This coursework aims at controlling the distance $d(t)$ of a robot R1 relative to a robot R2 moving at a speed $s(t)$ so that it follows a reference $d_r(t)$. All quantities are in SI units (metres, seconds, m/s, etc). Assume the following:

- the distance $d(t)$ is measured and can be used for control feedback.
- the initial conditions are: $d(0) = 1, \dot{d}(0) = 0, s(0) = 0$

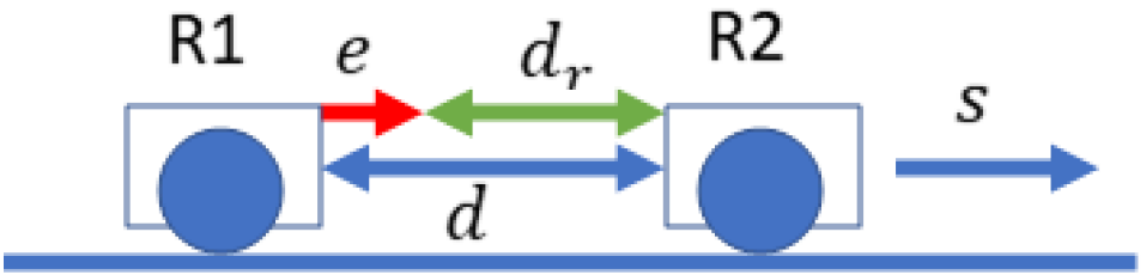

Figure 1: Robot diagram.

Question 1

[70 marks]
Assume the velocity of R1 is directly and instantly controlled by an input $u(t)$.

a

[8 marks]
Model the system as a continuous state-space representation, considering as inputs the control of R1 and the speed of R2 (respectively $u(t)$, $s(t)$, and as output the distance $d(t)$.

State: distance $d(t) \rightarrow x = d$
Input: Speed of R1 $\rightarrow f(u(t), s(t))$
Output: distance between R1 & R2 $y = x = d$
Dynamics: $\dot{x} = \dot{d} = s(t) - u(t)$

$$\delta d = d - d(0)$$

Laplace transform

$$sX(s) = sD(s) = S(s) - U(s)$$

Transfer Function

$$\frac{D(s)}{S(s) - U(s)} = \frac{1}{s}$$

At equilibrium:

$$\frac{dx}{dt} = \dot{x} = g(x, f(u, s)) = 0$$

b

[20 marks][MATLAB]
Implement a Simulink model that simulates the described system. Consider that:

- *The speed of R2 (which is unknown to R1) follows:*

$$s(t) = \begin{cases} 0.1t, & \text{if } 0 \leq t < 4 \\ 0.4, & \text{if } 4 \leq t < 8 \\ 0.8 - 0.05t, & \text{if } 8 \leq t < 12 \\ 0.2, & \text{if } 12 \leq \text{inf} \end{cases}$$

- *The reference distance is constant* $d_r = 1$
- *The input of R1* $u(t)$ *is generated by a continuous PID controller. Tune this controller so that* $d(t)$ *follows the reference* d_r *, justify your choice of parameters, making sure the system is stable.*

