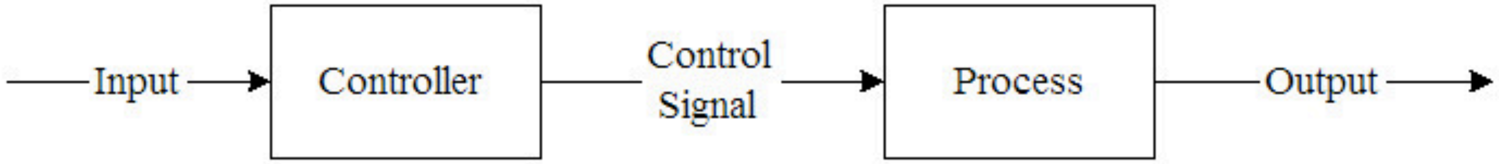
KIDOBOTIKZ™
Engineering Real Engineers

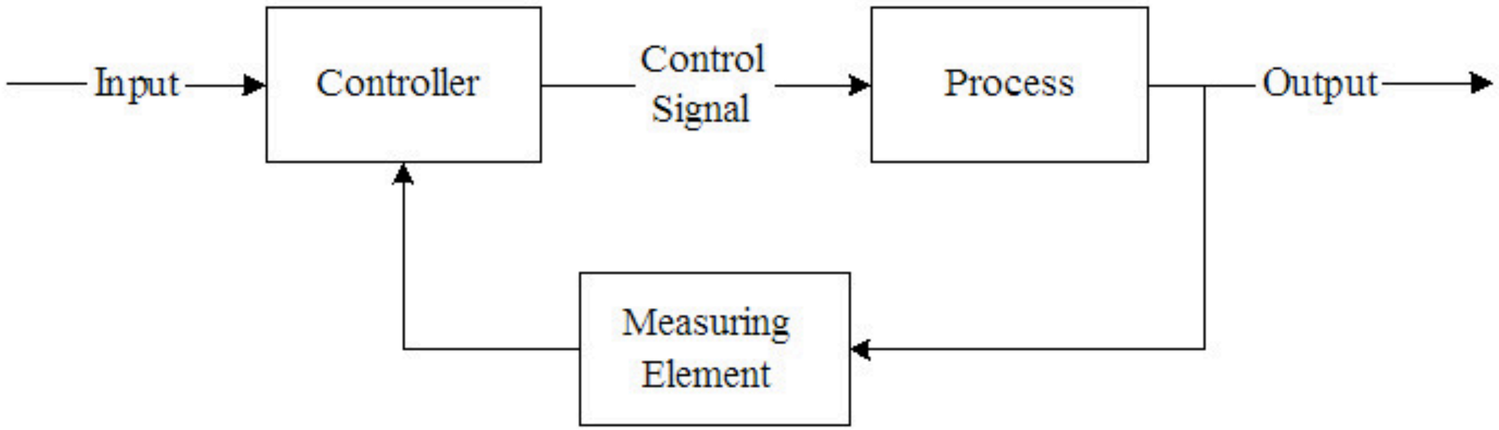
PID CONTROL

LINE TRACER ROBOT

TYPES OF SYSTEMS



