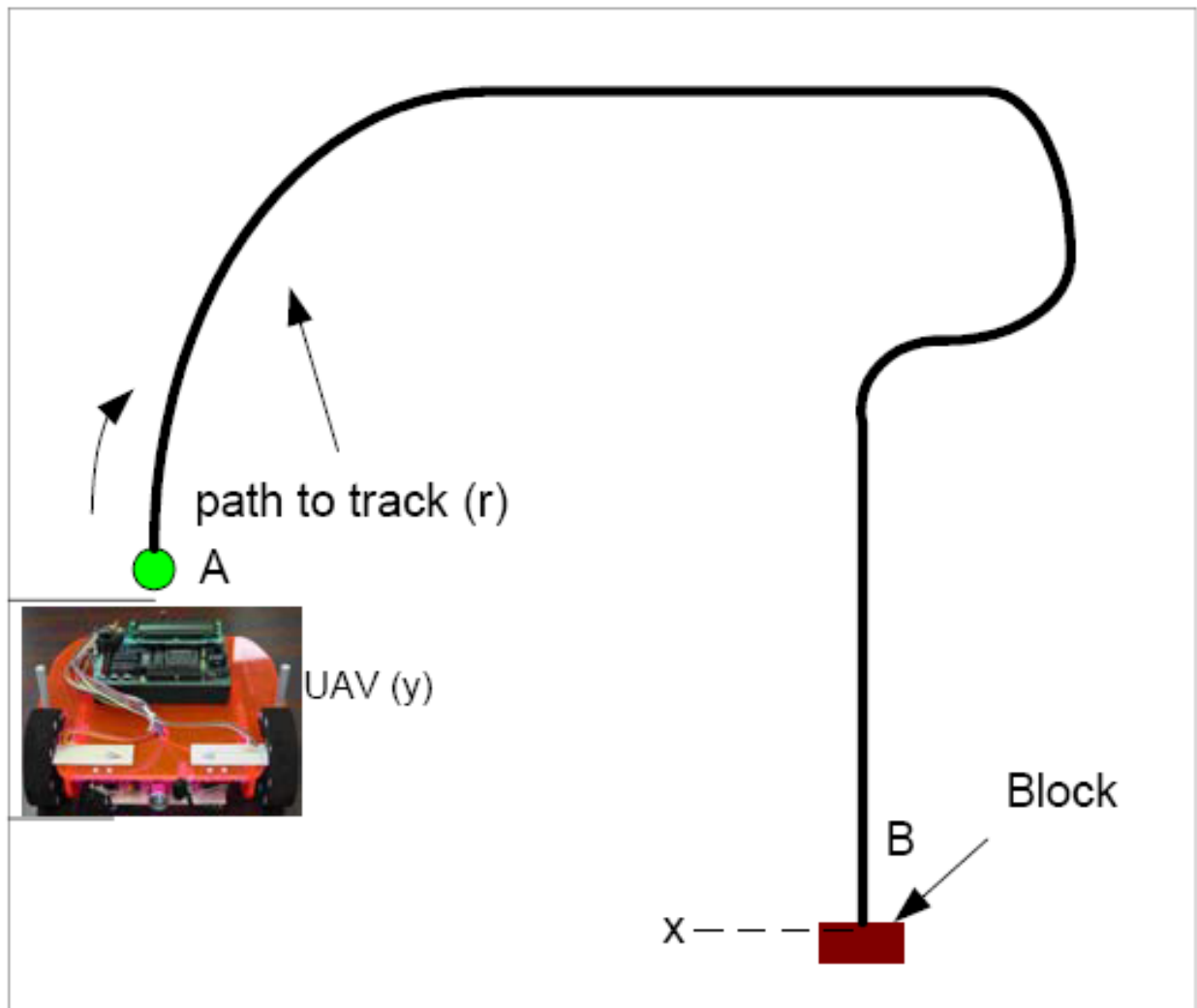# Line Follower(PD Control)

Why make a line follower?
Ans:As human beings , we respond to stimuli.Our actions are due to the feedback given by our sensory organs.A line follower is the simplest way to demonstrate that.

