TMR4160
Computer Methods for the Engineer
Dynamic Positioning

Kandidater: 769545

May 2018

Abstract

This report is the final delivery in the course "TMR4160 - Computer Methods for the Engineer" at NTNU and constitutes 100% of the grade. The aim of the course is to teach engineering students basic C programming and to apply computer methods to engineering applications.

I chose the dynamic positioning project and the assignment was to create a program that dynamically positions the boat in a tank at a target position. The given setup had a system that restricted the boat to only move longitudinally and pulled the boat backwards. The setup was fitted with devices to measure the position of the boat and control its motor.

The program is implemented using a PID control algorithm. It controls the boats motor based on input from the potensiometer. The algorithm was tuned using the Ziegler-Nichols method, but experimental testing and tuning resulted in the best performance. P, PI and PID algorithms where tested. It was revealed that the I-term was necessary for the boat to remove the steady state error.

The final result was that the boat was successfully controlled by the program, but further improvements of the algorithm and the program could have done it more reliable.

Contents

List of Figures

Listings

Chapter 1

Introduction

This report presents and documents the project assignment for the course TMR4160, Computer Methods for the Engineer. The project was to develop a program in C for dynamic positioning of a small boat in a tank.

The report will include

- Theory for dynamic positioning
- How the theory is implemented
- The results and plots
- The source code in the appendix

1.1 Assignment specification

A program using PI and PID algorithm was to be developed to dynamically position the boat at a target position set by the user. The program where to be fully written in the C language.The program was required to log the deviation of position, input to servo and other relevant variables to a file for plotting and documentation of the programs performance.

1.2 Project setup

A small motorboat is put in a rectangular tank filled with water. The tank is fitted with a rope stabilization system that restricts the boat to only move longitudinally. At both ends of the tank there is a pulley with a rope over. The two ropes are connected to the front end and back end of the boat. At the other end of the ropes are weights. The rearmost weight is heavier, thus the boat is pulled backwards. Commercial sensors and control products called Phidgets, made by Phidget Inc., are used to measure the position of the boat and input wanted thrust to the boats motor.

The motor is controlled by a servomotor controller that receives input thrust from the program via USB. The rearmost pulley is fitted with a Phidget potentiometer that has variable resistance depending on the position of the pulley. The potentiometer is fitted with three wires that are connected to an interface kit card (PhidgetInterfaceKit 8/8/8). This card measures the voltage over the potentiometer and transmits it to the computer via USB.

The Phidget devices are delivered by Phidget Inc. who produce commercial sensor and control devices. All necessary software and libraries for communicating with the devices, including tutorials and example code, is found on the Phidget website.

Chapter 2

Theory

2.1 Control System

The aim of a control system is to control a dynamic, often unstable, system reliably and efficiently. Often the process is affected by unknown or unpredictable forces called disturbances. In a typical control system a feedback loop is used to measure a process variable(PV), calculate an appropriate response with a control algorithm and then apply the response. A control algorithm will typically estimate the response based on the difference between the PV and a set point, which is the desired value of the PV. The feedback loop is illustrated in figure 2.1.

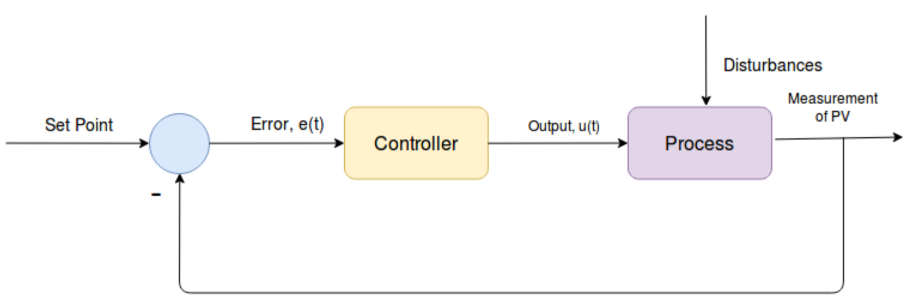

Figure 2.1: Feedback Loop

2.2 PID algorithm

The most common real-time control algorithm in the industry is the Proportional-Integral-Derivative(PID) algorithm, or PID. It is simple and often as good as much more complex algorithms.[3] The algorithm is based on equation 2.1. The equation takes an error as input and sums up the P, I and D contributions to the response, $u(t)$. The response is used to affect the system in order to stabilize it at the wanted set point. The P, I and D terms are tuned by the gain constants K_p, K_i, K_d. The tuning of these are critical to get a stable system. The proportional function is the base function and a simple P algorithm often works quite good alone. The integral and derivative terms are merely attempts to fix very common downsides of the P-controller.

Equation 2.1 is continous while computers are discrete. In the real world discrete numerical algorithms are implemented based on the continous analytical equation by using the definition of integration and derivation. The bias is added as a baseline to systems where the zero output does not map properly to the process.

$$ThePIDEquationu(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) + bias \tag{2.1}$$

2.3 Proportional Response

The proportional component attempts to achieve set point by giving an output that is proportional to the error and the K_p constant determines the scale of the response. In general, increasing K_p will increase the speed of the control system response.[3] When a system controlled by a P-controller settles, it sometimes settles at a value below the wanted set point. This is called a steady state error. Testing with a too low K_p and gradually increasing it generally leads to something like the following:

- The system does not adjust
- The system adjusts slowly and settles
- Perfect adjustment
- The system overshoots or oscillates but settles eventually

- The system oscillates around the set point without settling
- The system oscillates out of control

2.4 Integral Response

The integral component sums the error term over time. Thus it responds to even small errors and gradually adjusts the response. This effect is used to avoid the steady state error of the P-controller. As the intergral functions is sensitive to small errors and sums these up over time, a large K_i constant should be avoided. Large K_i constants can lead to saturation. It is called integral windup when the integral term drives the controller to saturation without reaching the set point. This can be avoided by keeping K_i small and restricting extreme I-term values with an if clause. When the I-term is added to a perfectly tuned P-controller, K_p should be lowered because their responses are added.

2.5 Derivative Response

The derivative component responds proportionally to the change rate of the error and is often introduced to calm the proportional term. It causes the output to decrease if the process variable increases rapidly, and vica versa. Increasing K_d on a overhooting system, will generally lead to less overshooting, but the derivative function is very sensitive. Therefore the K_d constant should be small. When the derivative term is introduced to a perfectly tuned P- or PI-controller, K_p can be increased because the D-term contributes to avoid over- or undershooting.

