File: C:\Users\M4rc05\Documents\Vex\Starstruck\2223-G\4-4-2017\PD test\PD test.c

```
#pragma config(Sensor, dgtl1,  leftEncoder,    sensorQuadEncoder)
#pragma config(Sensor, dgtl3,  rightEncoder,   sensorQuadEncoder)
#pragma config(Motor,  port1,           RightMotor,    tmotorVex393 HBridge, openLoop, reversed)
#pragma config(Motor,  port10,          LeftMotor,     tmotorVex393_HBridge, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

float Kp = 1, Ki = 0.0025,Kd = 15;
int lastErrorLeft = 0, lastErrorRight = 0,    //For now they will be 0
integralLeft = 0, integralRight = 0,          //For now they will be 0
derivativeLeft, derivativeRight,              //No initialization needed
timeToWait = 20,                              //After each cycle
outputLeft = 0, outputRight = 0,              //To be determined
errorBand=0,                                  //Modify for precision
errorLeft = errorBand, errorRight = errorBand;//These values have to be equal or grater than errorBand !!!!

void moveForward(int speed = 100, long pulses = 360){                                  //Use predetermined values of speed and pulses
  for(int c = 1; SensorValue[rightEncoder]<=pulses || SensorValue[rightEncoder]<=pulses; c++){  //Only run if there is still distance needed to tra
    errorLeft = errorBand <= errorLeft ? pulses - SensorValue[leftEncoder] : 0;                 //There is only error while value is bigger than er
    errorRight = errorBand <= errorRight ? pulses - SensorValue[rightEncoder] : 0;              //There is only error while value is bigger than er

    integralLeft += errorLeft/(timeToWait)*c;
    integralRight += errorRight/(timeToWait)*c;

    derivativeLeft = (lastErrorLeft-errorLeft)/(timeToWait);
    derivativeRight = (lastErrorRight-errorRight)/(timeToWait);

    outputLeft = (Kp*errorLeft) + (Ki*integralLeft) + (Kd*derivativeLeft);
    outputRight = (Kp*errorRight) + (Ki*integralRight) + (Kd*derivativeRight);

    motor[LeftMotor] = outputLeft <= speed ? outputLeft : speed;
    motor[RightMotor] = outputRight <= speed ? outputRight : speed;

    lastErrorLeft = errorLeft;
    lastErrorRight = errorRight;

    datalogDataGroupStart();

    datalogAddValue(1, SensorValue[leftEncoder]);
    datalogAddValue(2,SensorValue[rightEncoder]);
    datalogDataGroupEnd();
```

```
    wait1Msec(timeToWait);
  }
}
float batteryLevel = nImmediateBatteryLevel;
task main(){
  datalogClear();
  SensorValue[rightEncoder] = 0;
  SensorValue[leftEncoder] = 0;
  moveForward(127,360);
}
```