Industrial Automation
Position and Temperature

Thijs van Boesschoten, Hussam Ayoub, Niousha Moshirzedeh
Fontys University of Applied sciences
December 26, 2019

Abstract

This document will go through the process of automating an industrial process using industrial equipment in the form of PLCs.
This will entail the tools used for development, the design of the application and breakdown of the problem in states, using matlab to generate values for the PID system used in the program and binding it together using the Structured Text programming language.

Version Management

Version	Changes	Reason
v0.1	All	First Draft

Contents

1 Introduction

For this project, a company assigned us to automate the process of a movable tank. The tank will be used to contain and mix liquids(chemicals) in a temperature controlled environment.

The general tasks that the system should be able to fulfill are:

1. Fill liquid A into the tank.
2. Fill liquid B into the tank.
3. Stir liquids A&B while heated to a temperature setpoint.
4. Move the tank to a position setpoint.
5. Empty the tank.
6. Return the tank to the default position.

To fulfill these functionalities the following hardware is installed and can be manipulated by our program:

- Value A, B & C
- Stirrer
- Level Switch S1, S2 & S3
- Heater
- Temperature Sensor
- Servo Motor

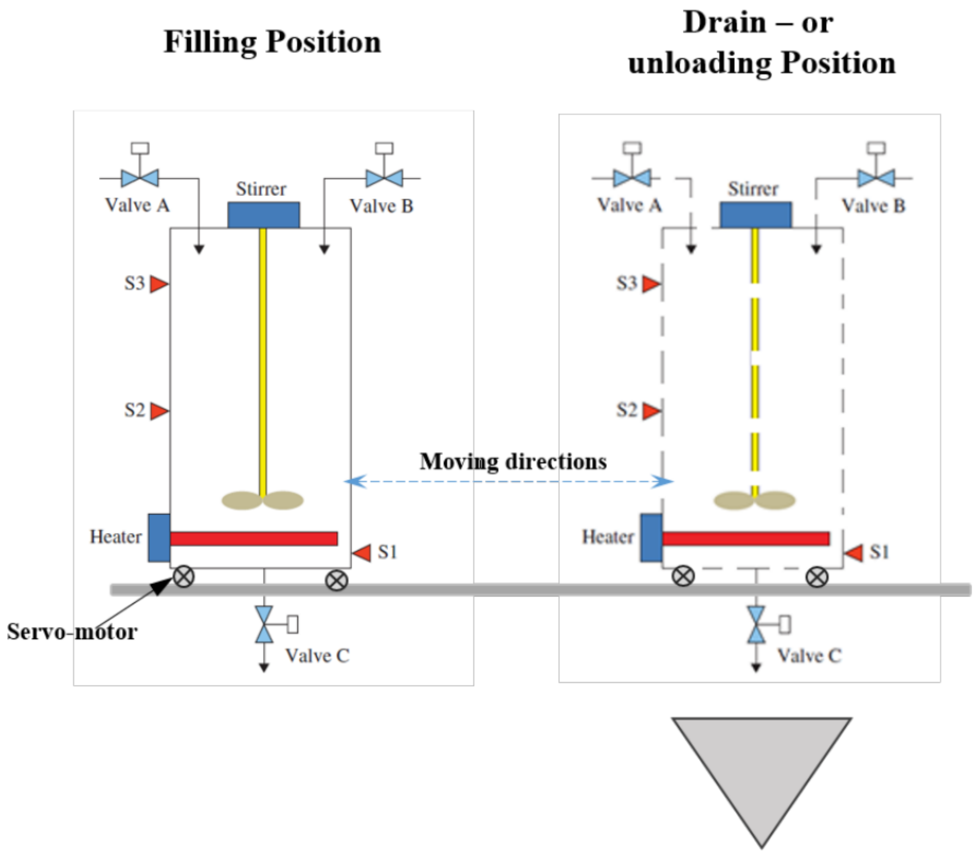


Figure 1: Production set-up

2 Tools & Equipment

For the development of our program, Twincat 3.0 will be used as the work that is provided will be in the form of a Twincat 3.0 archived project. Extracting this project shows that the basics like I/O and a base program has already been implemented. However this implementation is very crude and can use more refinement. In this manner, the first thing to be improved will be the visualization.