Generate plots of R1's velocity and the distance $d(t)$ *over 16 seconds*

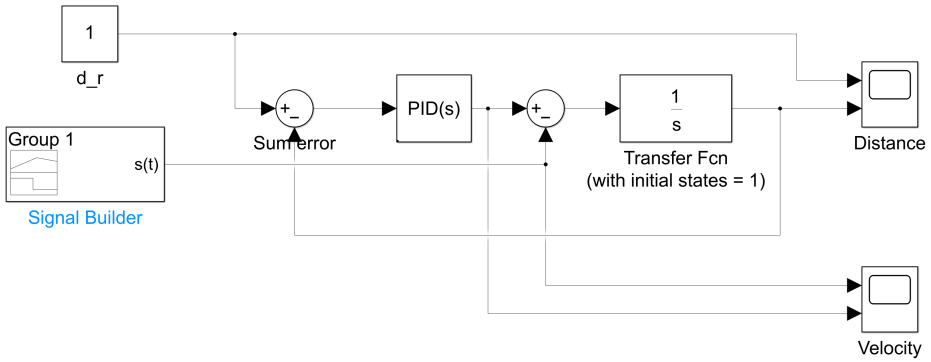

Figure 2: Simulink Model

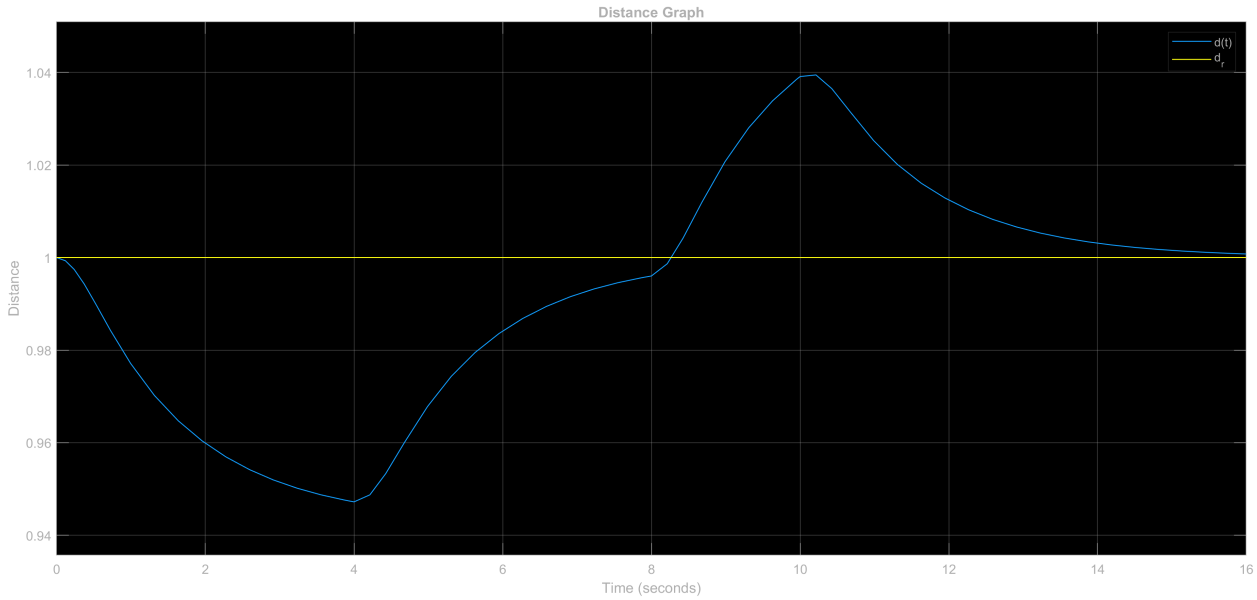

Figure 3: Distance Graph 1b

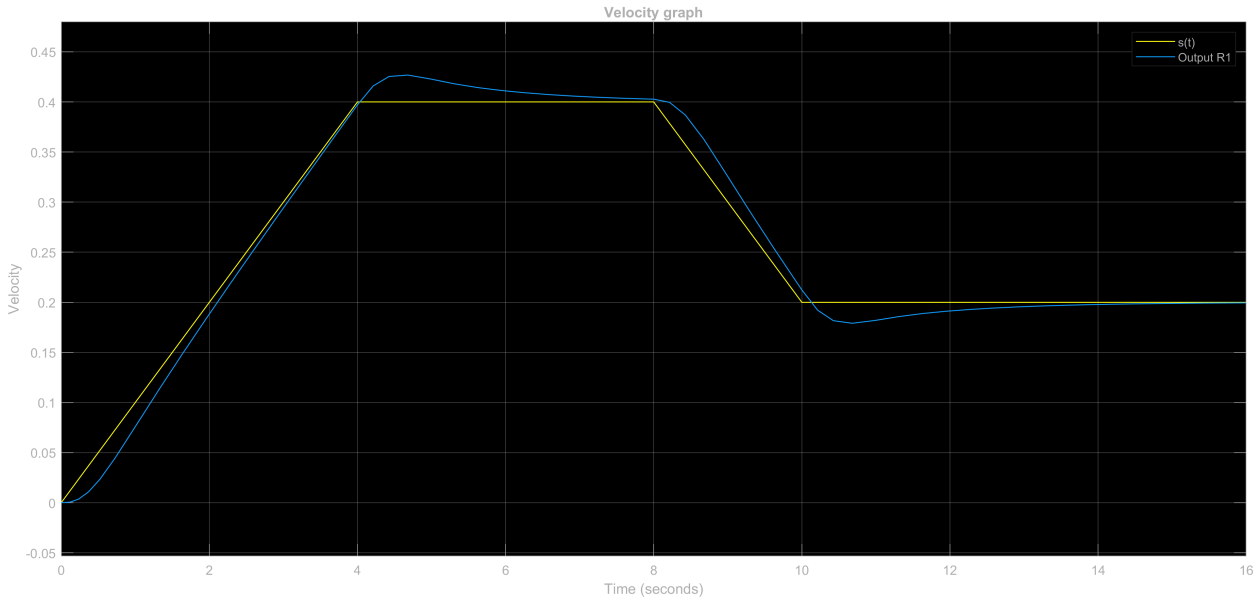

Figure 4: Velocity Graph 1b

The PID constants were tuned with the Simulink built in tuner and chosen to have a fast response to match the signal, as shown in figure below, PID paramters as follows:

- $P = 1.549$
- $I = 0.483$
- $D = -0.36$

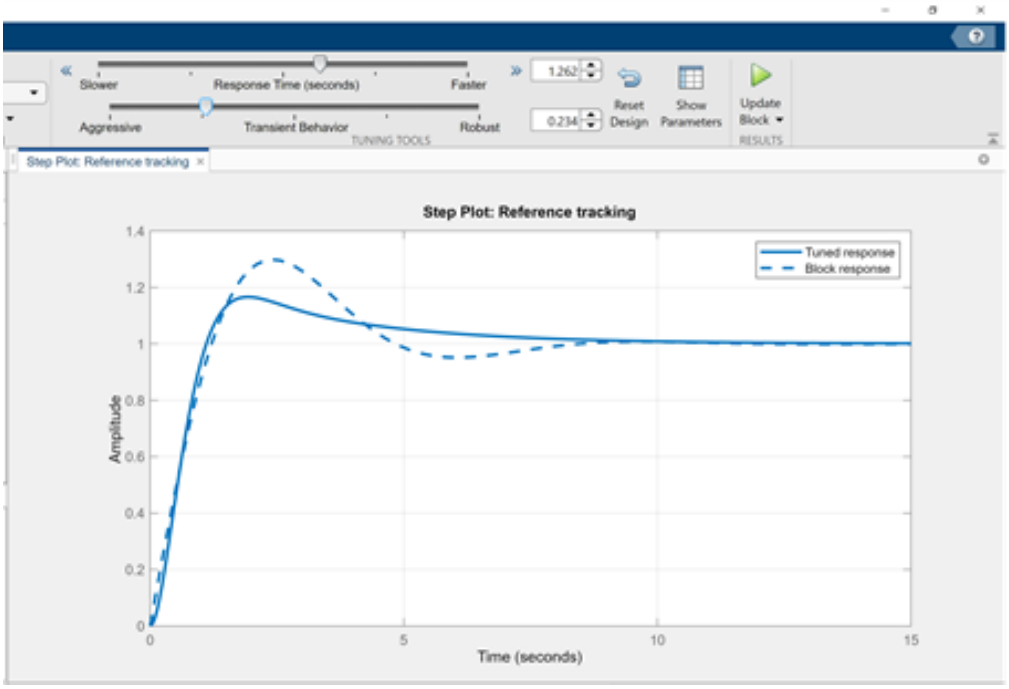

Figure 5: PID Parameter tuning

c

[8 marks]
Suppose we want to implement a discrete, digital version of the PID controller from the previous question, with a sampling time of 0.2 seconds. Derive its expression in the form of a discrete digital filter.

