Documentation

Documentation of a software PID controller

bilinear_PID

Lukas Rumpel

Abstract

A software PID controller should be developed because the current solutions were not satisfactory, as they merely reflected the concept. The goal was to implement a digital representation of a continuous-time PID controller, as established control design methods should be applied. The source code is written in C, so implementation on microcontrollers with integrated floating-point unit is possible.

Contents

List of Figures

Abbreviations Index

PID	proportional integral derivative
T	sampling time
K_P	proportional gain
K_I	integral gain
K_D	derivative gain

1 Design

1.1 Model of the controller

The basic structure chosen for the controller is a parallel PID controller, as it avoids having differential equations of higher order (> 1st degree). The model is depicted graphically below:

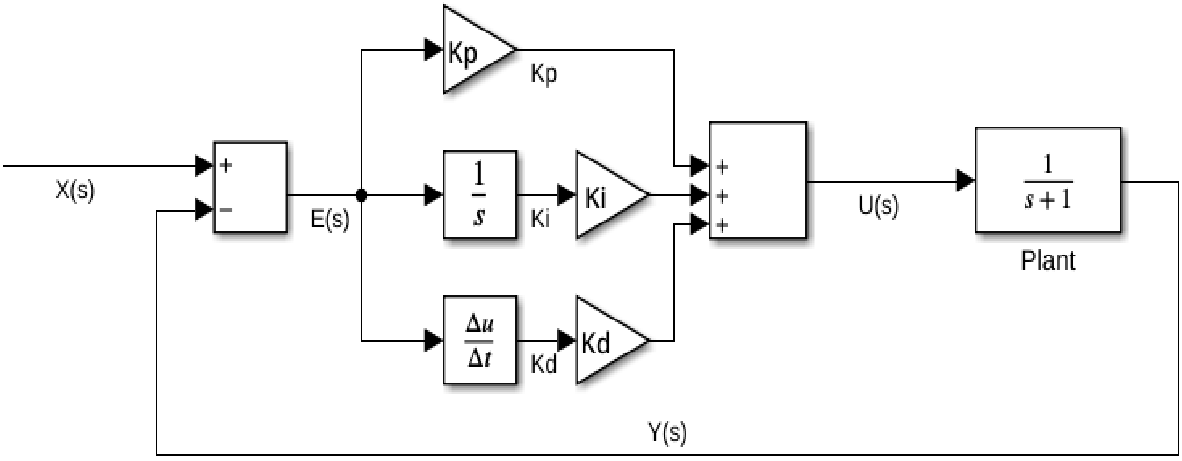

Fig. 1: Simulink model of the desired controller

1.2 Derivation

The differential equation of the controller is as follows:

$$u_{PID}(t) = K_P + K_I \cdot \int e(t)dt + K_D \cdot \tfrac{d}{dt}e(t) \quad (1)$$

Where e(t) is the control error.

$$e(t) = x(t) - y(t); \quad (2)$$

Transformed into the frequency domain using the Laplace transformation, the following equation arises:

$$U_{PID}(s) = K_P + K_P \cdot E(s)\frac{1}{s} + K_D \cdot E(s) \cdot s \tag{3}$$

The equation 2 is now transformed into the z-domain using the bilinear transformation.

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{4}$$

$$U_(z) = K_p + K_I \cdot E(z) \cdot \frac{1}{\frac{2}{T}\frac{z-1}{z+1}} + K_D \cdot E(z) \cdot \frac{2}{T}\frac{z-1}{z+1} \tag{5}$$

Which leads to this transfer function:

$$H_{PID}(z) = \frac{z^2 \cdot (2K_P T + K_I T^2 + 4K_D) + z \cdot (2K_I T^2 - 8K_D) + K_I T^2 4K_D - 2K_P T}{2Tz^2 - 2T} \tag{6}$$

Therefore, the impulse response is:

$$u(k) = \frac{e(k) \cdot (2K_P T + K_I T^2 + 4K_D) + e(k-1) \cdot (2K_I T^2 - 8K_D) + e(k-2) \cdot (K_I T^2 4K_D - 2K_P T) + u(k-1) \cdot (2T)}{2T} \tag{7}$$