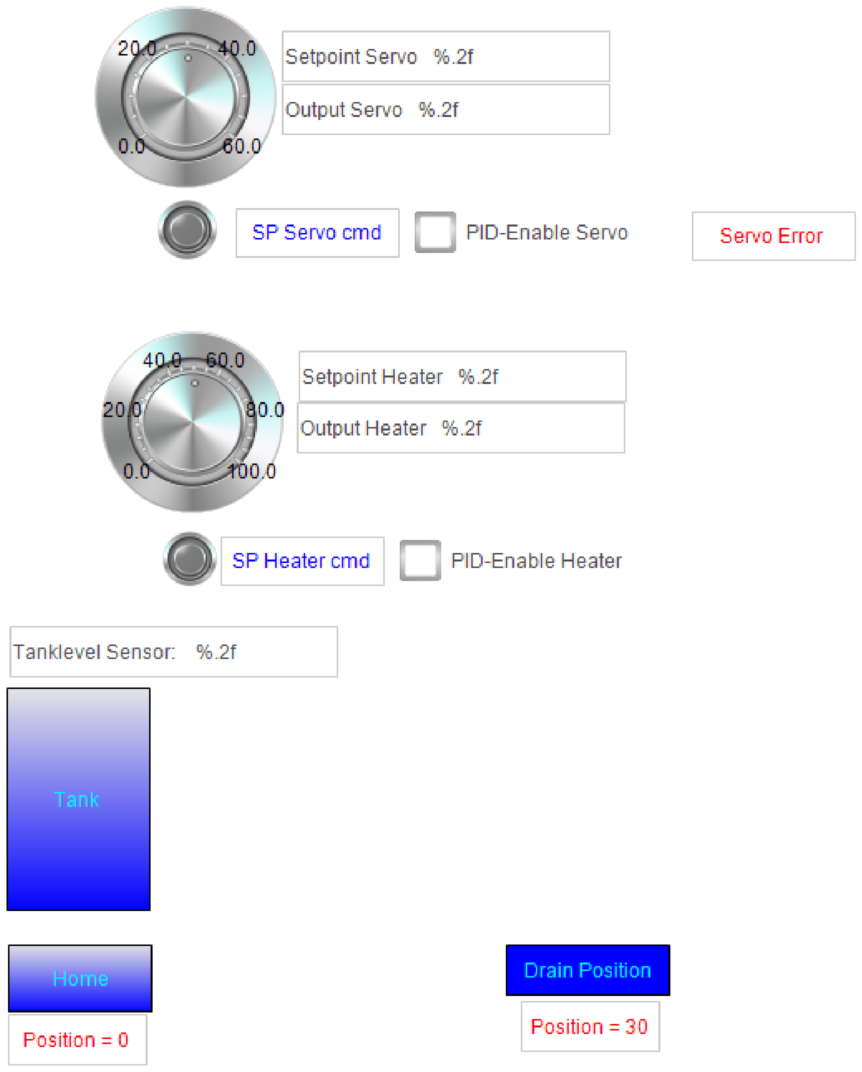


Figure 2: Old Visualization

Currently the UI is rather cluttered and not all information like the valves and stirrer are displayed.

As such a new visualization has been created to show all input information and the states of the outputs.

