ELEC 442 101

Introduction to Robotics

Assignment 5

Zhi Chuen Tan **Hubert Shim**

65408361 **98504806**

THE UNIVERSITY
OF BRITISH COLUMBIA
Applied Science

Electrical and
Computer
Engineering

1 Two-Link Manipulator Open-Loop Simulation

The Euler-Lagrange approach results in a generalizaed manipulator dynamics equation of the form:

$$\underbrace{D(\boldsymbol{q})}_{\substack{\text{manipulator}\\\text{inertia matrix}}}\ddot{\boldsymbol{q}} + \underbrace{C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}}_{\substack{\text{accounts for}\\\text{Coriolis \& Centripetal}\\\text{terms}}} + \underbrace{G(\boldsymbol{q})}_{\substack{\text{accounts for}\\\text{gravitational \& other}\\\text{potential energy terms}}} = \underbrace{\boldsymbol{u}}_{\substack{\text{generalized}\\\text{motor}\\\text{forces}}} + \underbrace{J_n^T\begin{bmatrix}\underline{\boldsymbol{f}}_e\\ \underline{\boldsymbol{\tau}}_e\end{bmatrix}}_{\text{forces from environment}}$$

As the robot does not interact with the environment, we can ignore the last term. Rearranging this equation then gives us:

$$\ddot{\boldsymbol{q}} = D^{-1}(\boldsymbol{q})[\boldsymbol{u} - C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - G(\boldsymbol{q})], \tag{1}$$

where

$$\boldsymbol{q} = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$$

$$\boldsymbol{u} = \begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}$$

$$D(\boldsymbol{q}) = \begin{bmatrix}(m_1+m_2)l_1^2 + m_2l_2^2 + 2m_2l_1l_2\cos\theta_2 & m_2l_2^2 + m_2l_1l_2\cos\theta_2\\ m_2l_2^2 + m_2l_1l_2\cos\theta_2 & m_2l_2^2\end{bmatrix}$$

$$C(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix}-2m_2l_1l_2(\sin\theta_2)\dot{\theta_2} & -m_2l_1l_2(\sin\theta_2)\dot{\theta_2}\\ m_2l_1l_2(\sin\theta_2)\dot{\theta_1} & 0\end{bmatrix}$$

$$G(\boldsymbol{q}) = \begin{bmatrix}(m_1+m_2)gl_1\cos\theta_1 + m_2gl_2\cos(\theta_1+\theta_2)\\ m_2gl_2\cos(\theta_1+\theta_2)\end{bmatrix}$$

$$T = \frac{1}{2}\dot{\boldsymbol{q}}^T D(\boldsymbol{q})\dot{\boldsymbol{q}}$$

$$V = m_1gl_1\sin\theta_1 + m_2g[l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2)]$$

$$\Rightarrow E_T = T + V$$

The simulation results for each run can be found in section 1.1.3.

1.1 Simulation

1.1.1 Simulink Block Diagrams

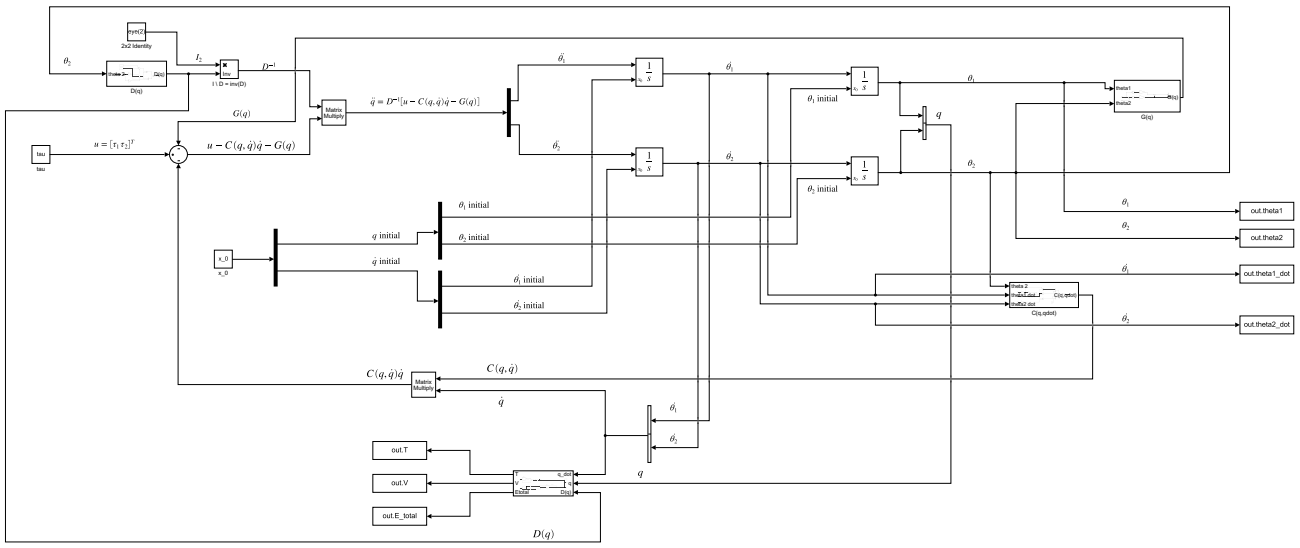

Figure 1: Simulink Block Diagram for the robot.

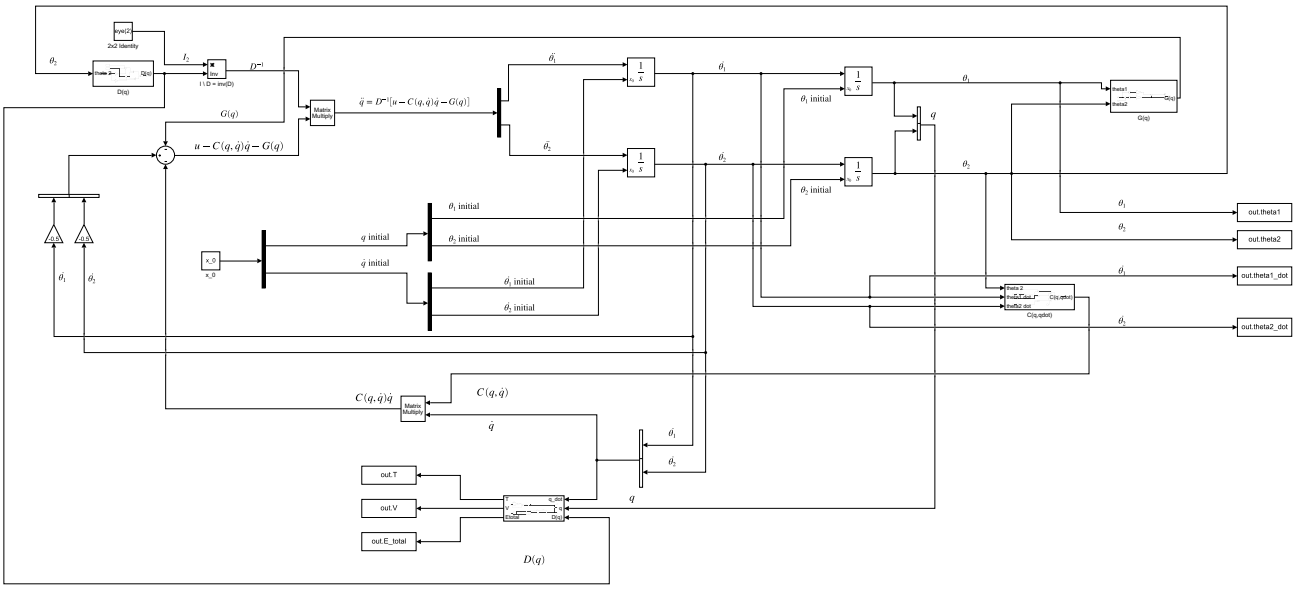

Figure 2: Simulink Block Diagram for the robot, used in part 1(c) to feed friction back into the system.)

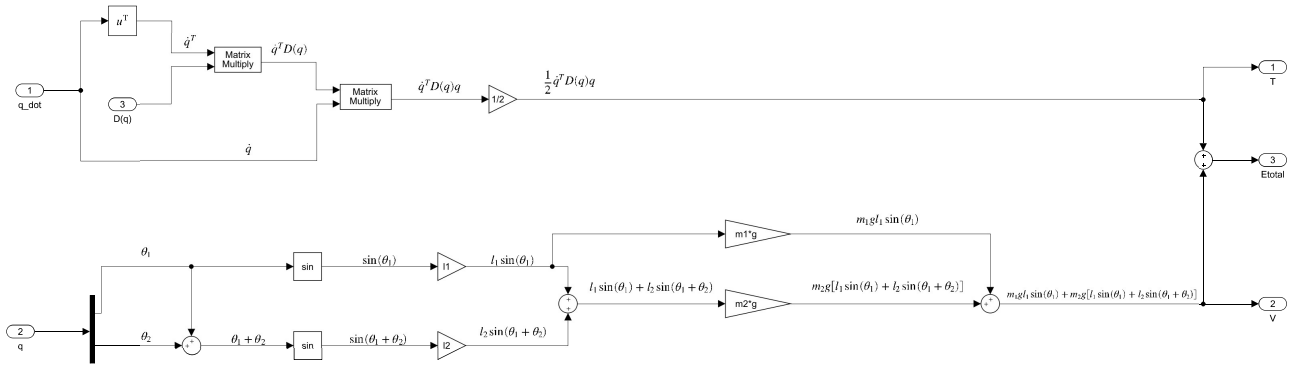

Figure 3: Simulink Block Diagram for the the subsystem used to calculate (potetial, kinetic, and total) energy.