1.3 Demonstration

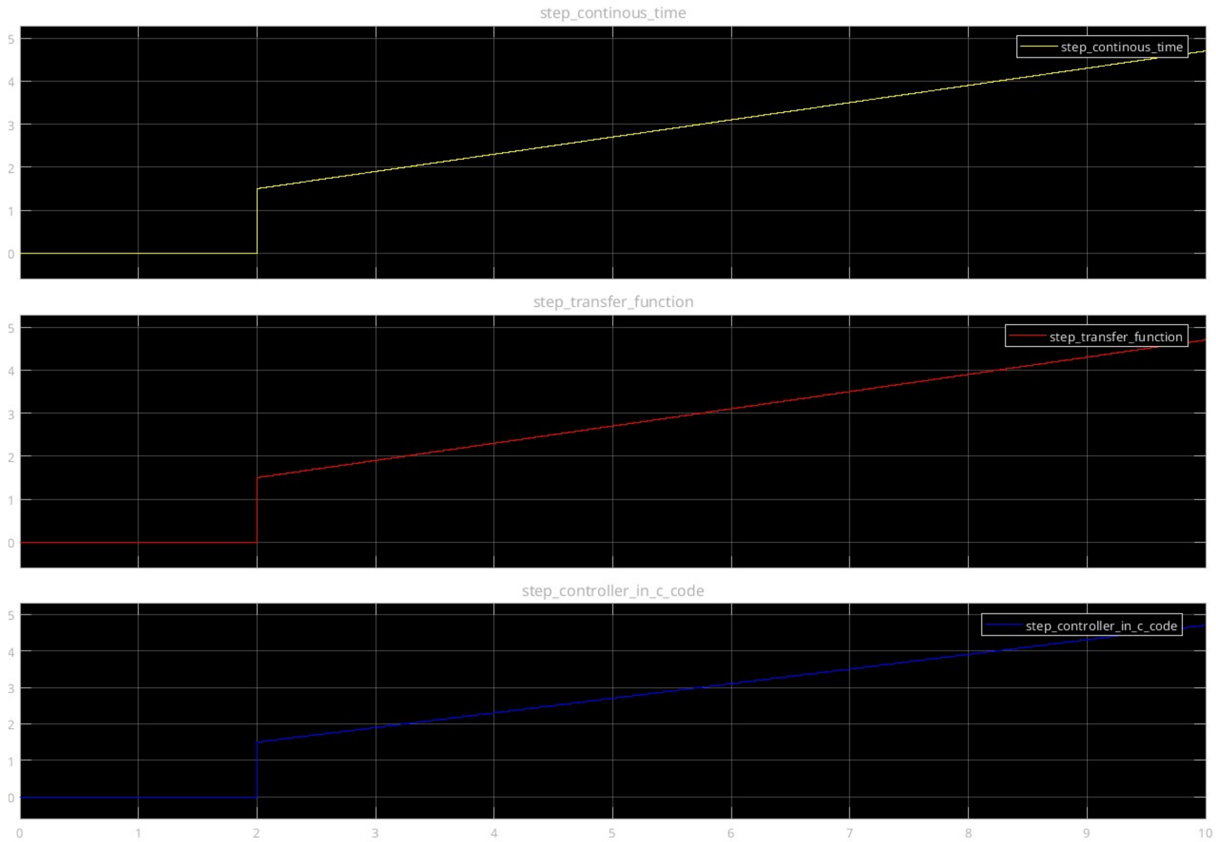

Fig. 2: step response of the designed controller: contious-time model vs. transfer function vs. c code

