# 1. Introduction

At the heart of the autonomous maze-solving robot is a 10-bits ADC Arduino microcontroller system which handles motion of the robot and its sensual capabilities. The main responsibility of the Arduino/Hardware team is therefore to build the robot, setting up basic movement and sensors functionalities and lastly providing an API for robot to communicate with other teams in the project.

# 2. Key Hardware Components

The important hardware needed for this project is outlined as below:

## 2.1. Arduino Uno R3

The Uno R3 is an entry-level Arduino with 10-bit ADC hardware, 12 pins of Digital I/O and 5 pins of Analog Input. It’s simple hardware interface and user-friendly IDE has become a standard for beginners in making their own embedded projects, such as this project.

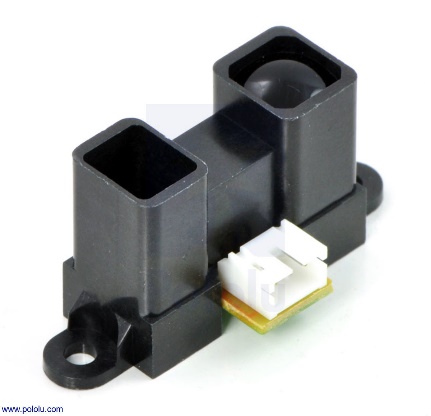
## 2.2. Pololu DualVNH5019 Motor Shield & 47:1 DC Motor with Encoder

The **DualVNH5019 motor driver shield** for Arduino [<http://www.pololu.com/catalog/product/2502>] make it easy to control the two bidirectional DC motors with the Arduino Uno. Together with the quadrature encoders, the motor shield and motors form the bulk of the robot control system, which allow for robust and accurate movements of robot as discuss in section 3.



## 2.3. Sharp IR Sensors & Power Regulator PCB

There are 2 types of IR sensors provided: **Sharp GP2Y0A21YK IR sensor** (effective range 10-80 cm) and the **Sharp GP2Y0A02YK IR sensor** (effective range 20-150cm). These sensors are interfaced to the Arduino microcontroller using SCSE fabricated power regulator PCB, which allow maximum 6 Sharp IR sensors to be connected at the same time. These sensors are important in obstacle avoidance of the robot.



## 2.4. SLA 6V Battery

The SLA 6V Battery is used to power the entire robot. Both the readings of the sensors and the performance of the motors are directly affected by the voltage of this battery. Hence close monitoring of voltage level during tuning or the robot is highly advised.



# 3. Software Library

Arduino is a very popular platform for hardware hobbyist. Therefore, there are many libraries by other developers that accelerate the development process of this autonomous robot.

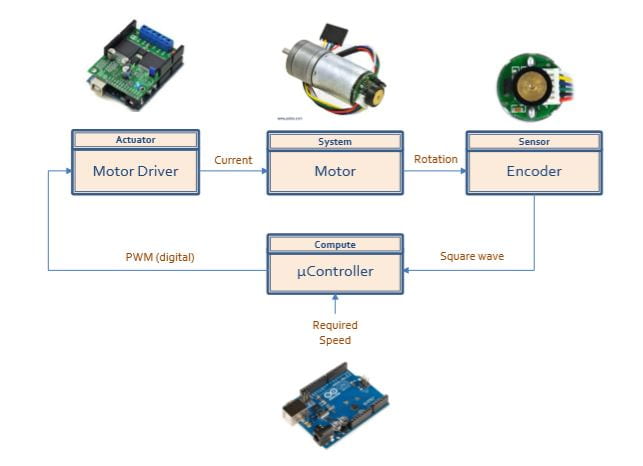
* PID\_v1.h (link) : A library to implement PID controller for the robot’s control system.
* SharpIR.h: A library to interface with Sharp IR sensors to get reading (in cm).
* DualVNH5019MotorShield.h: A library to interface with DualVNH5019MotorShield to control motors
* EnableInterrupt.h: A library to enable interrupts on digital pins of Arduino Uno. Without this library only limited number of pins can be enabled with interrupt.
* ArduinoSort.h: A library implementing insert sort. This sorting method is very useful during median filter when ordering of sample array is required.

# 4. Robot Motion and Control System

## 4.1. Overview of Closed Loop Control System.

It is easy to implement an Open Loop Control System for the movement of the robot. Just enable and supply a suitable PWM signal the motors to let them move straight, rotate right or left. Unfortunately, there are so many things that could go wrong in reality that make the robot stray away from its ideal movement. Characteristics of surface, difference in voltage supplied to motors, difference in motor’s error tolerance are a few to name. In short it is almost impossible for the robot to go straight with an open loop control system.

Luckily, a Closed Loop Control System may make the robot move relatively straighter via continuous feedback of motor actual speed and adjustments in motor speed at run-time, making them spin at relatively same time. Here is the rough idea behind the implementation of this feedback loop.



The microcontroller supplies the motors with PWM signals via the motor shield and make them spin. While the motors spin, the hall effect encoders are also spinning due to electromagnetic force from the motor and generate square wave signals.

Using edge-triggered interrupts, the Arduino microcontroller captures the rising/falling edge of the encoders’ signals and increase “ticks” count. This is done using two separate Interrupt Service Routine (ISR), which are triggered when their corresponding interrupt-enabled digital pins detect an edge change in their input (from the encoders). Using the tick value and time elapsed between each tick (rising/falling edges of square wave), we can calculate the actual speed of the motors since each DC motors generate approx. 2249 ticks (rising and falling edges of square waves across 2 channels in total) per full revolution. We used the EnableInterrupt.h library to implement interrupt and ISR in this case.

