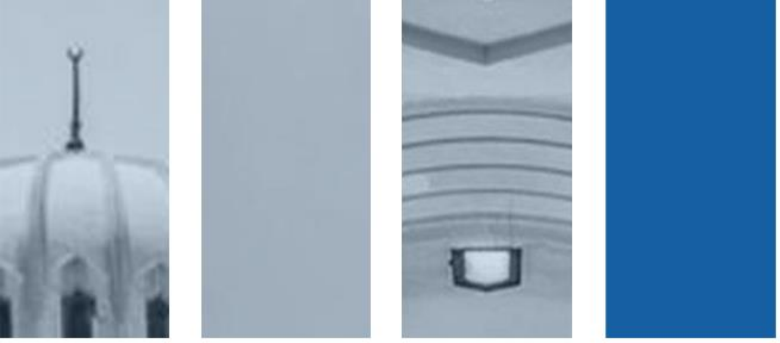


RESEARCH & PROJECT SUBMISSIONS

Program:

Course Code: CSE372

Course Name: Control Systems (2)

Examination Committee

Prof. Dr. Sherif Ali Mohamed Hammad

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

Student Personal Information for Group Work

Student Names:	Student Codes:	Student Tel.:	Student E-mail:
Ayman Mohamed Saad Eldeen Mohamed	14T0045	01099613699	14T0045@eng.asu.edu.eg
Bassem Osama Farouk AbdElWahab	1500400	01062456397	1500400@eng.asu.edu.eg
Khalid Shehab Shams Eldeen Abo Elyazeed	15x0031	01121276113	15x0031@eng.asu.edu.eg
Mohamed Samy Sadek Mohamed	1501214	01010751546	1501214@eng.asu.edu.eg
Kirollos Samir Asaad Qudas	1601023	01222433112	1601023@eng.asu.edu.eg

Plagiarism Statement

Signature/Student Name: ايمن محمد سعد الدين **Date: 14/6/2020**

باسم أسامة فاروق

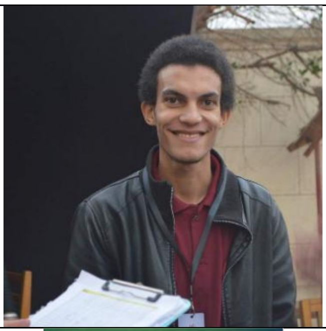
خالد شهاب

[illegible]

كيرلس سمير أسعد

Submission Contents

Personal photos & individual contributions:

Mohamed Samy Sadek Mohamed (C Code generation & illustration video)	
Bassem Osama Farouk AbdElWahab (Validation phase)	
Khalid Shehab Shams Eldeen Abo Elyazeed (.h parameters files generation & Report)	
Ayman Mohamed Saad Eldeen Mohamed (RST Parameters Calculation & Report)	
Kirollos Samir Asaad Qudas (Process simulation on GUI & Video editing)	

Contents

List of Figures

01

First Topic

ABSTRACT

1. ABSTRACT

Digital controllers became so popular due to low cost and flexibility to be changed. The main objective of this research is to build (.exe & installer) app which can simulate “a plant with its disturbances” controlled by RST Digital Controller. In this research, MATLAB GUI and functions are used to convert the plant from continuous to discrete and tune the RST controller parameters to get the target characteristics desired to the system & also to simulate the process. The presented app can generate the system & RST parameters on .h files. generation of C code is also supported by this app, which can run on Tiva C ARM-CORTEX M4.

02

Second Topic

PROBLEM STATEMENT

2. PROBLEM STATEMENT

2.1. Introduction

Digital controllers became so popular since they don't cost much, flexible to change later & more adjustable to environment disturbance as we can handle delays & loss of information. Digital systems could be made by

1. converting continuous controller to discrete one & use digital to analog converter.
2. converting the continuous plant and sensor to discrete plant and sensor.
3. tune a discrete controller with a continuous plant model using model base programming, for example: "Simulink".

"code generator" involves other resources or tools that help to turn out specific kinds of code. Developing tool to generate c code that run on the microcontroller will speed up the development process of digital micro controller and makes it easy to be done. By using the code generator utilize the advantage of digital controller (the flexibility to changes).

The Tiva C Series microcontrollers provide a broad portfolio of floating point enabled ARM Cortex-M4F MCUs, Figure 1. Designers who migrate to the Tiva C Series MCUs benefit from a balance between the performance needed to create highly responsive mixed-signal applications and the low power architecture required to address increasingly aggressive power budgets. Tiva C Series MCUs are supported by TivaWare™ for C Series, designed specifically for those customers who want to get started easily, write production-ready code quickly, and minimize their overall cost of software ownership.

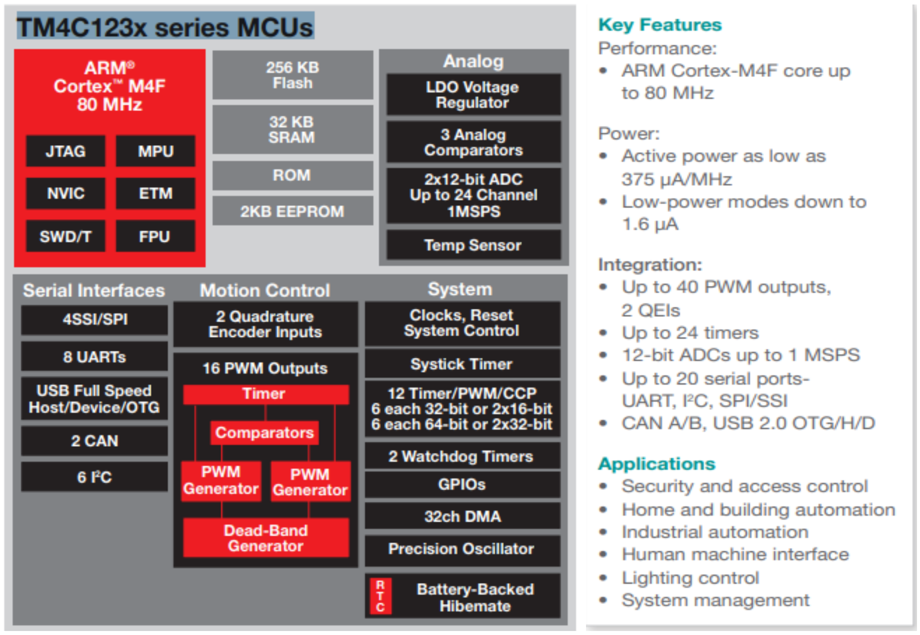


Figure 1 TM4C123x series MCUs Specification and architecture

2.2. Problem Statement

Developing code for control system is a long-time process due to additional requirements related to the system correctness and real time requirements. these are critical requirement for a control system as any fault my case damage to the system, reducing the system correctness or make it too risky for the surrounding environment. The development cycle of a software consists of several steps including requirement phase, design, development, testing & maintenance as shown in Figure 2.