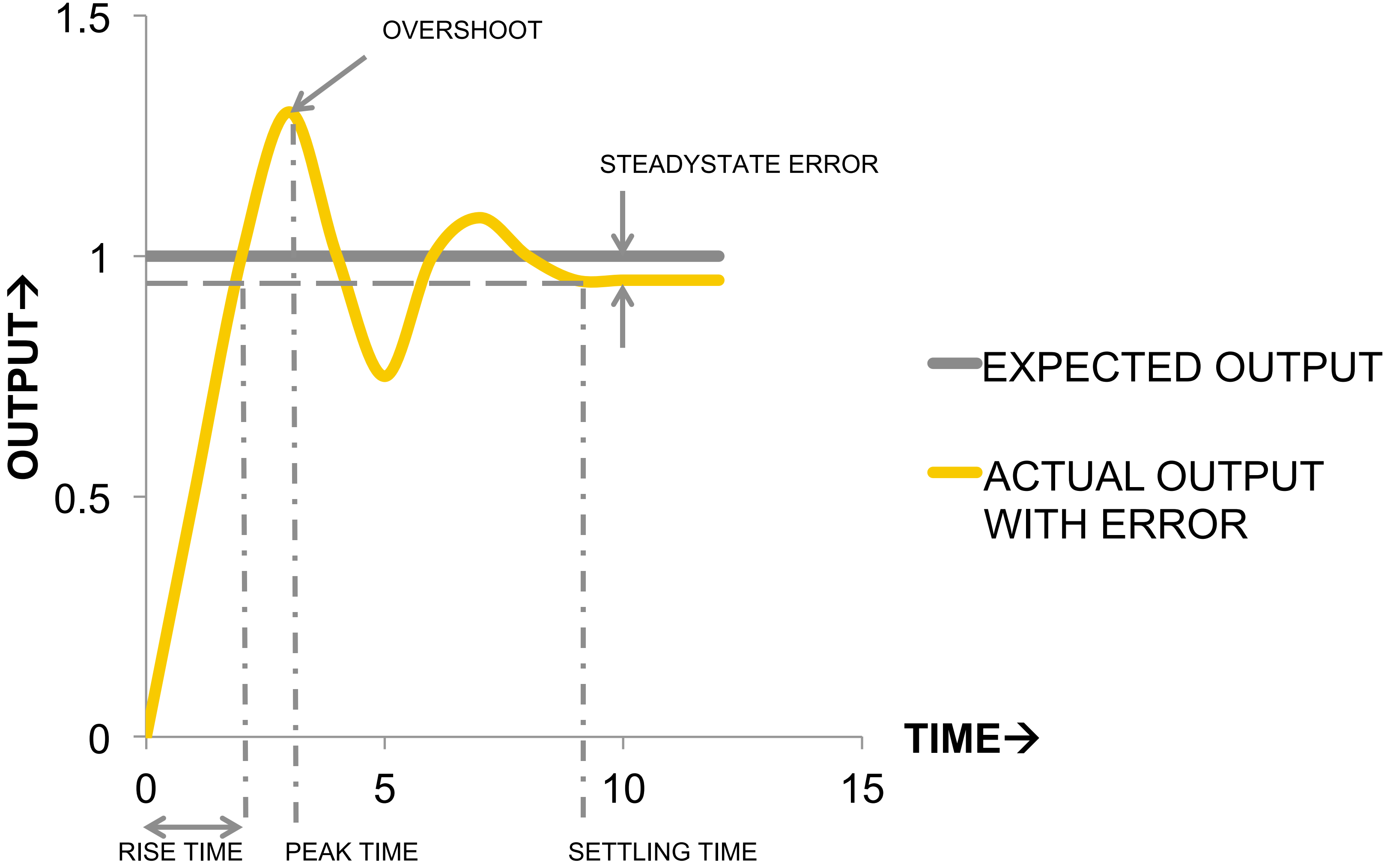
Open Loop System



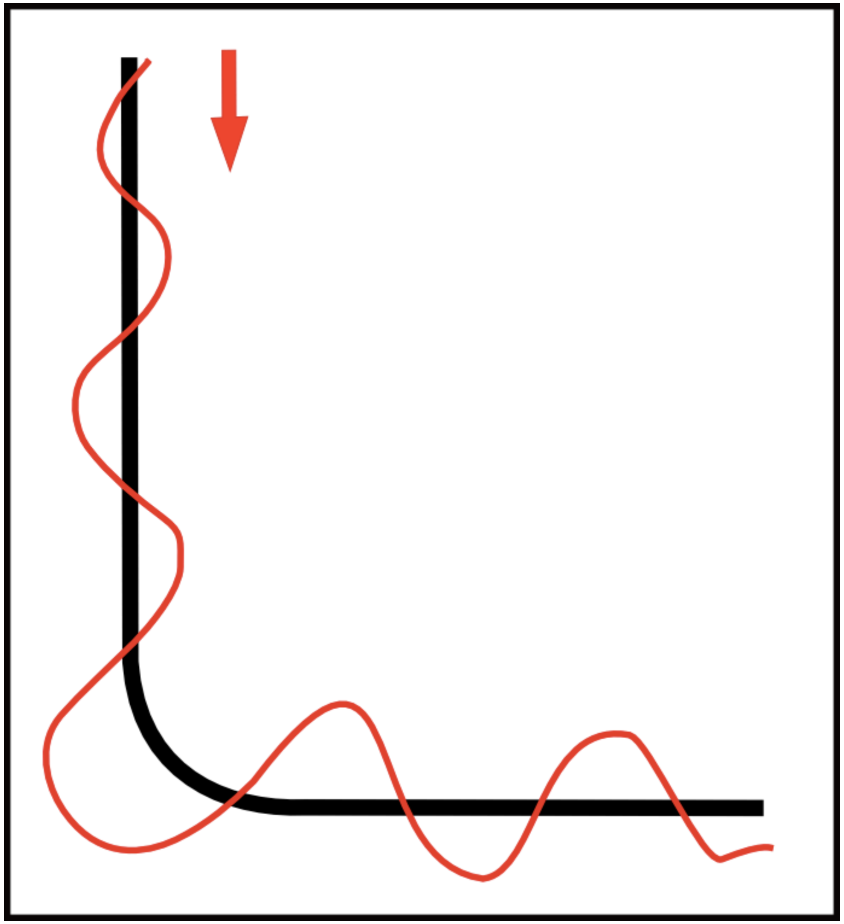
Closed Loop System

KiDOBOTiKZ™
Engineering Real Engineers

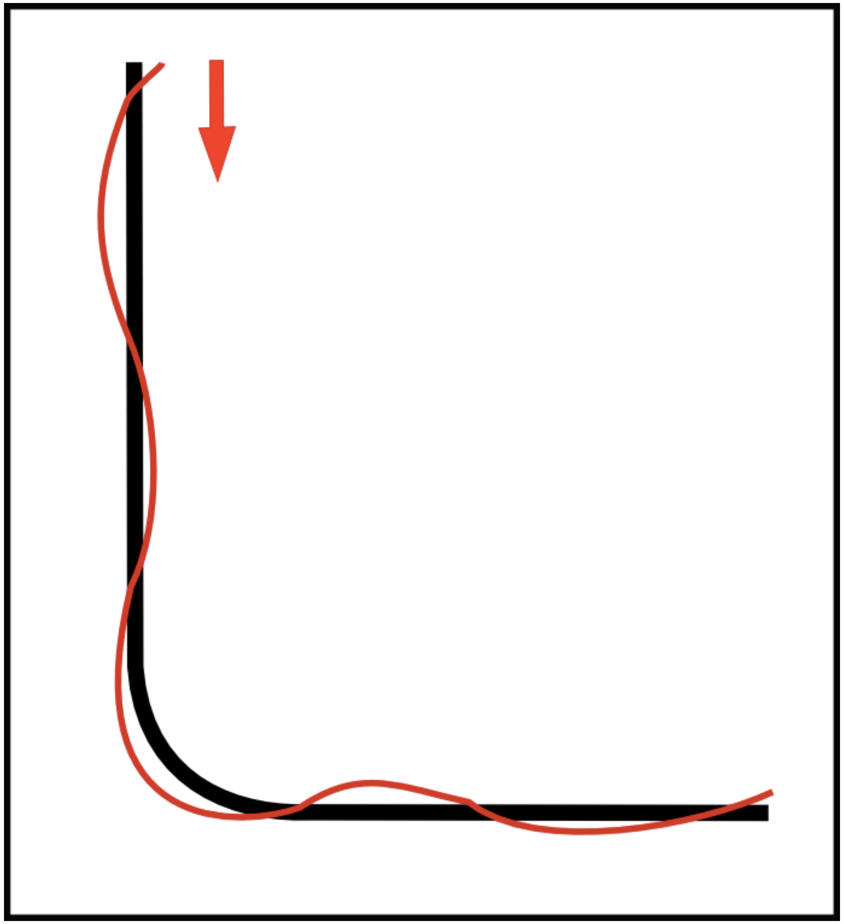
INSIGHT INTO AN ERROR



CONVENTIONAL ROBOT



PID BASED ROBOT



TERMS

Target – It is the position you want the line follower to always be(or try to be),that is, the center of the robot.