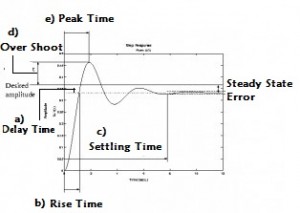
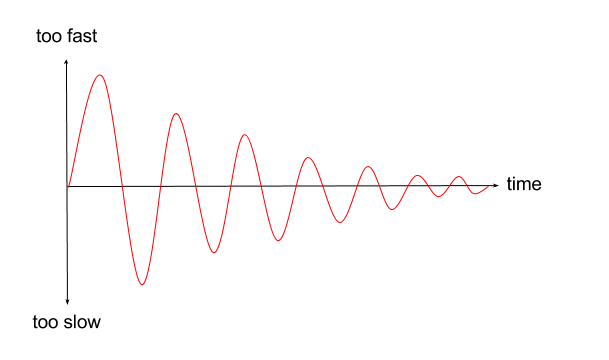
Arduino microcontrollers then compare the actual speed with whatever supplied and adjust the PWM/speed supply to the motor shield to ensure the two motors move as close to the ideal speed as possible. A PID controller is implemented here to actualise the above workflow as it is commonly used in robotics for the same purpose.

By trial and error, target tick value can also be determined to move the robot in exact distance forward or exact angle in left, right rotation. The motors will move until theirs tick values hit the target, where it will send a notification to the RPI and wait for the next command of movement.

## 4.2. PID Controller

A **PID controller** continuously calculates an **error** and applies a **corrective** **action** to resolve the error; in this case, the error is the motor spinning at the wrong speed and the corrective action is changing the PWM supply to the motor.

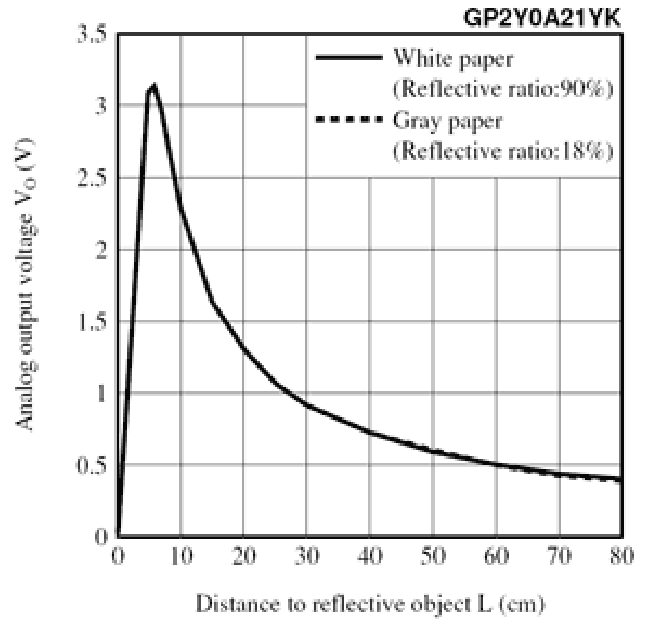
According to control theory, a stable system will have the following characteristics. Our goal is to keep our motor-encoder system bounded to this curve. The PID controller has 3 constants Kp, Ki, Kd, which serve to achieve this goal.



Increasing Kp will have the effect of reducing the rise time and will reduce but never eliminate the steady-state error. Increasing Ki will have the effect of eliminating the steady-state error, but it may make the transient response worse. Increasing Kd will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response. PID tuning is to determine the best combination of these constants for the ideal response of the robot motors.

# 5. Sensor Interface

IR Sensors can be integrated to the system using the SharpIR.h library. Both types of IR sensors have a bimodal characteristic of voltage reading vs distance, as shown in the picture below:



Hence it is crucial for us to determine distance threshold (blind spot) of sensors to be ignored. Luckily this is already done inside the library, where any distance below 10 cm or 20 cm is return as 9 cm and 19 cm for the corresponding IR sensors.

To improve the accuracy and precision of sensor reading, a sample of 50 sensor readings with delay of 1.5ms in between are collected. Median filtering with help of ArduinoSort.h library for sorting is then applied on these readings to output the most stable readings for use in dead recognition. This practice greatly reduces the false detection of obstacles, a common cause of phantom blocks during maze exploration.

# 6. Calibration for Exploration

For this project 2 methods of calibration have been devised: front calibration and right wall aligning calibration. The former makes sure the robot is parallel to a wall/ obstacle in front and the latter makes sure the robot is parallel to the wall on its right side. During the maze exploration, the ability to realign to a wall straightens the robot and prevents the robot from steering away from its original course.

Each calibration method has 2 phases: angle alignment and distance alignment.

* Angle alignment: reading from a pair of sensors – either the front and back sensors facing to the right OR the leftmost and rightmost sensors facing the front, are taken. The difference between these 2 values are calculated. Should this value be more than the tolerance of 0.03 cm , the robot will rotate left and right continuously in a short period until the robot is parallel to the walls/ obstacles.
* Distance alignment: Sensors value are once again taken, robot will then move forward and backward (for front distance calibration) or rotate right, repeat the same thing and rotate left (for right wall hugging calibration) until the distance of either one sensor and hence position of robot is within the 3x3 grid range away from the obstacles.

Calibrations during exploration are crucial for the robot to finish its maze solving mission as the closed loop control system described above cannot fully remove errors, but only somewhat reduce error. Hence frequent positional adjustment can take a robot far in its mission