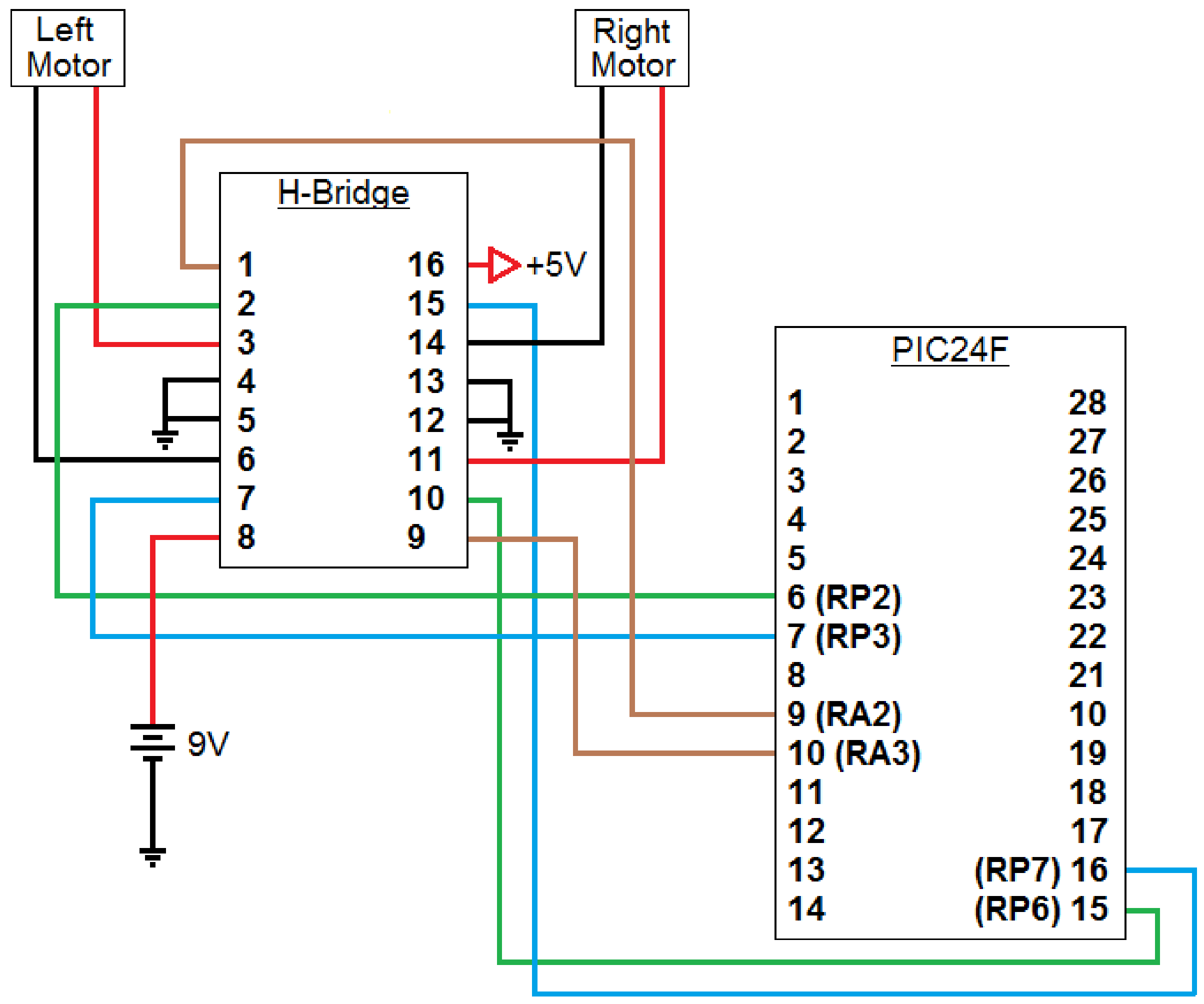
Design Summary:

Our design will consist of using 4 infrared transmitters and receivers to detect black tape lines on the floor. All emitters and receivers will be on the bottom of the car pointing down. There will be two receivers in the front, on either side of the black tape. The other two will be both centered in relation to the moving axis of the car. We will center one between the wheels so that as long as it is over a line, pivoting the car will not de-align it. The other will be in the center, but at the very front of the car, around ½ - ¾ of an inch ahead of the two side sensors. The emitters will be appropriately positioned (pointing down as well), so that the receivers receive maximal reflective light. Of course, the data received will be all handled in software, which will dictate what route the robot takes, and ensuring the line is being followed straight on.