Transfer function for the controller is as follows:

$$tf_{pid} = K_p + \frac{K_i}{s} + K_d s$$

with PID constants as follows:

- $P = 1$

- $I = 1$
- $D = 0$ rounded for simplicity

Gives:

$$tf_{pid} = \frac{11z - 9}{10z - 10}$$

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

$$\mathbf{y} = C\mathbf{x} + D\mathbf{u}$$

The solution of this linear differential equation is thus:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)\,d\tau$$

If u is constant then:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)\,d\tau$$

A is invertible then:

$$x(t) = e^{At}x(0) + A^{-1}(e^{At} - I)Bu$$

Digital system with dampling rate $\Delta t = 0.2$ seconds:

$$x(t + \Delta t) = e^{A\delta t}x(t) + A^{-1}(e^{A\Delta t} - I)Bu(t)$$

Discretise:

$$x[k + 1] = e^{A\Delta t}x[k] + A^{-1}(e^{A\Delta t} - I)Bu[k]$$

Using the Tustin method for continous to discrete transformations:

$$s = \frac{2}{T_s}\frac{z - 1}{z + 1}$$

$$\frac{1}{s} \rightarrow \frac{0.1z + 0.1}{z - 1}$$

However, using Forward Euler:

$$s = \frac{z - 1}{T_s}$$

$$\frac{1}{s} \rightarrow \frac{0.2}{z - 1}$$

$$\frac{Y(z)}{U(z)} = \frac{0.2}{z - 1} = \frac{0.2z^{-1}}{1 - z^{-1}}$$

Substitute in:

$$U(z)\dot{z}^{-1} = u[k - 1]$$

$$Y(z)\dot{z}^{-1} = y[k-1]$$
$$Y(z) = y[k]$$

Giving:

$$y[k] = T_s u[k-1] + y[k-1]$$

Where $T_s = 0.2$.

d

[8 marks][MATLAB]
Re-implement the Simulink model from 1b), but now with the digital PID controller derived in 1c) represented as a discrete transfer function. Generate plots of R1's velocity $u(t)$ and the distance $d(t)$ over 16 seconds. Comment on the performance of the digital PID controller, when compared to the continuous controller implemented in 1b).

Through the Tustin method, the discrete transfer function for the system can be calculated as follows:

$$s = \frac{2}{T_s}\frac{z-1}{z+1}$$
$$\frac{1}{s} \rightarrow \frac{0.1z + 0.1}{z-1}$$

The Tustin method to calculate the transfer function for the controller is as follows:

$$K_p + \frac{K_i}{s} + K_d s$$

with PID constants as follows:

- $P = 1$
- $I = 1$
- $D = 0$ rounded for simplicity

Gives:

$$tf_{pid} = \frac{11z - 9}{10z - 10}$$

The discrete controller would perform better if the sampling time (0.2) was lower, as the quantization error would be reduced, as the function would tend toward continuous as $T_s \rightarrow 0$.