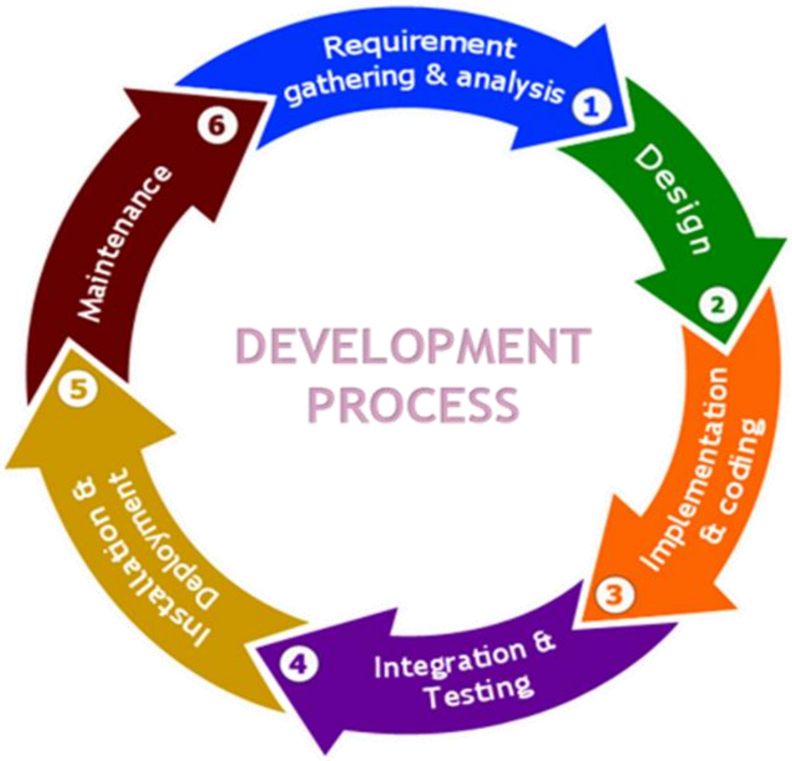


Figure 2 Steps of development process

Developing code generator tool will reduce the time taken in implementation & coding phase making the developing process shorter & give more time for testing phase, as it is very critical in digital controller with real time requirements. In additional to the code generation, using a digital controller will solve the problems of continuous controller high cost & the fact that it’s not flexible for changes.

03 SOLUTION DESIGN

Third Topic

3. Solution design

MATLAB GUI tools were used to simulate, generate the system parameters and the RST Controller subsystem code which will be run on Tiva. An .exe app & installer are then constructed to be easy to run without using the MATLAB. The generated .c code is in a general formula which has the constant RST Parameters included from the .h files. The useful screenshots are presented below in Figures 3, 4, 5 and 6.