If the measurement signal of the process variable is noisy or the set point is changed, the derivative term will spike. This is called derivative kick and can be avoided by restricting extreme D-term values with an if clause. The D-term could also be skiped when the set point is changed, because this would lead to a sudden change rate of the error which should not be relaxed by the derivative response.

2.6 Tuning and Ziegler-Nichols Method

PID controllers have to be tuned in order to work properly. There are many methods and the Ziegler-Nichols method is especially popular.[3] Experienced engineers can often predict suitable gain constants based on experience and their approach is often to start with the P-term.

In the Ziegler-Nichols method the I- and D-terms are kept zero by setting K_i and K_d to zero. The K_p gain constant is gradually increased until the system starts to oscillate. This value, K_c and the oscillation period, T_c, is used to calculate values for K_p, K_i and K_p in accordance with table 2.2. Even though there is a method, the final values should always be tested and fine tuned if necessary.

Control Type	K_P	K_I	K_D
P	$0.50K_c$		
PI	$0.45K_c$	$1.2K_p/T_c$	
PID	$0.60K_c$	$2K_p/T_c$	$K_pT_c/8$

Figure 2.2: Ziegler-Nichols method

Chapter 3

Environment and Architecture

3.1 Enviroment and framework

The program is completely written in C and is compiled with a GCC compiler that translates the C source code to executable machine code.

The program is written on the Ubuntu OS which is a UNIX-like OS and the code is not directly convertible to Windows. The time-related libraries and functions would have to be changed to make the program work on both platform. This was intentionally not done to keep things simple. CLion was used as IDE and during development time both CLion and terminal commands where used to execute the code. CMake was used to manage the build process.

3.2 Architecture

The architecture of the program is illustrated in figure 3.1. The source code is in the appendix and can be examined for more details. In brief main.c contains the main and is where the program is run. It includes a main.h header file that includes libraries, defines all necessary macros, declares global variables and declares the subroutines. The subroutines that has to do with the phidgets and that are directly called from main.c are located in phidget.c. This file includes phidgetHelper.c which contains subroutines that helps setUpPhidgets() to connect to the phidgets. phidgetHelper.c has not much in common with the rest of the program and is therefore only connected to phidget.c. This approach is inspired by the layered architecture.

