Control System

Project Report
Course Number: SOEN 385

By
Renaud Brunet, 26626742
Justin Trong Huan Do, 26600514
Patrick Nguyen, 26349285
Date Submitted:
April 15, 2016

Table of contents

Statement of need

Since Galileo first started gazing at the stars atop a mountain in Italy, the general design of a telescope has pretty much remained the same. The need for higher-resolution imaging to resolve far away objects requires telescopes that are far superior to a human pair of eyes has always been present to improve as a society by understanding our Universe.

Project Scope

The purpose of this project is to design a PID controller system for controllable telescope system that will be able to take digital pictures at different kind of positions in the sky. The controllable telescope will be able to find each star in the sky based on a list of coordinates of n=100 positions. The following model will be equipped with a digital camera that can take 2 pictures for each star at an average of 5 seconds(including the focus and saving pictures to a hard disk time). It consists of two identical DC motors, each supporting a maximum of 2% overshoot for each motor, as well as settling time less than 1 second. We start by looking at the system with our given transfer function $G(S) = \frac{5}{S(S-1)}$ and a closed-loop feedback $H(S) = \frac{1}{1+0.01S}$. From these functions, we analyze the stability of the system by adding P, PI, PD and PID controllers. The same steps are then repeated with a closed-loop feedback of H(S)=1.

Our project must also support a Graphical User Interface that allows us to input a position for our telescope to point to. Therefore, we will use MATLAB for the GUI, Simulink to model our control system, and Virtual Reality Toolbox to simulate our telescope.

Implementation

1. TelescopeModel.slx
The Simulink Model containing the whole closed loop system
2. TelescopePID.m
The MatLab code that will handle the GUI and the Simulink Model for inputs
3. Telescopegui.m
The MatLab code that allows the manual tuning
4. Teslescopegui.fig
The GUI that allows the simulation to start and terminate with PID configuration
5. TeslescopeVirtualWorld.wrl
The 3D model representation of the telescope

Implementation Requirements

The sky will be represented as a semi-sphere above:

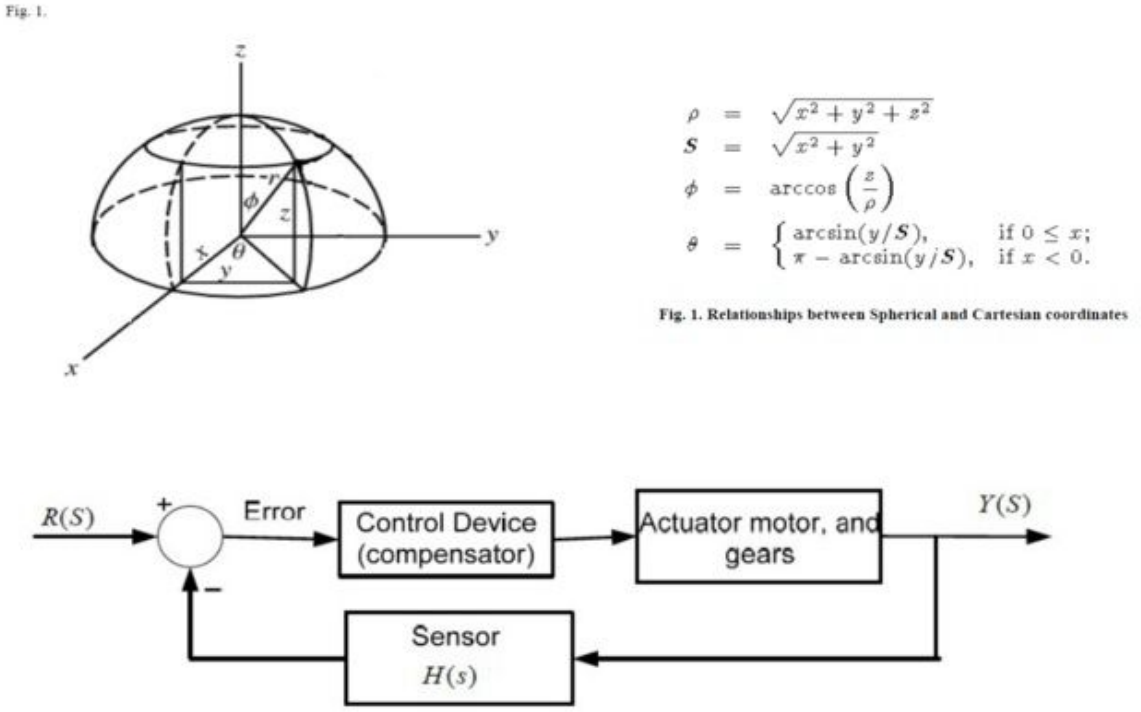


Fig. 2. Block diagram of each motor, its sensor and controller

The requirements, conditions, and assumptions of this system are as follows:

- The telescope is initially pointing to an initial position of (r_0, θ_0, ϕ_0)
- The position of a specific star *n* will be (r_n, θ_n, ϕ_n)
- The direction of the telescope will change from $\theta_0 \rightarrow \theta_1$ and from $\phi_0 \rightarrow \phi_1$, to reach the first star. It will repeat this process to reach all the other stars
- There will be two DC motors for both θ-direction and for ϕ-direction. The control system must send commands to each DC motors separately.
- θ and ϕ will both take in radians as a measurement. E.g: $\phi = -\frac{\pi}{2}$ means that the DC motor for ϕ will move 90 degrees counter clockwise
- Assume the motor transfer function is $G(S) = \frac{5}{S(S-1)}$
- Assume the sensor transfer function is $H(S) = \frac{1}{1+0.01S}$
- Both the motor and the controller of the system can have a maximum overshoot of 2%
- Everytime θ and ϕ performs a rotation, it must move by at least 5 degrees $(\frac{\pi}{36})$
- The time it takes for each rotation, including settling time, must be less than 1 second.

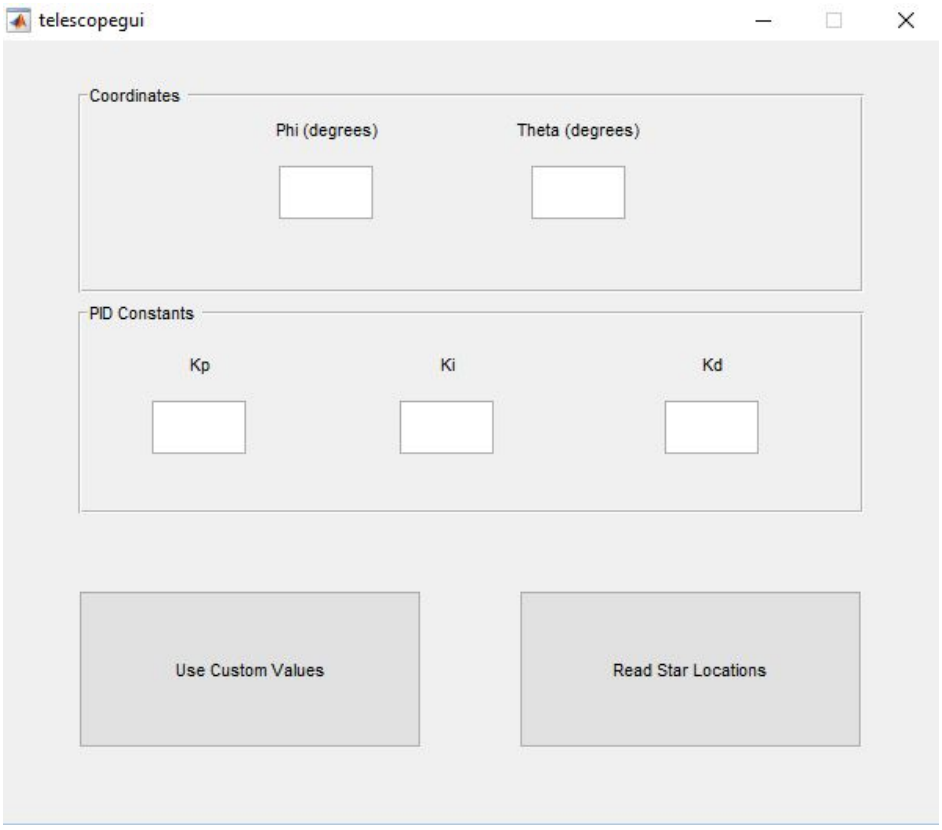
Graphical User Interface

With the help of the Graphical User Interface Design Environment (GUIDE) that is provided with MatLab, a simple GUI interface was created for the purposes of our project.