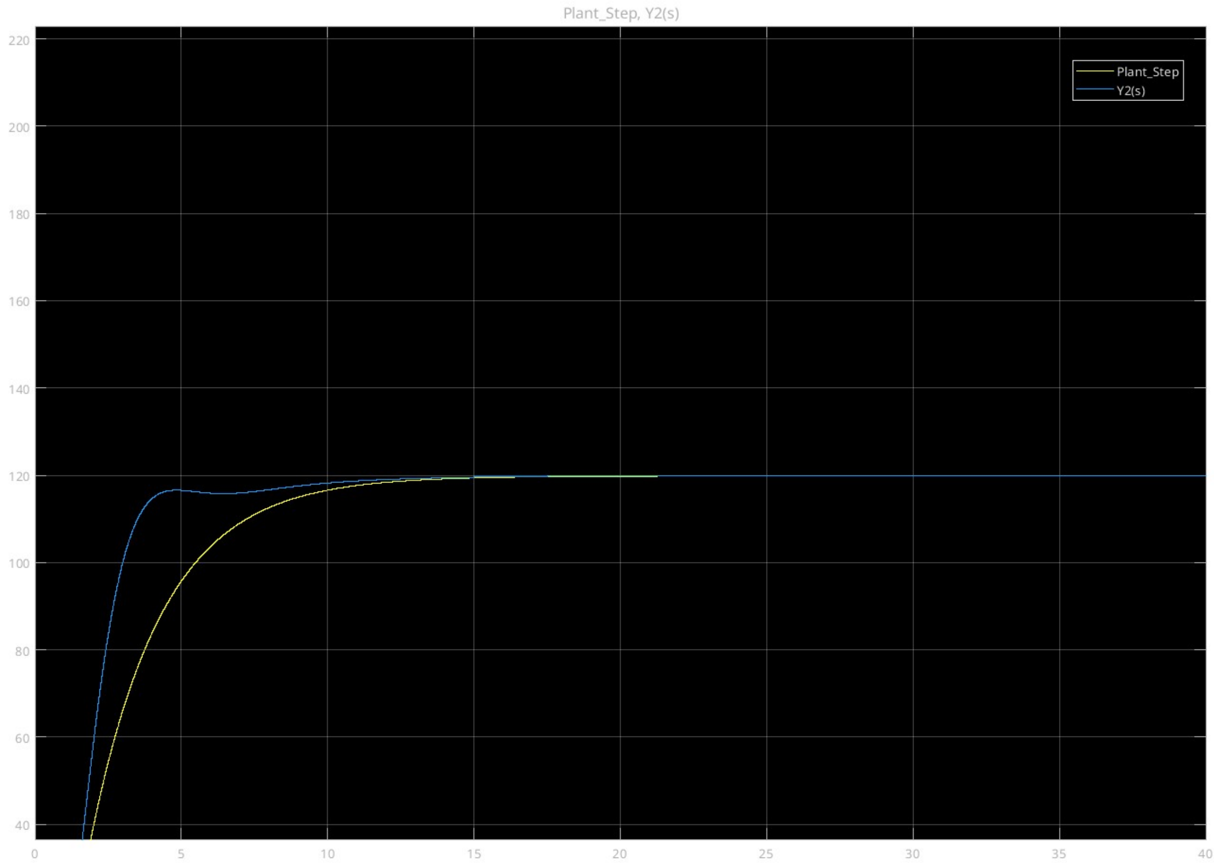

Fig. 3: the controller in a closed loop (blue)

2 Code

2.1 Header

Listing 1: header of the controller

```
#ifndef PID_H
#define PID_H

#include <stdint.h>
#include <stdbool.h>

extern void setKp(uint16_t newKp);
extern void setKi(uint16_t newKi);
extern void setKd(uint16_t newKd);
extern void setSamplingTime(double sampleTime);
extern void setUpperBoundSystemInput(double upperBound);
extern void setLowerBoundSystemInput(double lowerBound);
extern void getCurrentParameters(uint16_t* currKp, uint16_t* currKi, uint16_t*
    currKd, double* currUpperBound, double* currLowerBound, double* currTs);

extern double procPID(double current, double setPoint);

#endif // !PID_H
```