Current Position – It is the current position of the robot with respect to the line.

Error - It is the difference between the current position and the target. It can be negative, positive or zero.

Proportional – It tells us how far the robot is from the line like – to the right , to the extreme right, to the left or a little to the left. Proportional is the fundamental term used to calculate the other two.

Integral – It gives the accumulated error over time. It tells us if the robot has been on the line in the last few moments or not.

Derivative – It is the rate at which the robot oscillates to the left and right about the line.

PID

Proportional – It helps to reduce the error

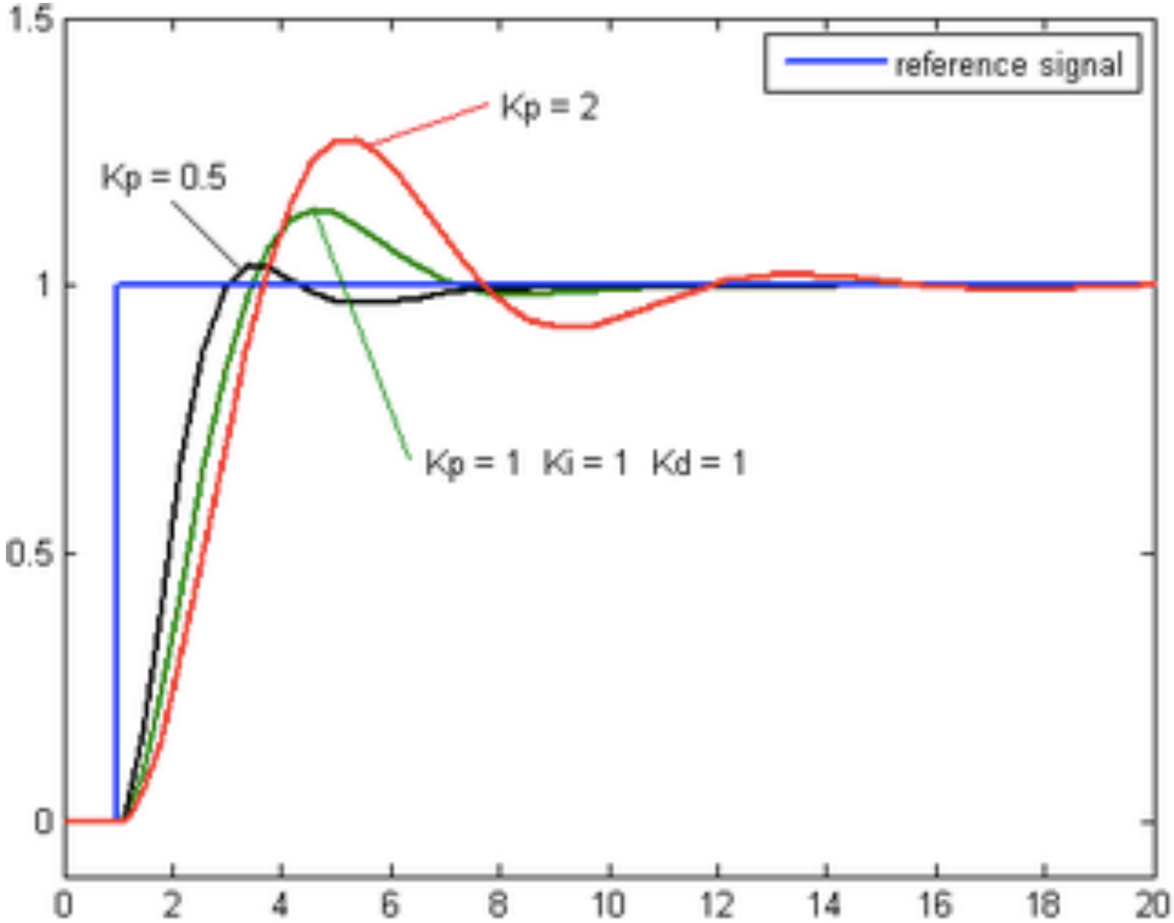
Integral – It helps to reduce the error faster