Cartesian Coordinate Panel: The user is able to input their cartesian coordinates for a star that they would like to observe. A script is run to convert the cartesian coordinates into spherical coordinates and those values (namely ϕ and θ) are sent to Simulink, which will in turn send commands to our Virtual Reality Model.

PID Constants: The values for Kp, Ki, and Kd can also be manually inputted in order to see the various effect that each value does to the system. This has helped us manual tune to find the proper value.

Read Star Locations: When pressed, the program will open the text file containing all the star locations, reading and then visiting each start, turn by turn. It will also calculate the total time to visit all 100 stars.



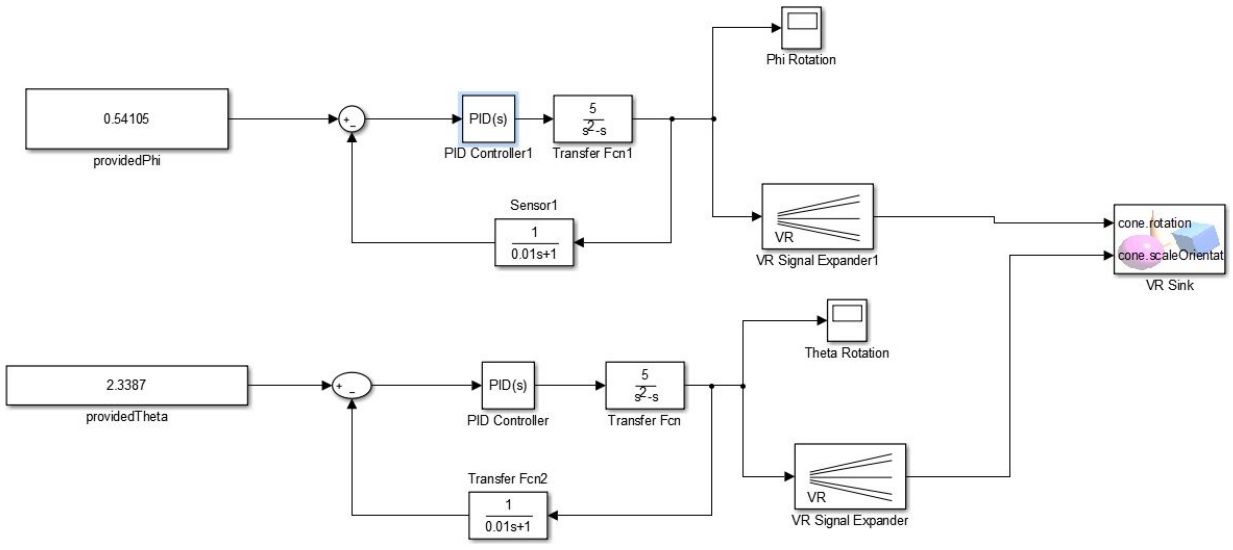
Simulink Model

We configured our design to have two systems composed of DC motors that controls the angles of rotation around a certain axis. Two inputs are given, in Phi and Theta just like shown in the Figure below.

The Simulink Model takes both phi and theta angles as input, feeding it to the PID controller and the motor(transfer function) in order to generate a stable system which will be runned in our virtual model that will rotate correctly based on the given star's position input.

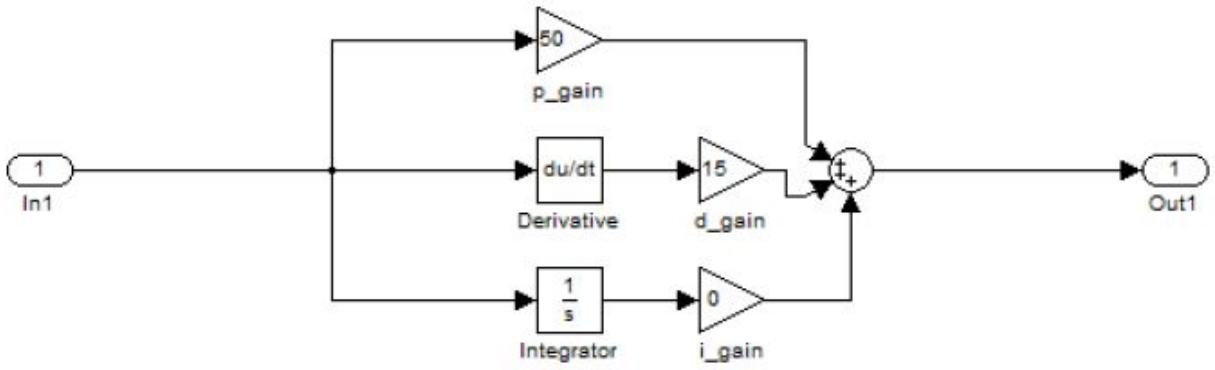
In order to stabilize the system and satisfy both the overshoot and settling time requirements, different PID controllers(P,PI,PD,PID) will be tested through trial and error in order to define which is the best one. In this section, we have

- The motor transfer function is $G(S) = \frac{5}{S(S-1)}$
- The sensor transfer function is $H(S) = \frac{1}{1+0.01S}$



The sensor is the feedback system that evaluates the output, and resends it through the PID+Transfer function in order to readjust the output.

Different values of Kp, Ki and Kd will be found in order to obtain a fast rise time, minimum overshoot and no steady-state error system.



CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	No Change

To adjust these parameters to get a good step response, we started by using the manual tuning for the PID.Our technique was composed of such.

1. Set all gains to zero.
2. Increase the P gain until the response to a disturbance is steady oscillation.
3. Increase the D gain until the the oscillations go away (i.e. it's critically damped).
4. Repeat steps 2 and 3 until increasing the D gain does not stop the oscillations.
5. Set P and D to the last stable values.
6. Increase the I gain until it brings you to the setpoint with the number of oscillations desired (normally zero but a quicker response can be had if you don't mind a couple oscillations of overshoot)

Due to the difficulty of the trial and error technique, we decided to used the PID Automatic Tuner provided by MatLab. It allows to achieve a good balance between both performance and robustness by having a two-degree of freedom PID controller on the time domain(Response Time and Transient Behavior). Since it gave us a GUI that was easy to use in order to modify and tune, we chose that technique.