3.1. Screenshots

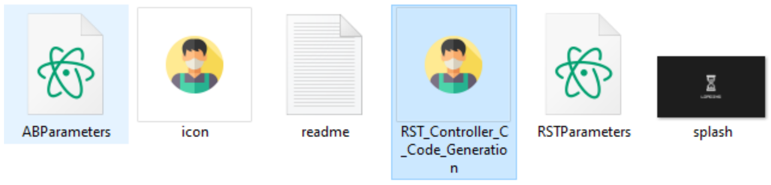


Figure 3 App folder contents

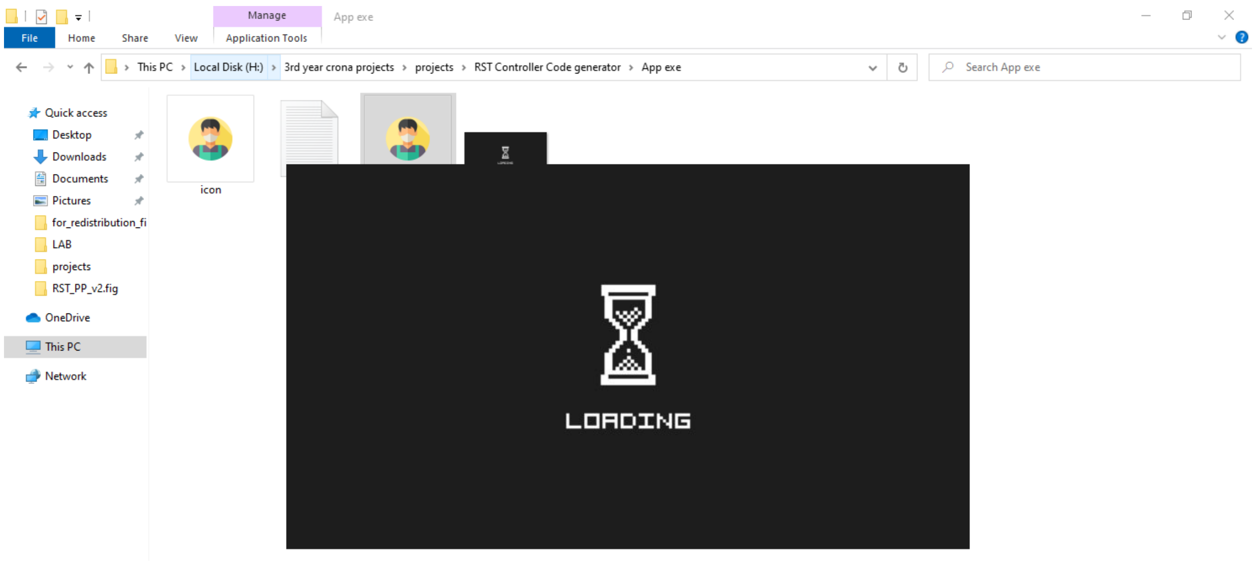
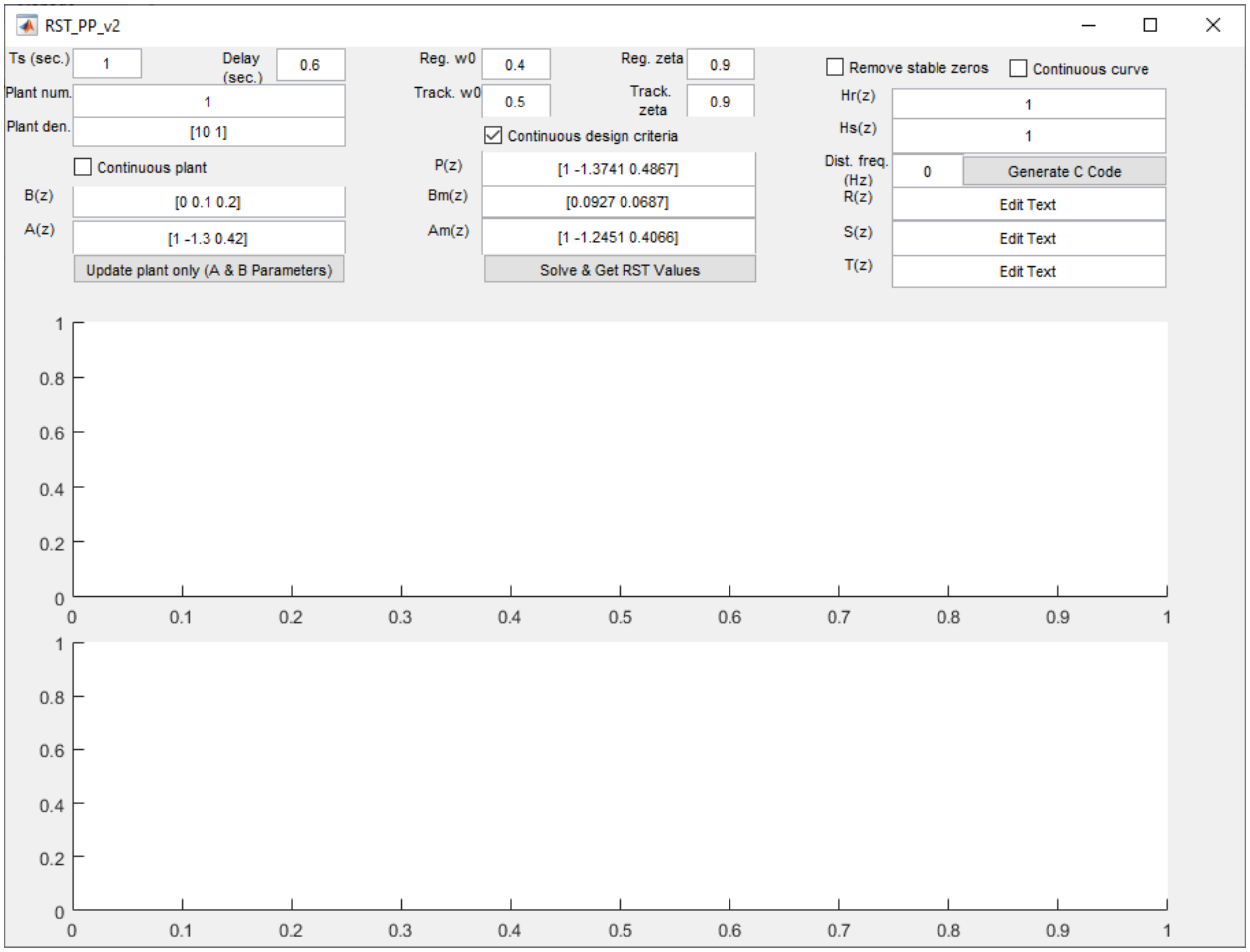
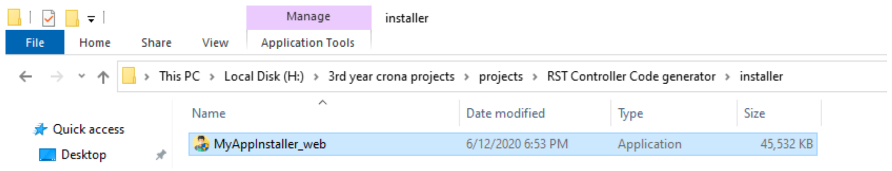


Figure 4 Loading window after clicking the .exe



Ts (sec.) 1
Delay (sec.) 0.6
Plant num. 1
Plant den. [10 1]
[ ] Continuous plant
B(z) [0 0.1 0.2]
A(z) [1 -1.3 0.42]
Reg. w0 0.4
Reg. zeta 0.9
Track. w0 0.5
Track. zeta 0.9
[✓] Continuous design criteria
P(z) [1 -1.3741 0.4867]
Bm(z) [0.0927 0.0687]
Am(z) [1 -1.2451 0.4066]
[ ] Remove stable zeros
[ ] Continuous curve
Hr(z) 1
Hs(z) 1
Dist. freq. (Hz) 0
R(z) Edit Text
S(z) Edit Text
T(z) Edit Text

Figure 5 GUI window



This PC › Local Disk (H:) › 3rd year crona projects › projects › RST Controller Code generator › installer

Name	Date modified	Type	Size
MyAppInstaller_web	6/12/2020 6:53 PM	Application	45,532 KB

Figure 6 App installer

3.2. Tool ideal scenario:

1. The user opens the installer and follows the installations steps, or just using the .exe file.
2. The user then inserts the actual Ap and Bp parameter for the plant.
3. The user clicks the "(A&B) Parameter" button and then, button callback function handle the click generating .h file includes all the system parameters is generated in the same directory of the app. (ABparameters.h).
4. The user inserts the system needed characteristics (regulation dynamics and tracking dynamics given).
5. The user clicks "solve & get RST Parameters" to get the Controller parameter on .h file in the same directory and to draw the plant output & step disturbance. (RSTParameter.h).
6. The user then clicks "generate C code" to generate two .h files for types and structs & .c file for the RST Controller output u(t) in term of reference input and plant feedback. (rtwtypes.h, Subsystem.h, Subsystem.c, main.c).