Derivative – it helps to predict the error and avoid it from happening

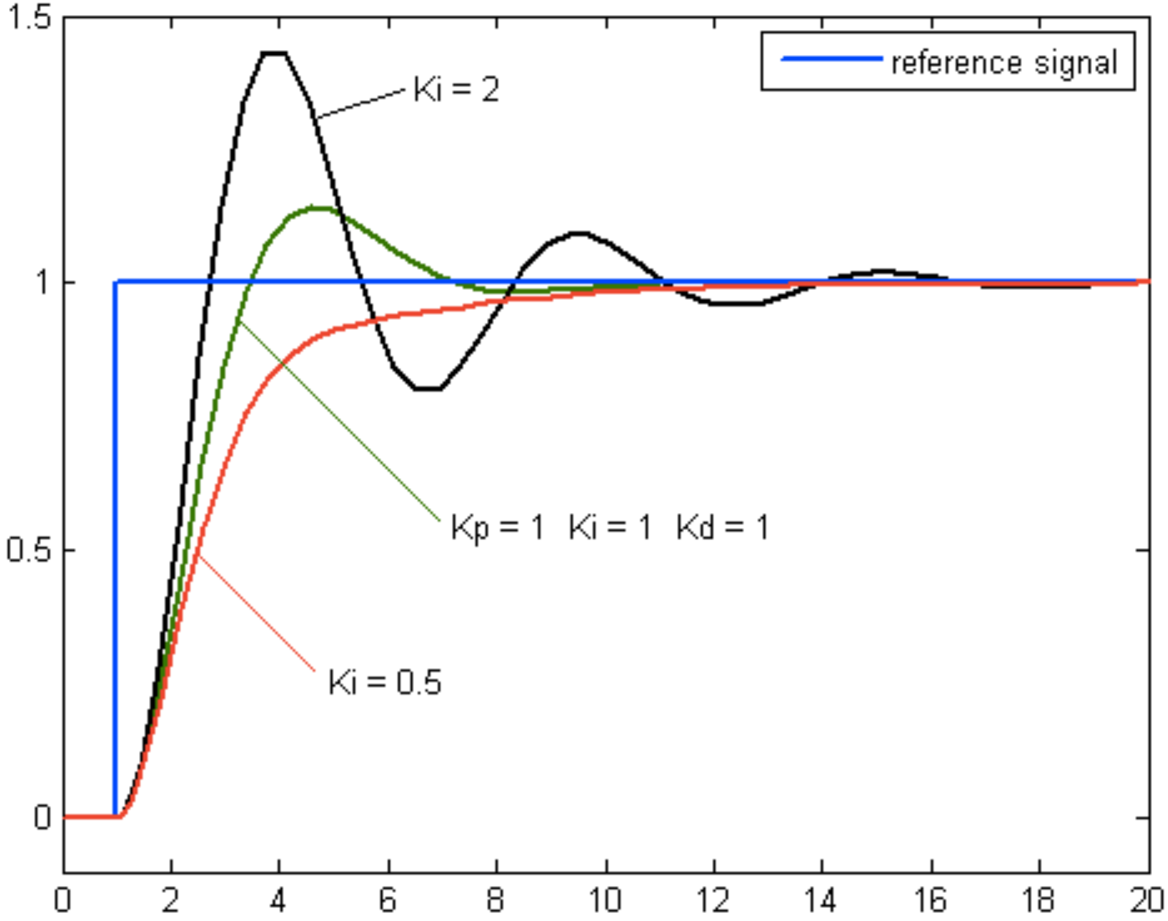
PID

What the controller does is first calculate the current position. Then calculate the error based on the current position. It will then command the motors to take a hard turn, if the error is high or a small turn if the error is low. Basically, the magnitude of the turn taken will be proportional to the error. This is a result of the Proportional control. Even after this, if the error does not decrease or decreases slowly, the controller will then increase the magnitude of the turn further and further over time till the robot centers over the line. This is a result of the Integral control. In the process of centering over the line the robot may overshoot the target position and move to the other side of the line where the above process is followed again. Thus the robot may keep oscillating about the line in order to center over the line. To reduce the oscillating effect over time the Derivative control is used.