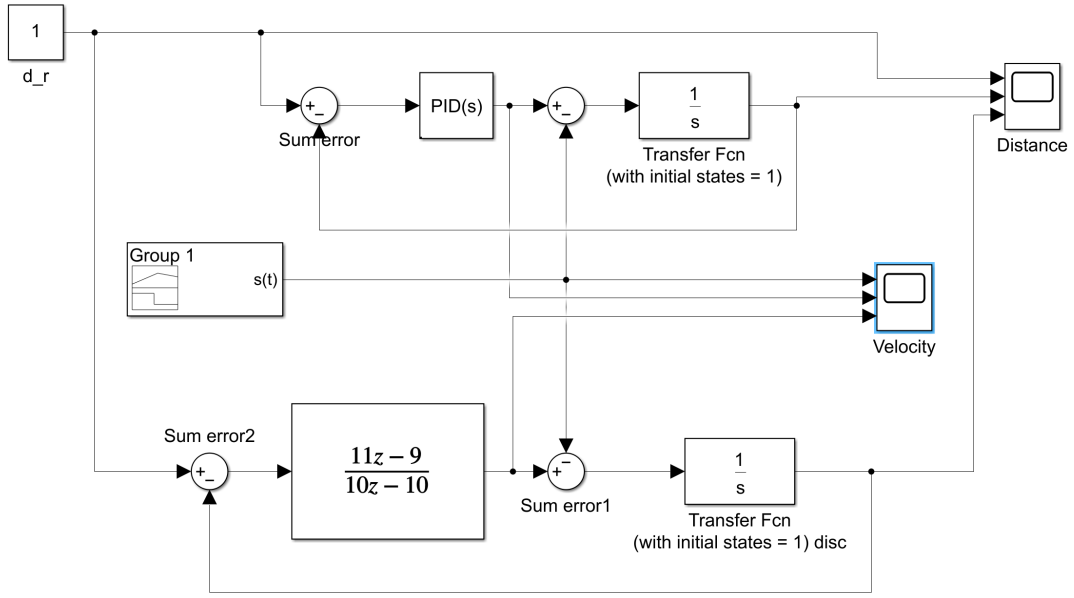

Figure 6: 1d Simulink Model

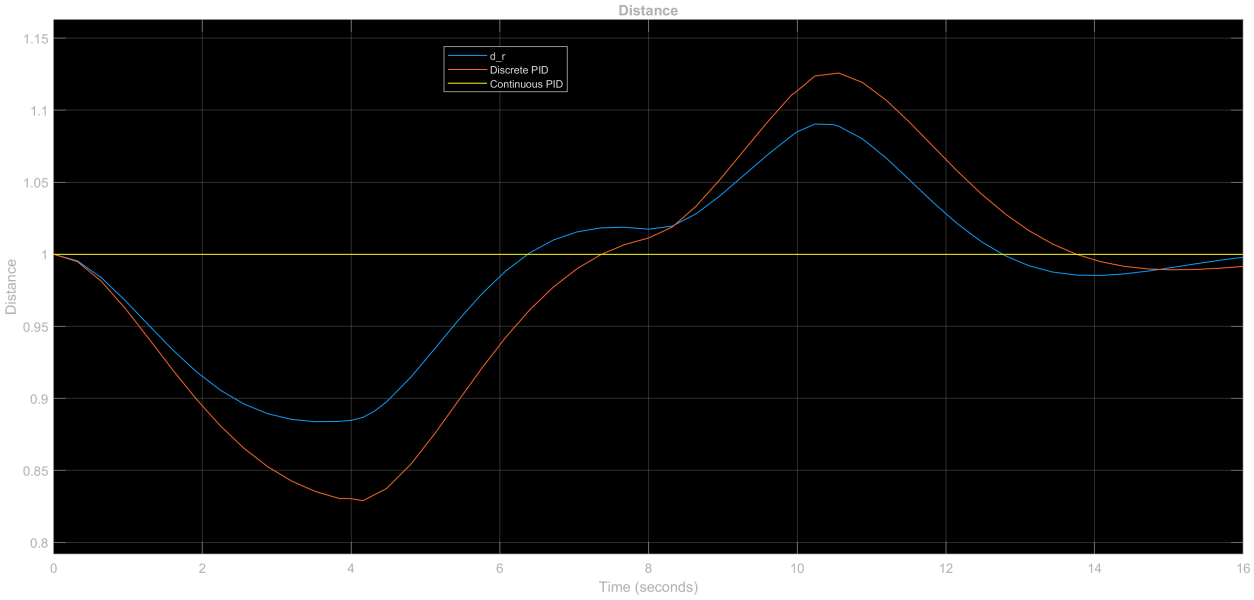

Figure 7: 1d Distance time graph

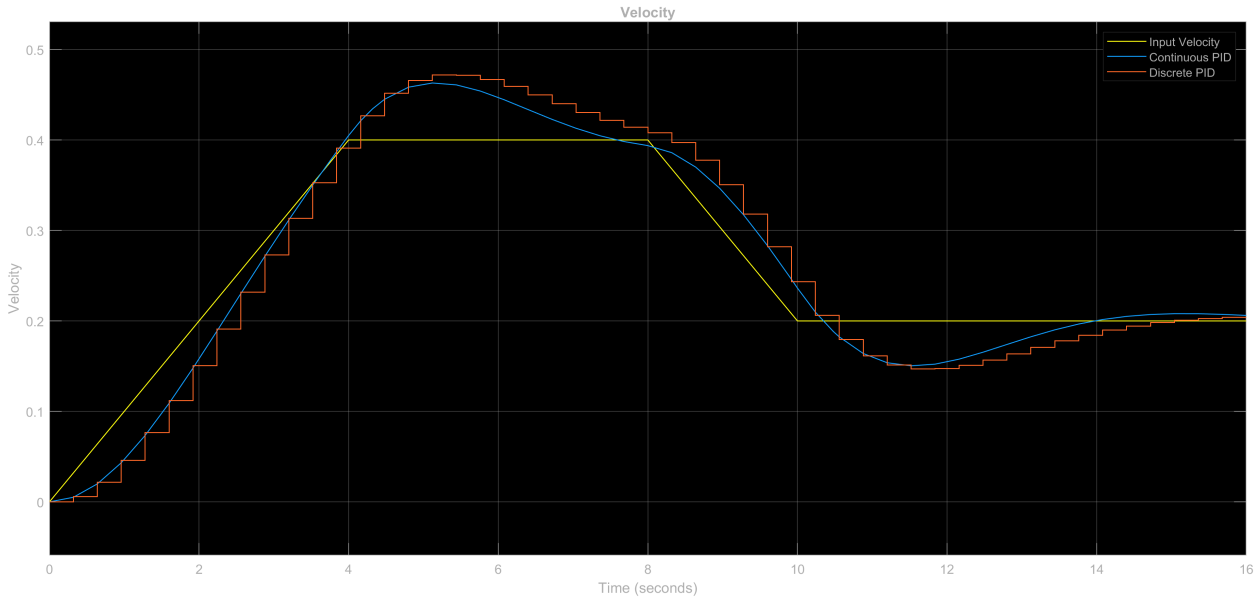

Figure 8: 1d Velocity time graph

e

[8 marks][MATLAB] *Re-implement the Simulink model from 1c), adding random Gaussian noise of 0.01 standard deviation to the control signal* $u(t)$ *and also to the sensor measurement of* $d(t)$*. Compare the performance of the controller against the noise-free scenario in d), and comment on the differences.*

To add Gaussian random noise with standard deviation 0.1 to input $u(t)$ and measurement $d(t)$, $u(t)$ was discretised to the fastest rate possible, then Gaussian noise was added via the Listing Gaussian Noise Matlab function block. Gaussian noise was also added in the feedback measurement. See figures below.
Due to the additive Gaussian white noise to the input signal and the feedback error, the model is not as robust, thus oscillates slightly more than the previous model around the correct values. Although the signal is still reasonable for distance and velocity outputs.

