Arduino Essentials: PID Control using Beard Library
(robotics_crash_course/arduino_code/libraries/PID_beard/PID_beard.cpp)

3 Examples

(1) Controlling distance (linear displacement)
(2) Controlling angular velocity
(3) Controlling angle (angular displacement)

```
  //Initialize PID configuration struct (order is important here)
PID_control_config_t config =
{
  .kp = 10,                  //set proportional gain
  .ki = 1,                   //set integral gain
  .kd = 1,                   //set derivative gain
  .lowerLimit = -255,        //set limit(s) for output of controller
  .upperLimit = 255,
  .sigma = 0.1,              //for the filter
  .ts = 0.05;                //same as dt (timestep)
  .errorDotEnabled = true,   //allows for derivative calculation
  .antiWindupEnabled = true //allows for anti-windup on integrator
};

  //timing variables needed for consistent dt
  //allows for consistent integral and derivative calculations
unsigned long prev, cur = 0;
unsigned long dt = 50; //in milliseconds

  //state machine variable
int state = 0; //start in state 0

  //for your controller, set a desired value (goal of the routine)
(1) float desDist = 30;  //desired cm from object
(2) float desOmega = 0;  //desired degrees per sec
(3) float desTheta = 90; //desired angular displacement (degrees)


   //control variables
(1) float curDist;  //tracking current distance
(2) float curOmega; //tracking current angular velocity
(3) float curTheta; //tracking current angle

(1) float output;        //stores value from the PID calculation
(2) int basePower = 180; //how fast you want to drive straight
(2) int diffPower;
(3) float omega;         //stores angular velocity from imu
(3) float power;         //stores value from PID calculation
```

```
void setup() {
  [set up any additional sensors you may need]

  Wire.begin();  // IMPORTANT! I2C needs to be initialized before the IMU!!
  imu.begin();
  imu.calibrate();  // make sure the robot stays still while the IMU is calibrated

  //initialize timing variables for dt
  cur = prev = millis();

    //set limits so that motor doesn't stall
  (1) controller.setDeadbands(-120, 120); //adjust as needed
}


void loop() {
  cur = millis(); //mark time enter loop

//state 0
  if(state == 0){
    if(cur - prev >= dt){ state = 1; }
  }

//state 1 for distance control
  if (state == 1){
    curDist = duration2centimeters(ultrasonic.pulse());

    if(curDist !=0){  //check sensor did not timeout
      output = controller.pid(desDist, curDist);
        //set power to motors (negative bc output > 0 means back up)
      rawMotorCtrl(-1*output, -1*output);
    }
      state = 0;
      prev=cur;
  }

//state 1 for angular velocity control
 if (state ==1){
    imu.update();
    curOmega = imu.getAngVelZ();

    diffPower = controller.pid(curOmega, desOmega);
      //if rotating left, want increase power to left motor to go straight
      // since error is negative, need to subtract to increase power
    rawMotorCtrl(basePower - diffPower, basePower);
alternate:
    rawMotorCtrl(basePower - diffPower, basePower + diffPower);

    state = 0;
    prev=cur;
  }
//state 1 for angular displacement control
if (state ==1){
   imu.update();
   omega = imu.getAngVelZ();

     // integrate angular velocity to get angle traveled
   curTheta = curTheta + (omega*(dt/1000.0))

   power = controller.pid(curTheta, desTheta);

   rawMotorCtrl(-1*power, 0);     //turn around right wheel
alternate
   rawMotorCtrl(-1*power, power); //turn around center of robot

   state = 0;
   prev=cur;
 }

} //end loop
```