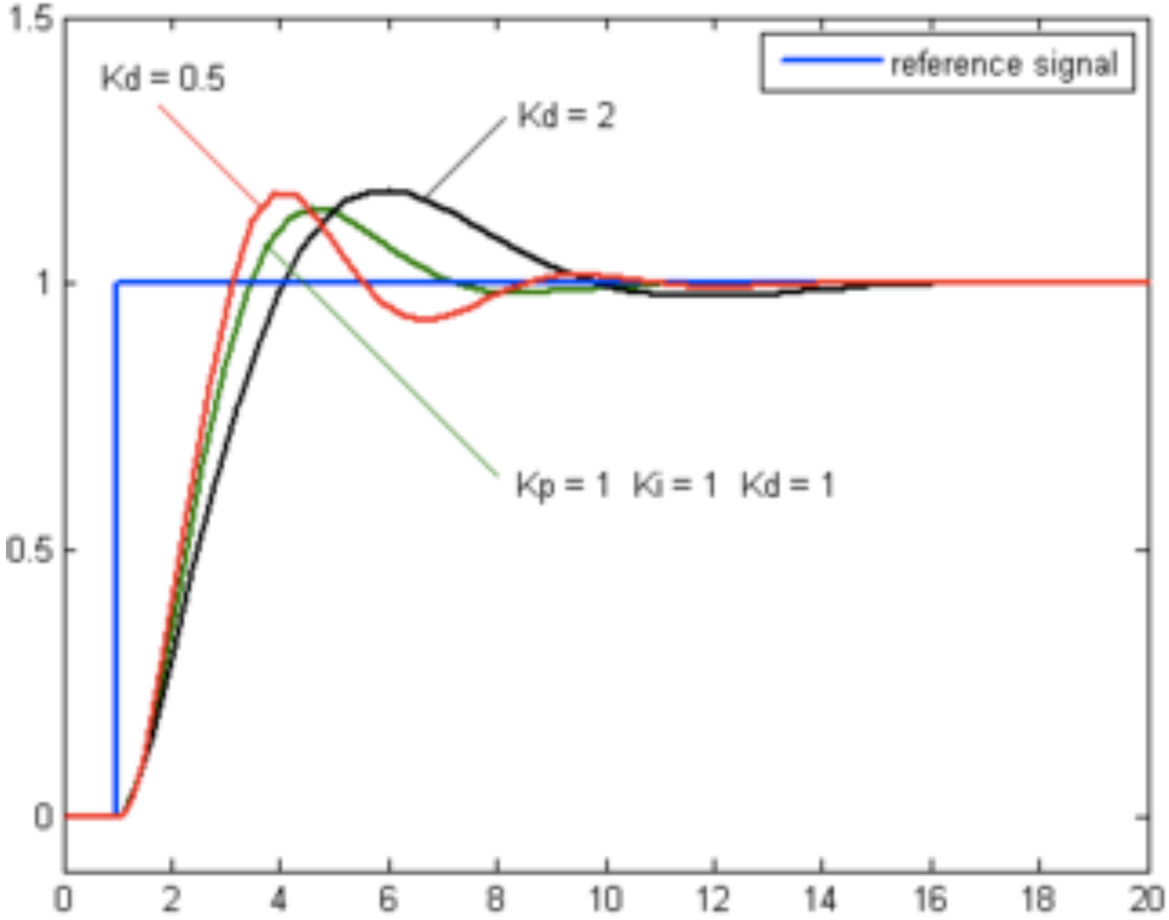
EFFECT OF PROPORTIONAL



EFFECT OF INTEGRAL



EFFECT OF DERIVATIVE



KiDOBOTiKZ™
Engineering Real Engineers

PID CALCULATION

```
Error = target_pos – current_pos          //calculate error
P = Error * Kp          //error times proportional constant gives P
I = I + Error           //integral stores the accumulated error
I = I * Ki              //calculates the integral value
D = Error – Previos_error     //stores change in error to derivate
Correction = P + I + D
```

The next step is to add this correction term to the left and right motor speed.