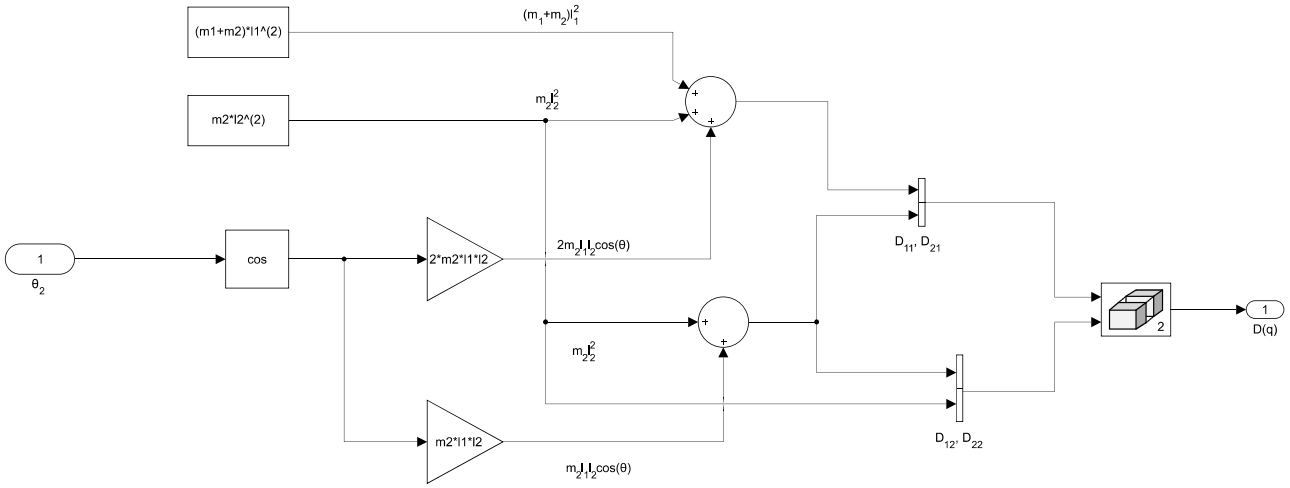

Figure 4: Simulink Block Diagram for the the subsystem used to calculate the $D(\boldsymbol{q})$ matrix.

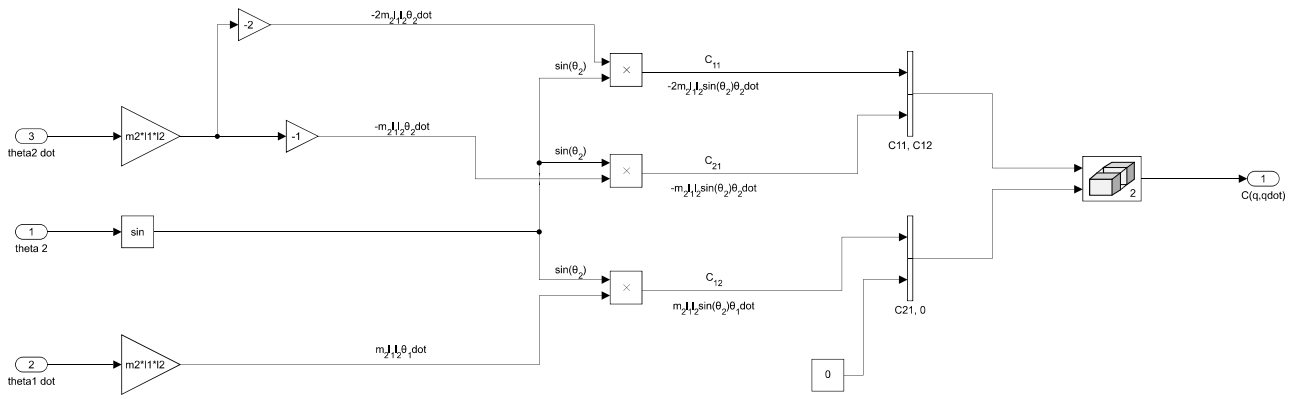

Figure 5: Simulink Block Diagram for the the subsystem used to calculate the $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ matrix.

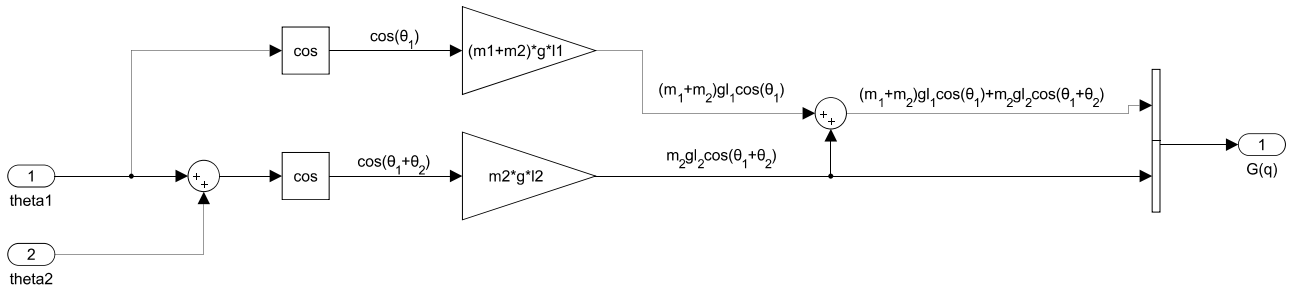

Figure 6: Simulink Block Diagram for the the subsystem used to calculate the $G(\boldsymbol{q})$ matrix.

1.1.2 MATLAB Code

The following code was used to simplify running the simulation, plotting the joint angles and velocities, and kinetic, potential, and total energies, and automatically saves the figures as `.fig` (and `.eps` or `.pdf` files for the LaTeXtypesetting, which makes these diagrams vector diagrams, i.e. they can be zoomed-in without significant aliasing):

```
function run_sim_asn5q1(x_0_in,tau_in)
    % Initialize constants
    x_0 = x_0_in;
    tau = tau_in;
    l1=1;l2=1;m1=1;m2=1;g=9.81;
```

```
6
7       % Run simulation
8       simOut = sim('asn5q1');
9
10      % Take output values from Simulink
11      theta1=simOut.get('theta1');
12      theta2=simOut.get('theta2');
13      theta1_dot=simOut.get('theta1_dot');
14      theta2_dot=simOut.get('theta2_dot');
15      T=simOut.get('T');
16      V=simOut.get('V');
17      E_total=simOut.get('E_total');
18
19      % Plots theta1
20      figure;
21      hold on;
22      view(2);
23      title('Plot for $\theta_{1}$', 'Interpreter', 'latex');
24      xlabel('Time (seconds)', 'Interpreter', 'latex');
25      ylabel('$\theta_{1}$', 'Interpreter', 'latex');
26      plot(theta1, 'Color', '#A2142F');
27      saveas(gcf, 'q1a_t1.fig'); % saves figure as .fig
28      saveas(gcf, 'q1a_t1', 'epsc'); % saves figure as .eps (for preparing text)
29
30      % Plots theta2
31      figure;
32      hold on;
33      view(2);
34      title('Plot for $\theta_{2}$', 'Interpreter', 'latex');
35      xlabel('Time (seconds)', 'Interpreter', 'latex');
36      ylabel('$\theta_{2}$', 'Interpreter', 'latex');
37      plot(theta2, 'Color', '#A2142F');
38      saveas(gcf, 'q1a_t2.fig'); % saves figure as .fig
39      saveas(gcf, 'q1a_t2', 'epsc'); % saves figure as .eps (for preparing text)
40
41      % Plots theta1_dot
42      figure;
43      hold on;
44      view(2);
45      title('Plot for $\dot{\theta}_{1}$', 'Interpreter', 'latex');
46      xlabel('Time (seconds)', 'Interpreter', 'latex');
47      ylabel('$\dot{\theta}_{1}$', 'Interpreter', 'latex');
48      plot(theta1_dot, 'Color', '#A2142F');
49      saveas(gcf, 'q1a_t1d.fig'); % saves figure as .fig
50      saveas(gcf, 'q1a_t1d', 'epsc'); % saves figure as .eps (for preparing text)
```

```
51
52      % Plots theta2_dot
53      figure;
54      hold on;
55      view(2);
56      title('Plot for $\dot{\theta}_{2}$', 'Interpreter', 'latex');
57      xlabel('Time (seconds)', 'Interpreter', 'latex');
58      ylabel('$\dot{\theta}_{2}$', 'Interpreter', 'latex');
59      plot(theta2_dot, 'Color', '#A2142F');
60      saveas(gcf, 'q1a_t2d.fig'); % saves figure as .fig
61      saveas(gcf, 'q1a_t2d', 'epsc'); % saves figure as .eps (for preparing text)
62
63      % Plots kinetic energy
64      figure;
65      hold on;
66      view(2);
67      title('Plot for Kinetic Energy, $T$', 'Interpreter', 'latex');
68      xlabel('Time (seconds)', 'Interpreter', 'latex');
69      ylabel('$T$', 'Interpreter', 'latex');
70      plot(T, 'Color', '#A2142F');
71      saveas(gcf, 'q1a_T.fig'); % saves figure as .fig
72      saveas(gcf, 'q1a_T', 'epsc'); % saves figure as .eps (for preparing text)
73
74      % Plots potential energy
75      figure;
76      hold on;
77      view(2);
78      title('Plot for Potential Energy, $T$', 'Interpreter', 'latex');
79      xlabel('Time (seconds)', 'Interpreter', 'latex');
80      ylabel('$V$', 'Interpreter', 'latex');
81      plot(V, 'Color', '#A2142F');
82      saveas(gcf, 'q1a_V.fig'); % saves figure as .fig
83      saveas(gcf, 'q1a_V', 'epsc'); % saves figure as .eps (for preparing text)
84
85      % Plots total energy
86      figure;
87      hold on;
88      view(2);
89      title('Plot for Total Energy, $T+V$', 'Interpreter', 'latex');
90      xlabel('Time (seconds)', 'Interpreter', 'latex');
91      ylabel('$T+V$', 'Interpreter', 'latex');
92      plot(E_total, 'Color', '#A2142F');
93      saveas(gcf, 'q1a_total.fig'); % saves figure as .fig
94      saveas(gcf, 'q1a_total', 'epsc'); % saves figure as .eps (for preparing text)
95  end
```