PID Design for H(S) = 1/(1+0.01S)

P-Controller

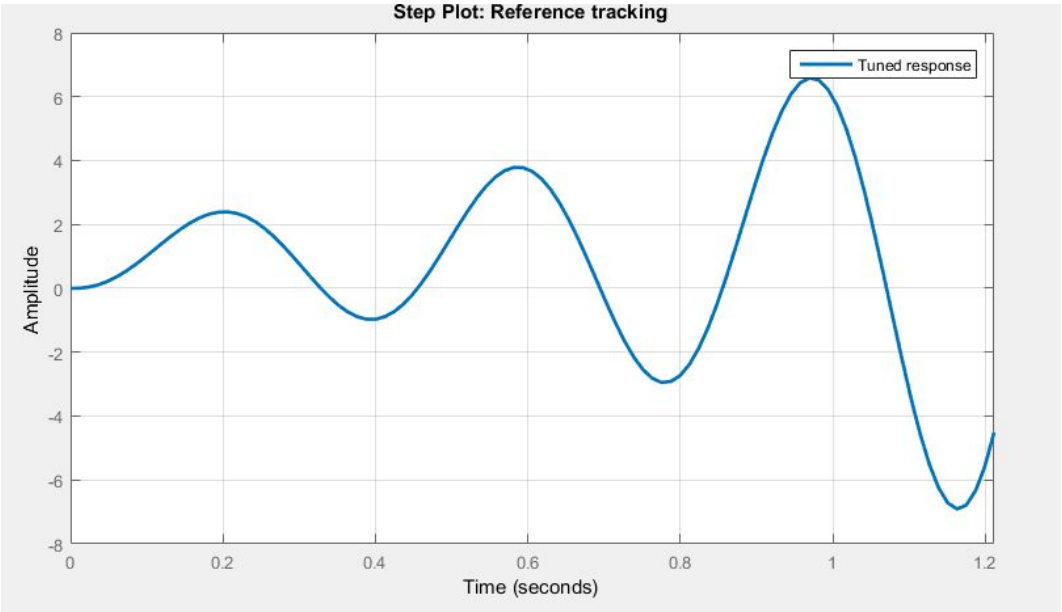


Figure 1 : Step Plot of P-Controller

Controller Parameters

	Tuned
P	55.3646
I	n/a
D	n/a
N	n/a

Performance and Robustness

	Tuned
Rise time	NaN seconds
Settling time	NaN seconds
Overshoot	NaN %
Peak	Inf
Gain margin	Inf dB @ NaN rad/s
Phase margin	-12.8 deg @ 16.5 rad/s
Closed-loop stability	Unstable

Figure 2 : Controller parameters of P-Controller

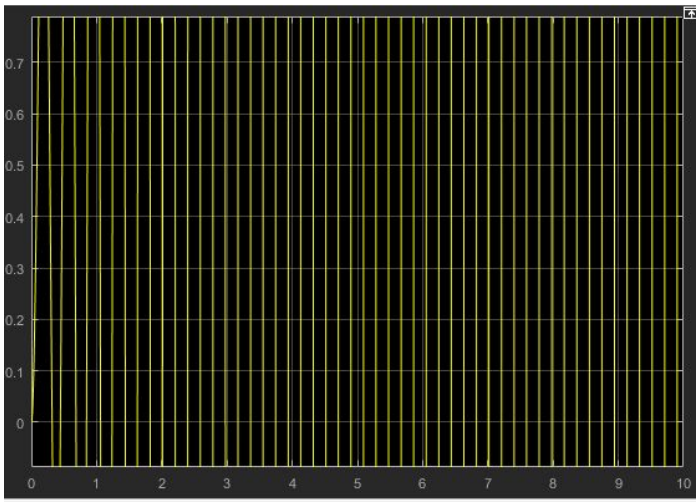


Figure 3 : Output of System with P-Controller

We use the automatic tuning technique to find the value for the P-controller and we found out that the P-controller is not enough to stabilize the system. As we can see in the graph, the system will never settle if we only use the P-controller because the value of P doesn't change the settle time so we are going to try with PI-controller.

PI-Controller

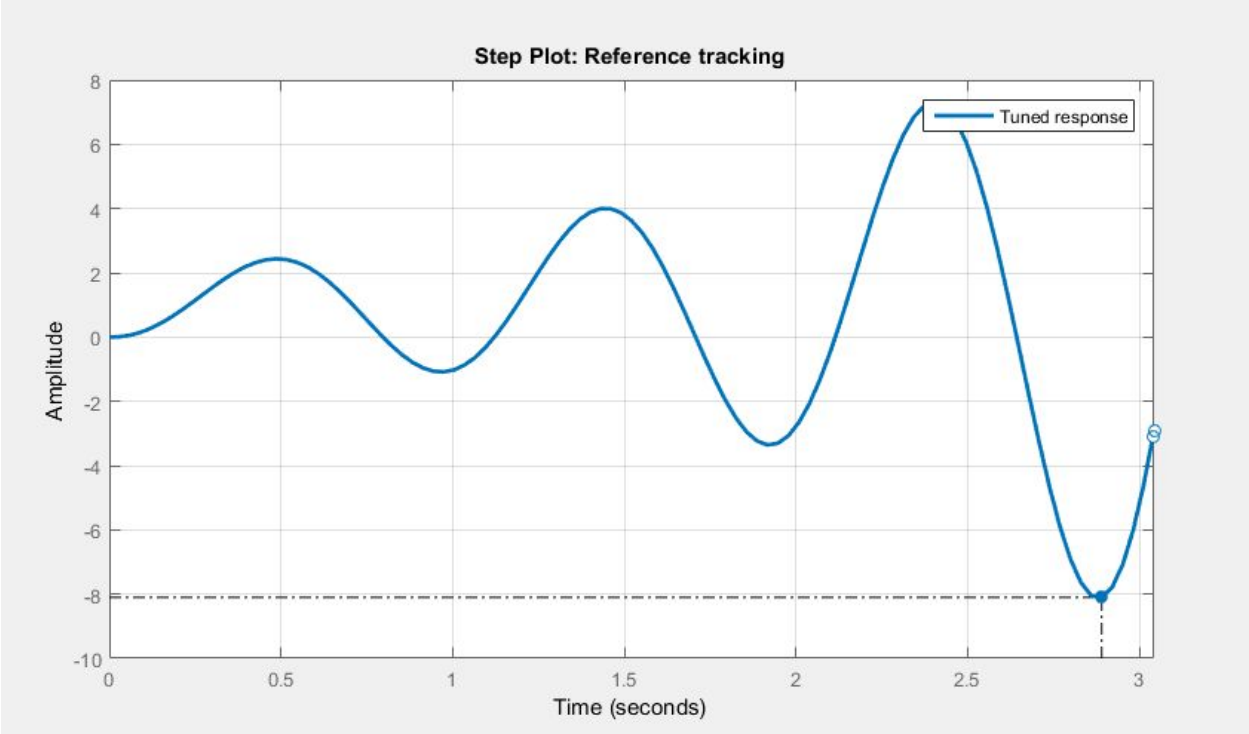


Figure 4 : Step Plot of PI-Controller

Controller Parameters

	Tuned
P	8.7586
I	1.0076
D	n/a
N	n/a

Performance and Robustness

	Tuned
Rise time	NaN seconds
Settling time	NaN seconds
Overshoot	NaN %
Peak	Inf
Gain margin	Inf dB @ NaN rad/s
Phase margin	-13.4 deg @ 6.57 rad/s
Closed-loop stability	Unstable