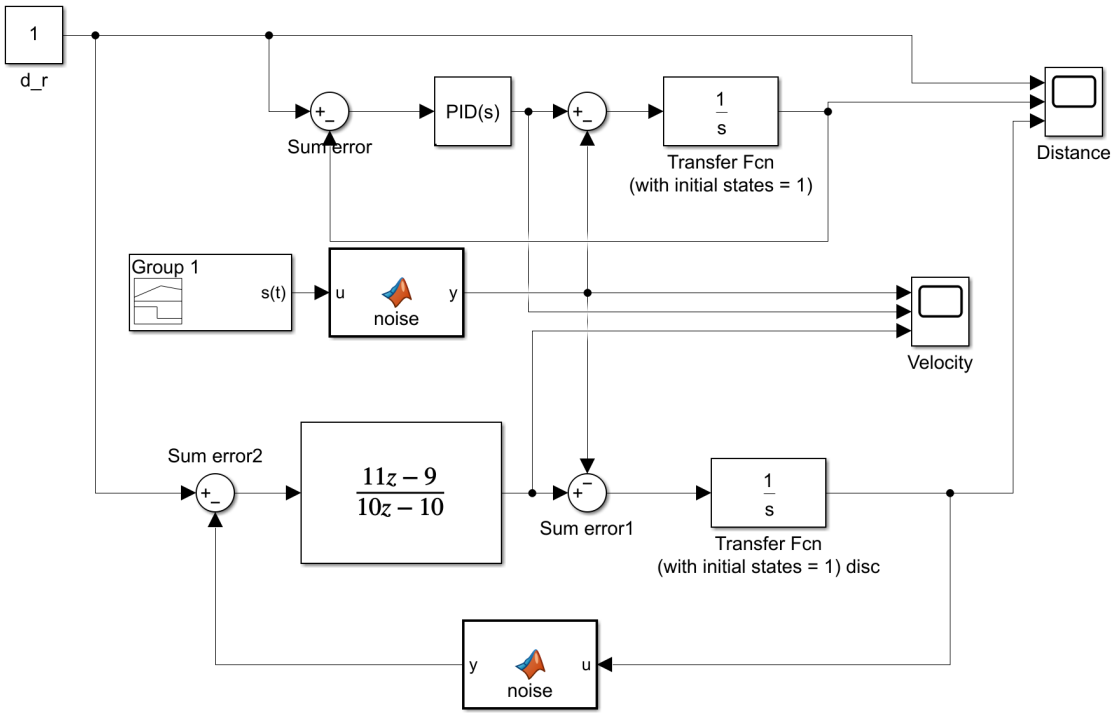

Figure 9: Gaussian Noise Simulink model

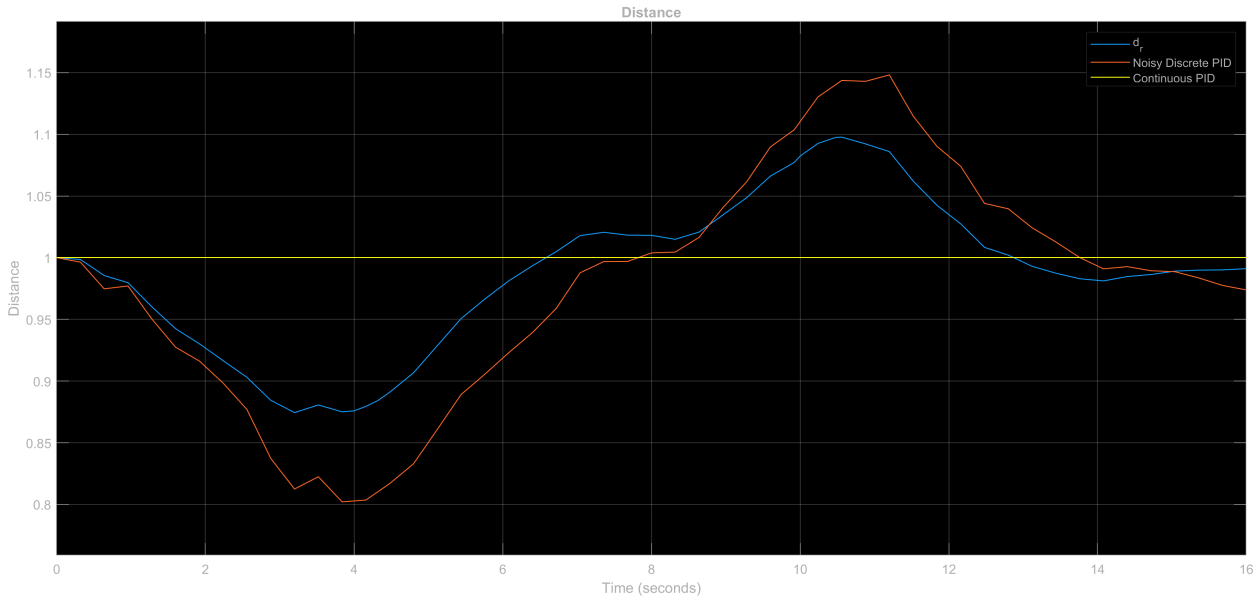

Figure 10: q1e Distance graph

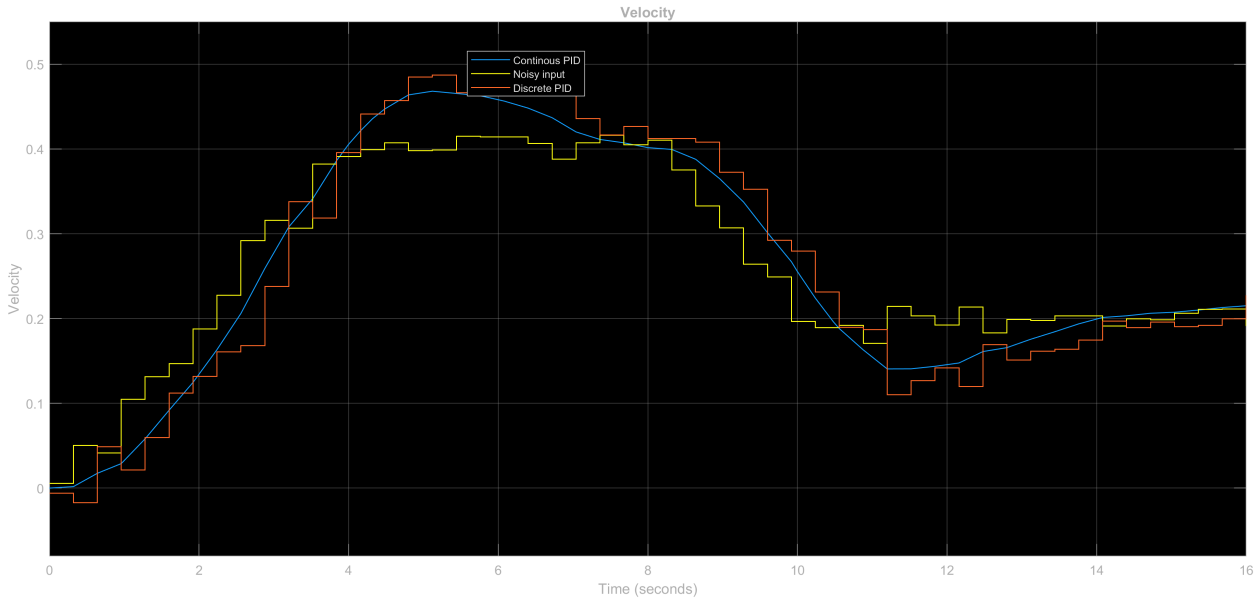

Figure 11: Noisy Input signal s(t) and output R1 Velocity

```
function y = noise(u)
% std = sqrt(var)
```

```
mean0 = 0;
standard_dev = 0.01;
y = u + standard_dev*randn(1)+ mean0;
```

Listing 1: 1e Gaussian noise Matlab function block

f

[18 marks][MATLAB] *In a Matlab .m script, re-implement a simulation of the noisy system in 1e), together with the discrete digital filter derived in 1c). Compare the results against the Simulink simulation in e).*