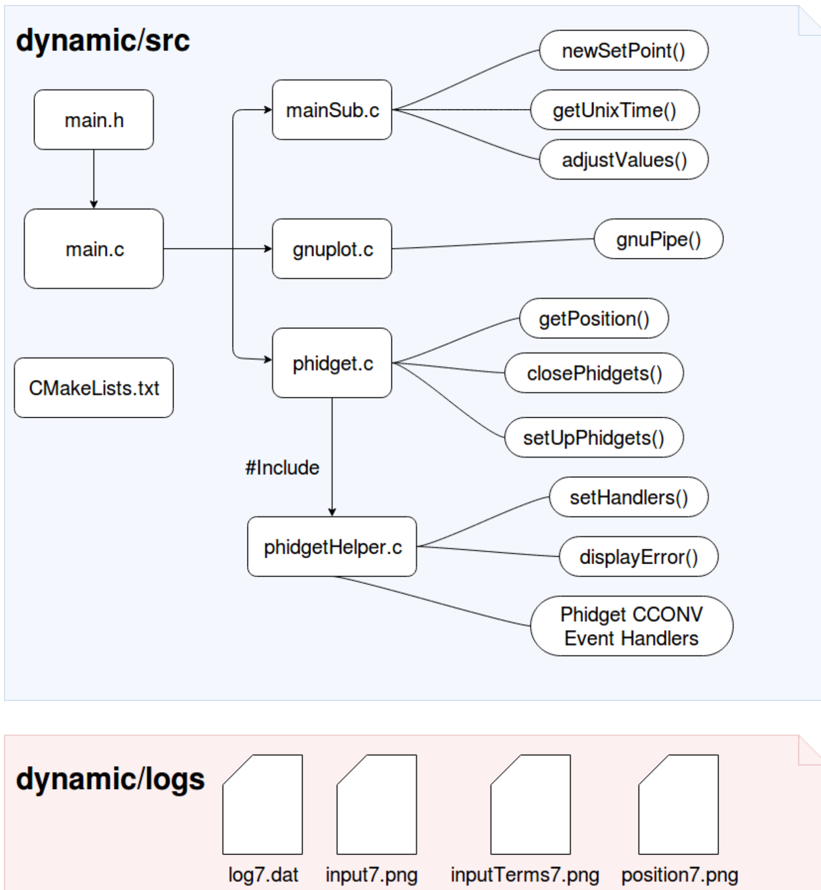

Figure 3.1: Structure of the program

3.3 Flow chart

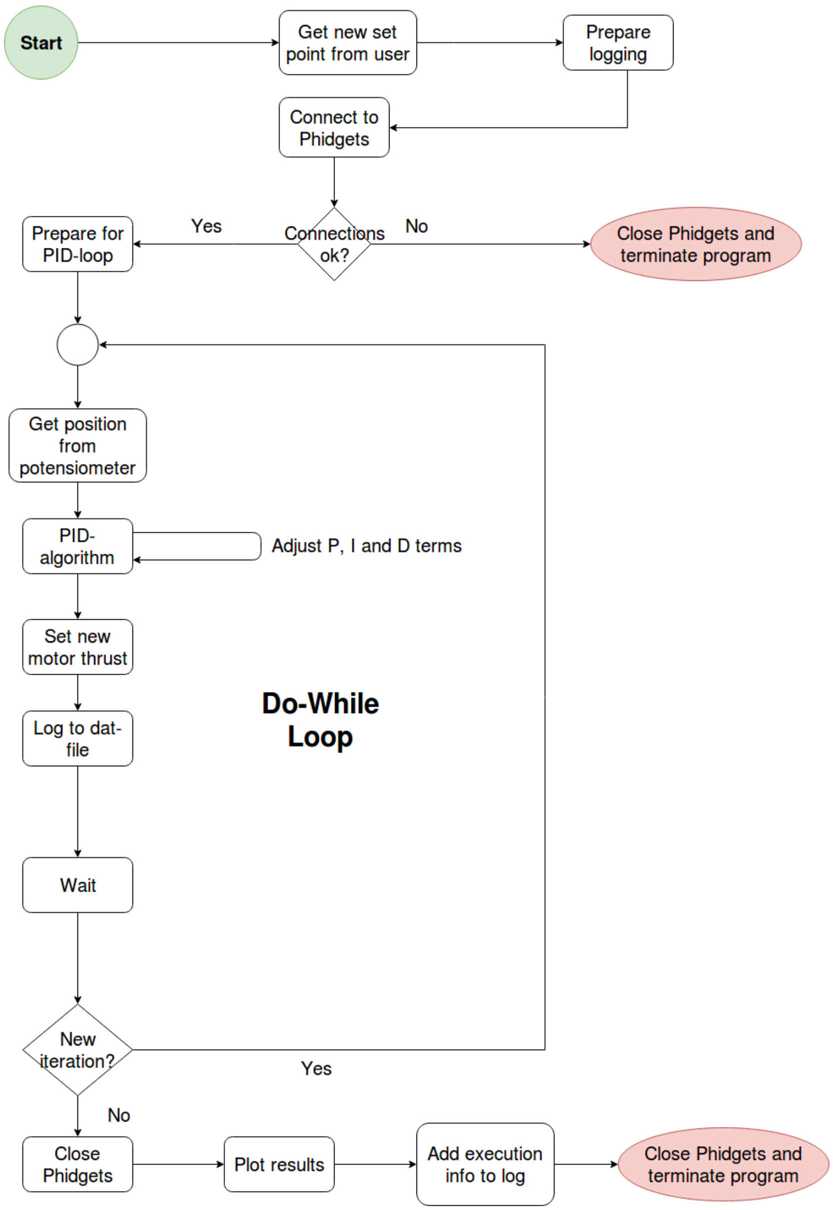

Figure 3.2: Flow chart of the program

Chapter 4

Implementation

4.1 Implementation of the PID algorithm

As stated earlier, the PID equation is continuous while computers only do discrete computations. The iteration of the PID alogrithm is a discrete version of the PID equation. dt, which is the duration between every iteration, is used to calculate approximate outputs of the integrative and derivative term. The apporximations are given in equation 4.2 and 4.1.

$$P_{term} = K_p * e \tag{4.1}$$

$$I_{term} = K_i * (I_{term} + e * dt) \tag{4.2}$$

$$D_{term} = K_d * (e - e_0)/dt \tag{4.3}$$

The program has clauses to prevent extreme values of the terms. The I-term is restricted to be between 80 and 0. The D-term is restricted to be between -20 and 20. The proportional term is not restricted directly, but the combined output, including the off set, is restricted to be between 160 and 85. The minimum value has a great effect on the performance of the algorithm, because without it the motor shuts off when it overshoots. This quickly results in undershooting and the result is oscillation.

4.2 Helper functions for main

mainSub.c contains helper subroutines for main. getUnixTime() is used to calculate the time passed. newSetPoint() asks the user for a new set point and redefines the setPoint macro if the input is valid. adjustValue() checks if the inputted value should be changed. It is used three times and was created to avoid repetitive code in main.

4.3 Communicating with the phidgets

In this project the phidget22 library was used. The subroutines in phidget.c are used to communicate with the phidgets. SetUpPhidgets() also uses subroutines in phidgetHelper.c.phidghetHelper.c is a slightly modified version of PhidgetHelperFunctions.c that can be downloaded from Phidget Inc.'s home page.[2] The modifications where cautious because I have had many problems with connecting to the phidgets and therefore wanted to stay close to the original example codes.

getPosition() receives voltage intensity from the potentiometer and uses this to calculate the current position in cm.

4.4 Implementation rationale

The clauses to check the output value of the PID algorithm was deliberately not put in setMotorSpeed(), even though such clauses are common to put in setters. The checkers where put in PID algorithm so that the program would not think it was setting a new input thrust that was actually changed.[1] Also it is important to have the correct output for plotting.

In the return statement a off set constant is added. This was done because the servo motor reversed below 66 and forwarded over 71. As the boat is being pulled backwards, the performance drops dramatically when the program sets the motor in reverse. The PID equation is implemented to output a positive value for forward and a negative value for backwards. The I-term and D-term are checked individually. The off set is added and finally the whole term is checked and adjusted if necessary. Then the output is given as new thrust to the motor.

During the development, the PID calculations where placed in its own subroutine, compute(). As the subroutine had many inputs and many variables where changed and these variables where needed in the main for plotting and for next iteration, it was much easier to put all of the PID code directly in the main. The PID algorithm is also core to the program, so it makes sense to have it in the main program. adjustValue() was a better way to simplify and shorten the code.

Chapter 5

Visualization

5.1 Logging

Before the loop, where the PID is, a DAT-file is created. In every iteration of the loop, data is written to the DAT-file in the following format:

[Time(sek)] [Measured position(cm)] [prc-Error] [Output] [Error(cm)]

prc-Error is the error given as a percentage of the initial error. After the loop the DAT-file is closed, then the function gnuPipe is run. gnuPipe creates and saves two plots. Both have time along the x-axis.

5.2 Gnuplot

Plotting was done using Gnuplot in gnuPipe(). It requires that Gnuplot is installed on the computer. gnuPipe() creates three plots; positon1.png, input1.png and subInput1.png.

The first plot plots the position as a function of time along the xy axis. The reference is given as horizontal line. Along the xy2 axes, the error is plotted as a percentage, where initial error equals 100%. Error was deliberately not given in cm, as this would be a perfect mirror of the position graph.

The second plot plots the output from the PID algorithm along the xy axis and the error along xy2. The third plot is similar to the second, but the error is removed and replaced with the components of P, I and D. This plot is particularly useful as it can be used to analyze the different contributions and give a understanding of how the PID works and can be improved.

Chapter 6

Results

6.1 Ziegler-Nichols

The boat started to oscillate with $K_c = 1.6$ and $T_c = 2.9$ seconds. Using Ziegler-Nichols resulted in $K_p = 0.96$, $K_i = 0.47$ and $K_d = 0.34$. This gave a quite good result where the boat overshoot, undershoot and settled at target position after 7.3 seconds.

6.2 P controller

The first try with a P-controller was with $K_p = 1.1$ seen in figure6.1. Note that a off set of 71 is added to the final output. The boat both overshoots and settles with a steady state error(SSE) of 10 cm below set point. Many other K_p values where tried, but the boat always settles with SSE because the boat is being dragged backwards.

6.3 PI controller

After a few tries, a PI-controller tuned with $K_p = 0.8$ and $K_i = 0.4$ was quite successful. It both overshoot and undershoot, but reached and maintained set point after only 6 seconds. Its plots are in figure 6.2.