The equations used in code and parameters generation are shown in Figures 7, 8 and 9.

$$u(t) = \frac{T(q^{-1})y^*(t+d+1) - R(q^{-1})y(t)}{S(q^{-1})}$$

$$S(q^{-1})u(t) + R(q^{-1})y(t) = GP(q^{-1})y^*(t+d+1) = T(q^{-1})y^*(t+d+1)$$

$$S(q^{-1}) = 1 + q^{-1}S^*(q^{-1})$$

$$u(t) = P(q^{-1})Gy^*(t+d+1) - S^*(q^{-1})u(t-1) - R(q^{-1})y(t)$$

$$y^*(t+d+1) = \frac{B_m(q^{-1})}{A_m(q^{-1})}r(t)$$

$$A_m(q^{-1}) = 1 + q^{-1}A_m^*(q^{-1})$$

$$y^*(t+d+1) = -A_m^*(q^{-1})y(t+d) + B\_m(q^{-1})r(t)$$

$$B_m(q^{-1}) = b_{m0} + b_{m1}q^{-1} + b_{m2}q^{-2} + \cdots \qquad A_m(q^{-1}) = 1 + a_{m1}q^{-1} + a_{m2}q^{-2} + \cdots$$

Figure 7Used Equations in code & parameters generation-a

Plant: d=0

$$B(q^{-1}) = 0.1q^{-1} + 0.2\,q^{-2}$$

$$A(q^{-1}) = 1 - 1.3q^{-1} + 0.42q^{-2}$$

Tracking dynamics:

$$B_m(q^{-1}) = 0.0927 + 0.0687q^{-1}$$

$$A_m(q^{-1}) = 1 - 1.2451q^{-1} + 0.4066q^{-2}$$

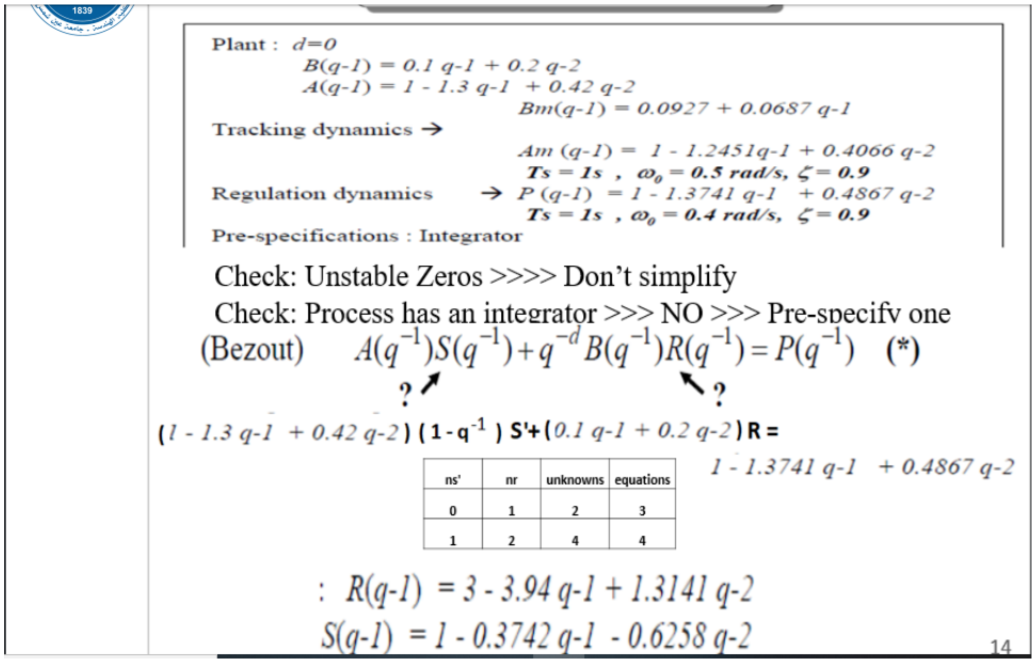
T_s = 1s , ω_0=0.5 rad/s , ζ=0.9

Regulation dynamics:

$$P(q^{-1}) = 1.1.3741q^{-1} + 0.4867q^{-2}$$

T_s = 1s , ω_0=0.4 rad/s , ζ=0.9

Figure 8 Used Equations in code & parameters generation-b



Plant : $d=0$

$B(q\text{-}1) = 0.1\ q\text{-}1 + 0.2\ q\text{-}2$

$A(q\text{-}1) = 1 - 1.3\ q\text{-}1 + 0.42\ q\text{-}2$

$Bm(q\text{-}1) = 0.0927 + 0.0687\ q\text{-}1$

Tracking dynamics →

$Am\ (q\text{-}1) = 1 - 1.2451q\text{-}1 + 0.4066\ q\text{-}2$

$Ts = 1s\ ,\ \omega_0 = 0.5$ rad/s, $\zeta = 0.9$

Regulation dynamics → $P\ (q\text{-}1) = 1 - 1.3741\ q\text{-}1 + 0.4867\ q\text{-}2$

$Ts = 1s\ ,\ \omega_0 = 0.4$ rad/s, $\zeta = 0.9$

Pre-specifications : Integrator

Check: Unstable Zeros >>>> Don't simplify

Check: Process has an integrator >>> NO >>> Pre-specify one

(Bezout) $$A(q^{-1})S(q^{-1}) + q^{-d}B(q^{-1})R(q^{-1}) = P(q^{-1}) \quad (*)$$

? ↗ ↖ ?

$(1 - 1.3\ q\text{-}1 + 0.42\ q\text{-}2)\ (1 - q^{-1})\ S' + (0.1\ q\text{-}1 + 0.2\ q\text{-}2)\ R =$

$1 - 1.3741\ q\text{-}1 + 0.4867\ q\text{-}2$

ns'	nr	unknowns	equations
0	1	2	3
1	2	4	4

: $R(q\text{-}1) = 3 - 3.94\ q\text{-}1 + 1.3141\ q\text{-}2$

$S(q\text{-}1) = 1 - 0.3742\ q\text{-}1 - 0.6258\ q\text{-}2$

14

Figure 9 Used Equations in code & parameters generation-c

3.3. Block Diagrams & signals

The block diagram and signals are shown in Figure 10.

Hint (Subsystem represents the RST Controller. The inputs to the subsystem are the reference input & the feedback from the plant & output is u(t)).