See below the MATLAB code used to simulate the noisy system with discrete digital filter. The system is not as robust to noise as the output $d(t)$ distance struggles to keep up with the d_r.

```
%Make Signal s(t)
ts = 0.2
total_time = 16
%s = [0, 0.1, 0.2, 0.3, 0.4, 0.4, 0.4, 0.4, 0.4, 0.3, 0.2,0.2, 0.2, 0.2,
    0.2, 0.2, 0.2];
s = [];
s = [s,0:0.02:0.4];
s = [s, linspace(0.4,0.4,4/ts)];
s = [s, linspace(0.4,0.2,2/ts)];
s = [s, linspace(0.2,0.2,6/ts)];

%Make Time, 16s duration with 0.2s intervals
t = 0:ts:total_time;

%Reference distance
d_r = 1;

%Add gaussian noise [See noise function]
s_noisy= noise(s);


%Generate Discrete transfer fcn PID
sys_discrete = tf([11 -9], [10 -10], ts);

%Generate Continous transfer fcn integrator
sys_cont = tf([1],[1 0])

%Run signal through continous transfer function integrator
[d,t] = lsim(sys_cont, s_noisy,t);

%Find error in signal
err = d_r - d;
noisy_err = noise(err)
```

```
33
34 %Run signal through discrete trasnfer fcn PID
35 [s_1,t] = lsim(sys_discrete, d_r + noisy_err ,t)
36 %Run signal
37 [d,t] = lsim(sys_cont, s_1,t)
38
39 %WHAT CONVERGENCE LIMIT DO YOU SET?
40
41
42 %Plot
43 figure
44 subplot(2,2,1)
45 [x,y] = lsim(sys_cont, s_1, t)
46 plot(y,x)
47 title('Output Distance')
48 xlabel('Time (seconds)')
49 ylabel('Distance')
50 hold on
51 plot(t, ones(1, length(t)))
52 legend('Output distance', 'd_r')
53
54 subplot(2,2,2)
55 lsim(sys_discrete, d_r + noisy_err , t)
56 title('Output Velocity')
57 xlabel('Time (seconds)')
58 ylabel('Velocity')
59
60 subplot(2,2,3)
61 plot(t,s)
62 title('Input Velocity')
63 xlabel('Time (seconds)')
64 ylabel('Velocity')
65
66 subplot(2,2,4)
67 lsim(sys_cont, s_1, t)
68 title('Output Distance')
69 xlabel('Time (seconds)')
70 ylabel('Distance')
71
72 function y = noise(u)
73 % std = sqrt(var)
74 mean = 0;
75 standard_dev = 0.01;
76 y = u + standard_dev*randn(1)+ mean;
77 end
```

Listing 2: 1f Code

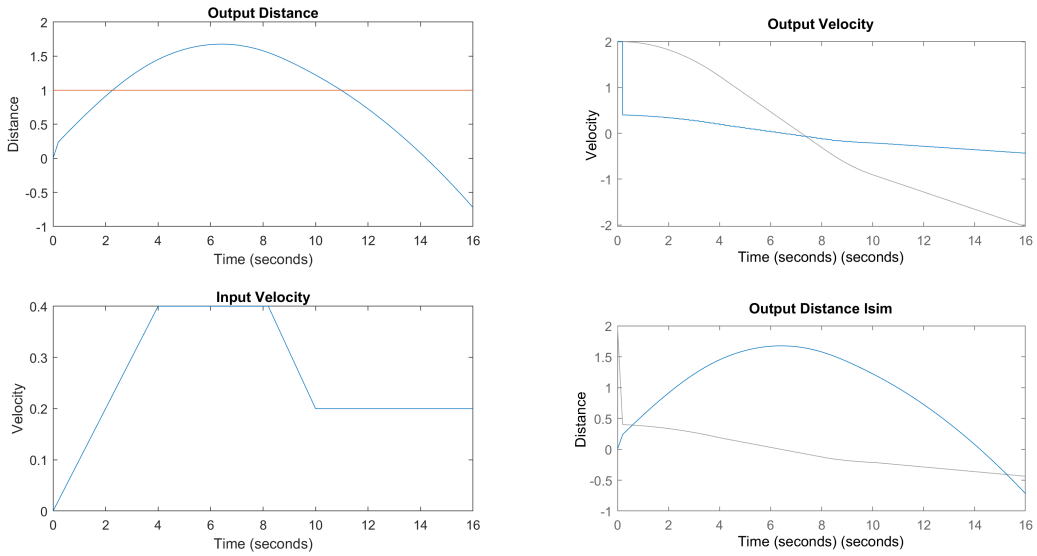

Figure 12: Distance and Velocity simulations

Question 2

[30 marks]
Assume now that the input of R1 $u(t)$ controls its acceleration instead. Also assume that R2 is stopped ($s(t) = 0$).

a

[8 marks]
Determine the open-loop transfer function of this system considering as input the reference $u(t)$ and as the output the distance $d(t)$.

State: $x = d$
Input: $u(t) = a(t) = \ddot{d}$
Output: $y = x = d(t)$
Dynamics: $\dot{d}_2 - \dot{d}_1 = 0 - \int_0^t u(t)dt$

$$\rightarrow sD(s) = -\frac{U(s)}{s}$$

$$\frac{D(s)}{U(s)} = -\frac{1}{s^2}$$

b

[4 marks][MATLAB]
Investigate the viability of using a proportional controller for this system, through a root locus analysis.

```
%Proportional controller
sys = tf([-1],[1 0 0])
sisotool(sys)
```

