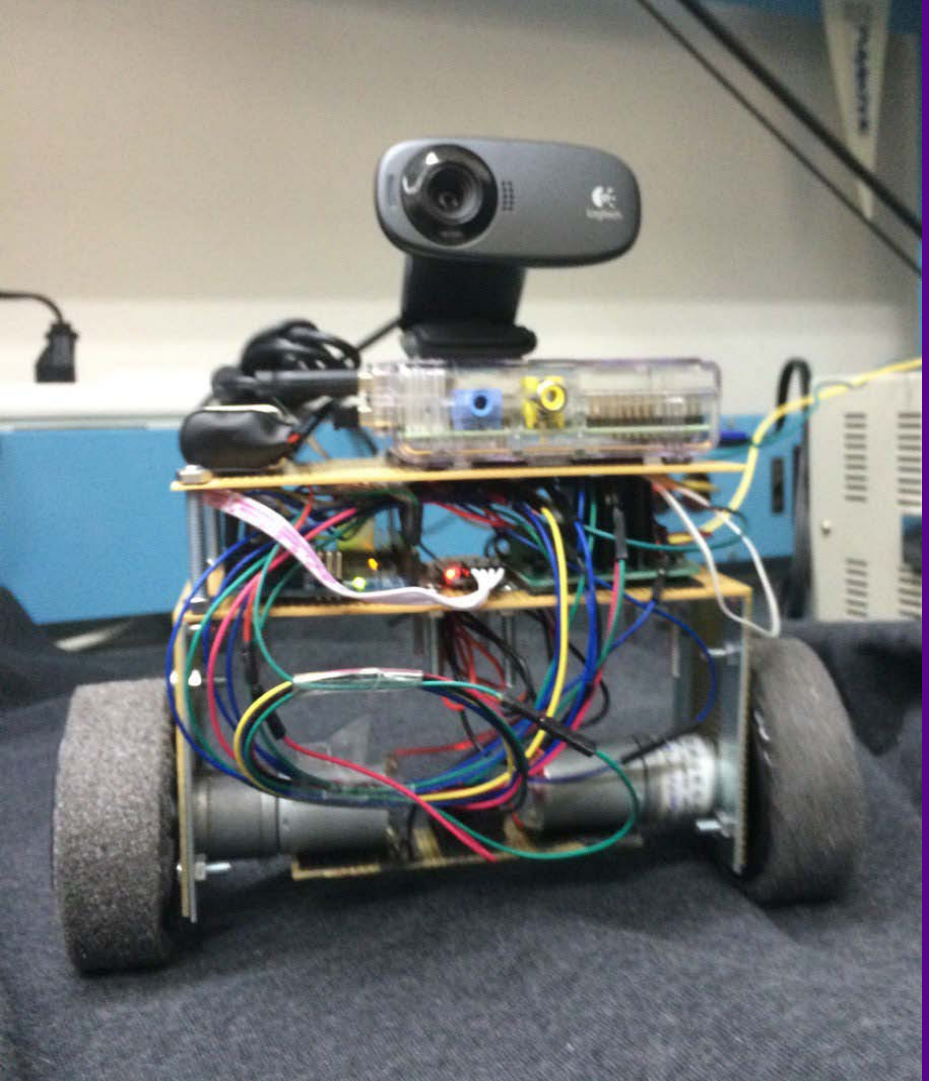
NYU POLYTECHNIC SCHOOL OF ENGINEERING

Vision enabled Self Balancing Segway Robot

Craig
Parth
Santosh

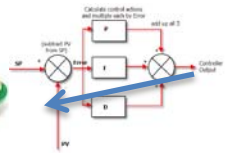


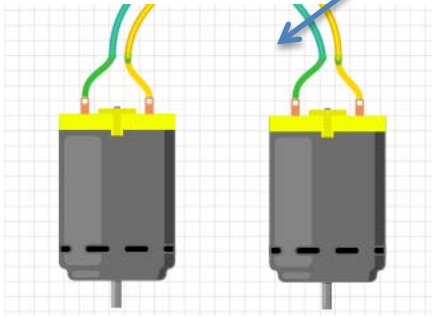
NYU POLYTECHNIC SCHOOL OF ENGINEERING

Today’s Agenda

- Over view
- PID Controller
- Self Balancing Code (Arduino)
- Trained signs
- CV code (RPi)
- Yet to be implemented (To do list)
- Video demo

Over view





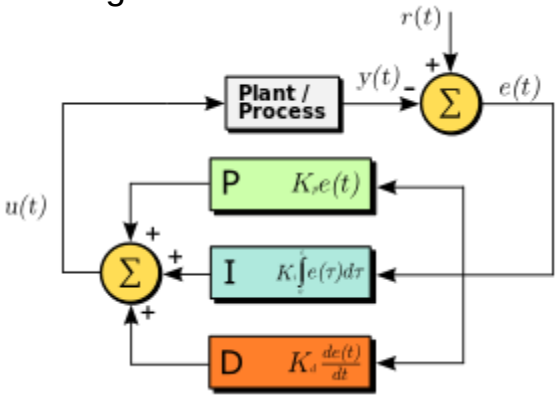
PID THEORY

The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining as the controller output, the final form of the PID algorithm is:

Where

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

Kp: Proportional gain, a tuning parameter
Ki: Integral gain, a tuning parameter
Kd: Derivative gain, a tuning parameter
e(t): Error e(t)=SV-PV
t: Time or instantaneous time (the present)



TUNING PARAMETERS

```
double aggKp=49, aggKi=1.6, aggKd=0.8;
double consKp=48,consKi=3.5, consKd=0.6;
while (!mpuInterrupt && fifoCount < packetSize) {
        Input = ypr[1] * 180/M_PI;

        double gap = abs(Setpoint-Input); //distance away from setpoint
        if(gap<5)
        {  //we're close to setpoint, use conservative tuning parameters
          myPID.SetTunings(consKp, consKi, consKd);
        }
        else
        {
          //we're far from setpoint, use aggressive tuning parameters
          myPID.SetTunings(aggKp, aggKi, aggKd);
        }
```

The Beginner's PID

Here's the PID equation as everyone first learns it:

$$\text{Output} = K_P e(t) + K_I \int e(t)\,dt + K_D \frac{d}{dt} e(t)$$

$$\text{Where : } e = \text{Setpoint} - \text{Input}$$

This leads pretty much everyone to write the following PID controller:

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
void Compute()
{
   /*How long since we last calculated*/
   unsigned long now = millis();
   double timeChange = (double)(now - lastTime);

   /*Compute all the working error variables*/
   double error = Setpoint - Input;
   errSum += (error * timeChange);
   double dErr = (error - lastErr) / timeChange;

   /*Compute PID Output*/
   Output = kp * error + ki * errSum + kd * dErr;

   /*Remember some variables for next time*/
   lastErr = error;
   lastTime = now;
}

void SetTunings(double Kp, double Ki, double Kd)
{
   kp = Kp;
   ki = Ki;
   kd = Kd;
}
```

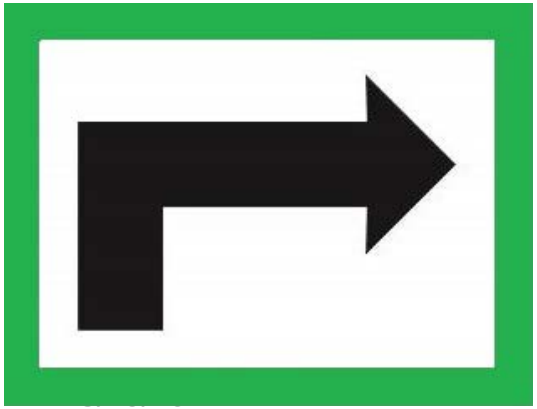
revision1

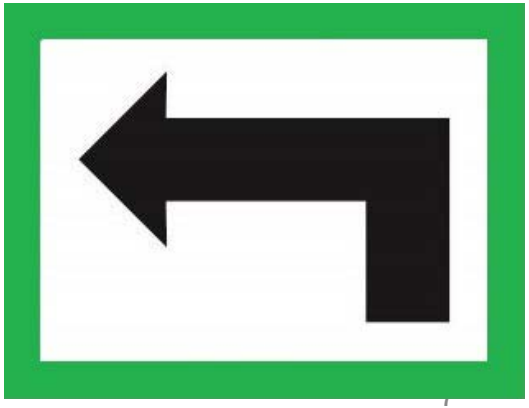
```
void driveMotor(Motor motor, Direction dir, double pwm)
{
        int dirPin, dirPinl, pwmPin;
        if(motor == leftMotor)
        {
                dirPin = leftMotorA;
                dirPinl = leftMotorAl;
                pwmPin = leftMotorB;
        }
        else if(motor == rightMotor)
        {
                dirPin = rightMotorA;
                dirPinl = rightMotorAl;
                pwmPin = rightMotorB;
        }
        else return;

        pwm = round(constrain(pwm,0.0,255.0));

        if(dir == forward)
        {
                digitalWrite(dirPin, LOW);
                digitalWrite(dirPinl, HIGH);
                analogWrite(pwmPin, (int)pwm);
        }
        else if(dir == still)
        {
          digitalWrite(dirPin,LOW);
          digitalWrite(dirPinl,LOW);
          analogWrite(pwmPin, 0);
        }
}
```

Trained Signs





- **Find Image contours**

Opencv method: findContours

- **Find approx rectangular contours**

Opencv method: approxPolyDP

- **Isolate sign and correct perspective**

Opencv method: WrapPerspective

- **Binarize and Match**

Bitwise XOR

- **RPi and Arduino**

Wiring library

- **Result**

It worked pretty well on PC but when ported to RPi its laggy even with modest overclocking.

To do list

- **As of now the bot is balancing and if we try to drive it, it looses stability.**

Video demo


Thank you