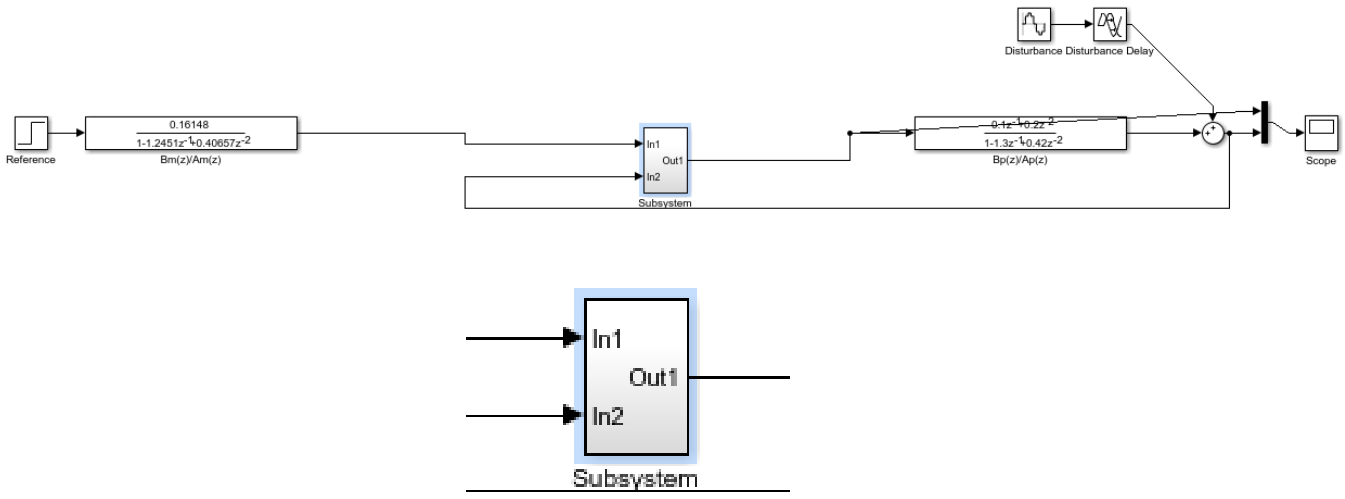
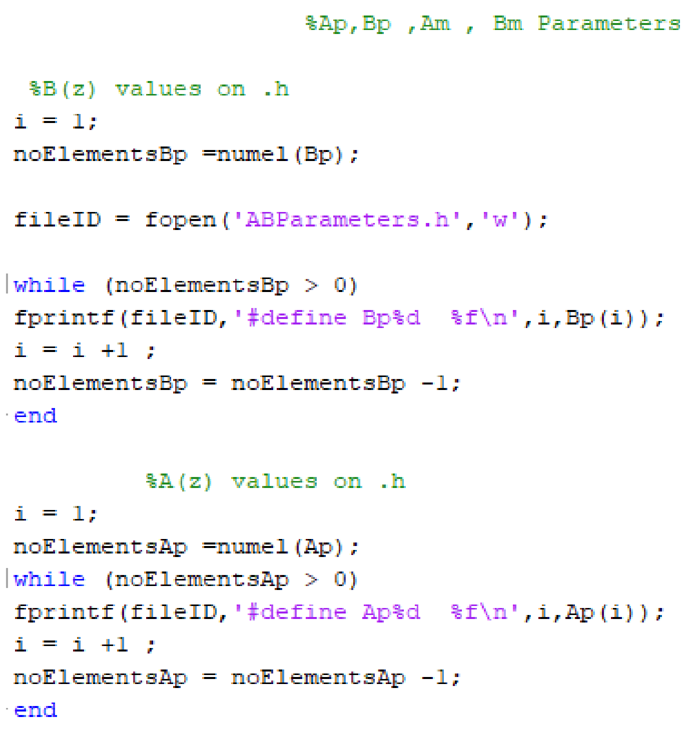


Figure 10 Block Diagrams & signals

3.4. Code .m Screenshots



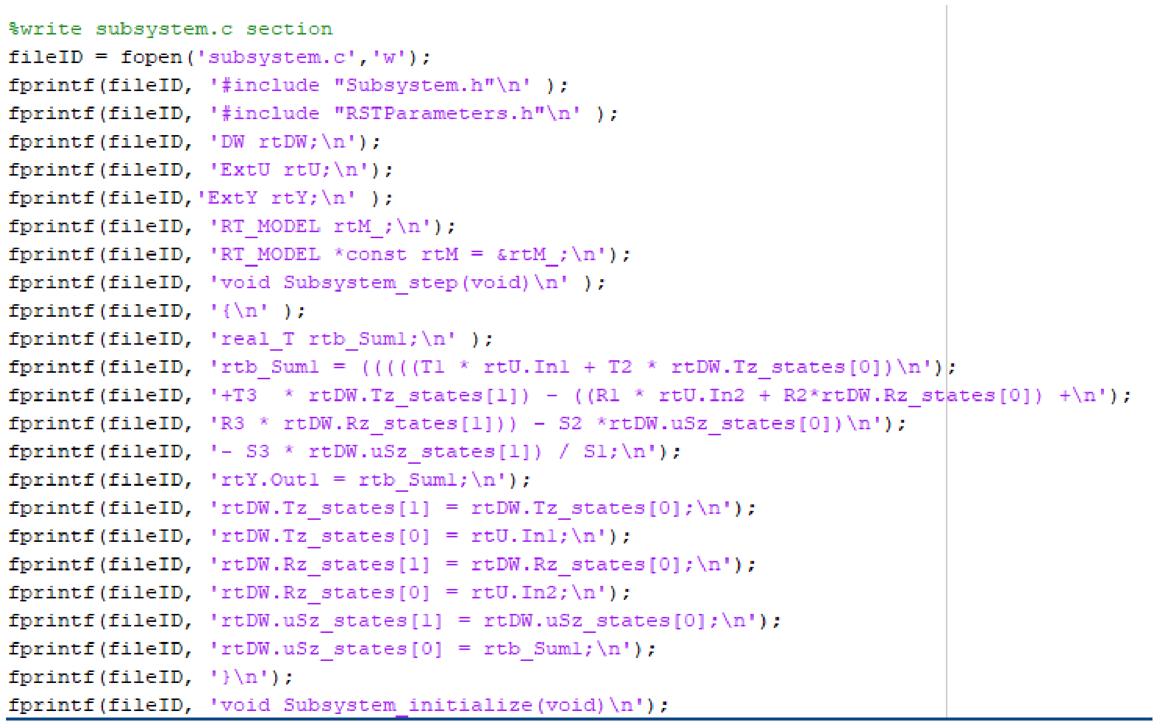
```
                    %Ap,Bp ,Am , Bm Parameters

 %B(z) values on .h
i = 1;
noElementsBp =numel(Bp);

fileID = fopen('ABParameters.h','w');

while (noElementsBp > 0)
fprintf(fileID,'#define Bp%d  %f\n',i,Bp(i));
i = i +1 ;
noElementsBp = noElementsBp -1;
end

          %A(z) values on .h
i = 1;
noElementsAp =numel(Ap);
while (noElementsAp > 0)
fprintf(fileID,'#define Ap%d  %f\n',i,Ap(i));
i = i +1 ;
noElementsAp = noElementsAp -1;
end
```



```
%write subsystem.c section
fileID = fopen('subsystem.c','w');
fprintf(fileID, '#include "Subsystem.h"\n' );
fprintf(fileID, '#include "RSTParameters.h"\n' );
fprintf(fileID, 'DW rtDW;\n');
fprintf(fileID, 'ExtU rtU;\n');
fprintf(fileID,'ExtY rtY;\n' );
fprintf(fileID, 'RT_MODEL rtM_;\n');
fprintf(fileID, 'RT_MODEL *const rtM = &rtM_;\n');
fprintf(fileID, 'void Subsystem_step(void)\n' );
fprintf(fileID, '{\n' );
fprintf(fileID, 'real_T rtb_Sum1;\n' );
fprintf(fileID, 'rtb_Sum1 = (((((T1 * rtU.In1 + T2 * rtDW.Tz_states[0])\n');
fprintf(fileID, '+T3  * rtDW.Tz_states[1]) - ((R1 * rtU.In2 + R2*rtDW.Rz_states[0]) +\n');
fprintf(fileID, 'R3 * rtDW.Rz_states[1])) - S2 *rtDW.uSz_states[0])\n');
fprintf(fileID, '- S3 * rtDW.uSz_states[1]) / S1;\n');
fprintf(fileID, 'rtY.Out1 = rtb_Sum1;\n');
fprintf(fileID, 'rtDW.Tz_states[1] = rtDW.Tz_states[0];\n');
fprintf(fileID, 'rtDW.Tz_states[0] = rtU.In1;\n');
fprintf(fileID, 'rtDW.Rz_states[1] = rtDW.Rz_states[0];\n');
fprintf(fileID, 'rtDW.Rz_states[0] = rtU.In2;\n');
fprintf(fileID, 'rtDW.uSz_states[1] = rtDW.uSz_states[0];\n');
fprintf(fileID, 'rtDW.uSz_states[0] = rtb_Sum1;\n');
fprintf(fileID, '}\n');
fprintf(fileID, 'void Subsystem_initialize(void)\n');
```