Listing 1: MATLAB code used to simulate the system for questions 1(a-b)

```
function run_sim_asn5q1c(x_0_in)
    % Initialize constants
    x_0 = x_0_in;

    l1=1;l2=1;m1=1;m2=1;g=9.81;

    % Run simulation
    simOut = sim('asn5q1c');

    % Take output values from Simulink
    theta1=simOut.get('theta1');
    theta2=simOut.get('theta2');
    theta1_dot=simOut.get('theta1_dot');
    theta2_dot=simOut.get('theta2_dot');
    T=simOut.get('T');
    V=simOut.get('V');
    E_total=simOut.get('E_total');

    % Plots theta1
    figure;
    hold on;
    view(2);
    title('Plot for $\theta_{1}$', 'Interpreter', 'latex');
    xlabel('Time (seconds)', 'Interpreter', 'latex');
    ylabel('$\theta_{1}$', 'Interpreter', 'latex');
    plot(theta1, 'blue');
    saveas(gcf, 'q1c_t1.fig'); % saves figure as .fig
    saveas(gcf, 'q1c_t1', 'epsc'); % saves figure as .eps (for preparing text)

    % Plots theta2
    figure;
    hold on;
    view(2);
    title('Plot for $\theta_{2}$', 'Interpreter', 'latex');
    xlabel('Time (seconds)', 'Interpreter', 'latex');
    ylabel('$\theta_{2}$', 'Interpreter', 'latex');
    plot(theta2, 'blue');
    saveas(gcf, 'q1c_t2.fig'); % saves figure as .fig
    saveas(gcf, 'q1c_t2', 'epsc'); % saves figure as .eps (for preparing text)

    % Plots theta1_dot
```

```
42      figure;
43      hold on;
44      view(2);
45      title('Plot for $\dot{\theta}_{1}$', 'Interpreter', 'latex');
46      xlabel('Time (seconds)', 'Interpreter', 'latex');
47      ylabel('$\dot{\theta}_{1}$', 'Interpreter', 'latex');
48      plot(theta1_dot, 'blue');
49      saveas(gcf, 'q1c_t1d.fig'); % saves figure as .fig
50      saveas(gcf, 'q1c_t1d', 'epsc'); % saves figure as .eps (for preparing text)
51
52      % Plots theta2_dot
53      figure;
54      hold on;
55      view(2);
56      title('Plot for $\dot{\theta}_{2}$', 'Interpreter', 'latex');
57      xlabel('Time (seconds)', 'Interpreter', 'latex');
58      ylabel('$\dot{\theta}_{2}$', 'Interpreter', 'latex');
59      plot(theta2_dot, 'blue');
60      saveas(gcf, 'q1c_t2d.fig'); % saves figure as .fig
61      saveas(gcf, 'q1c_t2d', 'epsc'); % saves figure as .eps (for preparing text)
62
63      % Plots kinetic energy
64      figure;
65      hold on;
66      view(2);
67      title('Plot for Kinetic Energy, $T$', 'Interpreter', 'latex');
68      xlabel('Time (seconds)', 'Interpreter', 'latex');
69      ylabel('$T$', 'Interpreter', 'latex');
70      plot(T, 'blue');
71      saveas(gcf, 'q1c_T.fig'); % saves figure as .fig
72      saveas(gcf, 'q1c_T', 'epsc'); % saves figure as .eps (for preparing text)
73
74      % Plots potential energy
75      figure;
76      hold on;
77      view(2);
78      title('Plot for Potential Energy, $T$', 'Interpreter', 'latex');
79      xlabel('Time (seconds)', 'Interpreter', 'latex');
80      ylabel('$V$', 'Interpreter', 'latex');
81      plot(V, 'blue');
82      saveas(gcf, 'q1c_V.fig'); % saves figure as .fig
83      saveas(gcf, 'q1c_V', 'epsc'); % saves figure as .eps (for preparing text)
84
85      % Plots total energy
86      figure;
```

```
87      hold on;
88      view(2);
89      title('Plot for Total Energy, $T+V$', 'Interpreter', 'latex');
90      xlabel('Time (seconds)', 'Interpreter', 'latex');
91      ylabel('$T+V$', 'Interpreter', 'latex');
92      plot(E_total, 'blue');
93      saveas(gcf, 'q1c_total.fig'); % saves figure as .fig
94      saveas(gcf, 'q1c_total', 'epsc'); % saves figure as .eps (for preparing text)
95
96
97
98
99
100 end
```

Listing 2: MATLAB code used to simulate the system for question 1(c)

1.1.3 Simulation Results

All of the diagrams below are vector diagrams and can be zoomed in without significant aliasing.