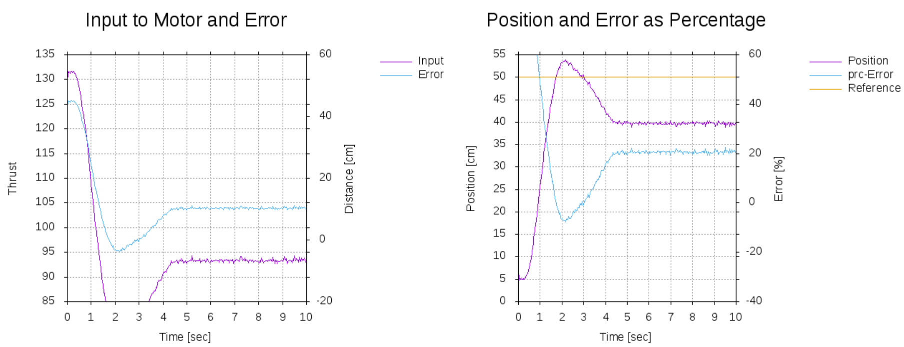

Figure 6.1: Position with P-controller, $K_p = 1.1$

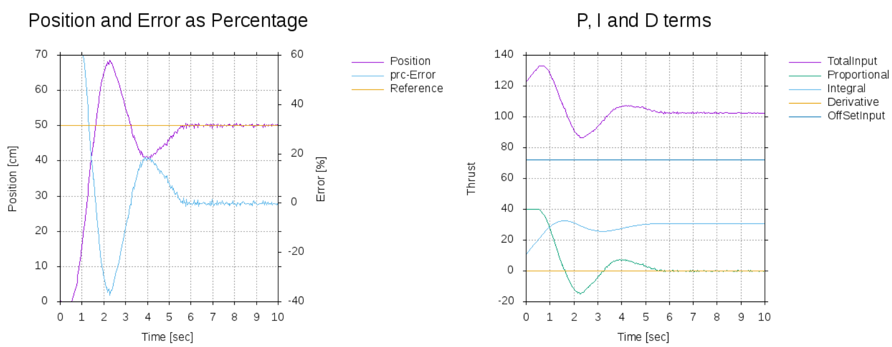

Figure 6.2: Position and input with PI-controller, $K_p = 0.8$ and $K_i = 0.4$.

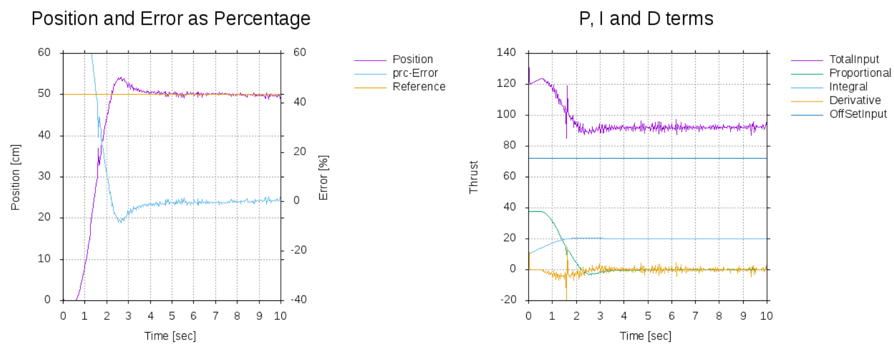

Figure 6.3: Position and input with PID-controller, $K_p = 0.75$, $K_i = 0.15$ and $K_d = 0.1$.

6.4 PID controller

The PID-controller was tuned experimentally. With $K_p = 0.75$, $K_i = 0.15$ and $K_d = 0.1$ it reached and maintained set point after 4 seconds. It can be seen in figure 6.3. The input spikes briefly around 1.6 seconds.

Chapter 7

Discussion

The purpose of the project was to create a program to dynamically position the boat at a given target position. This was done with phidget devices, phidget API and by implementing a PID algortihm in C. The project was successful, but the program can be further improved and tuned to become more reliable.

To tune the boat experimentally turned out to give slightly better results than the Ziegler-Nichols method. The best results where achieved with the PID algorithm. It settled at set point after only 4 seconds. The PI was quite similar, and settled after 6 seconds.

As figure 6.3 shows, it is a large spike around 1.6 seconds. This is caused by the derivative term and is probably due to noise in the signals from the potentiometer. The derivative contribution could have been improved to avoid this. For example by only updating the derivative term every tenth iteration. Another solution could be to base the derivative on more than the current and last error. For example, if the five last errors where used, one could implement a derivative function with smoother contributions.

Experimentation that is not shown in the results also showed that the minimum input to the motor had huge impact on the performance. When the minimum input was raised to 85, the motor would always push forward. The result was that when the boat overshoot, it would slowly fall back to set point. Without continuous forward push from the motor, the boat mostly oscillated.

7.1 Modifications of the program

The onVoltageChangeHandler is fired every time the boat changes position. In addition, it can be configured to restrict firing with a minimum time interval or a minimum change. Many students I have talked with, put the the PID algorithm inside the handler so that a new motor speed would be set every time the boat changed position. In addition, this function makes it easier to implement other features in the main, such as the possibility of changing the set point during execution of the program. This was considered, but not done due to the wish to create most of the program independently.

7.2 Errors and flaws

Analyzing the plots and seeing the prints on screen revealed that the PID is iterated many times without the motor changing the thrust. The iteration interval was lowered to 30 ms, but the problem still persisted. It seems like the motor servo was the bottleneck of the program.

Another source of error might be that the program does not accurately measure the time between loop. The real time between iterations might be somewhat higher than the value used in the PID algorithm. This problem is is quite inconsequential because the gain constants are tuned experimentally. If it turns out that the time between iterations is very uneven, this would however have a negative effect on the algorithm.

The basin is very small and the boat is therefore affected by the water flow in the basin and the waves. These are created by the boat and in open water it would not reflect back on the boat.

Bibliography

[1] Brett Beauregard. *Improving the beginners PID*. 2011. URL: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/%3F. (viewed: 22.04.2018).

[2] Phidgets Inc. *PhidgetInterfaceKit*. 2017. URL: https://www.phidgets.com/?tier=3&catid=2&pcid=1&prodid=1021%3F. (viewed: 22.04.2018).

[3] National Instruments. *PID Theory Explained*. 2011. URL: http://www.ni.com/white-paper/3782/en/. (viewed: 02.05.2018).