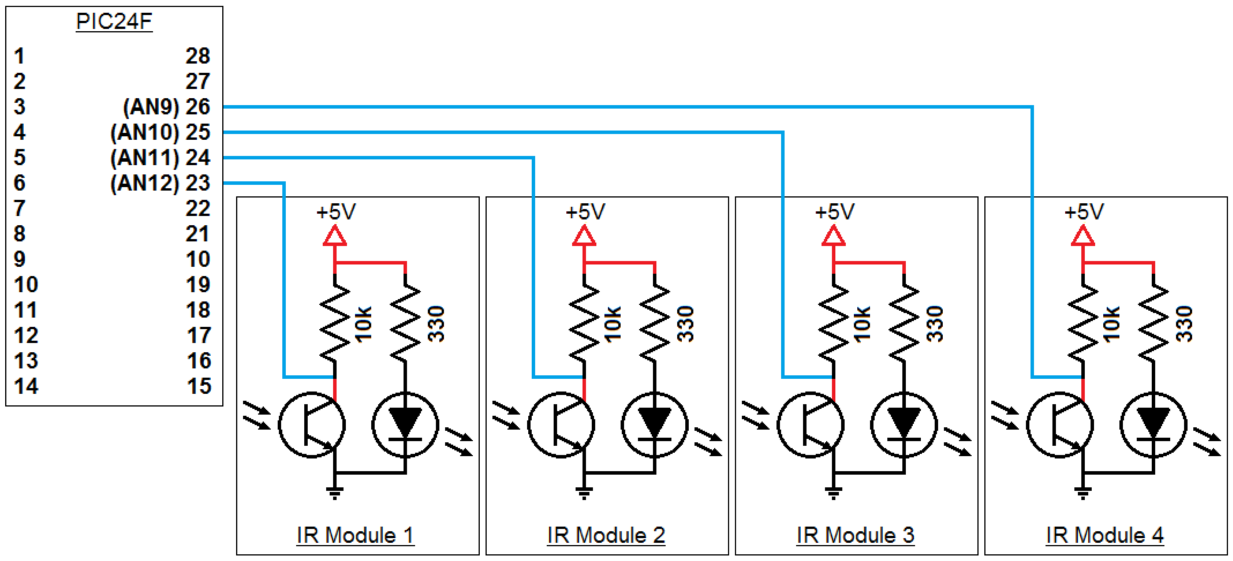
The extra sensor we will implement will be a pair of hypersonic emitters/detectors. This outputs a pulse based on distance an object is from the sensor. The distance formula can be calculated using a formula provided in the datasheet. We are planning on using two of these on the front as a right and left detector. This should help in object detection (like if there is a car stopped in the middle of the line), and knowing which way to go around would be optimal. Also, wall detection and avoidance would be a useful function these would provide. Additionally, we are planning on using a higher voltage source on the H-bridge, so as to get more speed out of our robot.

Hardware Design:

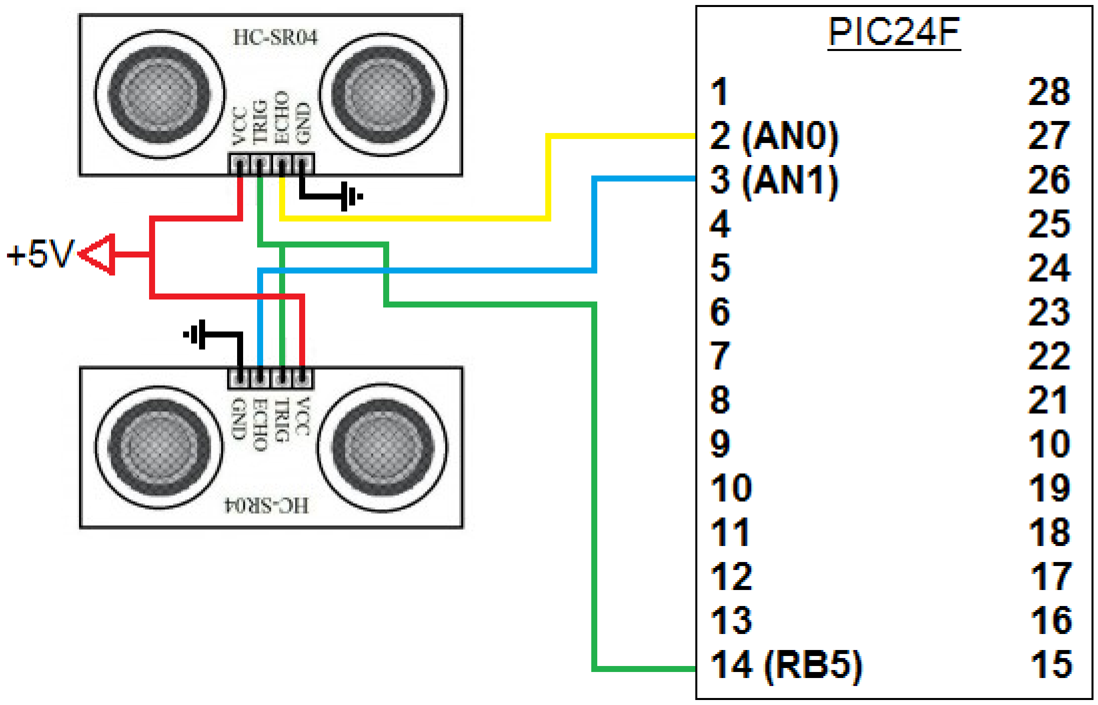
Circuit Diagrams:



All of the pins on the H-Bridge will be connected using connectors.
Pins 6, 7, 9, 10, 15, 16 on the PIC24F microcontroller will be connected using connectors.
Both the left and right motors will be connected using connectors.
All grounds and sources will be connected using connector.



Pin 23 – 26 on the PIC24F microcontroller will be connected using connectors.
All photo transistors, LEDs, and resistors will be soldered to vector boards to create individual IR modules.
Output from the IR modules will be connected using connectors.
All grounds and sources will be connected using connectors.
Resistor values are just example right now, formal values will be chosen later.



Pins 1, 2, 14 on the PIC24F microcontroller will be connected using connectors.
Pins TRIG and ECHO on both HC-SR204 modules will be connected using connectors.
All grounds and sources will be connected using connectors.

Parts:

Parts List				
Quantity	Part	Part Number	Approx. Cost (ea)	Source
2	SunFounder Ultrasonic Module	HC-SR04	$4.50	Amazon.com
1	Duracell 9V Battery	MN1604	$4.00	BestBuy

Microcontroller Pins:

PIC24F Pin Guide		
Pin #	Use	Device
1	-	
2	Digital Input	Ultrasonic Module
3	Digital Input	Ultrasonic Module
4	Debugging	PIC
5	Debugging	PIC
6	PWM	H-Bridge
7	PWM	H-Bridge
8	-	
9	Control (Digital Output)	H-Bridge
10	Control (Digital Output)	H-Bridge
11	Debugging	PIC
12	Debugging	PIC
13	-	
14	OCM Dual Compare	Ultrasonic Module
15	PWM	H-Bridge
16	PWM	H-Bridge
17	-	
18	-	
19	-	
20	-	
21	-	
22	-	
23	Analog Input	IR Module 1
24	Analog Input	IR Module 2
25	Analog Input	IR Module 3
26	Analog Input	IR Module 4
27	-	
28	-	

Software Design:

Software Overview:

The software will handle the controls of the robot following the line. The outline is as follows, and assumes operation from the main function of the program, with any variables mentioned pre-defined. A traditional state machine is not (visually) utilized because it is generally less helpful to myself, and this outline provides a better medium for greater detail.