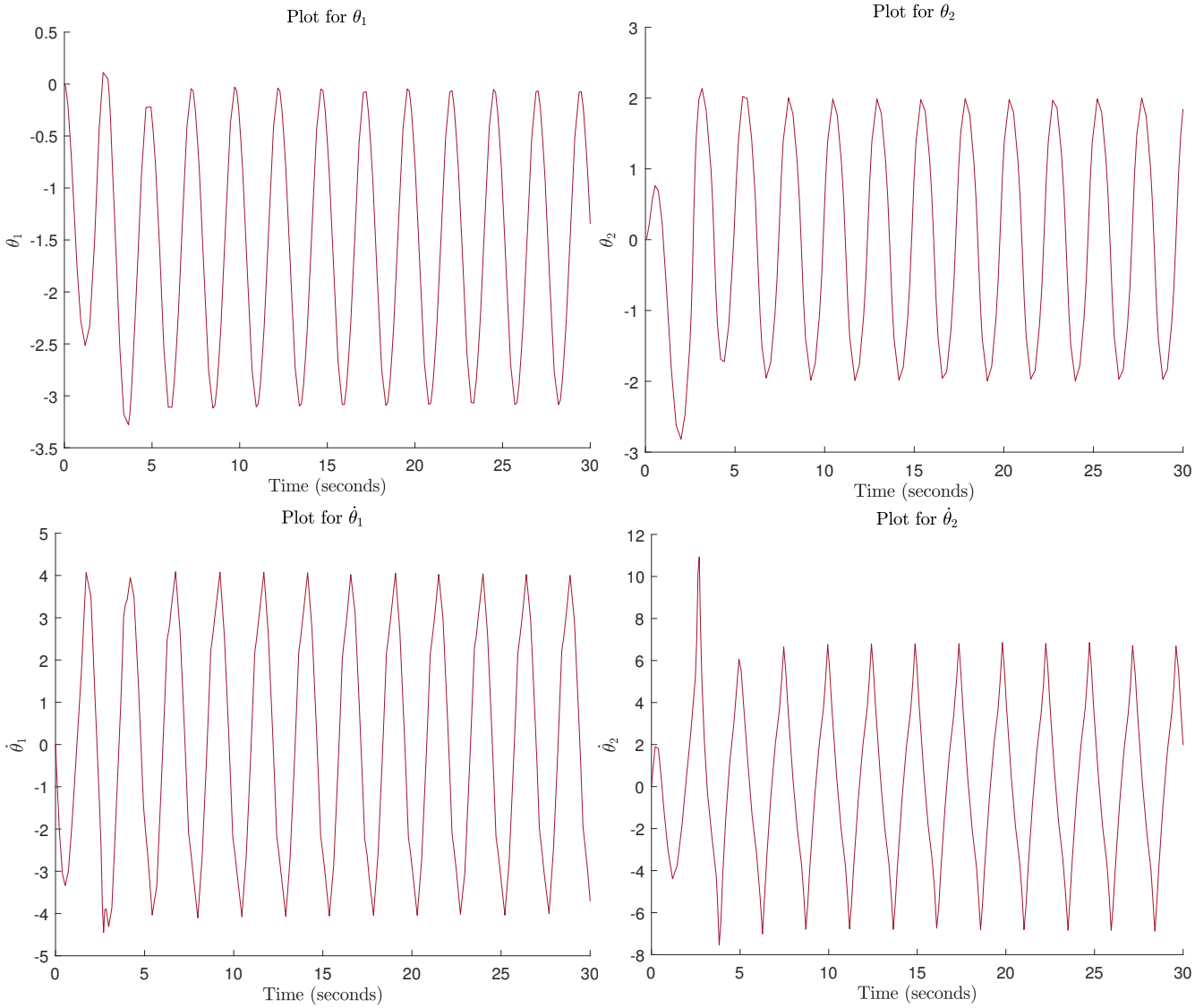

Figure 7: Joint angle and velocity plots for $x(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$, $\tau_1 = \tau_2 = 0$

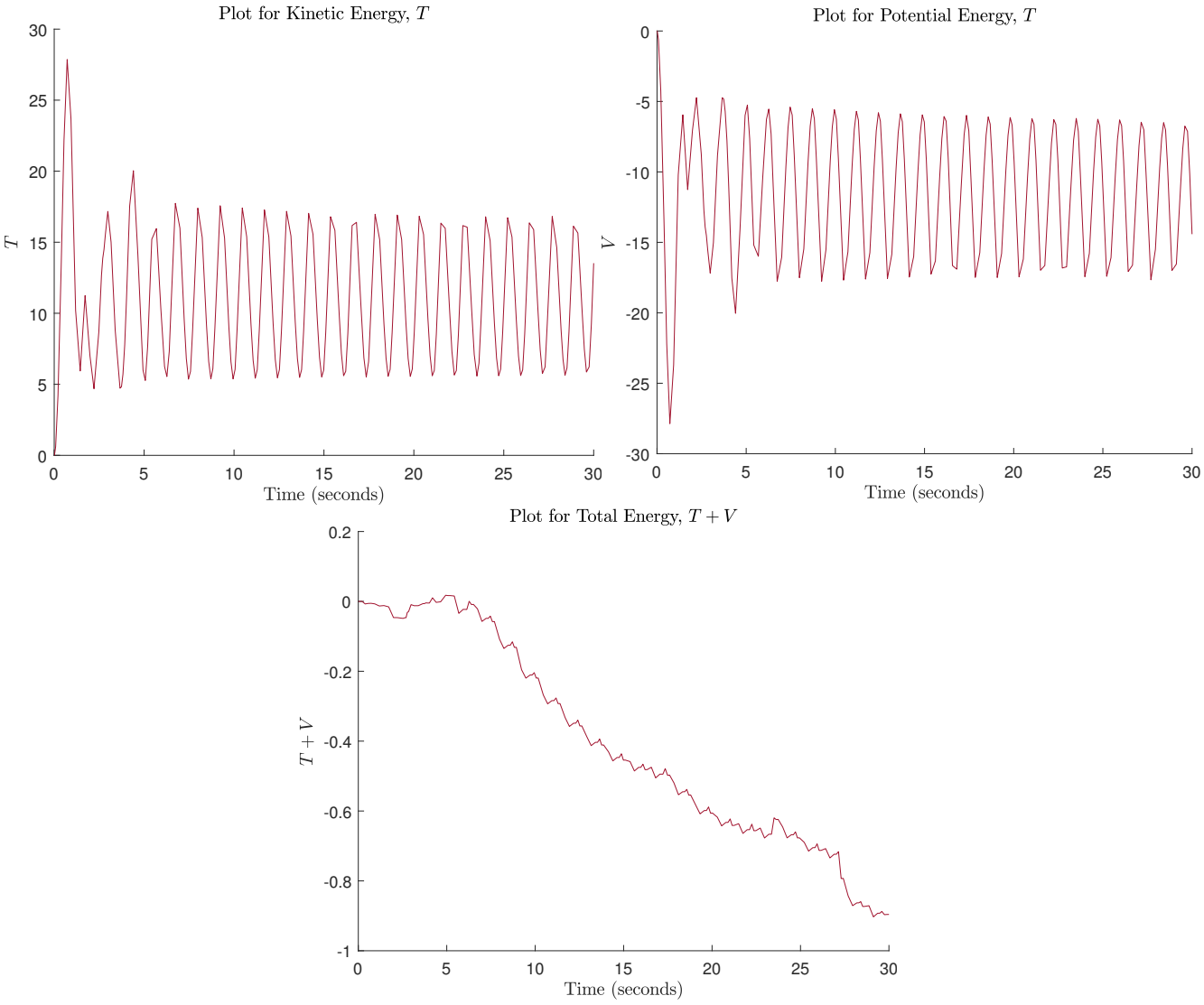

Figure 8: Energy plots for $x(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$, $\tau_1 = \tau_2 = 0$

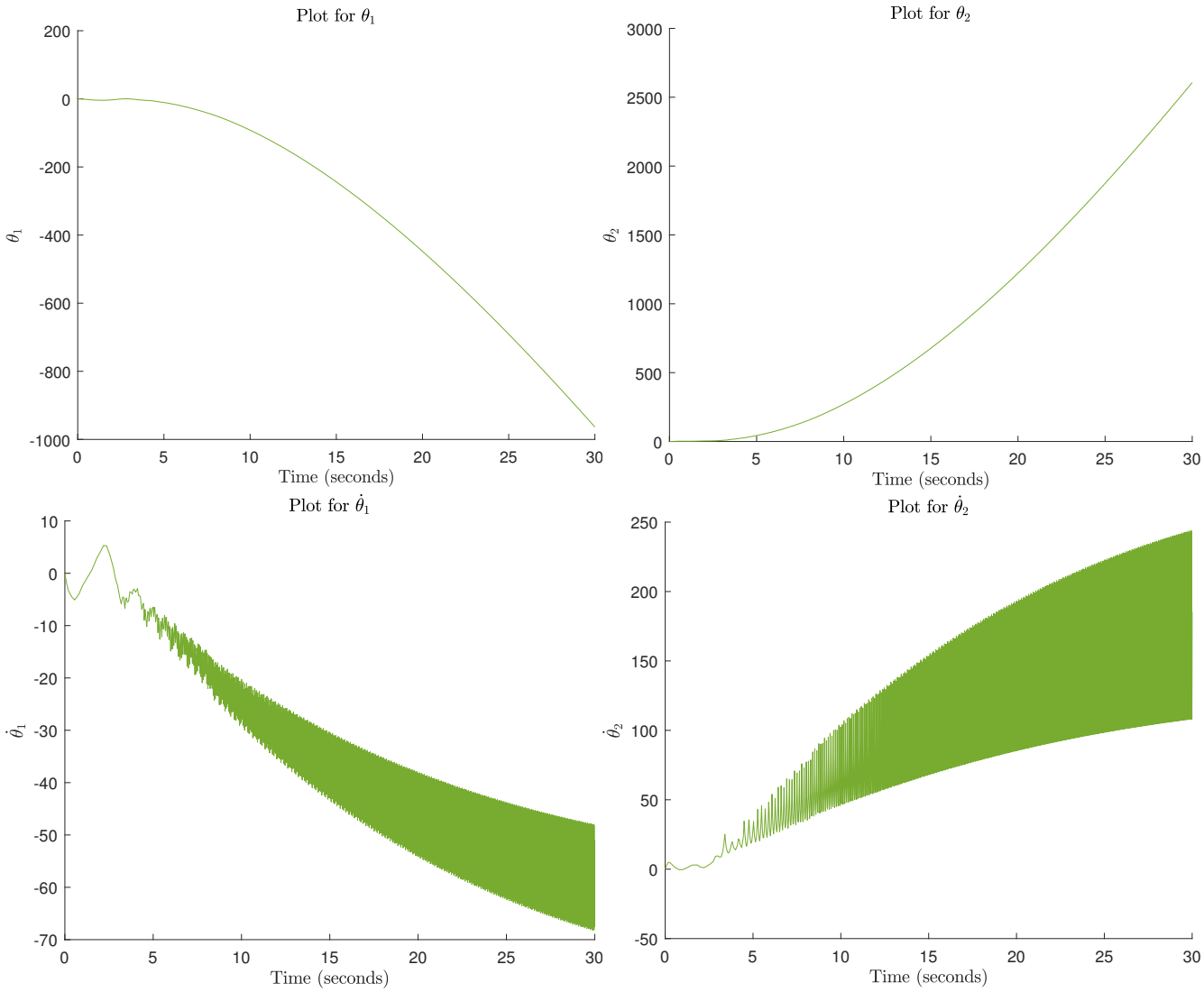

Figure 9: Joint angle and velocity plots for $x(0) = \begin{bmatrix} 0 & \frac{\pi}{2} & 0 & 0 \end{bmatrix}^T$, $\tau_1 = 0$, $\tau_2 = 5$