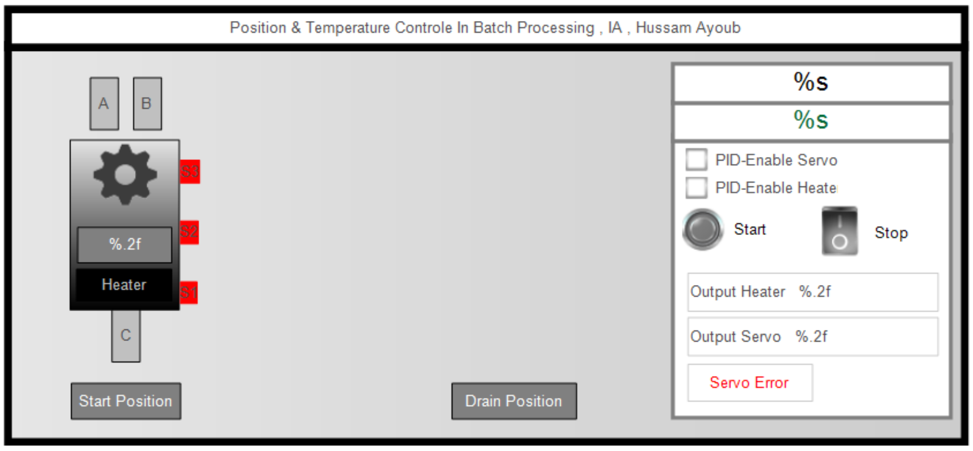


Figure 3: State Diagram of the process

3 Program Design

Having set the parameters of the project and a way to visualize the result, the process is the next thing in line. To start of has a state diagram been made to show the manner at which the program should operate given the criteria.