Appendix A

Source Code

```
/*dynamic/
    src_______________________________________________________________

Name: main.h
-------------------------------------------------------------------------------

Purpose:  main.h defines constants, declares variables and includes
     libraries for all
          subroutines except for phidgetHelper.c, which is included
     directly to phidget.c


-----------------------------------------------------------------------------------

--------------------------------Parameters-------------------------------------


__Name__                   __Type__              __Description__
setPoint                   macro                 Target position for the
     vessel.
Kp                         macro                 Proportional gain
    constant.
Ki                         macro                 Integrative gain
    constant.
Kd                         macro                 Derivative gain
    constant.
endLoopTime_sec            macro                 Duration of the loop[
    sec].
loopIteration_ms           macro                 Time between PID
    iterations.

fileNum                    macro                 Number to identify logs
     and plots.

outMax                     macro                 Max thrust to motor
    before saturation.
```

```
outMin                  macro               Min thrust to avoid stopped propellers.
offSetMotor             macro               Offset to adjust the motors shifted input.

maxPos                  macro               Max length the boat can travel. (67 cm)
maxVolt                 macro               Max voltage from potensiometer. 5V when pos=0.
minVolt                 macro               Min voltage from potensiometer. ca 3.6V, pos=67.

pos                     float               Measured position of the vessel.
startPos                float               Start position of the vessel.
error                   float               Deviation between pos and setPoint.
lastError               float               Error in last PID computation.
output                  float               Output from PID algorithm. New thrust to motor.
iTerm                   float               Integrative component of PID.
dTerm                   float               Derivative component of PID.

startTime_sec           double              The time before the loop begins.

datFile                 FILE*               File pointer to log file.
fileName                Char[32]            Where the log is stored. --> /logs/log[fileNum].dat

voltage                 double              The voltage measured by the potensiometer.

potHandle               PhidgetVoltageInputHandle       Phidget handle for potensiometer.
motorHandle             PhidgetRCServoHandle            Phidget handle for servo motor.


Subroutines & location:

    newSetPoint()           mainSub.c
    getUnixTime()           mainSub.c
    adjustValue()           mainSub.c
    gnuPipe()               gnuplot.c
    setUpPhidets()          phidget.c
    getPosition()           phidget.c
    closePhidgets()         phidget.c
    setMotorThrust()        phidget.c
--------------------------------------------------------------------------------

Made by student: 69545
```

```
--------------------------------------------------------------------------------
    */

#ifndef SRC_MAIN_H
#define SRC_MAIN_H


//__________Include libraries__________
#include <stdio.h>
#include <phidget22.h>
#include <time.h>
#include <unistd.h>


//Obs: phidget.c includes phidgetHelper.h and its subroutines

//__________Define constant variables__________

//PID related variables
#define setPoint 35
#define Kp 1.0
#define Ki 0.18
#define Kd 0.1
#define endLoopTime_sec 15
#define loopIteration_ms 30

//File number for plots and logs
#define fileNum 17

//Thrust limits for motor: [0,66] -> Reverse and [71, ~180] ->
    Forward
#define outMax 160
#define outMin 85
#define offSetMotor 72

//Potensiometer constants
#define maxPos 67
#define maxVolt 5
#define minVolt 3.6


//__________Variables__________

//PID variables
float pos, startPos;
float error, lastError;
float output, dTerm, iTerm;
double startTime_sec;

//Logging variable
FILE *datFile;

//Potensiometer variable
double voltage;

//__________Subroutines__________
```

```
//Phidget subroutines used directly by main
float getPosition(PhidgetVoltageInputHandle pot);
int setUpPhidgets(PhidgetVoltageInputHandle potHandle,
    PhidgetRCServoHandle motorHandle);
void closePhidgets(PhidgetRCServoHandle motorHandle,
    PhidgetVoltageInputHandle potHandle);
void setMotorThrust(PhidgetRCServoHandle motorHandle, float output)
    ;

//Other subroutines
void newSetPoint();
void gnuPipe(const char *fileName);
double getUnixTime();
void adjustValue(float *value, int max, int min, char name[16]);

/* OBS: phidgetHelper.c is included directly to phidget.c */


#endif //SRC_MAIN_H
```

Listing A.1: main.h

```c
/*dynamic/src--------------------------------------------------------------
Name: main.c
------------------------------------------------------------------------------
Purpose: The program will ask the user for a desired target position[cm] of the
        boat and then it will use a PID-algorithm to maintain that position.
        Parameters as position, input (and its components), error and set point
        will be logged and plotted with Gnuplot. Logs and plots are saved in
        dynamic/logs.

Parameters:
Check main.h

Subroutines & location:

    newSetPoint()           mainSub.c
    getUnixTime()           mainSub.c
    adjustValue()           mainSub.c
    gnuPipe()               gnuplot.c
    setUpPhidets()          phidget.c
    getPosition()           phidget.c
    closePhidgets()         phidget.c
    setMotorThrust()        phidget.c
-------------------------------------------------------------------------
Made by student: 69545
------------------------------------------------------------------------------*/

#include "main.h"

int main(int argc, char* argv[]) {

    //Ask user for new target position for boat (Set Point is redefined)
    newSetPoint();

    //__________Logging file__________
    //Create dat-file for logging
    char fileName[32];
    sprintf (fileName, "../logs/log%i.dat", fileNum);
    printf("Logs are saved in file: %s\n", fileName);
    datFile = fopen(fileName, "w");


    //__________Connect to Phidgets__________

    //Declare and Create Potensiometer handle for measuring of position
    PhidgetVoltageInputHandle potHandle = NULL;
    PhidgetVoltageInput_create(&potHandle);
```

```
    //Declare and Create Motor handle for setting motor speed
    PhidgetRCServoHandle motorHandle = NULL;
    PhidgetRCServo_create(&motorHandle);

    //Connect to the potensiometer and motor
    if (setUpPhidgets(potHandle, motorHandle)){
        printf("Problems with connecting to Phidgets. Program is terminating");
        return 1;
    }


    //__________Prepare for control loop__________

    //Gives smaller dTerm in first iteration -> Smaller derivative kick
    lastError = 20;

    startPos =  getPosition (potHandle);
    startTime_sec = getUnixTime();

    //__________Control loop__________
    do {
        //__________PID Algorithm__________

        //Get measured current position of boat
        pos = getPosition(potHandle);

        //Set error
        error = (float) setPoint - pos;

        //Calculate the integral-term and check that it is not too big or small
        iTerm +=  Ki*error*loopIteration_ms/1000;

        //Avoid extreme values
        adjustValue(&iTerm, 80, 0, "iTerm");

        //Calculate the derivative-term and limit its impact if necessary
        dTerm = (float) Kd * ( (error - lastError) *1000/loopIteration_ms );

        //Avoid extreme values
        adjustValue(&dTerm, 20, -20, "dTerm");

        //Sum up the P, D, I and offSetMotor terms
        output = (float) Kp*error + iTerm + dTerm + offSetMotor;

        //Adjust for saturation
        adjustValue(&output, outMax, outMin, "output");

        //Set new motor thrust
        setMotorThrust(motorHandle , output);

        //__________Logging and preparation for next loop__________
```

```c
        //Print output and the terms to screen
        printf("Output = %f + %2f + %2f = %f\tDeviation: %f\n", Kp*error, iTerm, dTerm, output, error);

        //Print to logging file
        //Format: time[s], position[cm], prc-error, output, error[cm], proportional, integral, derivative, OffsetMotor
        fprintf(datFile, "%f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%i\n",getUnixTime()-startTime_sec, pos, 100*(error/(startPos-setPoint)), output, error,Kp*error, iTerm,dTerm, offSetMotor);

        //Prepare for next loop
        lastError = error;

        //Sleep for loopIteration_ms milli seconds
        usleep(loopIteration_ms *1000);

        //Check if the loop should break
    } while(endLoopTime_sec > getUnixTime()-startTime_sec);

    printf("PID Control Loop Ended\n");

    //Safely end the connection with the potensiometer and motor
    closePhidgets(motorHandle,potHandle);

    //Close logging file
    fclose(datFile);

    //Generate plots with Gnuplot
    gnuPipe(fileName);

    //Adding execution info to logging file after the gnuplots have been created
    datFile = fopen(fileName, "a");
    fprintf(datFile, "Execution Info: \nSet Point: %i\tIteration time: %i\nKp: %f\tKi: %f\tKd: %f\t\n",(int) setPoint,(int) loopIteration_ms,Kp,Ki,Kd);
    fclose(datFile);

    return 0;

}
```