Sensing a line and maneuvering the robot to stay on course, while constantly correcting wrong moves using feedback mechanism forms a simple yet effective closed loop system.

*Before you go through this tutorial , it is highly recommended you go through the tutorial on microcontrollers and have a fair idea on the basics. Uc used in the tutorial is based on avr architecture.

Advantages of uc :

In the tutorial on line following using logic gates etc we have observed that we switch off one motor one let the other move till the bot corrects itself.There are a few disadvantages in this :

1.Your intensity of correction completely depends on the distance between the two motors.Since one is stationary and the other is moving , their distance forms the radius of curvature that the machine takes while correcting itself.Now the path can be completely irregular.For every irregular curve you will get very inefficient line following.You machine will simply wobble along its path wasting power,time etc.

With a uc, you can actuate a motor to run at a certain rpm and the other to move at another .This can generate any radius of curvature and give the most efficient line followers as you see on youtube videos.
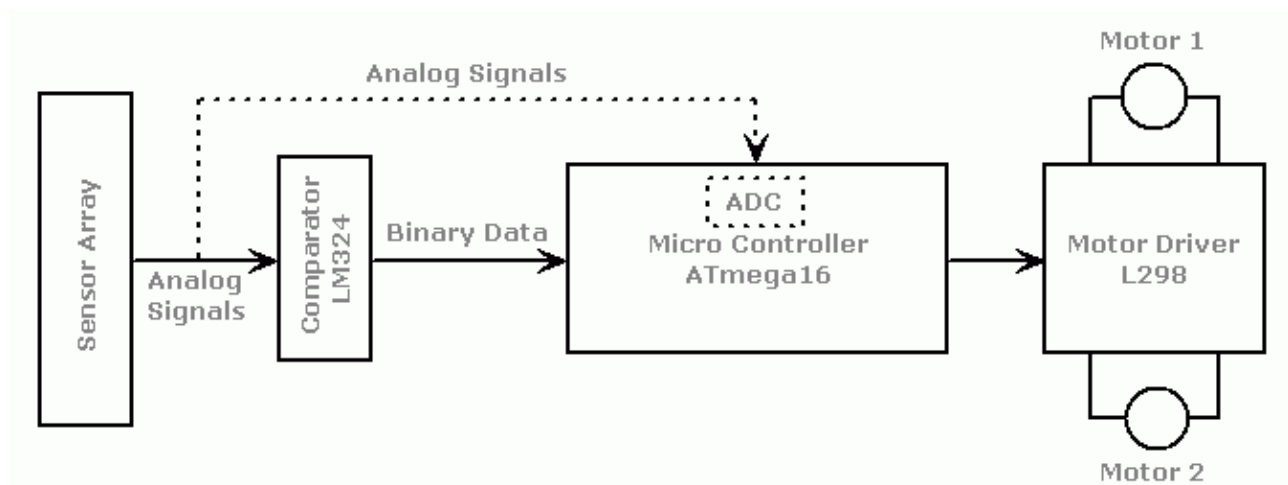

Basic Block Diagram

*You can use external LM324 when you are using a large array of sensors and assign weights to each sensor(explained ahead).But it is **recommended to use the ADC of the uc.** You can also use an l293d or some other driver in place of the l298.

<div align="center">Building the line follower :</div>

Before we begin , lets get to know a few terms that will be used ahead :