2.2 Source

Listing 2: source of the controller

```
#include "pid.h"
#include <stdio.h>
#include <stdint.h>

uint16_t Kp = 100;
uint16_t Ki = 0;
uint16_t Kd = 0;
double Ts = 1;
double upperBoundU_s = 255.0;
double lowerBoundU_s = -255.0;
```

```
double xStorage[3] = {0, 0, 0};
double yStorage[3] = {0,0,0};

/*
  * Desc.: Sets the proportional gain of the PID controller.
  * @param: (uint16_t) newKp: proportional gain
  * @return: none
  */
void setKp(uint16_t newKp){
    Kp = newKp;
}

/*
  * Desc.: Sets the integral gain of the PID controller
  * @param: (uint16_t) newKi: integral gain
  * @return: none
  */
void setKi(uint16_t newKi){
    Ki = newKi;
}

/*
  * Desc.: Sets the derivative gain of the PID controller.
  * @param: (uint16_t) newKp: proportional gain [percentage]
  * @return: none
  */
void setKd(uint16_t newKd){
    Kd = newKd;
}

/*
  * Desc.: Sets the upper bound of the control variable.
  * @param: (double) upperBound: maximal value of the control variable [
     percentage]
  * @return: none
  */
void setUpperBoundSystemInput(double upperBound){
    upperBoundU_s = upperBound;
}

/*
  * Desc.: Sets the lower bound of the control variable.
  * @param: (double) upperBound: minimal value of the control variable [
     percentage]
```

```
  * @return: none
  */
void setLowerBoundSystemInput(double lowerBound){
    lowerBoundU_s = lowerBound;
}

/*
  * Desc.: Sets the sampling time for the controller.
  * @param: (double) sampleTime: sampling time [seconds]
  * @return: none
  */
void setSamplingTime(double sampleTime){
    Ts = sampleTime;
}

/*
  * Desc.: A getter for all of the current controller parameters.
  * @param: (uint16_t*) currKp
  * @param: (uint16_t*) currKi
  * @param: (uint16_t*) currKd
  * @param: (uint16_t*) currUpperBound
  * @param: (uint16_t*) currLowerBound
  * @param: (uint16_t*) currTs
  * @return: none
  */
void getCurrentParameters(uint16_t* currKp, uint16_t* currKi, uint16_t* currKd,
    double* currUpperBound, double* currLowerBound, double* currTs){
    *currKp = Kp;
    *currKi = Ki;
    *currKd = Kd;
    *currUpperBound = upperBoundU_s;
    *currLowerBound = lowerBoundU_s;
    *currTs = Ts;
}

/*
  * Desc.: This is the main function of the controller.
            This function is called during the setting of the sampling interval.
            It calculates the error between the input and the feedback value,
                performs the necessary value shifting, and computes the new
                control value.
            Essentially, it represents a continuous-time PID controller
                transformed into the Z-domain using the bilinear transformation.
  * @param: (double) current: current input
```

```
  * @param: (double) setPoint: desired setpoint
  * @return: none
  */
double procPID(double current, double setPoint){
    double u_s = 0.0;
    double error = setPoint-current;
    double xTempStorage[3] = {0,0,0};
    double yTempStorage[3] = {0,0,0};
    xTempStorage[0] = xStorage[0];
    xTempStorage[1] = xStorage[1];
    xTempStorage[2] = xStorage[2];
    yTempStorage[0] = yStorage[0];
    yTempStorage[1] = yStorage[1];
    yTempStorage[2] = yStorage[2];
    xStorage[0] = error;
    xStorage[1] = xTempStorage[0];
    xStorage[2] = xTempStorage[1];
    u_s = ((xStorage[0]*(2*(Kp/100)*Ts + (Ki/100)*(Ts*Ts) + 4*(Kd/100))) + (
        xStorage[1] * (2*(Ki/100)*(Ts*Ts) - 8*(Kd/100))) + (xStorage[2]*((Ki
        /100)*(Ts*Ts) + 4*(Kd/100) - 2*(Kp/100)*Ts)) + (2*Ts*yStorage[1]))/(2*Ts
        );
    yStorage[0] = u_s;
    yStorage[1] = yTempStorage[0];
    yStorage[2] = yTempStorage[1];
    return u_s;
}
```