Listing A.2: main.c

```c
#include "main.h"



void newSetPoint(){
    /* Prompts a new setPoint from the user. Checks if input is valid.
     *
     * Valid -> redefine setPoint to new value.
     * Not valid -> Use  default value
     */
    printf("Enter a target position[cm] as integer between 10 and 60:  \n");
    int num;
    char term;
    if(scanf("%d%c",&num,&term)!=2|| term !='\n'||num<10 || num>60)
        printf("Invalid input. Target position set to default: %i\n", setPoint);
    else {
        #undef setPoint
        #define setPoint num
        printf("Valid Input. Target position set to %i\n", setPoint);
    }
}

double getUnixTime() {

    struct timespec tv;

    if(clock_gettime(CLOCK_REALTIME, &tv) != 0) return 0;

    return (tv.tv_sec + (tv.tv_nsec / 1000000000.0));
}

void adjustValue(float *value, int max, int min, char name[16] ){
    /*  Changes value to limit set by max or min if limit breached. */

    if (*value>max){
        *value = max;
        printf("[Maximum %s reached] --> Adjusted to %i \n", name, max);
    } else if (*value<min){
        *value = min;
        printf("[Minimum %s reached] --> Adjusted to %i \n", name, min);
    }
}
```

Listing A.3: mainSub.c

```c
#include "main.h"
#include "phidgetHelper.c"


int setUpPhidgets(PhidgetVoltageInputHandle potHandle, PhidgetRCServoHandle motorHandle) {
    /* Sets up the connections with the potensiometer and the motor (Phidget devices).
     * Returns 0 if connections are successful, otherwise 1.
     * Prints relevant messages to the screen.
     */

  PhidgetReturnCode prc;

  //Set Potensiometer channel and serial number
  Phidget_setDeviceSerialNumber((PhidgetHandle) potHandle, 76241);
  Phidget_setChannel((PhidgetHandle) potHandle, 0);

  //Set event handlers for potensimeter and motor
  if (setHandlers(potHandle, motorHandle) ){
        goto error;
  }

  printf("Waiting for attach reply from potensiometer...\n");
  prc = Phidget_openWaitForAttachment((PhidgetHandle) potHandle, 5000);
  if (prc != EPHIDGET_OK) {
    if (prc == EPHIDGET_TIMEOUT) {
      printf("Potensiometer channel did not attach after 5 seconds: please check that the device is attached\n");
    }
        goto error;
  }

    printf("Waiting for attach reply from motor...\n");
  prc = Phidget_openWaitForAttachment((PhidgetHandle)motorHandle, 5000);
  if (prc != EPHIDGET_OK) {
    if (prc == EPHIDGET_TIMEOUT) {
      printf("Motor channel did not attach after 5 seconds: please check that the device is attached\n");
    }
        goto error;
  }

  printf("Setting motor target thrust to 100 and engaging motor\n");
  prc = PhidgetRCServo_setTargetPosition((PhidgetRCServoHandle) motorHandle, 100);
  if (prc != EPHIDGET_OK) {
    printf("Failed to set motor target thrust\n");
  }
  prc = PhidgetRCServo_setEngaged((PhidgetRCServoHandle) motorHandle, 1);
  if (prc != EPHIDGET_OK) {
    printf("Failed to engage motor\n");
    goto error;
```

```c
  }

  printf("Connected to Phidgets successfully!\n");
  return 0;


  //If setup is unsuccessful, close device connections safely
  error:
    closePhidgets(motorHandle, potHandle);
    return 1;
}

float getPosition(PhidgetVoltageInputHandle pot) {
    /*Returns the position of the boat in cm */

  //Get current voltage from potensiometer. Returns max voltage(=5) when boat is on the back end.
  PhidgetVoltageInput_getSensorValue(pot, &voltage);

  //Return position in cm. Equation units: [cm -  V *(cm/V)]
  return (float) ( (maxPos) - (voltage - minVolt) * (maxPos / (maxVolt - minVolt)) );
}

void setMotorThrust(PhidgetRCServoHandle motorHandle, float thrust) {
  /*Sets new target position (thrust) for the motor*/

  PhidgetRCServo_setTargetPosition(motorHandle , thrust);
}

void closePhidgets(PhidgetRCServoHandle motorHandle, PhidgetVoltageInputHandle potHandle){
    /* Closes connections with the phidgets safely*/

    //Shutting motor off and closing the channel
    PhidgetRCServo_setEngaged((PhidgetRCServoHandle) motorHandle, 0);
  Phidget_close((PhidgetHandle)motorHandle);
  PhidgetRCServo_delete(&motorHandle);

    //Closing the potensiometer channel
    Phidget_close((PhidgetHandle)potHandle);
    PhidgetVoltageInput_delete(&potHandle);
}
```