Error : Your machine being exactly on the line with the respective sensors being on the either side of the line denotes zero error.While if the sensory array is on one side of the line denotes positive error and on the other side

denotes negative error.
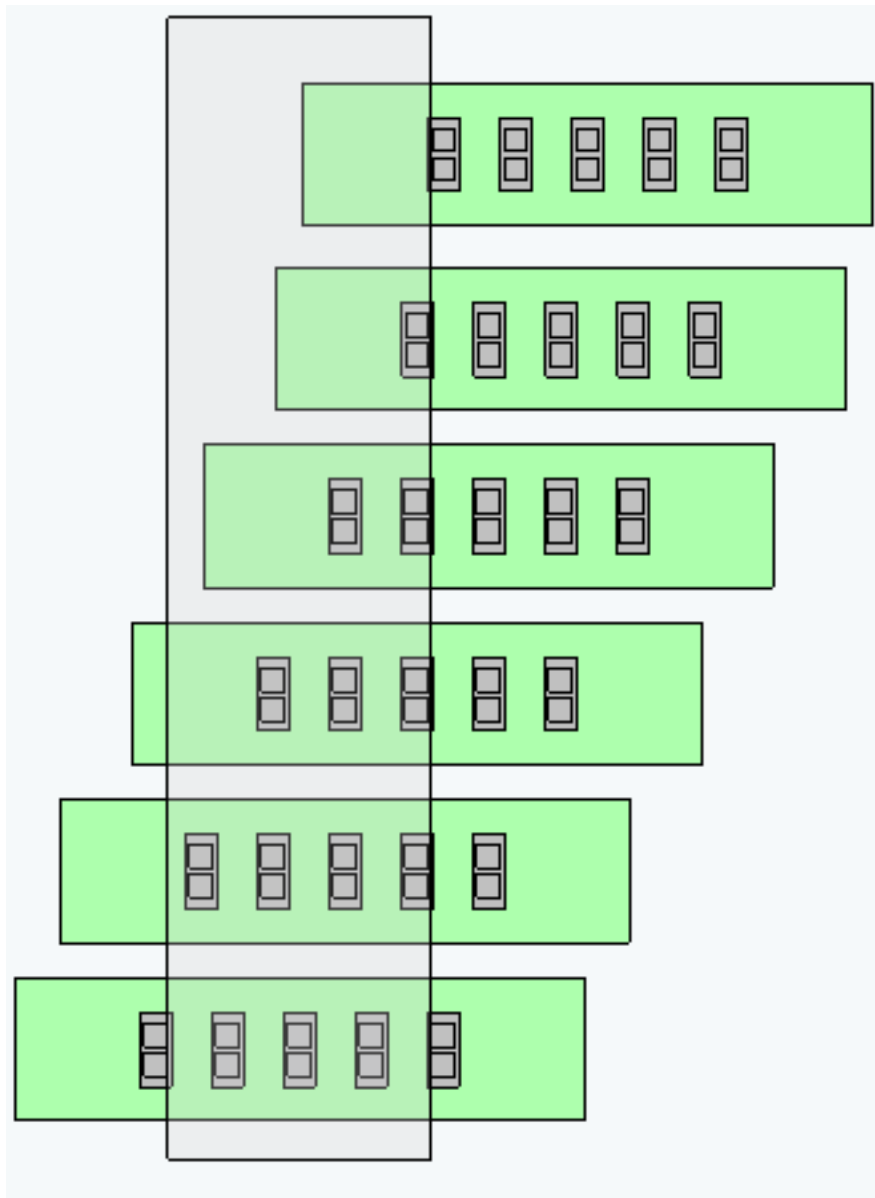
Correction : Denotes the magnitude of the parameter needed to bring the system to zero error.

So our job basically boils down to finding the error term and the correction term and to reduce the error to zero.

Sensors :
It has been observed that to attain efficient line following for any track we need not more than two sensors on either sides of the line for motors with speed within 500rpm(Lamington Dc motors).But it is upto the maker on the number of sensors , the quality of the motor etc.
1.Please refer to the tutorial for line following for beginners to make the

sensors and placing the sensors on the board.

2.Do not feed the values to the comparator(LM324) and take the analogue values to the uc starightaway if using ADC of uc.(Refer datasheet of your uc).

3.Every photodiode behaves differently from the other photodiode.So we have to claibrate our sensors(in our code) so that the logical part of the code deals with the same range for every sensor.