Listing 3: 2b Root Locus MATLAB Code

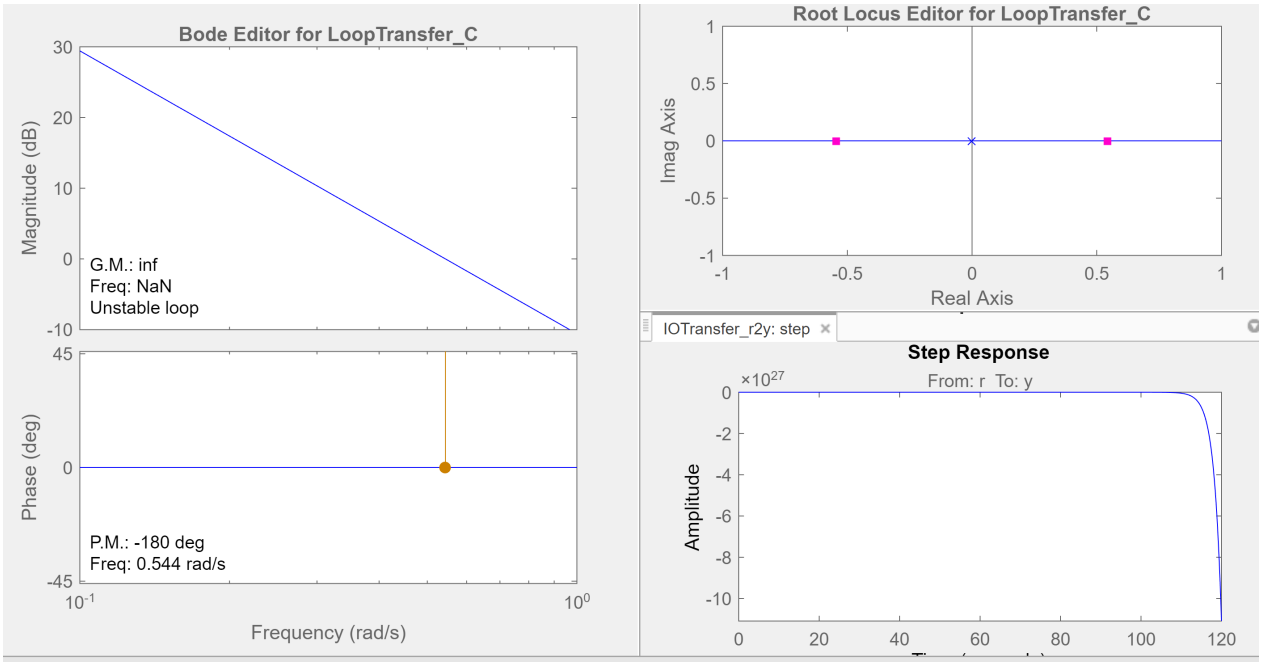

Figure 13: Root Locus Analysis

Root locus shows us system stability The root locus plot shown in the figure above, for a transfer function $-\frac{1}{s^2}$ shows the poles are always on the real axis, therefore whatever the Proportional gain values are changed to, they will always be real. Thus this system is suitable for a P controller. A stable loop was found to be 45^0.

c

[18 marks][MATLAB]
Implement a Simulink model that simulates the described system. Consider that:

- *The reference distance is variable and should follow:*

$$d_r(t) = \begin{cases} 1.5, & \text{if } 0 \leq t < 4 \\ 2.3 - 0.2t, & \text{if } 4 \leq t < 8 \\ 0.7, & \text{if } 8 \leq t < 12 \end{cases}$$

- *The input of R1 $u(t)$ is generated by a lead or a lag compensator. Tune its parameters, by conforming to suitable stability margins.*

Generate plots of R1's velocity and the distance $d(t)$ over 16 seconds

To determine the stable parameters of the lead or lag compensator, pole placement was performed as follows:

$$\frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{K\frac{s-1}{s+b}\frac{-1}{s^2}}{1 + k\frac{s+a}{s+b}\frac{-1}{s^2}}$$

$$= \frac{-Ks - Ka}{s^3 + bs^2 - Ka - Ka}$$

Solving for K with poles (roots) at 1, 2, 3:

$$s^3 + bs^2 - Ks - Ka = s^3 + 6s + 11s + 6$$

$$K = -11$$

$$a = 0.55$$

$$b = 6$$

Thus the Lead or lag compensator is:

$$11(\frac{s - 0.55}{s + 6})$$

Inputting this to a root locus analysis with code shown below, the stable loop was found with a phase margin 0^0 degrees. See figure below. This system has a stable loop and well damped oscillations, thus stable.

```
sys = tf([-1],[1 0 0])
ctr = tf([1 -0.55],[1  6])
sisotool(sys,ctr)
```

Listing 4: Sisotool

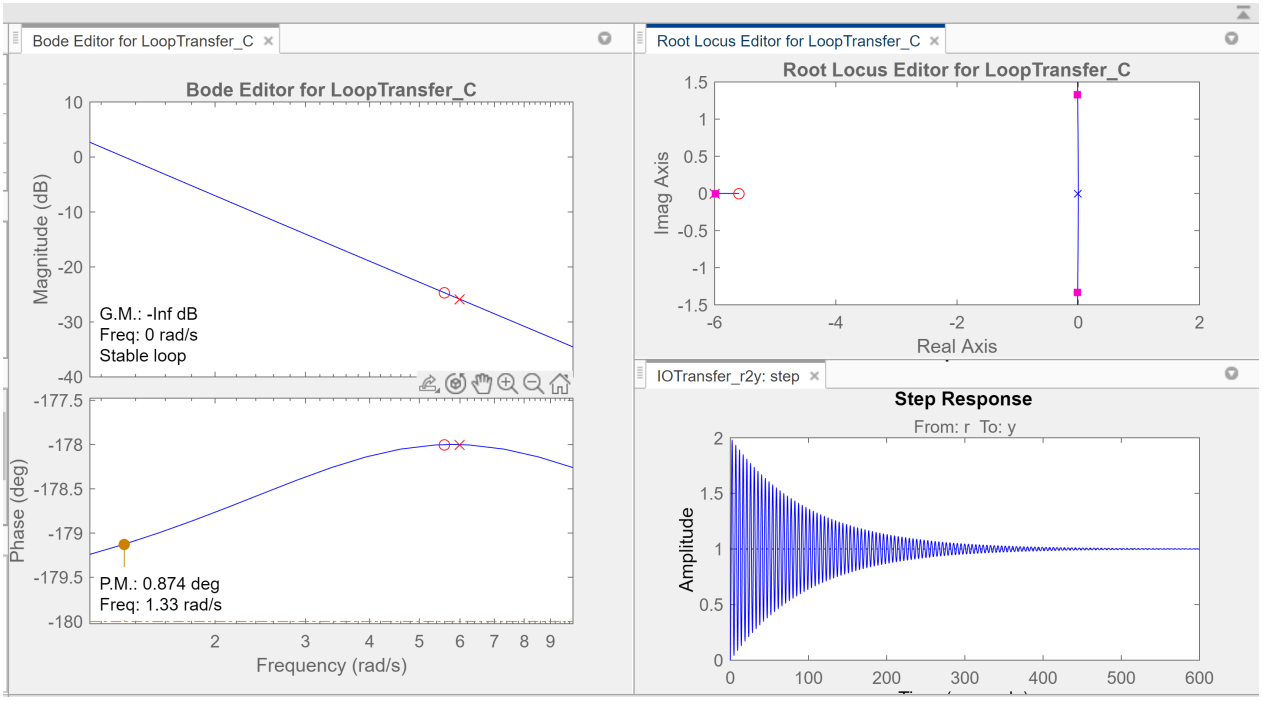

Figure 14: Root Locus

Pole placement with all poles at 1, 1, 1 was also tried, giving:

$$K = -3$$

$$a = \frac{1}{3}$$

$$b = -3$$

However this gave an instable system.
Inputting the lead or lag compensator and associated gain into the simulink model gives the distance and velocity time graphs Figure 17 shown.

