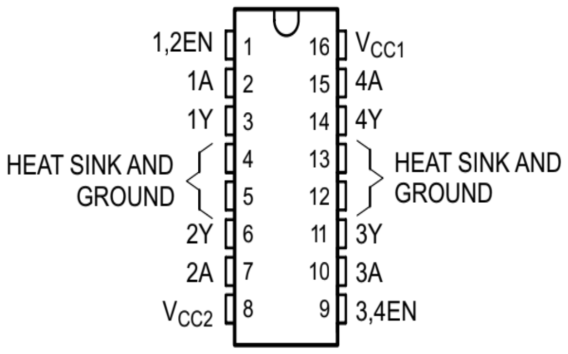
The following is a brief account of my final Open Project Space assignment.

Abstract

The goal of this project was to experiment with PID systems and create a miniature car that would parallel itself with a wall while traveling forward. Theoretically, the car would never hit the wall even if it was turning.

Experiment



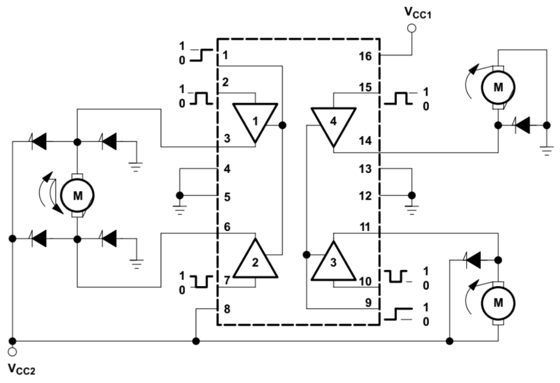


Figure 1: L293D IC

Figure 2: Circuit diagram

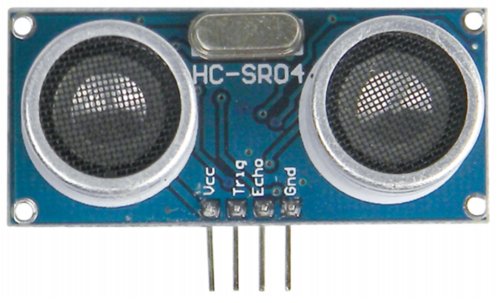
The L293D has two H-bridges which allows the driving of the car's motors. Described below are the appropriate ports for operating the car in the Arduino IDE.

- Power Supply
  - (pin16) VCC1 drives logic levels, connected to 5V
  - (pin8) VCC2 drives motor, connected to V_{in}
  - (pin4,5,12,13) connected to GND

- Direction Inputs
  - pin(2,7) IN1,IN2 left motors, connect to any digital pins
  - pin(10,15) IN3,IN4 right motors, connect to any digital pins

- Speed Inputs
  - pin(1,9) variable output to motor, connect to PWM pins

- Output
  - pin(3,6,11,14) L293D output, connect to motor leads



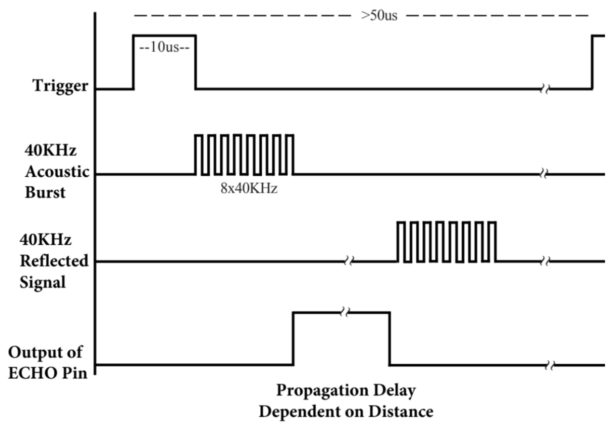


Figure 3: HC-SR04 ultrasonic sensor

Figure 4: Signals diagram of ultrasonic pulses

The ultrasonic sensor transmits and receives 40kHz ultrasonic pulses. To trigger a pulse, an electric input of at least 10μs should be applied to the “Trig” pin. At this state, the “Echo” pin would be pulled HIGH and an ultrasonic pulse would be sent out. Upon receiving a reflected pulse, the “Echo” pin would pull LOW. It is important to note that if nothing is reflected back, the sensor would simply time out and return the maximum value. The Propagation delay seen in Figure 4 shows the output of the ultrasonic sensor.

This distance can be simply calculated by applying the fact that the speed of sound is approximately 340 m/s. [distance = (duration/2)* 0.0343 - 10] in centimeters.

```
//------- UltraSound -------
duration = pulseIn(echoPin, HIGH); // Receive pulse
distance = (duration/2)* 0.0343 - 10;
distance -= 2;

//-------PID---------
delta_distance = distance - old_distance;
PID = kp * distance + kd * delta_distance;
analogWrite(rightSpd, 150 + (int) PID);
analogWrite(leftSpd, 175 - (int) PID);
old_distance = distance;
```

Figure 5: PID Control

Developing PID control involves taking the incoming sensor data and transmuting it into some legible for the motors. In this case, kp and kd are hard set values that determine the error that is currently existing and corrects it at the rate of kd. This way, kp and kd work in synergy to correct the path of the car and place it back on track in real time.

Conclusion

The idea of PID controlled movement can be applied to many cases in the real world such as cruise control and lance assist in the automotive world. This project not only showcases a primitive yet effective design to solving a real world issue, but it also gives insight to how PID can be used in a different manner to solve other problems that might require active assistance.