Figure 5 : Controller parameters of PI-Controller

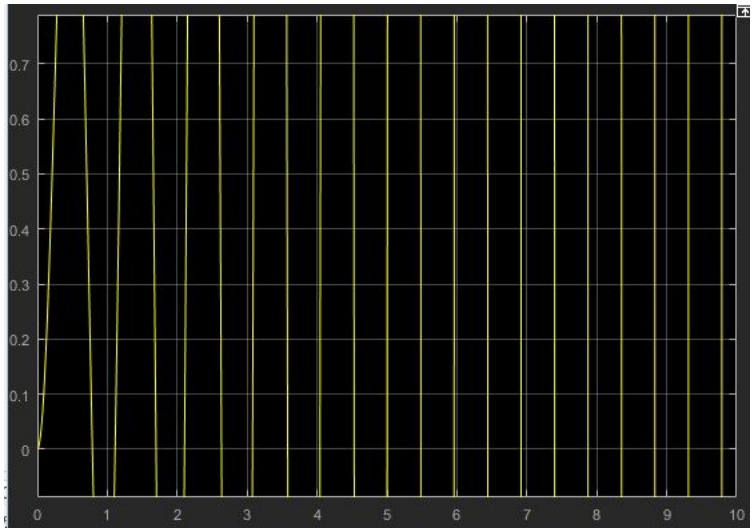


Figure 6 : Output of System with PI-Controller

We use the automatic tuning technique to find the value for the PI-controller and we found out that the PI-controller is not enough to stabilize the system. As we can see in the graph, the system will never settle if we only use the PI-controller because the P variable doesn't change the settle time and the I variable only increase the settle time so it doesn't help to stabilize the system so we are going to try with PD-controller.

PD-Controller

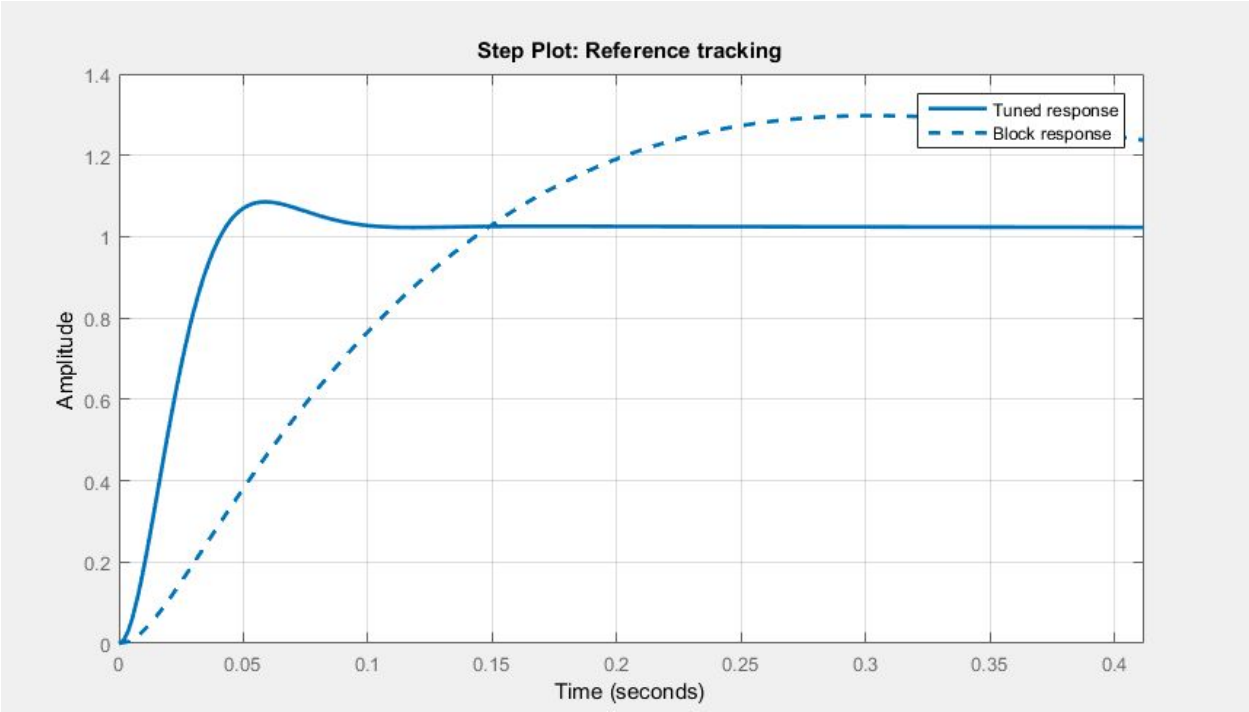


Figure 7 : Step Plot of PD-Controller

Controller Parameters

	Tuned	Block
P	4.5856	8.7586
I	n/a	n/a
D	10.7943	1.9762
N	5547.9785	1118.4618

Performance and Robustness

	Tuned	Block
Rise time	0.027 seconds	0.104 seconds
Settling time	0.725 seconds	0.688 seconds
Overshoot	8.56 %	29.8 %
Peak	1.09	1.3
Gain margin	-34.6 dB @ 0.657 rad/s	-19.5 dB @ 2.17 rad/s
Phase margin	61.9 deg @ 48.5 rad/s	55.5 deg @ 10.6 rad/s
Closed-loop stability	Stable	Stable

Figure 8 : Controller parameters of PD-Controller

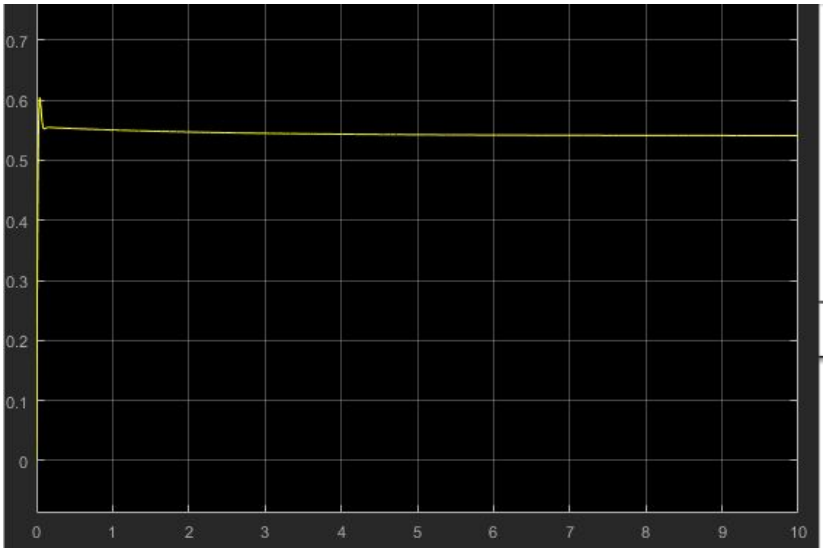


Figure 9 : Output of System with PD-Controller

We use the automatic tuning technique to find the value for the PD-controller and we found out that with the PD-controller it is possible to stabilize the system. As we can see in the graph, the system can settle using the PD-controller. Even if the P variable doesn't change the settle time and the D variable decrease the settle time so it doesn't help to stabilize the system and it also reduce the overshoot but the overshoot is still too high so we are going to try with PID-controller.