Listing A.4: phidget.c

```
#include "main.h"

/* Change handlers that are copied and slightly changed from Phidget homepage.
 *
 * CCONV is a defined "Calling Convention" for Phidget library.
 *
 * Motor Handler: OnPositionChangeHandler
 *
 * Potensiometer Handler: OnVoltageChangeHandler --> PotInterval set to 20
 *
 * Both: OnAttach-, OnDetach-, OnErrorhandler
 */

static void CCONV onAttachHandler_pot(PhidgetHandle ph, void *ctx) {
PhidgetReturnCode prc;
int32_t serialNumber;
PhidgetHandle hub;
int32_t hubPort;
int32_t channel = 0;
uint32_t potInterval_ms = 20;

printf("\tSetting potensiometers data interval to %i\n", potInterval_ms);
prc = PhidgetVoltageInput_setDataInterval((PhidgetVoltageInputHandle)ph, potInterval_ms);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Set DataInterval: \n\t");
displayError(prc);
return;
}

printf("\tSetting Voltage ChangeTrigger to 0.0\n");
prc = PhidgetVoltageInput_setVoltageChangeTrigger((PhidgetVoltageInputHandle)ph, 0.0);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Set VoltageChangeTrigger: \n\t");
displayError(prc);
return;
}

prc = Phidget_getDeviceSerialNumber(ph, &serialNumber);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Get DeviceSerialNumber: \n\t");
displayError(prc);
return;
}

prc = Phidget_getChannel(ph, &channel);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Get Channel: \n\t");
displayError(prc);
return;
}
```

```c
//Check if this is a VINT device
prc = Phidget_getHub(ph, &hub);
if (EPHIDGET_WRONGDEVICE != prc) {
prc = Phidget_getHubPort(ph, &hubPort);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Get HubPort: \n\t");
displayError(prc);
return;
}
printf("\n[Attach Event]:\n\t-> Serial Number: %d\n\t-> Hub Port: %d\n\t-> Channel %d\n\n", serialNumber, hubPort, channel);
}
else {
printf("\n[Attach Event]:\n\t-> Serial Number: %d\n\t-> Channel %d\n\n", serialNumber, channel);
}

channel = 0;
}

static void CCONV onDetachHandler_pot(PhidgetHandle ph, void *ctx) {
PhidgetReturnCode prc;
PhidgetHandle hub;
int serialNumber;
int hubPort;
int channel = 0;



prc = Phidget_getDeviceSerialNumber(ph, &serialNumber);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Get DeviceSerialNumber: \n\t");
displayError(prc);
return;
}

prc = Phidget_getChannel(ph, &channel);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Get Channel: \n\t");
displayError(prc);
return;
}

//Check if this is a VINT device
prc = Phidget_getHub(ph, &hub);
if (EPHIDGET_WRONGDEVICE != prc) {
prc = Phidget_getHubPort(ph, &hubPort);
if (EPHIDGET_OK != prc) {
fprintf(stderr, "Runtime Error -> Get HubPort: \n\t");
displayError(prc);
return;
}
printf("\n[Detach Event]:\n\t-> Serial Number: %d\n\t-> Hub Port: %d\n\t-> Channel %d\n\n", serialNumber, hubPort, channel);
}
else {
```

```
printf("\n[Detach Event]:\n\t-> Serial Number: %d\n\t-> Channel %d\
    n\n", serialNumber, channel);
}
channel = 0;

}

static void CCONV onErrorHandler_pot(PhidgetHandle phid, void *ctx,
     Phidget_ErrorEventCode errorCode, const char *errorString) {

fprintf(stderr, "[Phidget Error Event] -> %s (%d)\n", errorString,
    errorCode);
}

static void CCONV onAttachHandler_motor(PhidgetHandle phid, void *
    ctx) {
PhidgetReturnCode res;
int hubPort;
int channel;
int serial;

res = Phidget_getDeviceSerialNumber(phid, &serial);
if (res != EPHIDGET_OK) {
fprintf(stderr, "failed to get device serial number\n");
return;
}

res = Phidget_getChannel(phid, &channel);
if (res != EPHIDGET_OK) {
fprintf(stderr, "failed to get channel number\n");
return;
}

res = Phidget_getHubPort(phid, &hubPort);
if (res != EPHIDGET_OK) {
fprintf(stderr, "failed to get hub port\n");
hubPort = -1;
}

if (hubPort == -1)
printf("channel %d on device %d attached\n", channel, serial);
else
printf("channel %d on device %d hub port %d attached\n", channel,
    serial, hubPort);
}

static void CCONV onDetachHandler_motor(PhidgetHandle phid, void *
    ctx) {
PhidgetReturnCode res;
int hubPort;
int channel;
int serial;

res = Phidget_getDeviceSerialNumber(phid, &serial);
if (res != EPHIDGET_OK) {
fprintf(stderr, "failed to get device serial number\n");
return;
```

```
}

res = Phidget_getChannel(phid, &channel);
if (res != EPHIDGET_OK) {
fprintf(stderr, "failed to get channel number\n");
return;
}

res = Phidget_getHubPort(phid, &hubPort);
if (res != EPHIDGET_OK)
hubPort = -1;

if (hubPort != -1)
printf("channel %d on device %d detached\n", channel, serial);
else
printf("channel %d on device %d hub port %d detached\n", channel,
    hubPort, serial);
}

static void CCONV errorHandler_motor(PhidgetHandle phid, void *ctx,
     Phidget_ErrorEventCode errorCode, const char *errorString) {

fprintf(stderr, "Error: %s (%d)\n", errorString, errorCode);
}

static void CCONV onPositionChangeHandler(PhidgetRCServoHandle ch,
    void *ctx, double position) {

//printf("[Motor Event] New Thrust set to %f\n", position);

}

static void CCONV onVoltageChangeHandler(PhidgetVoltageInputHandle
    pvih, void *ctx, double volt) {

//printf("[Potensiometer Event] Voltage changed to %f\n", volt);
}

void displayError(PhidgetReturnCode returnCode) {
    /*Prints error message to screen*/

    PhidgetReturnCode prc;
    const char* error;

    prc = Phidget_getErrorDescription(returnCode, &error);
    if (EPHIDGET_OK != prc) {
        fprintf(stderr, "Runtime Error -> Getting ErrorDescription:
     \n\t");
        displayError(prc);
        return;
    }

    fprintf(stderr, "Desc: %s\n", error);
}


int setHandlers(PhidgetVoltageInputHandle potHandle,
```

```
    PhidgetRCServoHandle motorHandle) {
    /* Sets event handlers for the potensiometer and the motor.
     * These handlers will be fired if the event happens.
     * CCONV is defined "Calling Conventions" from Phidget library.
     *
     * Return 0 if successful, 1 otherwise. Relevant messages are printed to screen.
     */
    PhidgetReturnCode prc;

    printf("\nSetting potensiometer handlers: OnAttach, OnDetach, OnError and OnVoltageChange\n");
    prc = Phidget_setOnAttachHandler((PhidgetHandle) potHandle, onAttachHandler_pot, NULL);
    if (EPHIDGET_OK != prc) {
        fprintf(stderr, "Runtime Error -> Set Potensiometer Attach Handler: \n\t");
        displayError(prc);
        return 1;
    }

    prc = Phidget_setOnDetachHandler((PhidgetHandle) potHandle, onDetachHandler_pot, NULL);
    if (EPHIDGET_OK != prc) {
        fprintf(stderr, "Runtime Error -> Set Potensiometer Detach Handler: \n\t");
        displayError(prc);
        return 1;
    }

    prc = Phidget_setOnErrorHandler((PhidgetHandle) potHandle, onErrorHandler_pot, NULL);
    if (EPHIDGET_OK != prc) {
        fprintf(stderr, "Runtime Error -> Set Potensiometer Error Handler: \n\t");
        displayError(prc);
        return 1;
    }

    prc = PhidgetVoltageInput_setOnVoltageChangeHandler( potHandle, onVoltageChangeHandler, NULL);
    if (EPHIDGET_OK != prc) {
        fprintf(stderr, "Runtime Error -> Set Potensiometer OnVoltageChange Handler: \n\t");
        displayError(prc);
        return 1;
    }

    printf("Setting motor handlers: OnAttach, OnDetach, OnError and OnPositionChange\n");
    prc = Phidget_setOnAttachHandler((PhidgetHandle) motorHandle, onAttachHandler_motor, NULL);
    if (prc != EPHIDGET_OK) {
        fprintf(stderr, "Runtime Error -> Set Motor Attach Handler: \n\t");
        return 1;
    }
```

```
    prc = Phidget_setOnDetachHandler((PhidgetHandle) motorHandle,
    onDetachHandler_motor, NULL);
    if (prc != EPHIDGET_OK) {
        fprintf(stderr, "Runtime Error -> Set Motor Detach Handler:
     \n\t");
        return 1;
    }

    prc = Phidget_setOnErrorHandler((PhidgetHandle) motorHandle,
    errorHandler_motor, NULL);
    if (prc != EPHIDGET_OK) {
        fprintf(stderr, "Runtime Error -> Set Motor Error Handler:
    \n\t");
        return 1;
    }

    prc = PhidgetRCServo_setOnPositionChangeHandler( motorHandle,
    onPositionChangeHandler, NULL);
    if (prc != EPHIDGET_OK) {
        fprintf(stderr, "Runtime Error -> Set Motor
    OnPositionChange Handler: \n\t");
        return 1;
    }

    printf("All event handlers set\n");
    return 0;
}
```