h for here

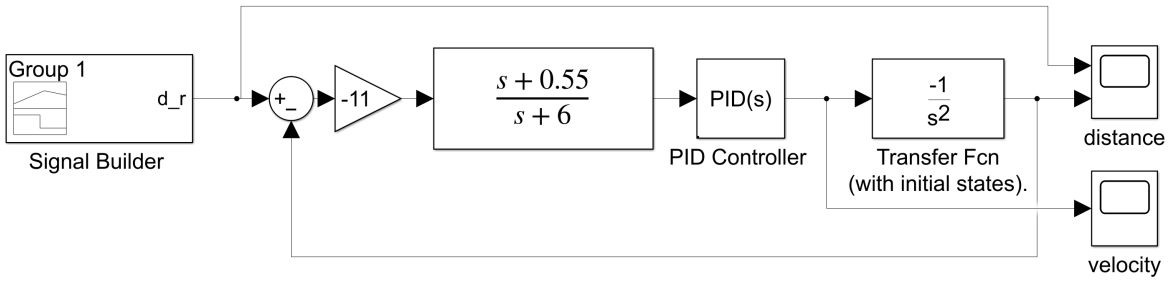

Figure 15: 2c Simulink Model

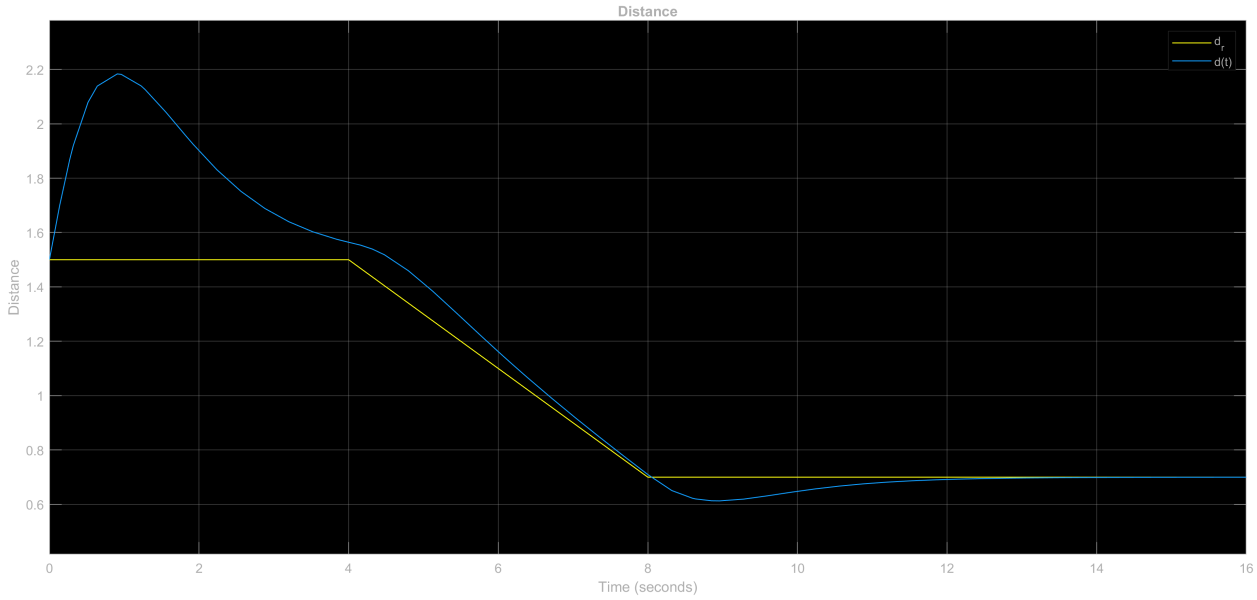

Figure 16: Q2c Distance Graph

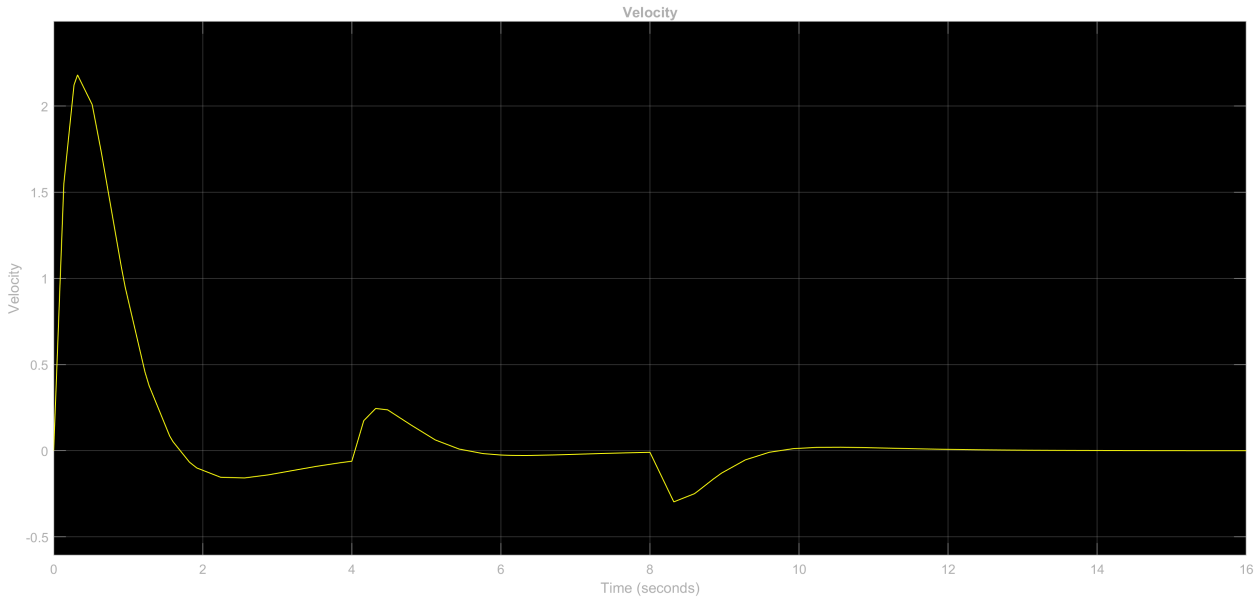

Figure 17: Q2c Velocity Graph