PID-Controller

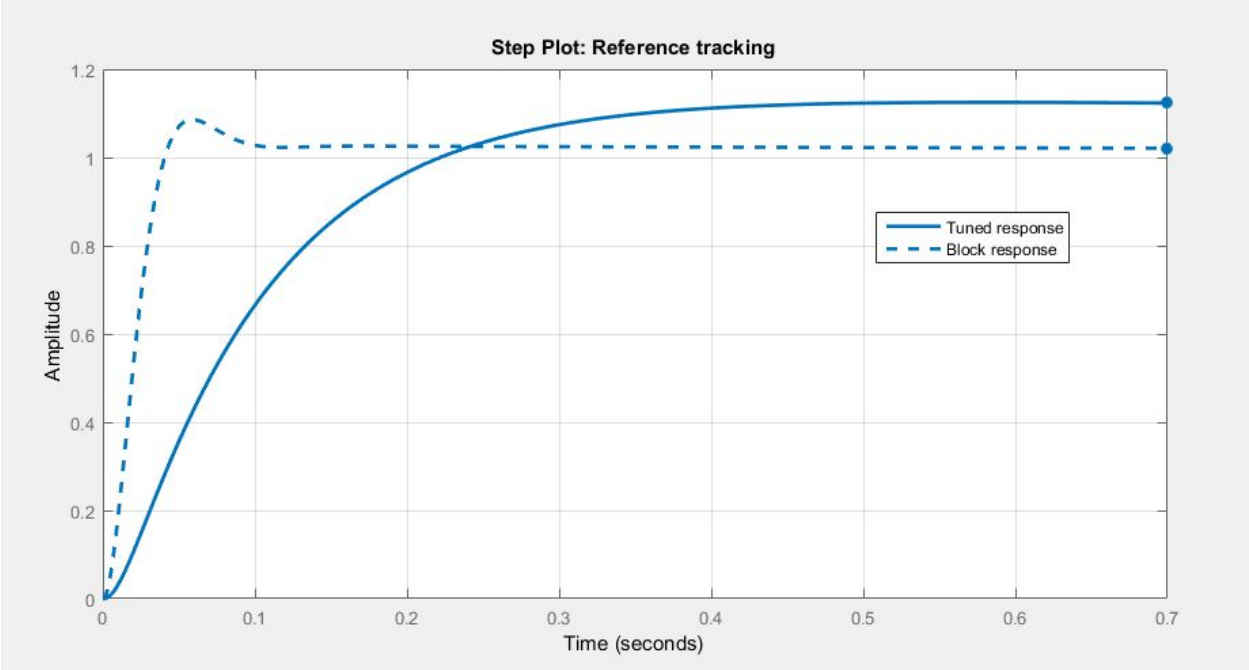


Figure 10 : Step Plot of PID-Controller

Controller Parameters

	Tuned	Block
P	0.52687	4.5856
I	0.030641	1.0076
D	2.0129	10.7943
N	1139.2555	5547.9785

Performance and Robustness

	Tuned	Block
Rise time	0.203 seconds	0.027 seconds
Settling time	0.333 seconds	0.79 seconds
Overshoot	1 %	8.57 %
Peak	1.12	1.09
Gain margin	-19.5 dB @ 0.53 rad/s	-32.9 dB @ 0.724 rad/s
Phase margin	76.6 deg @ 9.97 rad/s	61.9 deg @ 48.5 rad/s
Closed-loop stability	Stable	Stable

Figure 11 : Controller parameters of PID-Controller

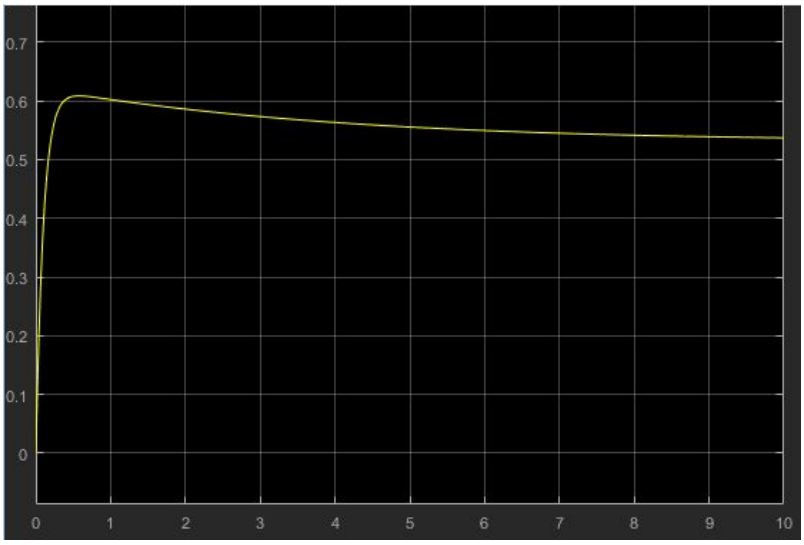


Figure 12 : Output of System with PID-Controller

We use the automatic tuning technique to find the value for the PID-controller and we found out that with the PID-controller it is possible to stabilize the system. As we can see in the graph, the system can settle using the PID-controller. Even if the P variable doesn't change the settle time and the D variable decrease the settle time so it doesn't help to stabilize the system and it also reduce the overshoot. The addition of the I variable allow us to reduce the overshoot to the number that we want.

```
clear all
clc

syms s

% G=5/((s)(s-1))

num=5;
den=sym2poly((s)*(s-1));

G=tf(num,den);

num1=1;
den1=sym2poly(1+0.01*s);

H=tf(num1,den1);

%%
Kp=0.52687;
Ki=0.030641;
Kd=2.0129;


C=pid(Kp,Ki,Kd);

T=feedback(C*G,H);

step(T);
```

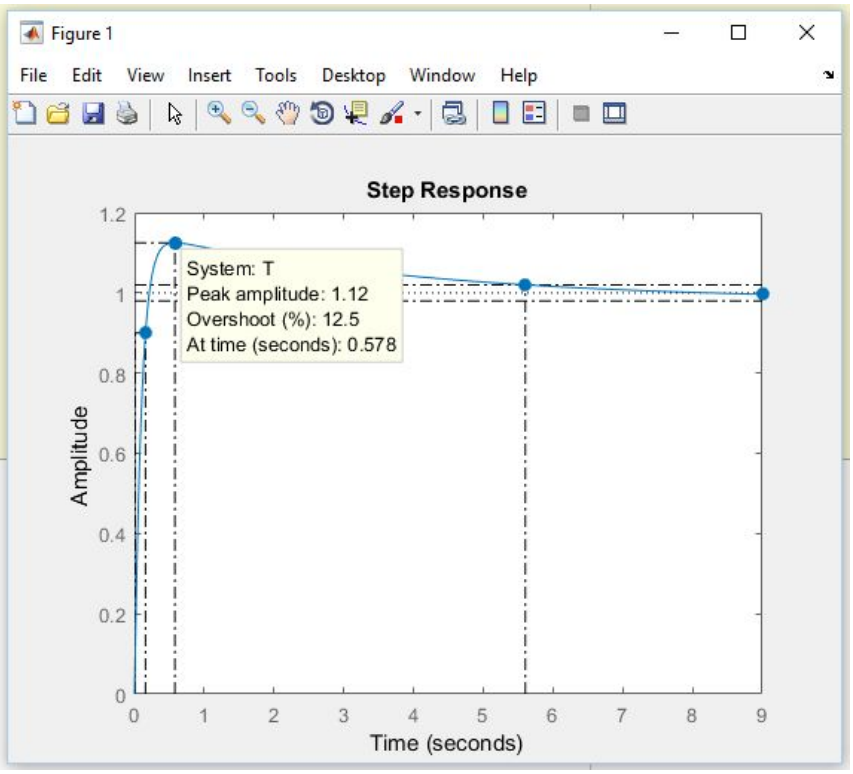


Figure 13: Manual Tuning for PID controller

Starting from the values found with the automatic tuning tool, we tried to use them with the manual tuning method. For some reason, it didn't give the same values even if we used the same Kp, Ki and Kd.