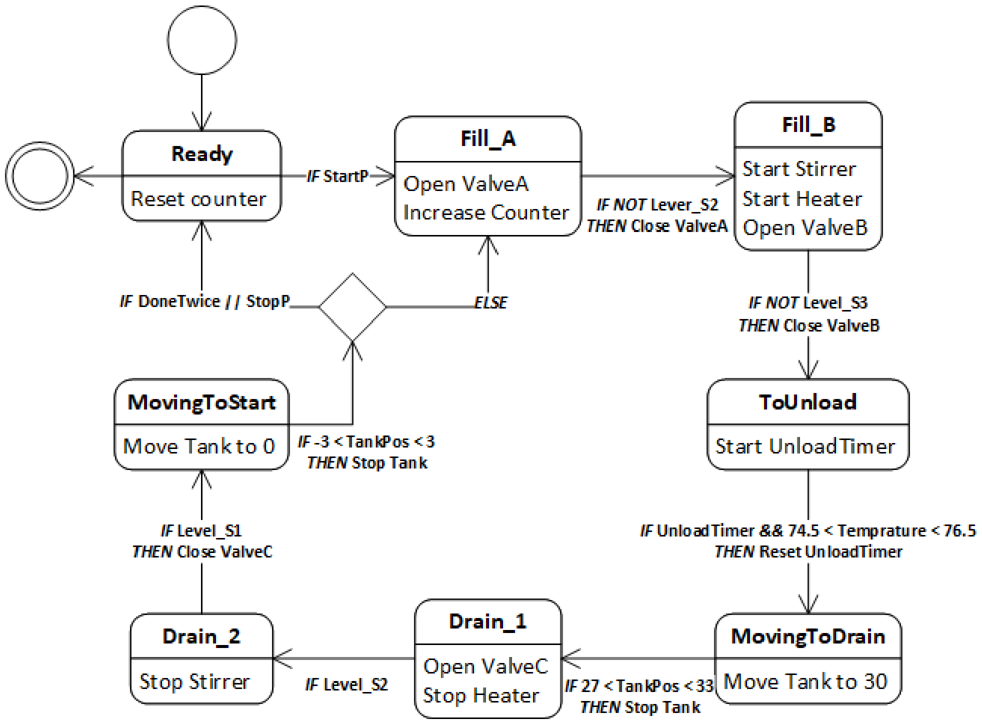


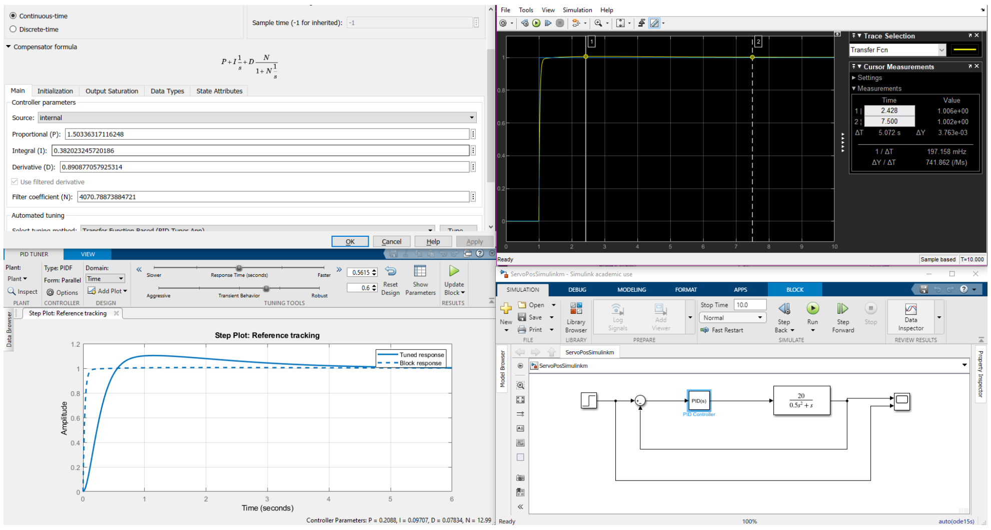
Figure 4: New Visualization

Most of this code has already been implemented, the main section that needed to be filled in was the MAIN section which holds the state of the program, and the PID controller (fBDISCRETE_PID) that needs to be completed.

4 PID & Matlab

Starting with the PID controller needed for the movement of the tank to go fluently, the first part is to understand the system.

In the code is a transfer function given of $\frac{20}{0.5s^2+s}$. The transfer function can be used in Matlab with a PID controller to get the P, I and D values for out system.



Compensator formula: $P + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}}$

Sample time (-1 for inherited): -1

Source: internal
Proportional (P): 1.50336317116248
Integral (I): 0.382023245720186
Derivative (D): 0.890877057925314
Use filtered derivative
Filter coefficient (N): 4070.78873884721

Type: PIDF
Form: Parallel
Domain: Time
Response Time (seconds): 0.5615
Transient Behavior: 0.6

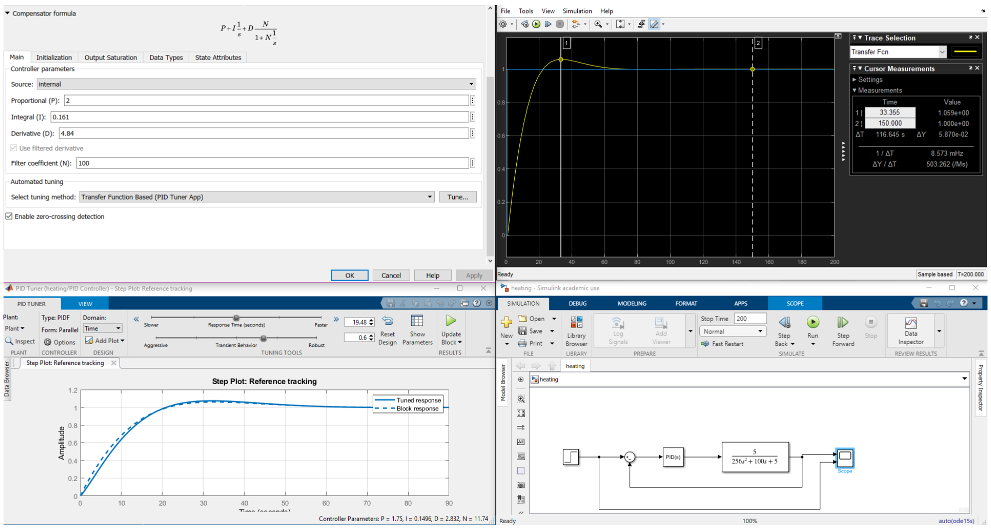
Cursor Measurements

	Time	Value
1	2.428	1.006e+00
2	7.500	1.002e+00
ΔT	5.072 s	ΔY 3.763e-03
1 / ΔT	197.158 mHz	
ΔY / ΔT	741.862 (/Ms)	

Controller Parameters: P = 0.2088, I = 0.09707, D = 0.07834, N = 12.99

Figure 5: PID Tuning for Servo

As can be seen in the image, the values of $P = 1.50336317116248$, $I = 0.382023245720186$ and $D = 0.890877057925314$ give an output that is fast and provides an overshoot that is within the 2% overshoot allowed.