INITIALIZATION

```
float Kp=0,Ki=0,Kd=0;
float error=0, P=0, I=0, D=0, PID_value=0;
float previous_error=0;
int sensor[5]={0, 0, 0, 0, 0};
int initial_motor_speed=100;

void read_sensor_values(void);
void calculate_pid(void);
void motor_control(void);

void setup()
{
 pinMode(9,OUTPUT); //Left Motor Pin 1
 pinMode(10,OUTPUT); //Left Motor Pin 2
 pinMode(5,OUTPUT); //Right Motor Pin 1
 pinMode(6,OUTPUT); //Right Motor Pin 2
Serial.begin(9600); //Enable Serial Communications
}
```

```
void read_sensor_values()
{
  sensor[0]=digitalRead(A0);
  sensor[1]=digitalRead(A1);
  sensor[2]=digitalRead(A2);
  sensor[3]=digitalRead(A3);
  sensor[4]=digitalRead(A4);

  if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&&(sensor[4]==0)&&(sensor[4]==1))
  error=4;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&&(sensor[4]==1)&&(sensor[4]==1))
  error=3;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&&(sensor[4]==1)&&(sensor[4]==0))
  error=2;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==1)&&(sensor[4]==1)&&(sensor[4]==0))
  error=1;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==1)&&(sensor[4]==0)&&(sensor[4]==0))
  error=0;
  else if((sensor[0]==0)&&(sensor[1]==1)&&(sensor[2]==1)&&(sensor[4]==0)&&(sensor[4]==0))
  error=-1;
  else if((sensor[0]==0)&&(sensor[1]==1)&&(sensor[2]==0)&&(sensor[4]==0)&&(sensor[4]==0))
  error=-2;
  else if((sensor[0]==1)&&(sensor[1]==1)&&(sensor[2]==0)&&(sensor[4]==0)&&(sensor[4]==0))
  error=-3;
  else if((sensor[0]==1)&&(sensor[1]==0)&&(sensor[2]==0)&&(sensor[4]==0)&&(sensor[4]==0))
  error=-4;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&&(sensor[4]==0)&&(sensor[4]==0))
    if(error==-4) error=-5;
    else error=5;

}
```

```
void calculate_pid()
{
    P = error;
    I = I + Error;
    D = error-previous_error;

    PID_value = (Kp*P) + (Ki*I) + (Kd*D);

    previous_error=error;
}
```