Controller	Overshoot(%)	Rise time(s)	Settling time(s)	Peak	Stability
P	-	-	-	Infinity	Unstable
PI	-	-	-	Infinity	Unstable
PD	8.56	0.027	0.725	1.09	Stable
PID	1	0.203	0.333	1.12	Stable

From this table, we can clearly see that both P and PI controllers are not a good choice for the design of our system due to the fact that no matter what values are input, it is unstable. As for the PD, it wasn't able to achieve the maximum overshoot of 2%, whereas the PID was able to successfully obtain 1% overshoot while having a rise and settling time under 1 second.

PID Design for $H(S) = 1$

P-Controller

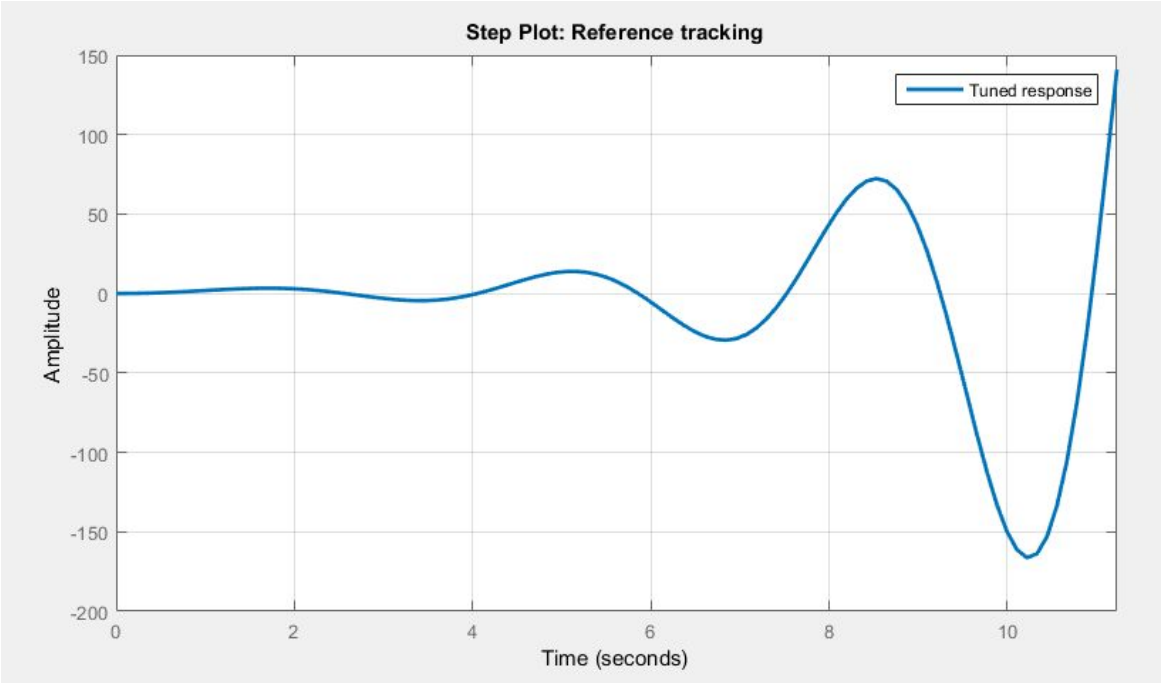


Figure 14 : Step Plot of P-Controller

Controller Parameters

	Tuned
P	1.6906
I	n/a
D	n/a
N	n/a

Performance and Robustness

	Tuned
Rise time	NaN seconds
Settling time	NaN seconds
Overshoot	NaN %
Peak	Inf
Gain margin	Inf dB @ Inf rad/s
Phase margin	-19.5 deg @ 2.82 rad/s
Closed-loop stability	Unstable

Figure 15 : Controller parameters of P-Controller

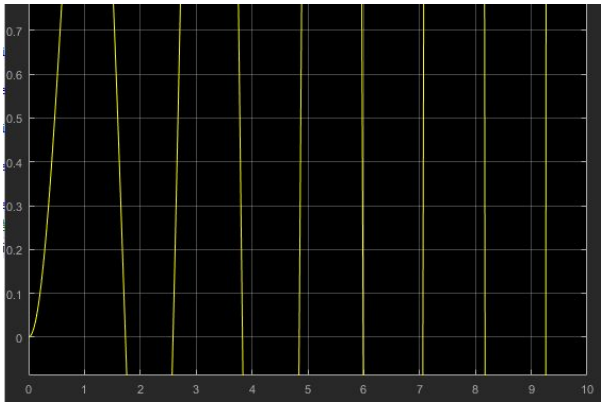


Figure 16 : Output of System with P-Controller

PI-Controller

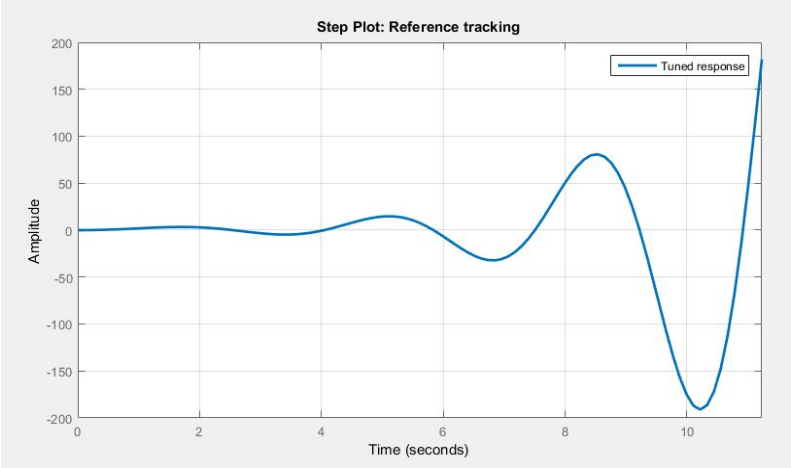


Figure 17 : Step Plot of PI-Controller

Controller Parameters

	Tuned
P	0.72745
I	0.022675
D	n/a
N	n/a

Performance and Robustness

	Tuned
Rise time	NaN seconds
Settling time	NaN seconds
Overshoot	NaN %
Peak	Inf
Gain margin	Inf dB @ Inf rad/s
Phase margin	-30.3 deg @ 1.78 rad/s
Closed-loop stability	Unstable