Listing A.5: phidgetHelper.c

```c
#include "main.h"

void gnuPipe(const char *fileName){
    /*  GnuPipe plots with gnuplots by opening a terminal and entering Gnuplot commands.
     *
     *  It reads the log from fileName[dat], which has following format:
     *  time [sec], position[cm], prc-Error, input , error[cm], P-term, I-term, D-term, Offset
     *
     */


    printf("Creating Gnuplots using a Gnuplot-Pipe\n");

    //Open terminal to write gnuplot commands.
    FILE *gPipe = popen("gnuplot", "w");
    if (gPipe==NULL) {
        printf("Error opening pipe to GNU plot. Check if you have it! \n");
    }

    /* Plotting the position and error-percentage as a function of time */
    fprintf(gPipe, "set key reverse Left outside\n");
    fprintf(gPipe, "set grid x y\n");
    fprintf(gPipe, "set title 'Position and Error as Percentage' font ',20'\n");
    fprintf(gPipe, "set xlabel 'Time [sec]';set ylabel 'Position [cm]'; set y2label 'Error [%%]'\n");
    fprintf(gPipe, "set autoscale xy\n");
    fprintf(gPipe, "set y2range [-40:60]; set y2tics nomirror 20\n");
    fprintf(gPipe, "set tics out\n");
    fprintf(gPipe, "plot '%s' w l lt 1 t 'Position',", fileName);
    fprintf(gPipe, "'%s' u 1:3 w l lt 3 t 'prc-Error' axes x1y2, %d w l lt 4 t 'Set point\n", fileName, setPoint);
    fprintf(gPipe, "set term png\n");
    fprintf(gPipe, "set output '../logs/position%d.png'\n", fileNum);
    fprintf(gPipe, "replot\n");

    /* Plotting the output to motor and error as a function of time */
    fprintf(gPipe, "reset\n");
    fprintf(gPipe, "set key reverse Left outside\n");
    fprintf(gPipe, "set grid x y\n");
    fprintf(gPipe, "set title 'Input to Motor and Error' font ',20'\n");
    fprintf(gPipe, "set xlabel 'Time [sec]';set ylabel 'Thrust'\n");
    fprintf(gPipe, "set tics out\n");
    fprintf(gPipe, "set y2label 'Distance [cm]'\n");
    fprintf(gPipe, "set y2tics nomirror 20\n");
    fprintf(gPipe, "plot '%s' using 1:4 w l lt 1 t 'Input',", fileName);
```

```c
    fprintf(gPipe, "'%s' using 1:5 w l lt 3 t 'Error' axes x1y2\n",
     fileName);
    fprintf(gPipe, "set term png\n");
    fprintf(gPipe, "set output '../logs/input%d.png' \n", fileNum);
    fprintf(gPipe, "replot\n");

    /* Plotting the output and the P, I and D contributions */
    fprintf(gPipe, "reset\n");
    fprintf(gPipe, "set key reverse Left outside\n");
    fprintf(gPipe, "set grid x y\n");
    fprintf(gPipe, "set title 'P, I and D terms' font ',20'\n");
    fprintf(gPipe, "set xlabel 'Time [sec]';set ylabel 'Thrust'\n")
    ;
    fprintf(gPipe, "set tics out\n");
    fprintf(gPipe, "plot '%s' using 1:4 w l lt 1 t 'TotalInput',",
    fileName);
    fprintf(gPipe, "'%s' using 1:6 w l lt 2 t 'Proportional','%s'
    using 1:7 w l lt 3 t 'Integral'," , fileName, fileName);
    fprintf(gPipe, "'%s' using 1:8 w l lt 4 t 'Derivative',%d w l
    lt 6 t 'OffSetInput\n ", fileName, offSetMotor);

    fprintf(gPipe, "set term png\n");
    fprintf(gPipe, "set output '../logs/inputTerms%d.png' \n",
    fileNum);
    fprintf(gPipe, "replot\n");
    fprintf(gPipe, "reset\n");

    //Close the pipe
    fclose(gPipe);

    printf("gnuPipe finished. Plots are in the logs folder.\n");
}
```

Listing A.6: gnuPlot.c