Driver hardware:

1.Pwm signals to the input pins of the driver will drive the motor at a rpm dependent on the duty cycle of the pwm.

2.We can configure different pins of the uc to generate pwm signals or use pins that are pwm generators.

3.We can send pwm signals to the enable pins of the driver or to the input signals of the driver.It does not make much of a difference just that in former case we need two pwm generators for bidirectional control of two motors(one each) and for the latter case we need four pwm channels for bidirectional control of two motors(two for each).

Algorithm:

Note: This discussion of PID is intended for hobby robotics and line following. It is not a perfect example of PID and is simplified in some aspects to make it easier to understand. If you are looking for more detailed information, perform a Google search on "PID controller" for numerous sources of information.

We sample our sensor inputs and generate ADC values of each sensor.We multiply them by their repective weights and add to get a sum for each: left and right array of sensors.The net sum will be zero if the line follower is on track and if offtrack will generate a +ve/-ve error term.

 Now we begin with the discussion on pid
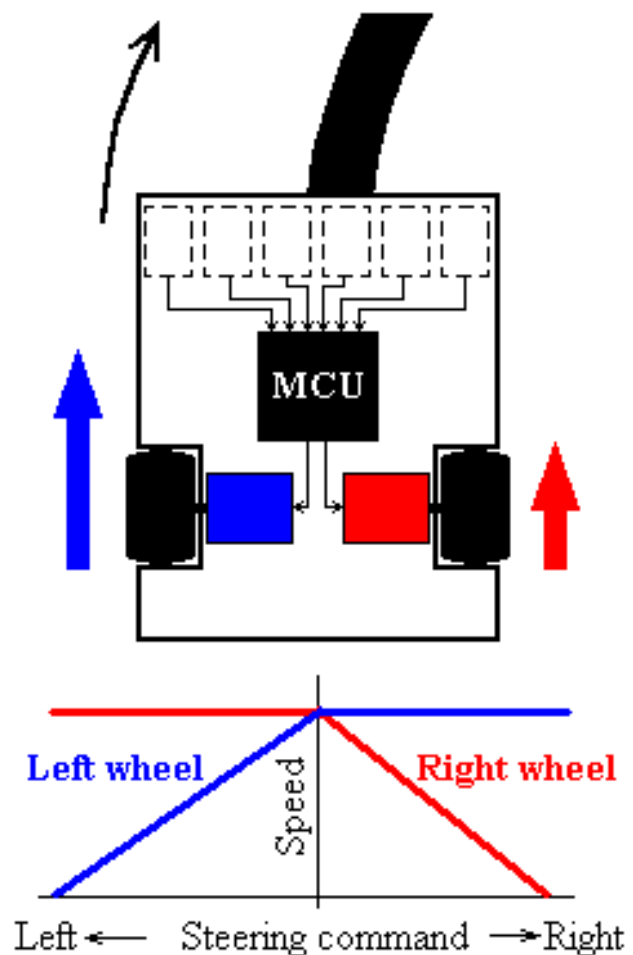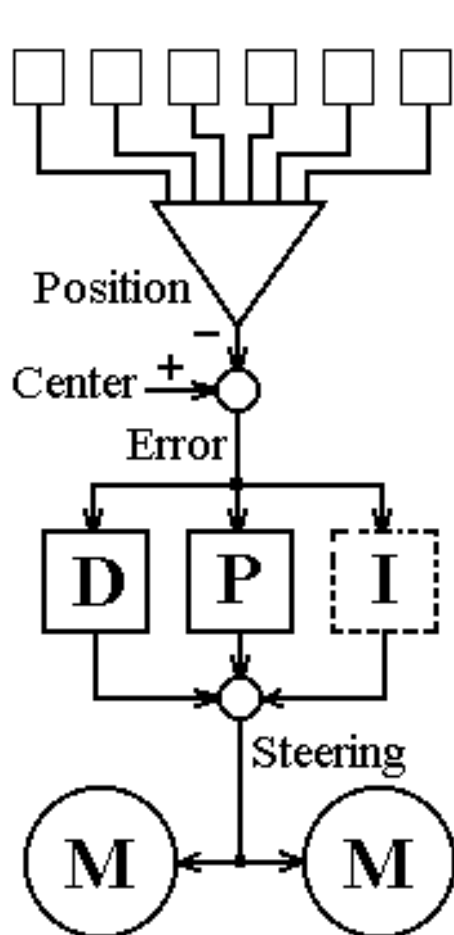
A **PID** controller is a mathematically-based routine that processes sensor data and uses it to control the direction (and/ or speed) of a robot to keep it on course.