Figure 18 : Controller parameters of PI-Controller

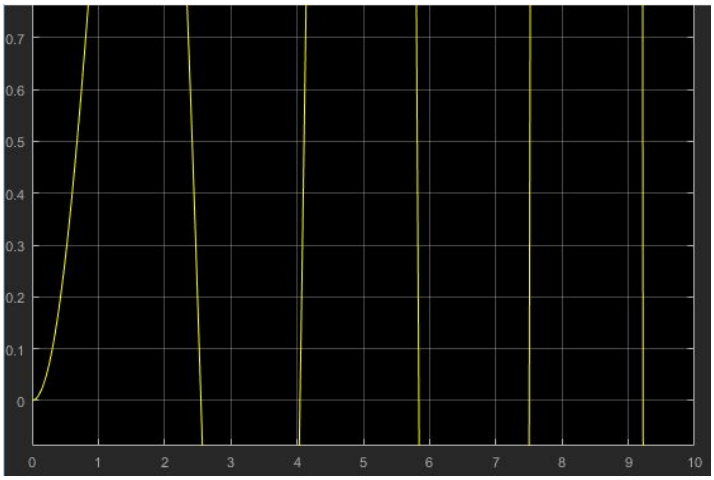


Figure 19 : Output of System with PI-Controller

PD-Controller

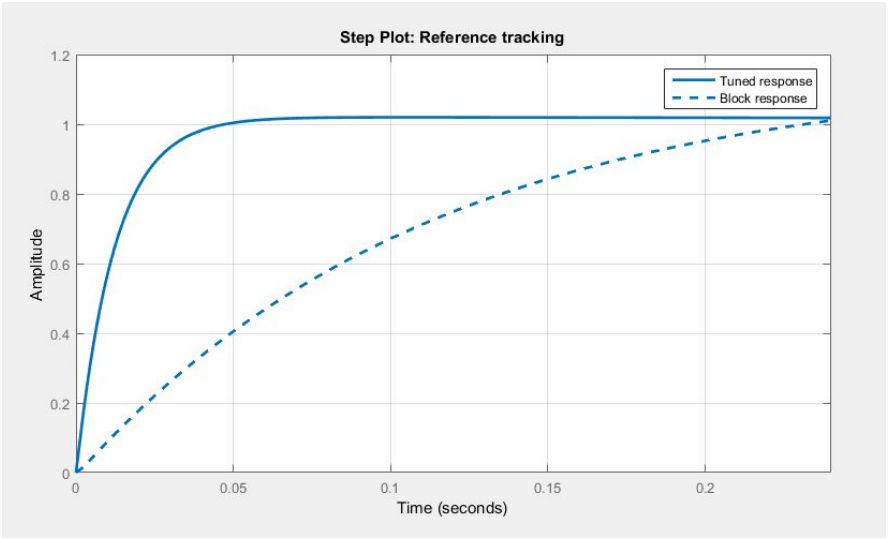


Figure 20 : Step Plot of PD-Controller

Controller Parameters

	Tuned	Block
P	12.1485	0.72745
I	n/a	n/a
D	16.6629	2.0129
N	9521.4397	1139.2555

Performance and Robustness

	Tuned	Block
Rise time	0.0247 seconds	0.162 seconds
Settling time	0.0392 seconds	5.12 seconds
Overshoot	1.98 %	12.8 %
Peak	1.02	1.13
Gain margin	-38.4 dB @ 0.854 rad/s	-20.1 dB @ 0.601 rad/s
Phase margin	88.3 deg @ 83.3 rad/s	81.7 deg @ 10 rad/s
Closed-loop stability	Stable	Stable

Figure 21 : Controller parameters of PD-Controller

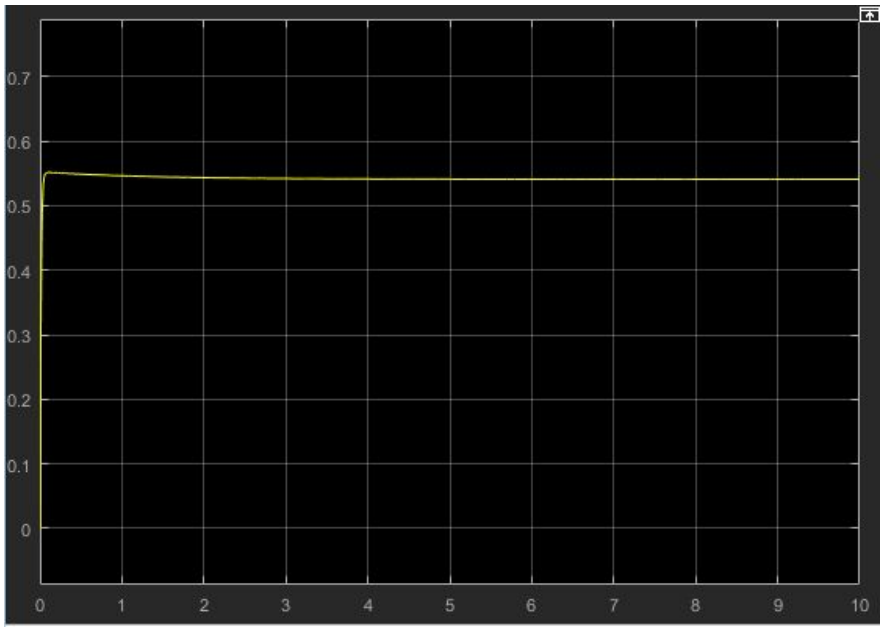


Figure 22 : Output of System with PD-Controller

PID-Controller

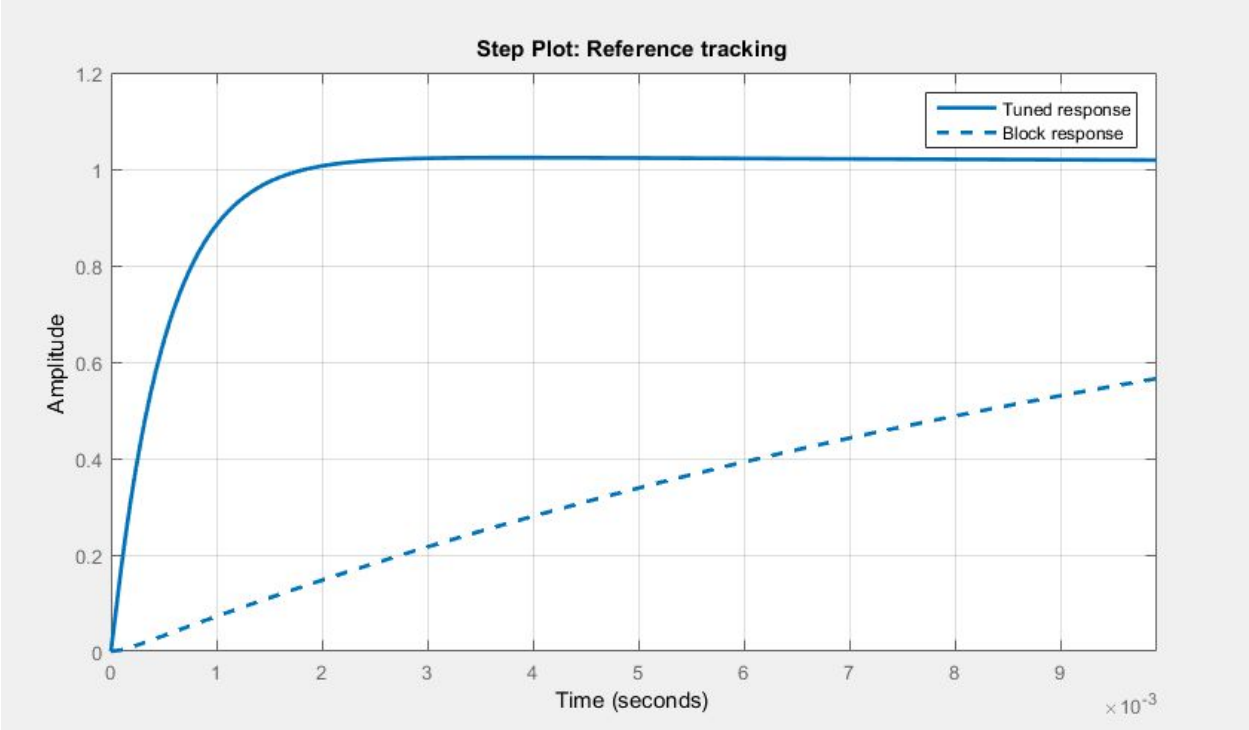


Figure 23 : Step Plot of PID-Controller

Controller Parameters

	Tuned	Block
P	21357.3518	12.1485
I	251321.8759	0.022675
D	403.2603	16.6629
N	230516.8216	9521.4397