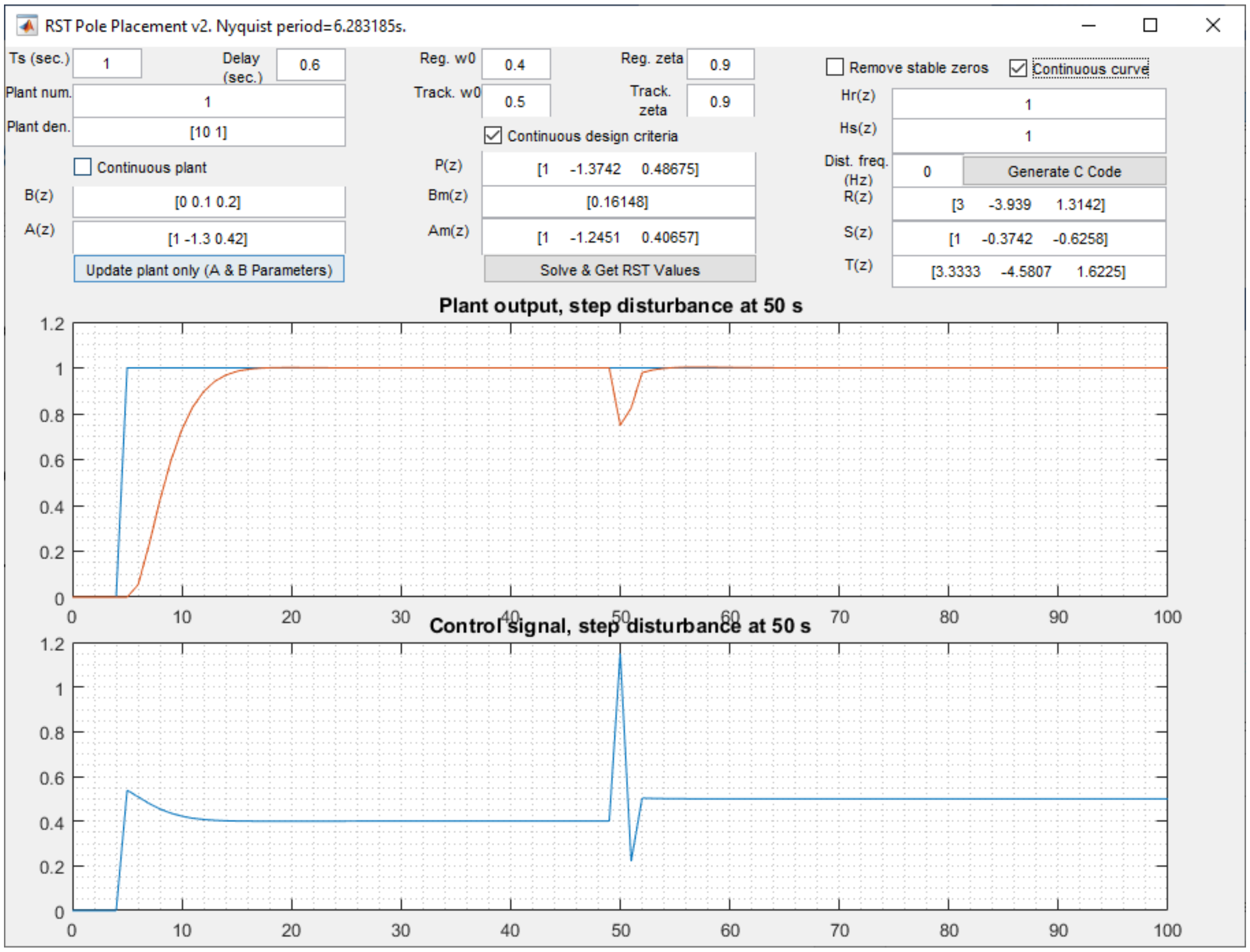
"all code can be found .m file call back functions"

04 *Fourth Topic* VALIDATION AND RESULTS

4. VALIDATION AND RESULTS

4.1. Testing the .h files parameters using the lecture example

Pole Placement: Part 1 lecture


RST Pole Placement v2. Nyquist period=6.283185s.