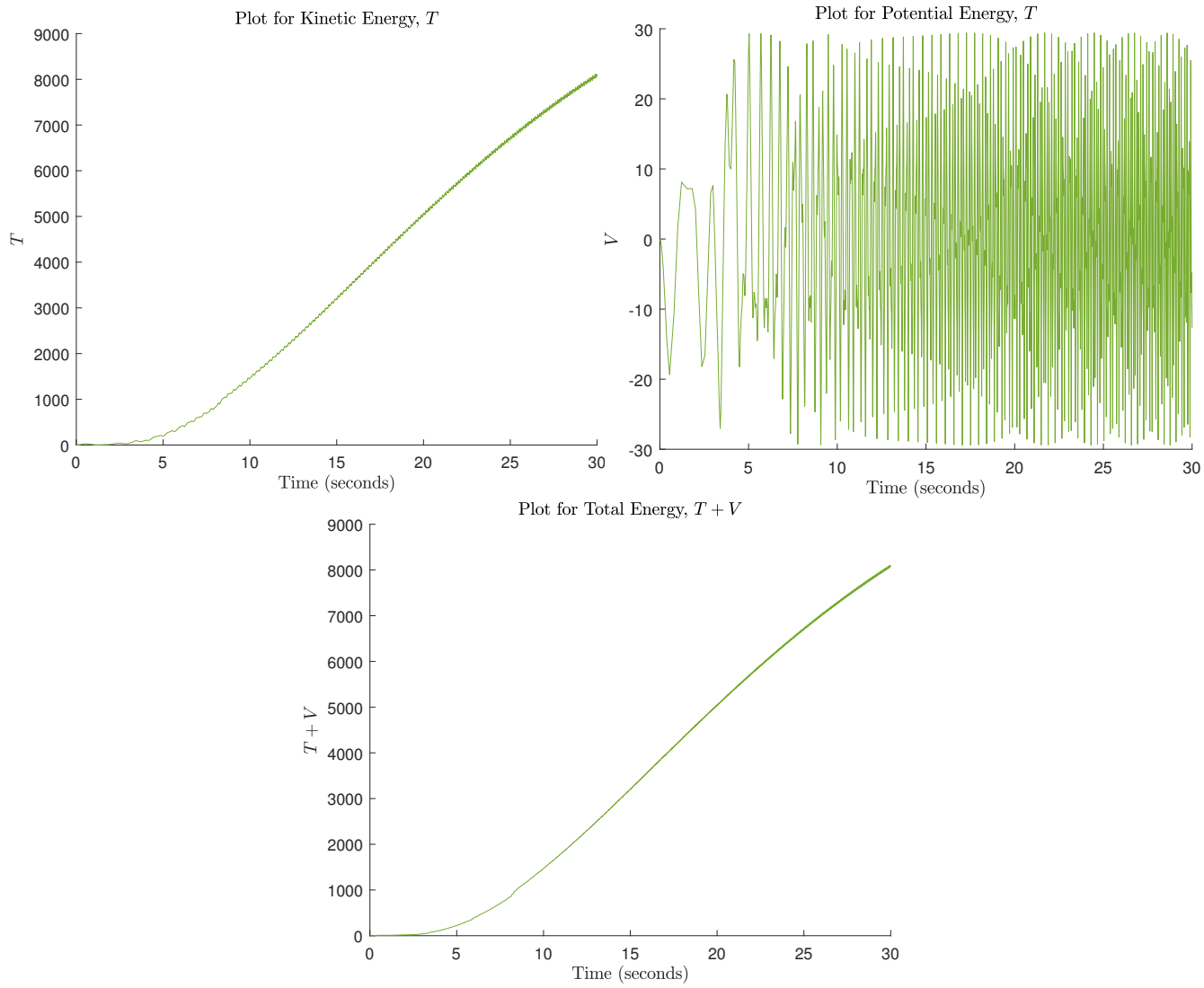

Figure 10: Energy plots for $x(0) = \begin{bmatrix} 0 & \frac{\pi}{2} & 0 & 0 \end{bmatrix}^T$, $\tau_1 = 0$, $\tau_2 = 5$

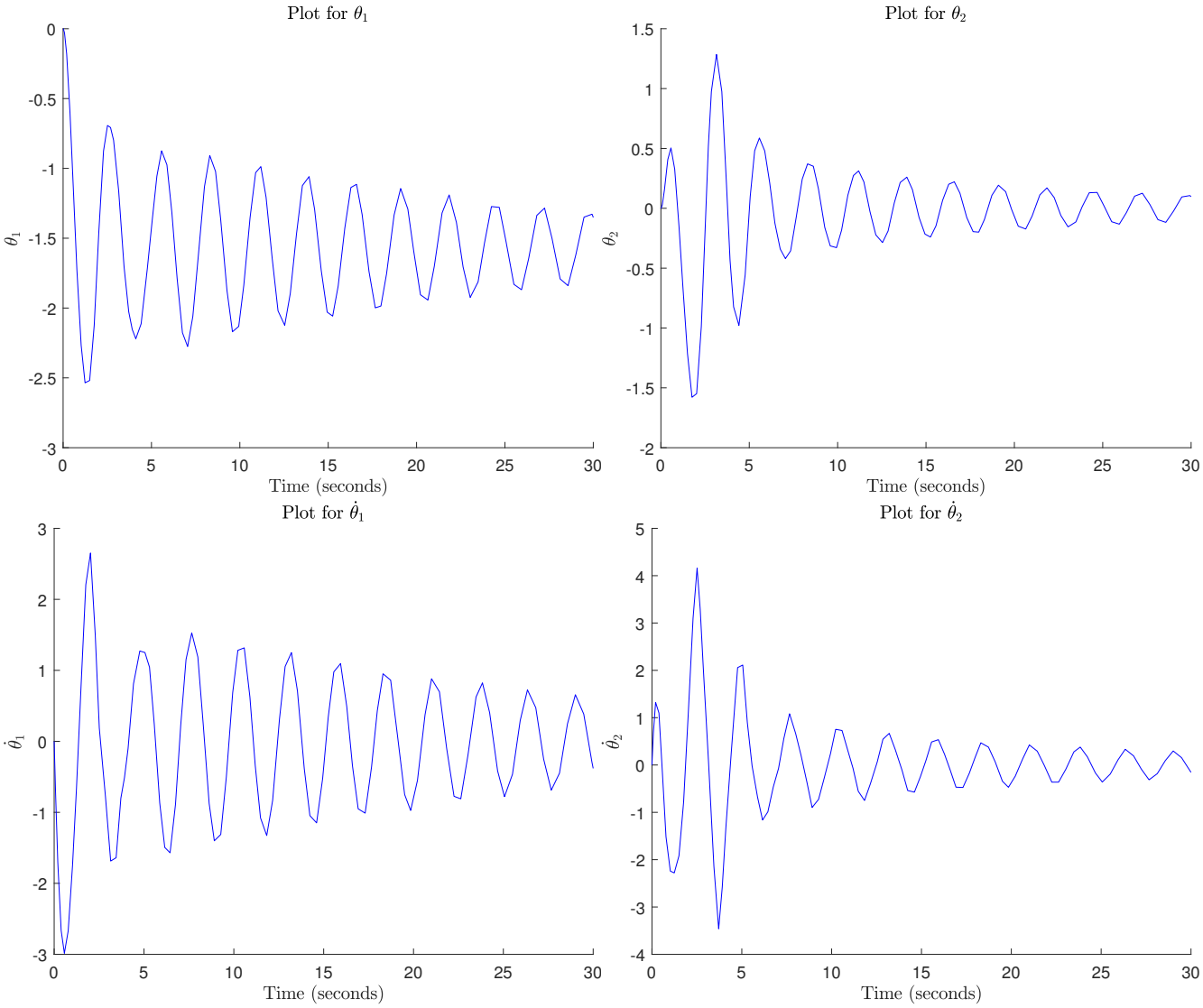

Figure 11: Joint angle and velocity plots for $x(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$, $\tau_1 = -0.5\dot{\theta}_1$, $\tau_2 = -0.5\dot{\theta}_2$