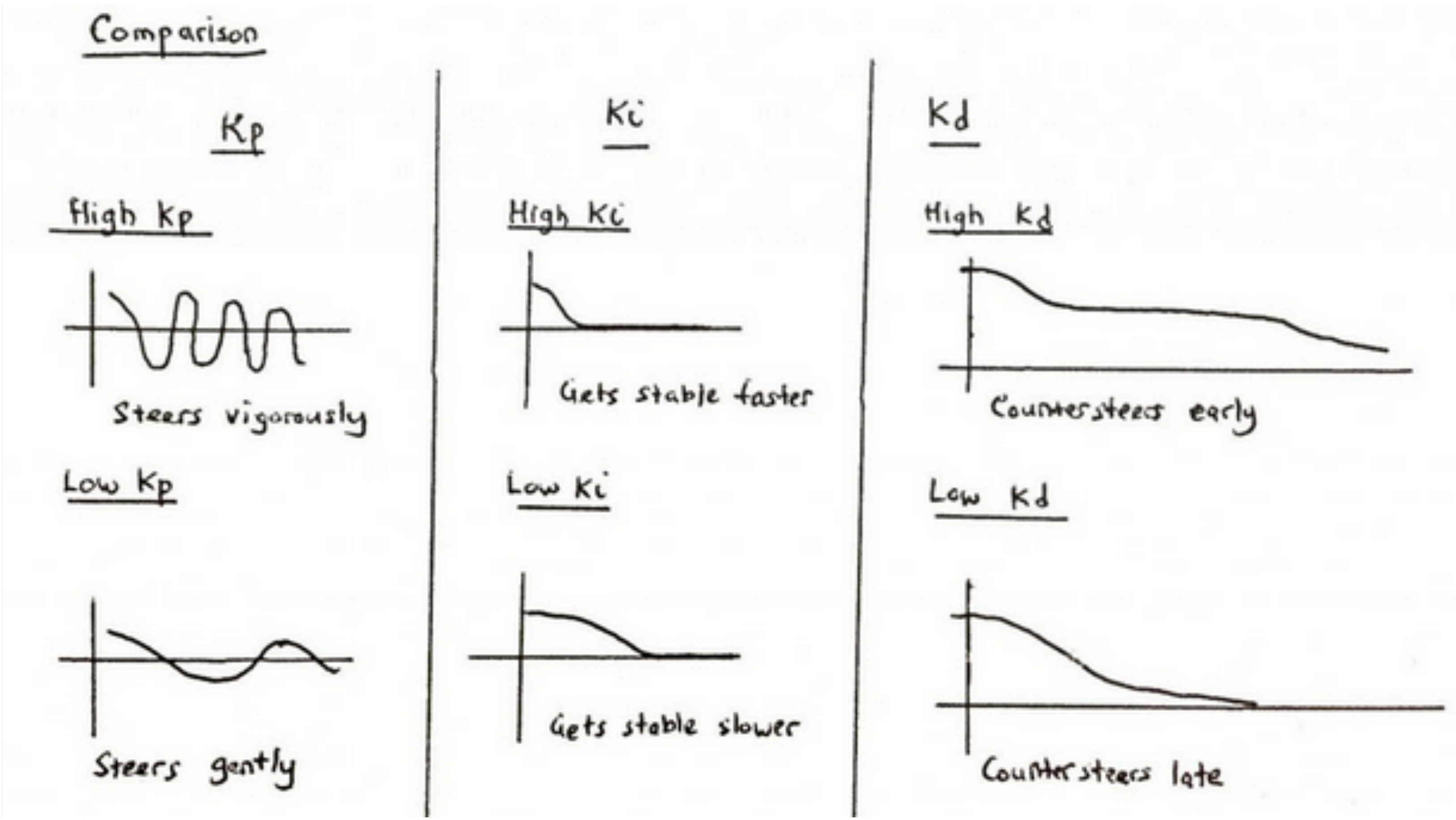
```
void motor_control()
{
   // Calculating the effective motor speed:
   int left_motor_speed = initial_motor_speed-PID_value;
   int right_motor_speed = initial_motor_speed+PID_value;

   // The motor speed should not exceed the max PWM value
   constrain(left_motor_speed,0,255);
   constrain(right_motor_speed,0,255);

   //following lines of code are to make the bot move forward
   /*The pin numbers and high, low values might be different
   depending on your connections */

   analogWrite(9,initial_motor_speed-PID_value);   //Left Motor Speed
   analogWrite(5,initial_motor_speed+PID_value);  //Right Motor Speed
   digitalWrite(10,LOW);
   digitalWrite(6,LOW);

}
```

```
void loop()
{
    read_sensor_values();
    calculate_pid();
    motor_control();
}
```

PID VALUES

- Start with Kp, Ki and Kd equalling 0 and work with Kp first. Try setting Kp to a value of 1 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.
- Once the robot is able to somewhat follow the line, assign a value of 1 to Kd (skip Ki for the moment). Try increasing this value until you see lesser amount of wobbling.
- Once the robot is fairly stable at following the line, assign a value of 0.5 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.
- 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.

PID VALUES


Comparison
Kp
Ki
Kd
High Kp
Steers vigorously
Low Kp
Steers gently
High Ki
Gets stable faster
Low Ki
Gets stable slower
High Kd
Countersteers early
Low Kd
Countersteers late

KIDOBOTIKZ™

Engineering Real Engineers

WWW.KIDOBOTIKZ.COM