#ifndef PID\_h

#define PID\_h

#define LIBRARY\_VERSION 1.2.1

#include <iostream>

using namespace std;

class PID

{

public:

//Display functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double GetKp(); // These functions query the pid for interal values.

double GetKi(); // they were created mainly for the pid front-end,

double GetKd(); // where it's important to know what is actually

int GetMode(); // inside the PID.

int GetDirection(); //

//commonly used functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PID(double\*, double\*, double\*, // \* constructor. links the PID to the Input, Output, and

double, double, double, int); // Setpoint. Initial tuning parameters are also set here.

// (overload for specifying proportional mode)

bool Compute(); // \* performs the PID calculation. it should be

// called every time loop() cycles. ON/OFF and

// calculation frequency can be set using SetMode

// SetSampleTime respectively

void SetOutputLimits(double, double); // \* clamps the output to a specific range. 0-255 by default, but

// it's likely the user will want to change this depending on

// the application

//available but not commonly used functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void SetTunings(double, double, // \* While most users will set the tunings once in the

double); // constructor, this function gives the user the option

// of changing tunings during runtime for Adaptive control

void SetTunings(double, double, // \* overload for specifying proportional mode

double, int);

void SetControllerDirection(int); // \* Sets the Direction, or "Action" of the controller. DIRECT

// means the output will increase when error is positive. REVERSE

// means the opposite. it's very unlikely that this will be needed

// once it is set in the constructor.

void SetSampleTime(int); // \* sets the frequency, in Milliseconds, with which

// the PID calculation is performed. default is 100

//Constants used in some of the functions below

#define AUTOMATIC 1

#define MANUAL 0

#define DIRECT 0

#define REVERSE 1

#define P\_ON\_M 0

#define P\_ON\_E 1

double dispKp; // \* we'll hold on to the tuning parameters in user-entered

double dispKi; // format for display purposes

double dispKd; //

double kp; // \* (P)roportional Tuning Parameter

double ki; // \* (I)ntegral Tuning Parameter

double kd; // \* (D)erivative Tuning Parameter

double\* output;

double\* input;

double\* setpoint;

int controllerDirection;

double \*myInput; // \* Pointers to the Input, Output, and Setpoint variables

double \*myOutput; // This creates a hard link between the variables and the

double \*mySetpoint; // PID, freeing the user from having to constantly tell us

// what these values are. with pointers we'll just know.

unsigned long lastTime;

double outputSum, lastInput;

unsigned long SampleTime;

double outMin, outMax;

bool inAuto, pOnE;

};

#endif