Compensator formula: $P + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}}$

Source: internal
Proportional (P): 2
Integral (I): 0.161
Derivative (D): 4.84
Use filtered derivative
Filter coefficient (N): 100
Select tuning method: Transfer Function Based (PID Tuner App)
Enable zero-crossing detection

PID Tuner (heating/PID Controller) - Step Plot: Reference tracking

Type: PIDF
Form: Parallel
Domain: Time
Response Time (seconds): 19.48

Cursor Measurements

	Time	Value
1	33.355	1.059e+00
2	150.000	1.000e+00
ΔT	116.645 s	ΔY 5.870e-02
1 / ΔT	8.573 mHz	
ΔY / ΔT	503.262 (/Ms)	

Controller Parameters: P = 1.75, I = 0.1496, D = 2.832, N = 11.74

Figure 6: PID Tuning for Heater

Doing the same for the heater yields the values $P = 2$, $I = 0.161$ and finally $D = 4.84$.

These values for the PID controllers come in useful, however can be utilized better by programmers in the "Advanced" PID - Algorithm [1] form.

$A = (k_p + k_i\frac{T}{2} + \frac{k_d}{T})$, $B = (-k_p + k_i\frac{T}{2} - \frac{2k_d}{T})$ and finally $C = \frac{k_d}{T}$

5 Implementation

With all the theoretical pieces together it is time to put things into action, starting of with the state machine in the MAIN file. Simply by using the following states programmed in:

```
TYPE SeqState :
(
        Ready,
        Fill_A ,
        Fill_B ,
        Heating ,
        ToUnload ,
        MovingToDrain ,
        MovingToStart ,
        Drain_1 ,
        Drain_2
);
END_TYPE
```

These programmed states can then be used in the main to go through in the form of a switch statement:

```
CASE States OF // Sequence State
SeqState.Ready:
        StateString := 'Ready!';
        Counter := 0; // sequence counter
        HeaterON := FALSE;
        IF GVL_IO.StartPr THEN // check if start button pressed
                States := SeqState.Fill_A; // go to state Fill A
        END_IF

SeqState.Fill_A: // sequence state fill A
        TotalStart := TRUE;
        ...
```

Activating the Servo is done as follows:

```
//
// *************  Servo with PID Controller  **************
//
IF (MoveTank AND NOT GVL_IO.xGenCtrl_Error ) THEN
        pidServo.Setpoint:=GVL_IO.SetPoint_ServoX;
END_IF

pidServo.Kp:= 1.503; // Kp parameter
 pidServo.Ki:= 0.382 ; // Ki parameter
 pidServo.Kd:= 0.89 ; // Kd parameter

pidServo.MV_max_sat:= 10; // To protect Servo
pidServo.MV_min_sat:= −10;
pidServo.xExt_Error:= Gvl_IO.xGenCtrl_Error;
pidServo.Yprocess:= LREAL_TO_REAL(GVL_IO.Output_ServoX);
pidServo(PID_enable:=PID_Servo_enable);
```

This then leads into the fBDISCRETE_PID function block which does the following to utilize the PID controller:

```
Perr:= Setpoint − Yprocess;

A := (Kp + Ki ∗ (Tsample/2) + (Kd/Tsample));
B := (−Kp + Ki ∗(Tsample/2) − (2∗Kd/Tsample));
C :=  Kd/Tsample;

Mv_out:= prevMV_out +  A∗Perr + B ∗ Perr1 + C ∗ Perr2;

// Update  samples
prevMV_out := MV_out; // prevMV_out = MV_out[k−1]
Perr2 := Perr1;       // e[k−2]
Perr1 := Perr;        // e[k−1]
```

References

1. Oswald Figaro. Industrial Automation. *intro module*, 5:17–21.