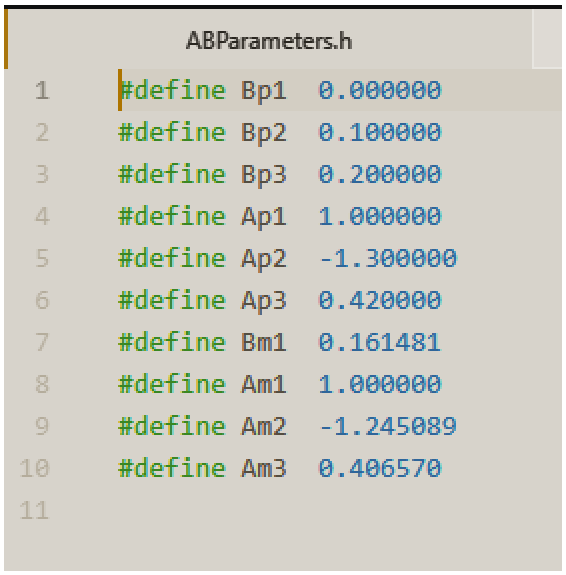
Ts (sec.) 1 | Delay (sec.) 0.6
Plant num. 1
Plant den. [10 1]
☐ Continuous plant
B(z) [0 0.1 0.2]
A(z) [1 -1.3 0.42]

Reg. w0 0.4 | Reg. zeta 0.9
Track. w0 0.5 | Track. zeta 0.9
☑ Continuous design criteria
P(z) [1 -1.3742 0.48675]
Bm(z) [0.16148]
Am(z) [1 -1.2451 0.40657]

☐ Remove stable zeros | ☑ Continuous curve
Hr(z) 1
Hs(z) 1
Dist. freq. (Hz) 0
R(z) [3 -3.939 1.3142]
S(z) [1 -0.3742 -0.6258]
T(z) [3.3333 -4.5807 1.6225]

Update plant only (A & B Parameters)
Solve & Get RST Values
Generate C Code
Plant output, step disturbance at 50 s
Control signal, step disturbance at 50 s
1.2
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
70
80
90
100

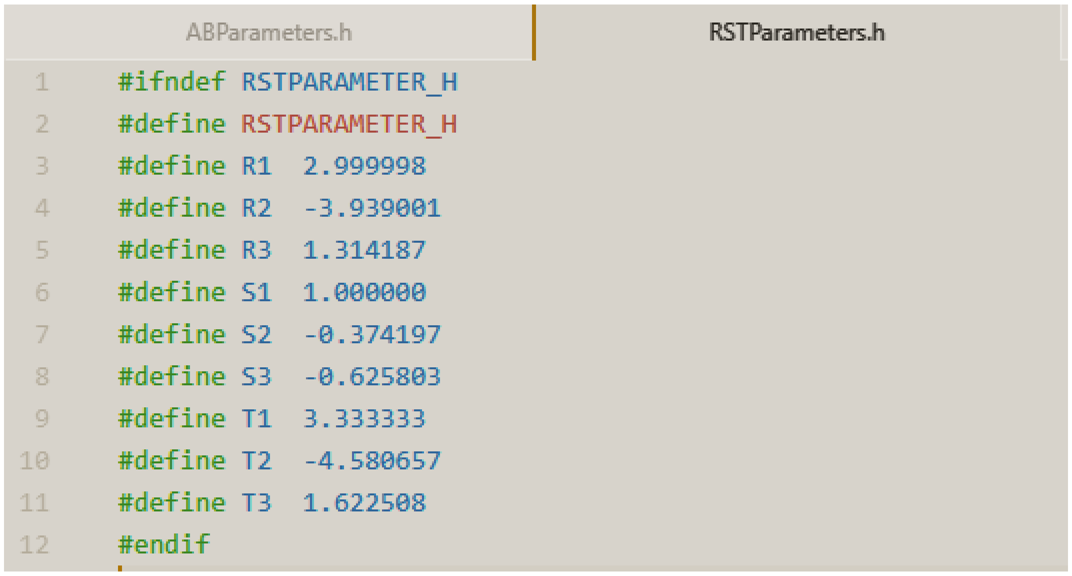
Figure 11 GUI lecture example results



ABParameters.h

```
#define Bp1  0.000000
#define Bp2  0.100000
#define Bp3  0.200000
#define Ap1  1.000000
#define Ap2  -1.300000
#define Ap3  0.420000
#define Bm1  0.161481
#define Am1  1.000000
#define Am2  -1.245089
#define Am3  0.406570
```