Performance and Robustness

	Tuned	Block
Rise time	0.001 seconds	0.0247 seconds
Settling time	0.00839 seconds	0.0392 seconds
Overshoot	2.38 %	1.98 %
Peak	1.02	1.02
Gain margin	-44 dB @ 26 rad/s	-38.4 dB @ 0.855 rad/s
Phase margin	88 deg @ 2.02e+03 ra...	88.3 deg @ 83.3 rad/s
Closed-loop stability	Stable	Stable

Figure 24 : Controller parameters of PID-Controller

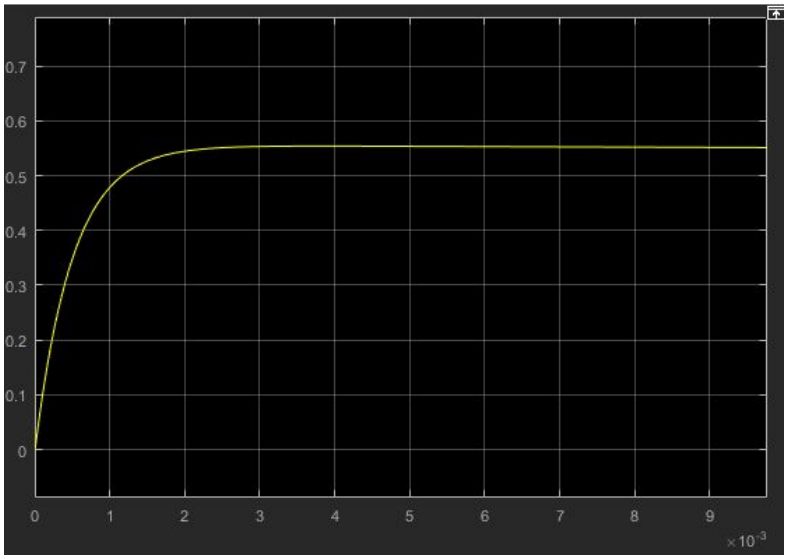
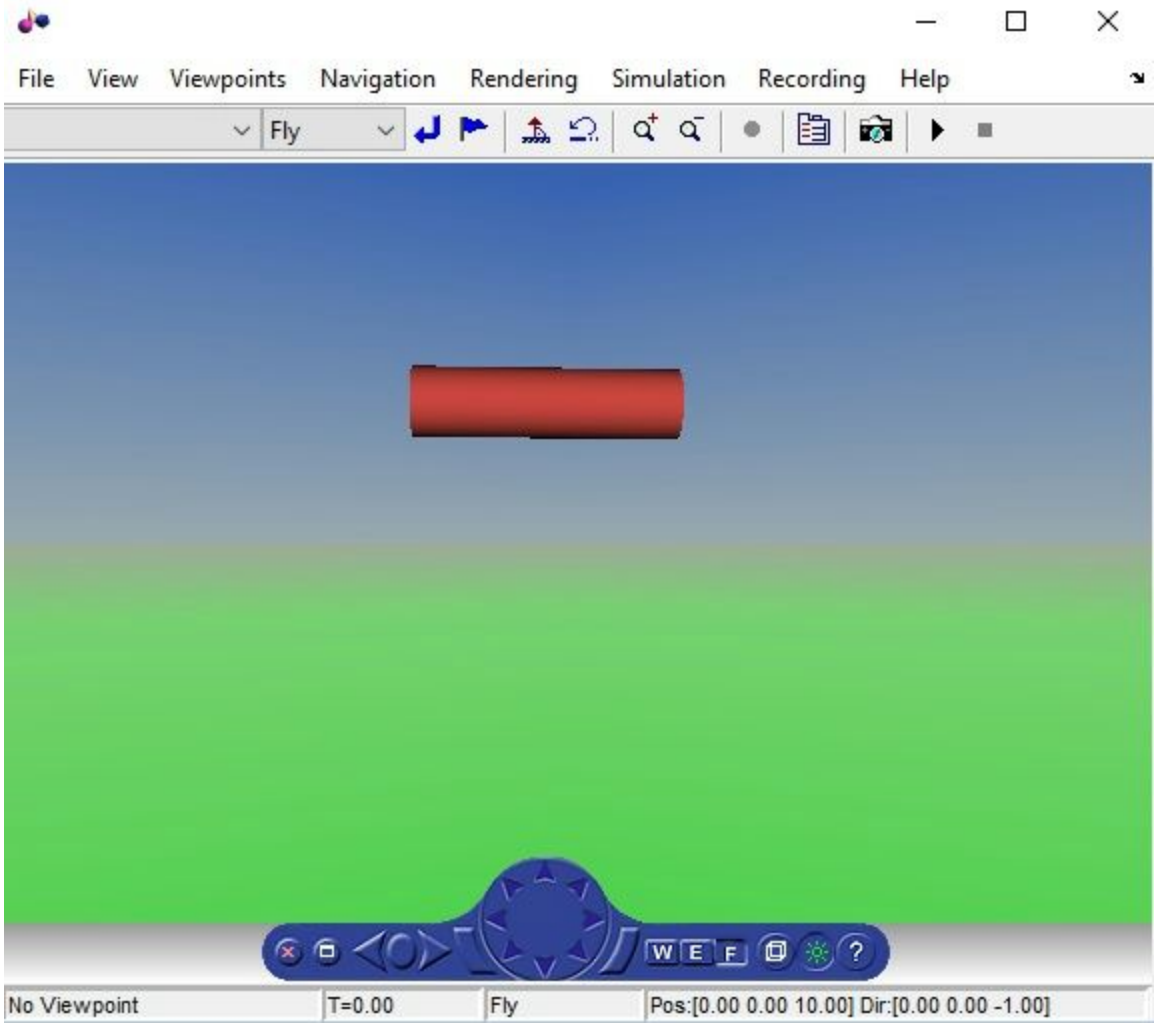


Figure 25 : Output of System with PID-Controller

Controller	Overshoot(%)	Rise time(s)	Settling time(s)	Peak	Stability
P	-	-	-	Infinity	Unstable
PI	-	-	-	Infinity	Unstable
PD	1.98	0.0247	0.0392	1.02	Stable
PID	2.38	0.001	0.00893	1.02	Stable

From this table, we can clearly see that both P and PI controllers are not a good choice for the design of our system due to the fact that no matter what values are input, it is unstable. As for the PD, it wasn't able to achieve the maximum overshoot of 2%, whereas the PID was able to obtain 2.38% overshoot, which is a little bit higher, while having a rise and settling time under 1 second.

Virtual Reality Toolbox



We created a model in V-Realm in order to simulate our telescope. Our VR model receives angles of rotation from the Simulink model. The telescope should be made to receive input and rotate itself to point towards the desired position. However, due to time constraint and limited team member skill set, we weren't able to get the animation working as desired. The telescope does not rotate around the proper axes required, only around one axe at the time.Furthermore, for some reason, we weren't able to make it rotate from the last known position, so it always had to reset to the initial position. However, we were able to make the system take input every 5 seconds, letting the time for the telescope to capture images for each single star. Thus this is the only part of the project that has not been fully completed.

Conclusion

At first, we tried to use the trial and error technique time, but we had difficulty finding the value for P, I and D values so the PID Tuner was used to find the appropriate PID values for our system. Using the PID Tuner, we were not able to find a stable output that met the specifications when just using P, PI, and PD. It shows that it was necessary to use the P, I and D in order to properly stabilize the system and meet the requirements of less than 2% overshoot. We were able to get an overshoot of 1% for the H(S) = 1/(1+0.01S) design. We learned that it was difficult to design a proper controller for a simple system We gained a new appreciation for controllers that can stabilize and meet requirements for complex systems.