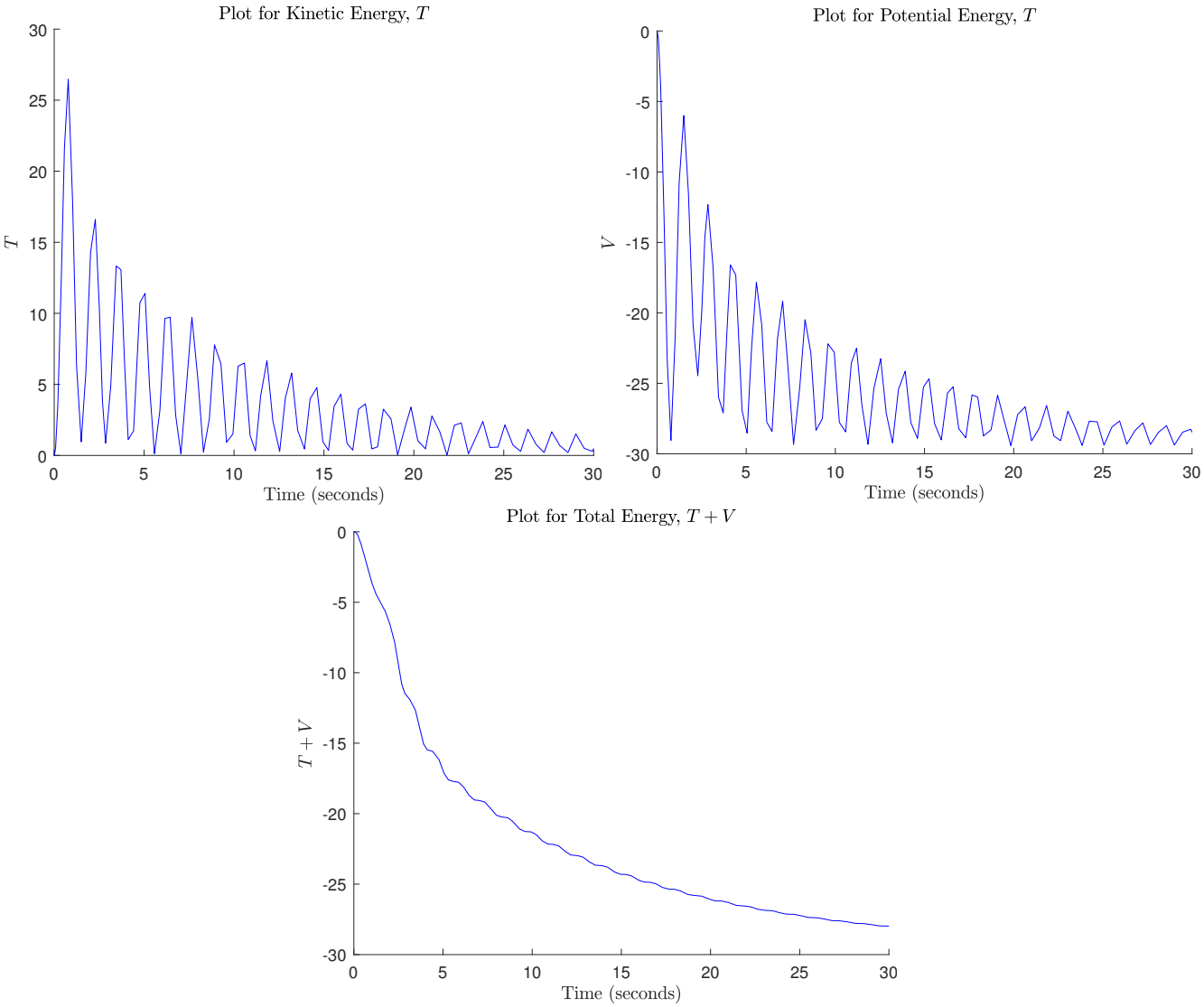

Figure 12: Energy plots for $x(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$, $\tau_1 = -0.5\dot{\theta}_1$, $\tau_2 = -0.5\dot{\theta}_2$

1.1.4 Comments on Simulation Results

a) As we can see the joint angles as well as their derivatives are oscillating, this makes sense for in this case the system as there is 0 motor torque in the motor, putting the robot into a completely undamped state. We see similar patterns in the kinetic and potential energies, with their total energy gradually decreasing.

b) With the second motor on, joint angle 2 will increase overtime, and joint angle 1 will decrease over time due to gravity. The joint velocities also display the non-linear relationship of the joint angles over time, while still displaying how joint 1's angle decreases and joint 2's angle increases over time. The kinetic Energy is constantly growing as motor 2's torque is constant, and while link 1 is going downwards link 2 is going up, causing the pattern in potential energy seen on the graph.

c) Friction causes the system to resemble that of an underdamped response, reflected in the joint angles and velocities. Because the amplitude of oscillation of the joint velocities decay over time, we see the kinetic energy gradually decreasing over time as well.

2 Closed-Loop Controller Implementation

2.1 Joint Space Control

The control law for this controller is

$$\underbrace{\boldsymbol{u}}_{\substack{\text{generalized}\\ \text{motor torques}}} = \underbrace{G(\boldsymbol{q})}_{\substack{\text{gravity}\\ \text{and other}\\ \text{potential energy terms}}} + \begin{bmatrix} K_p & K_v \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_d - \boldsymbol{q} \\ -\dot{\boldsymbol{q}} \end{bmatrix}$$

$$\boldsymbol{u} = G(\boldsymbol{q}) + K_p(\boldsymbol{q}_d - \boldsymbol{q}) - K_v \underbrace{\dot{\boldsymbol{q}}}_{\substack{\text{constant}\\ \text{set-point}}},$$

where we set (for this question):

$$x(0) = \begin{bmatrix} \frac{\pi}{2} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{q}_d = \begin{bmatrix} 0 \\ \frac{\pi}{2} \end{bmatrix}$$

$$K_p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$K_v = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The simulation results for this run can be found in section 2.1.3.

2.1.1 Simulink Block Diagrams

The robot block in Figure 1 was made into a subsystem and connected to a PD + Gravity controller block, shown in Figure 13 below:

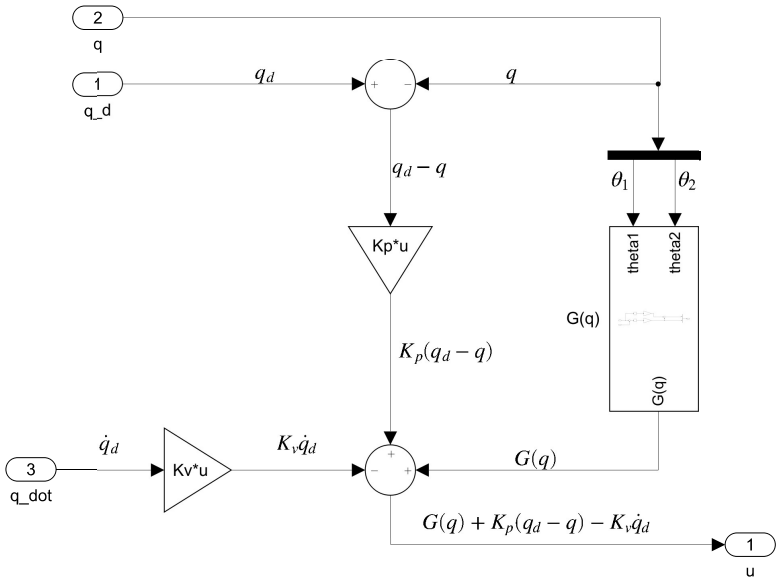

Figure 13: Simulink Block Diagram for the PD+Gravity Controller.

The overall system is shown in Figure 14:

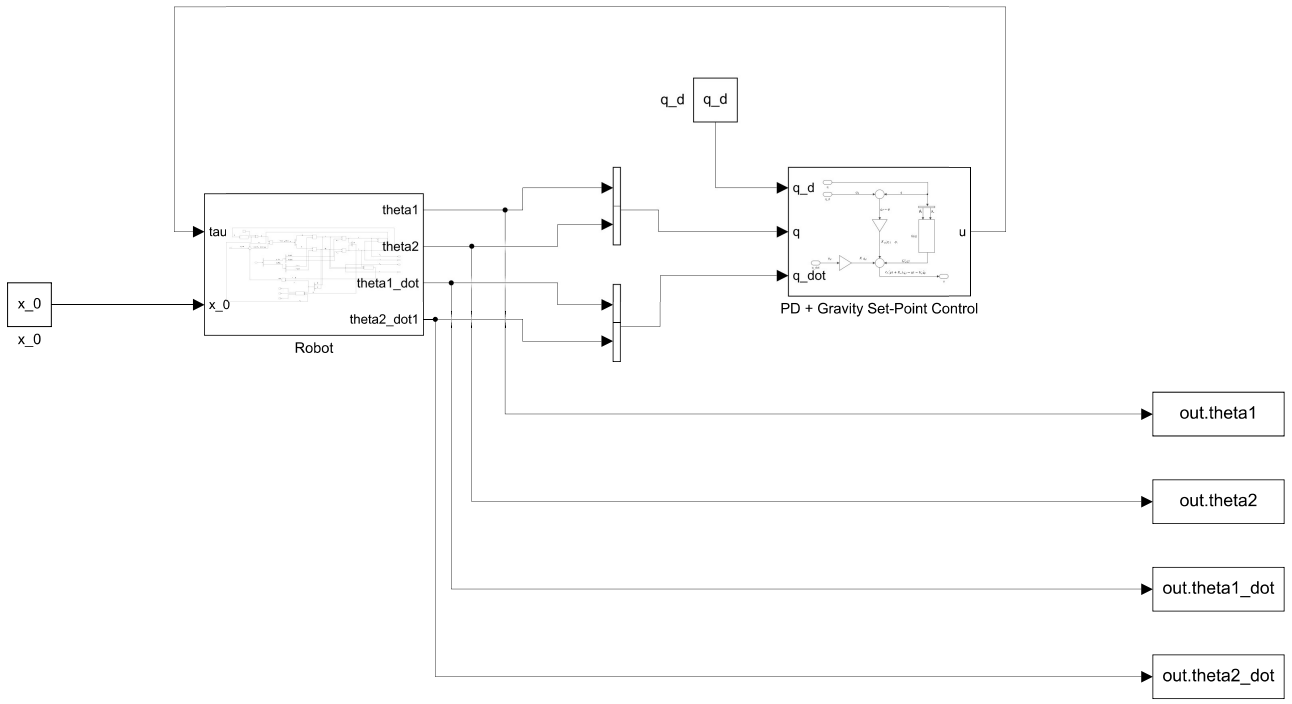

Figure 14: Simulink Block Diagram for the robot connected to the PD+Gravity controller.

2.1.2 MATLAB Code

As in Listing 1, and Listing 2, a MATLAB function was written to automate simulation and saves the plots as `.fig` (and `.pdf`) files. The code is shown in Listing 3:

```
1  function plot_asn5q2a()
2      % Run simulation
3      simOut = sim('stiffness');
4
5      % Take output values from Simulink
6      theta1=simOut.get('theta1');
7      theta2=simOut.get('theta2');
8
```

```
    % Plots theta1
    figure;
    hold on;
    view(2);
    title('Plot for $\theta_{1}$', 'Interpreter', 'latex');
    xlabel('Time (seconds)', 'Interpreter', 'latex');
    ylabel('$\theta_{1}$', 'Interpreter', 'latex');
    plot(theta1, 'm');
    saveas(gcf, 'q2a_t1.fig'); % saves figure as .fig
    saveas(gcf, 'q2a_t1', 'epsc'); % saves figure as .eps (for preparing text)

    % Plots theta2
    figure;
    hold on;
    view(2);
    title('Plot for $\theta_{2}$', 'Interpreter', 'latex');
    xlabel('Time (seconds)', 'Interpreter', 'latex');
    ylabel('$\theta_{2}$', 'Interpreter', 'latex');
    plot(theta2, 'm');
    saveas(gcf, 'q2a_t2.fig'); % saves figure as .fig
    saveas(gcf, 'q2a_t2', 'epsc'); % saves figure as .eps (for preparing text)
end
```

Listing 3: MATLAB code used to simulate the system for question 2(a).

2.1.3 Simulation Results

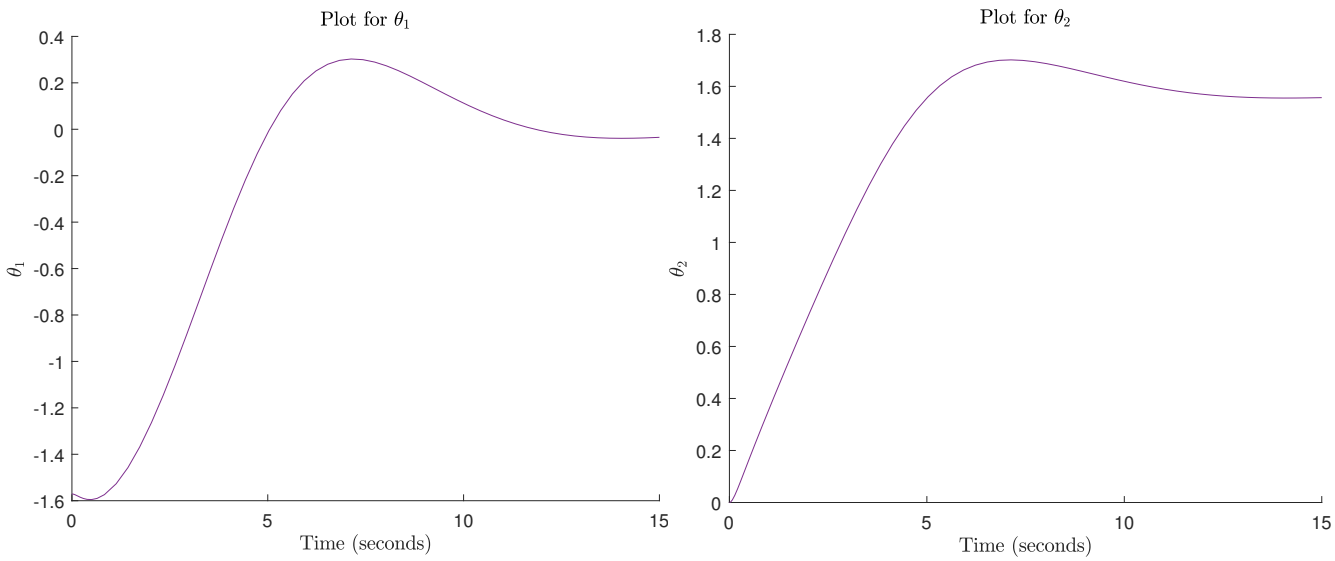

Figure 15: Joint angle plots for $x(0) = \begin{bmatrix} -\frac{\pi}{2} & 0 & 0 & 0 \end{bmatrix}^T$, $K_p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $K_v = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$, $\boldsymbol{q}_d = \begin{bmatrix} 0 & \frac{\pi}{2} \end{bmatrix}^T$

2.1.4 Comments on Simulation Results

With the PD controller added to the system, we see a more orderly response with no oscillation within the first 30 seconds, resembling more of a critically damped system than before. There is still slight overshoot in the response.

2.2 Task Space Control

The control law for the Task Space Stiffness Controller is:

$$\boldsymbol{u} = G(\boldsymbol{q}) + \underbrace{\underline{J}^T}_{\substack{\text{transpose of} \\ \text{manipulator} \\ \text{Jacobian}}} (\boldsymbol{q}) \begin{bmatrix} \underline{\boldsymbol{f}}_n \\ \underline{\boldsymbol{\tau}}_n \end{bmatrix},$$

where

$$\begin{bmatrix} \underline{\boldsymbol{f}}_n \\ \underline{\boldsymbol{\tau}}_n \end{bmatrix} = K_P \begin{bmatrix} \underset{\sim}{\boldsymbol{o}}_d - \underset{\sim}{\boldsymbol{o}}_n \\ \underline{\boldsymbol{e}} \end{bmatrix} + K_V \begin{bmatrix} \dot{\underset{\sim}{\boldsymbol{o}}}_d - \dot{\underset{\sim}{\boldsymbol{o}}}_n \\ \underline{\boldsymbol{\omega}}_d - \underline{\boldsymbol{\omega}}_n \end{bmatrix}, \tag{2}$$

and

$$\underset{\sim}{\boldsymbol{o}}_n = \underset{\sim}{\boldsymbol{o}}_2(\theta) = \begin{bmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

For a two-link planar RR manipulator, we have that:

$$\begin{bmatrix} \underset{\sim}{\boldsymbol{o}}_2 \\ \theta \end{bmatrix} = \underbrace{\begin{bmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \\ \theta_1 + \theta_2 \end{bmatrix}}_{\boldsymbol{f}(\boldsymbol{q})}$$

Differentiating this expression w.r.t time gives:

$$\begin{aligned} \begin{bmatrix} \dot{\underset{\sim}{\boldsymbol{o}}}_2 \\ \omega \end{bmatrix} &= \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} \\ &= \underbrace{\begin{bmatrix} -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) \\ 1 & 1 \end{bmatrix}}_{\text{Jacobian } \underline{J}} \dot{\boldsymbol{q}}, \end{aligned}$$

from which we can get:

$$\dot{\underset{\sim}{\boldsymbol{o}}}_n = \dot{\underset{\sim}{\boldsymbol{o}}}_2 = \begin{bmatrix} [-l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2)]\dot{\theta}_1 + [-l_2 \sin(\theta_1 + \theta_2)]\dot{\theta}_2 \\ [l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)]\dot{\theta}_1 + [l_2 \cos(\theta_1 + \theta_2)]\dot{\theta}_2 \end{bmatrix}$$

$$\underline{\boldsymbol{\omega}}_n = \underline{\boldsymbol{\omega}}_2 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