1. state = lineUpCar
    a. The checkSensors() will be called to check that the middle two sensors are lined up on the line. If the function reveals that the left and right sensors are not in the optimal positions (which will be found in practice), then the car will pivot until the other sensors are lined up. state = forward.
    b. It is possible this state is deemed not useful in the end.
2. state = forward
    a. The car will go forward at full speed, calling checkSensors() for the status of each sensor at some frequency (I'm thinking somewhere around every 5-50ms; this will be tuned. I will avoid using interrupts on this as much as possible, by checking flags/registers). Also, the hypersonic detectors will be going with similar frequency. If anything is detected, state = objectDetected.
    b. The car will adjust based on the left and right sensor values. They will be near the border of the line by default, so we will know based on one sensor getting more/less intense that the line is diverging away or getting closer. This will also need to be tuned once built.

        i. If either the left or right sensor goes over a line at any time, then there is a (potential) turn coming up. If this is a corner, then the front sensor will go off of a line after this happens. If this sequence happens and just the left sensor goes off, then state = turnLeft. If right sensor, then state = turnRight.
        ii. If both the left AND right sensors find a line, then state = horizontalLine.
        iii. If the left and right sensors don’t change, and the front one loses the line, we will proceed forward a little more (for under a second, though), then state = finished.

3. state = turnLeft
    a. The car will turn left by calling the turnDeg() function. This state will also be doing the checks to make sure the car actually gets back over the line in the end. So once the front middle sensor is over the line, the turning will lessen significantly until the back sensor is over the line. Then the car will pivot once the back sensor is lined up, so that the car is fully on line with the tape.
4. state = turnRight
    a. The car will turn right by calling the turnDeg() function. This state will also be doing the checks to make sure the car actually gets back over the line in the end. So once the front middle sensor is over the line, the turning will lessen significantly until the back sensor is over the line. Then the car will pivot once the back sensor is lined up, so that the car is fully on line with the tape.

5. state = objectDetected
    a. Nothing will really change here. The robot will proceed forward, doing the same checks of sensors, but also will be continuously checking the distance of the object.
        i. If the object is lost (a few checks reveals no object), state = forward. If a left or right turn is detected, state = left or right, respectively.
        ii. If the finish line condition is found, state = finishLine
        iii. If the object comes within some threshold, state = objectAvoidance.
6. state = objectAvoidance
    a. The car will circle around an object to the right, returning to the line after. If the line is lost (like an object was sitting on the corner of a turn), then findLine('l') will be called, so the car will circle until the line is again found. state = forward.
7. state = horizontalLine
    a. Basically is checking for a T-intersection versus the finish line. The car will proceed forward and if the front sensor loses the line, then we have a T-intersection. state = turnRight.
    b. As long as the front sensor is still on a line, the car keeps going forward. The left and right sensors will then lose the line, state = finishLine.
8. state = finishLine
    a. Continues on the line and then watches for the other two marks of the finish line. Once the third one is found, I call the turnAround() function and then state = lineUpCar.
9. state = finished
    a. The car will turn around (for show, really) and then stop, as we are back at the start.

Functions:

int checkSensors(**int ***middle, **int ***midFront, **int ***l, **int ***r)

-Puts the values of each sensor in the given pointers.

int initilizeIR()

-Initializes all pins, input and output, used for IR. Turns on IR emitters and ADC for receivers. Of course, #define statements will be used for pin names.

int findLine(**char** dir)

-Hopefully this will be called minimally, but if the line is lost altogether (isOnStraightLine() returns 0), this will be called, and the robot will start circling until the line is found again. dir will dictate which direction the robot starts circling.

void turnDeg(**int** deg)

-Turns without stopping. If cutting corners is breaking the rules, this is potentially only useful for turns within inner-triangles.

void pivotDeg(**int** deg)

-Stops car (if moving) and pivots. Good for error-correction and for sticking to corners closer.

int checkObjectDistance()

-Does everything for checking the distance. Sends the 10us ODC pulse to the hypersonic device, and handles the pulse it returns (related to the length of the pulse, given by a formula in the datasheet)

void turnAround()

-Called when the finish line is found. The robot will stop and pivot around, keeping the middle sensor lined up over the line. This might just call the pivotDeg() function, and may be deemed useless in the end.

Tests:

1. Test robot to see whether it can follow the black line.

   It will pass if the robot follows the black line. Fail for not following the black line.

2. When robot reach the three parallel lines whether the robot can turn around.

   It will pass if the robot successfully turn around. Fail for not turn around.

3. Whether the robot can pick correct line if there is more than one options (the loop) for the line.

   It will pass if the robot can pick the programmed line. Fail for going to wrong line.

4. The sensor and LCD can work together to show the distance of an object in front of sensor.

   It will pass if the LCD correctly display the distance. Fail for wrong distance.