Figure 12 Output .h file content - a

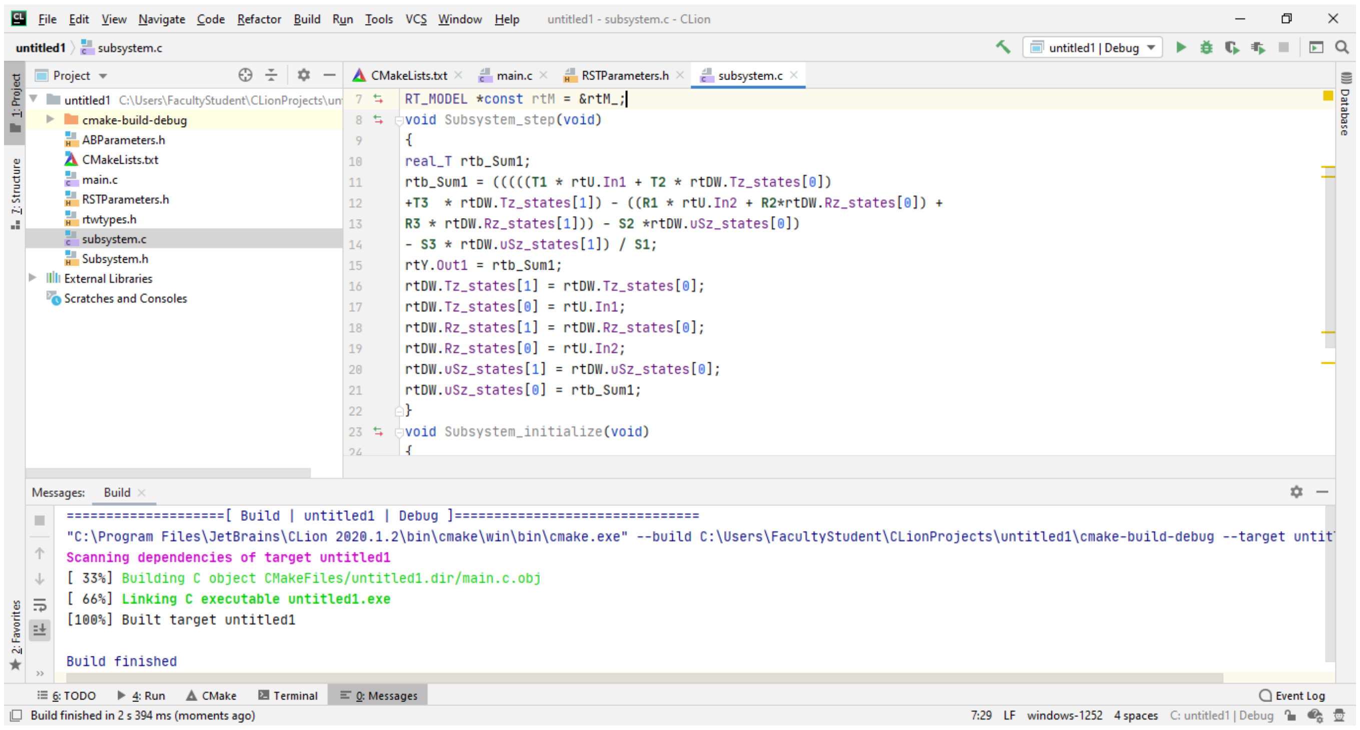


ABParameters.h RSTParameters.h

```
#ifndef RSTPARAMETER_H
#define RSTPARAMETER_H
#define R1  2.999998
#define R2  -3.939001
#define R3  1.314187
#define S1  1.000000
#define S2  -0.374197
#define S3  -0.625803
#define T1  3.333333
#define T2  -4.580657
#define T3  1.622508
#endif
```

Figure 13 Output .h file content - b

4.2. building the generated code in IDE.



```
RT_MODEL *const rtM = &rtM_;
void Subsystem_step(void)
{
real_T rtb_Sum1;
rtb_Sum1 = (((((T1 * rtU.In1 + T2 * rtDW.Tz_states[0])
+T3  * rtDW.Tz_states[1]) - ((R1 * rtU.In2 + R2*rtDW.Rz_states[0]) +
R3 * rtDW.Rz_states[1])) - S2 *rtDW.uSz_states[0])
- S3 * rtDW.uSz_states[1]) / S1;
rtY.Out1 = rtb_Sum1;
rtDW.Tz_states[1] = rtDW.Tz_states[0];
rtDW.Tz_states[0] = rtU.In1;
rtDW.Rz_states[1] = rtDW.Rz_states[0];
rtDW.Rz_states[0] = rtU.In2;
rtDW.uSz_states[1] = rtDW.uSz_states[0];
rtDW.uSz_states[0] = rtb_Sum1;
}
void Subsystem_initialize(void)
{
```

```
===================[ Build | untitled1 | Debug ]==============================
"C:\Program Files\JetBrains\CLion 2020.1.2\bin\cmake\win\bin\cmake.exe" --build C:\Users\FacultyStudent\CLionProjects\untitled1\cmake-build-debug --target untit
Scanning dependencies of target untitled1
[ 33%] Building C object CMakeFiles/untitled1.dir/main.c.obj
[ 66%] Linking C executable untitled1.exe
[100%] Built target untitled1

Build finished
```

```
Build finished in 2 s 394 ms (moments ago)                7:29  LF  windows-1252  4 spaces  C: untitled1 | Debug
```

Figure 14 Screenshot of correct build process

4.3. Checking the system performance in the graphs.

The system shows stability and fast response in handling the disturbance, Figure 15.

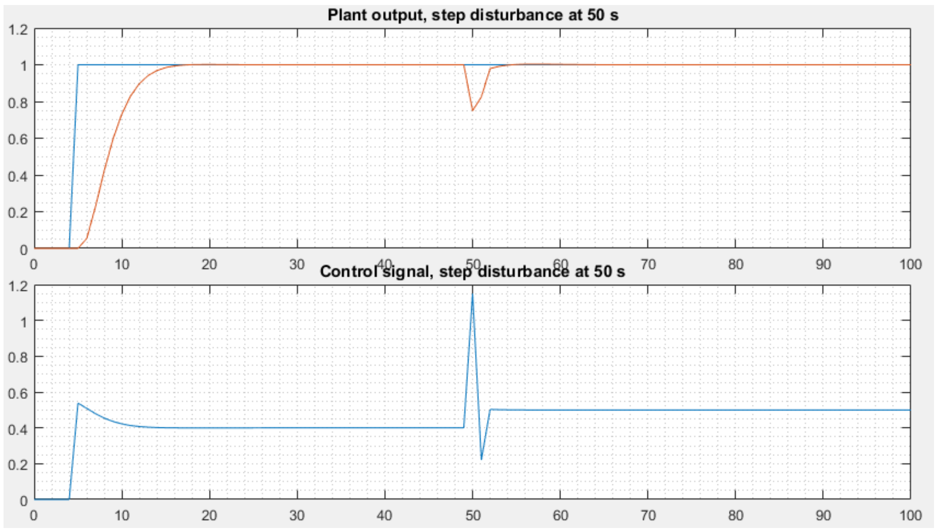


Figure 15 Stability and response of the plant

05

Fifth Topic

CONCLUSION

5. Conclusion

One of the RST method advantages consists in its capability to get a low order solution for most plants & performance criteria, which makes it simple to be implemented. The RST structure for discrete-time controllers provides an elegant design method, both for reference tracking and disturbance rejection. Using real-time operating system implementation for critical systems will boost performance & will make the system meets the time requirements.

KALMAN FILTER can be used to boost the performance of the controller especially in robotics applications, in KALMAN FILTER there are two states (state prediction & measurement update).

The weight of each states determined by the **GAUSSIEN** model; we have included in the project files a simple KALMAN FILTER implementation to estimate the state of multiple cars on a highway using noisy lidar and radar measurements, which can be used with the digital controller to get more accurate results.

Project files

One Drive -> https://bit.ly/2MTYMFh

Google Driver (alternative) -> https://bit.ly/2YB1GEi

Illustration video

Main Video

Validation video

Validation Video

06

Sixth Topic

REFERENCES

6. References

1- Prof. Dr. Sherif Hammad, Control Systems (2), CSE372, "Lecture Notes", Spring 2020.
2- Landau, Ioan Doré, and Gianluca Zito. Digital control systems: design, identification and implementation. Springer Science & Business Media, 2007.
3- MATLAB. (2018). 9.7.0.1190202 (R2019b). Natick, Massachusetts: The MathWorks Inc.
4- Keil uVision IDE software program.
5- CLion software program. Version: 2020.1.2, Build: 201.7846.88.