Proportional - measures how far your robot is away from the line. Proportional is the foundation for capturing the robots position using sensors. The more granular the sensor data is, the more accurately you can measure the robots position over the line.

Integral - measure the accumulated **Error** over time. The Integral value increases while the robot is not centered over the line. The longer the robot is not centered over the line, the higher the Integral value becomes.

Derivative - Measures the rate the robot is moving left-to-right or right-to-left. The faster the robot moves side-to-side, the higher the Derivative value is.

P-Factor (Kp) - is a constant value used to increase or decrease the impact of Proportional

I-Factor (Ki) - is a constant value used to increase or decrease the impact of Integral

D-Factor (Kd) - is a constant value used to increase or decrease the impact of Derivative

PID Formula

Proportional

Target Position minus the Measured Position = Difference
Difference times Kp= Proportional

Integral

Integral plus Difference = Integral (Integral stores the accumulated Difference value)
Integral times Ki = Integral

Derivative

Difference minus Previous Difference = Rate of Change
Rate of Change times Kd = Derivative (this is sometimes divided by the interval or time between measurements)

Proportional plus Integral plus Derivative = Control value used to adjust the robot's motion

That represents the math of PID but the real secret of it's usefulness is in tuning the PID controller to match the physical characteristics of your robot.

The control equation used is
$$\text{Correction} = (Kp*Error) + Kd*(error - error\_) + Ki* (^k_0 \Sigma err)$$

Tuning PID

Once you have PID running in your robot, you will probably notice that it still doesn't follow the line properly. It may even perform worse than it did with just proportional! The reason behind this is you haven't tuned the PID routine yet. PID requires the Kp, Ki and Kd factors to be set to match your robot's characteristics and these values will vary considerably from robot to robot. Unfortunately, there is no easy way to tune PID. It requires manual trial and error until you get the desired behavior. There are some basic guidelines that will help reduce the tuning effort.

1. Start with Kp, Ki and Kd equalling 0 and work with Kp first. Try setting Kp to a value (generally less than 10) and observe the robot. The goal is to get the robot to follow the line even if it is very wobbly. If the robot overshoots and loses the line, reduce the Kp value. If the robot cannot navigate a turn or seems sluggish, increase the Kp value. Try your level best to not not introduce Kd or Ki and fine tune Kp.
2. Once the robot is able to somewhat follow the line, assign a value (generally less than 1) to Kd (skip Ki for the moment). Try increasing this value until you see less wobble.
3. Generally finely tuned values of Kp and Kd is sufficient for pretty efficient line following and Ki need not be introduced.
4. Once the robot is fairly stable at following the line, assign a value of .5 to 1.0 to Ki. If the Ki value is too high, the robot will jerk left and right quickly. If it is too low, you won't see any perceivable difference. Since Integral is cumulative, the Ki value has a significant impact. You may end up adjusting it by .01 increments.
5. Once the robot is following the line with good accuracy, you can increase the speed and see if it still is able to follow the line. Speed affects the PID controller and will require retuning as the speed changes.

Badly tuned

Finely Tuned

Links for reference :
*See video of bot only with P control and bot with PID :
http://elm-chan.org/works/ltc/report.html
http://forum.pololu.com/viewtopic.php?f=23&t=521
http://www.robotroom.com/Jet.html
http://www.chibots.org/index.php?q=node/339
http://www.seattlerobotics.org/encoder/200106/linerigel.html