However, as this is a 2 DOF manipulator, we can rewrite Equation 2 as:

$$\begin{bmatrix} \underline{\boldsymbol{f}}_n \\ \underline{\boldsymbol{\tau}}_n \end{bmatrix} = K_P \begin{bmatrix} \underset{\sim}{\boldsymbol{o}}_d - \underset{\sim}{\boldsymbol{o}}_n \end{bmatrix} + K_V \begin{bmatrix} \dot{\underset{\sim}{\boldsymbol{o}}}_d - \dot{\underset{\sim}{\boldsymbol{o}}}_n \end{bmatrix},$$

which makes the control law:

$$\begin{aligned}
\boldsymbol{u} &= G(\boldsymbol{q}) + \underline{J}^T(\boldsymbol{q}) \begin{bmatrix} \underline{\boldsymbol{f}}_n \\ \underline{\boldsymbol{\tau}}_n \end{bmatrix} \\
&= G(\boldsymbol{q}) + \underline{J}^T(\boldsymbol{q})(K_P \left[\underset{\sim}{\boldsymbol{o}}_d - \underset{\sim}{\boldsymbol{o}}_2 \right] + K_V \left[\dot{\underset{\sim}{\boldsymbol{o}}}_d - \dot{\underset{\sim}{\boldsymbol{o}}}_2 \right]) \\
&= G(\boldsymbol{q}) + \underline{J}^T(\boldsymbol{q})(K_P \begin{bmatrix} o_{d_1} - l_1\cos(\theta_1) - l_2\cos(\theta_1+\theta_2) \\ o_{d_2} - l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2) \end{bmatrix} \\
&+ K_V \begin{bmatrix} \dot{o}_{d_1} - ([-l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2)]\dot{\theta}_1 + [-l_2\sin(\theta_1+\theta_2)]\dot{\theta}_2) \\ \dot{o}_{d_2} - ([l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2)]\dot{\theta}_1 + [l_2\cos(\theta_1+\theta_2)]\dot{\theta}_2) \end{bmatrix}),
\end{aligned}$$

and the Jacobian:

$$\underline{J} = \begin{bmatrix} -l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2) \\ l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2) \end{bmatrix}$$

2.2.1 Simulink Block Diagrams

The stiffness controller was implementated as shown in Figure 16, which was then connected to the robot as shown in Figure 17:

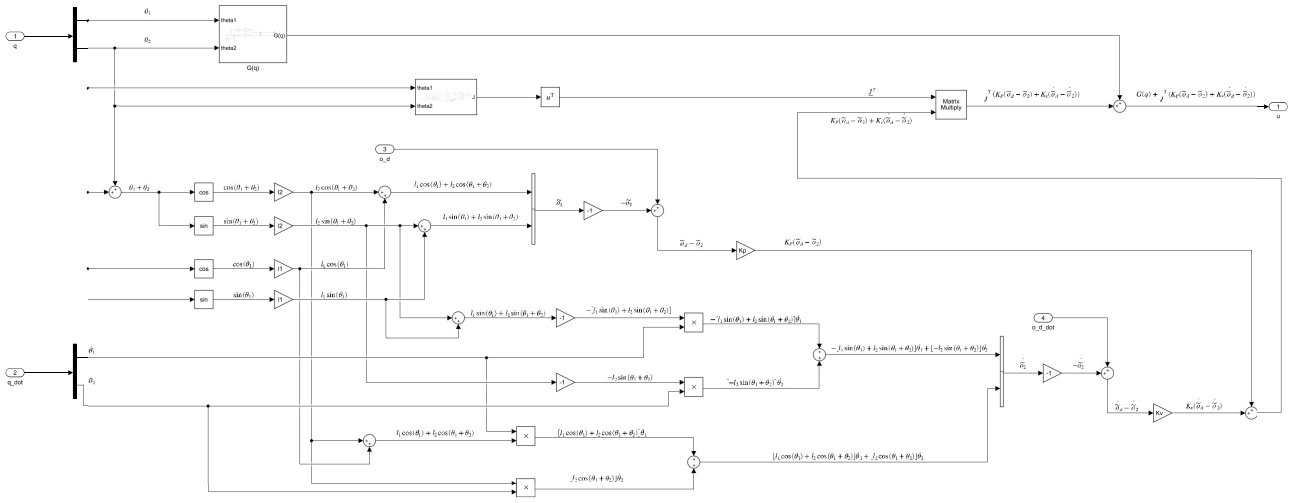

Figure 16: Simulink Block Diagram for the Task Space Stiffness Controller.

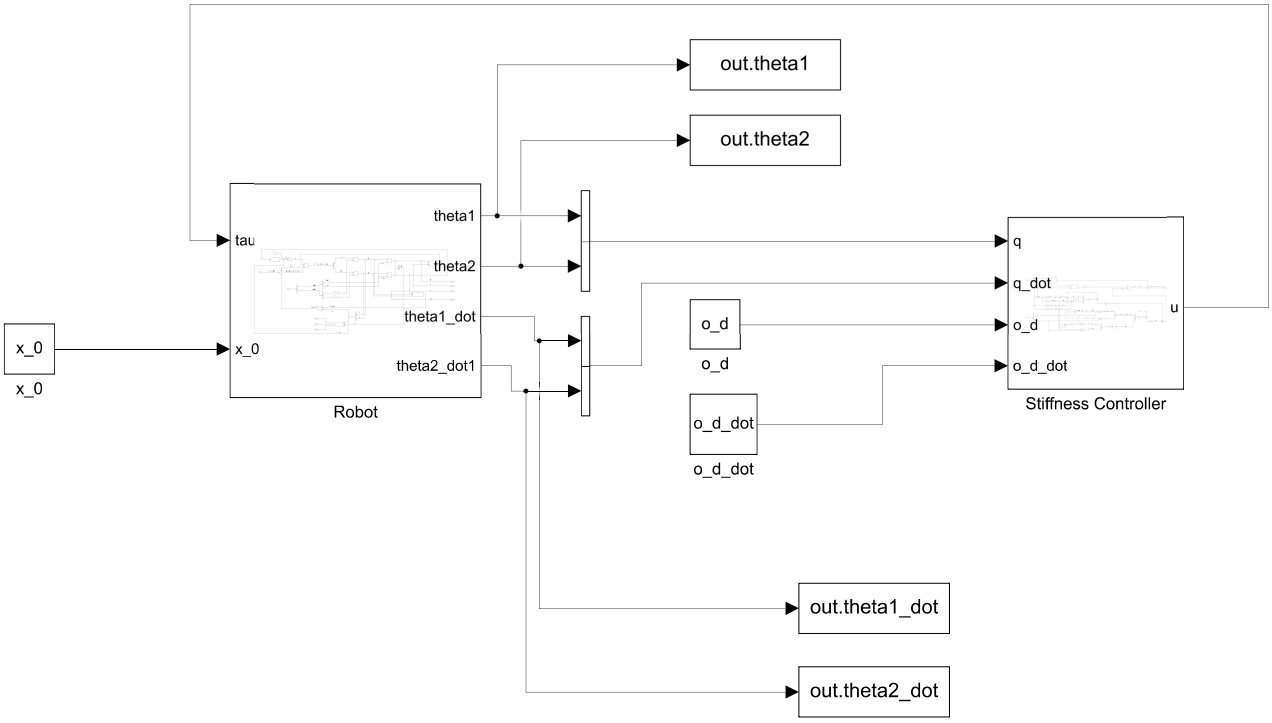

Figure 17: Simulink Block Diagram for the overall system.

2.2.2 MATLAB Code

A MATLAB function was written to automate simulation and saves the plots as `.fig` (and `.pdf`) files. The code is shown in Listing 4:

```
1  function plot_stiffness()
2      % Run simulation
3      simOut = sim('stiffness');
4
5      % Take output values from Simulink
6      theta1=simOut.get('theta1');
7      theta2=simOut.get('theta2');
8
```

```
    % Plots theta1
    figure;
    hold on;
    view(2);
    title('Plot for $\theta_{1}$', 'Interpreter', 'latex');
    xlabel('Time (seconds)', 'Interpreter', 'latex');
    ylabel('$\theta_{1}$', 'Interpreter', 'latex');
    plot(theta1, 'Color', '#D95319');
    saveas(gcf, 'q2b_t1_kp3.fig'); % saves figure as .fig
    saveas(gcf, 'q2b_t1_kp3', 'epsc'); % saves figure as .eps (for preparing text)

    % Plots theta2
    figure;
    hold on;
    view(2);
    title('Plot for $\theta_{2}$', 'Interpreter', 'latex');
    xlabel('Time (seconds)', 'Interpreter', 'latex');
    ylabel('$\theta_{2}$', 'Interpreter', 'latex');
    plot(theta2, 'Color', '#D95319');
    saveas(gcf, 'q2b_t2_kp3.fig'); % saves figure as .fig
    saveas(gcf, 'q2b_t2_kp3', 'epsc'); % saves figure as .eps (for preparing text)
end
```

Listing 4: MATLAB code used to simulate the system for question 2(b).

2.2.3 Simulation Results

The following simulations all use:

$$x(0) = \begin{bmatrix} -\frac{\pi}{2} & 0 & 0 & 0 \end{bmatrix}^T$$

$$K_v = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$

$$\underset{\sim}{o}_d = \underset{\sim}{o}_0 + \underline{C}_0 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

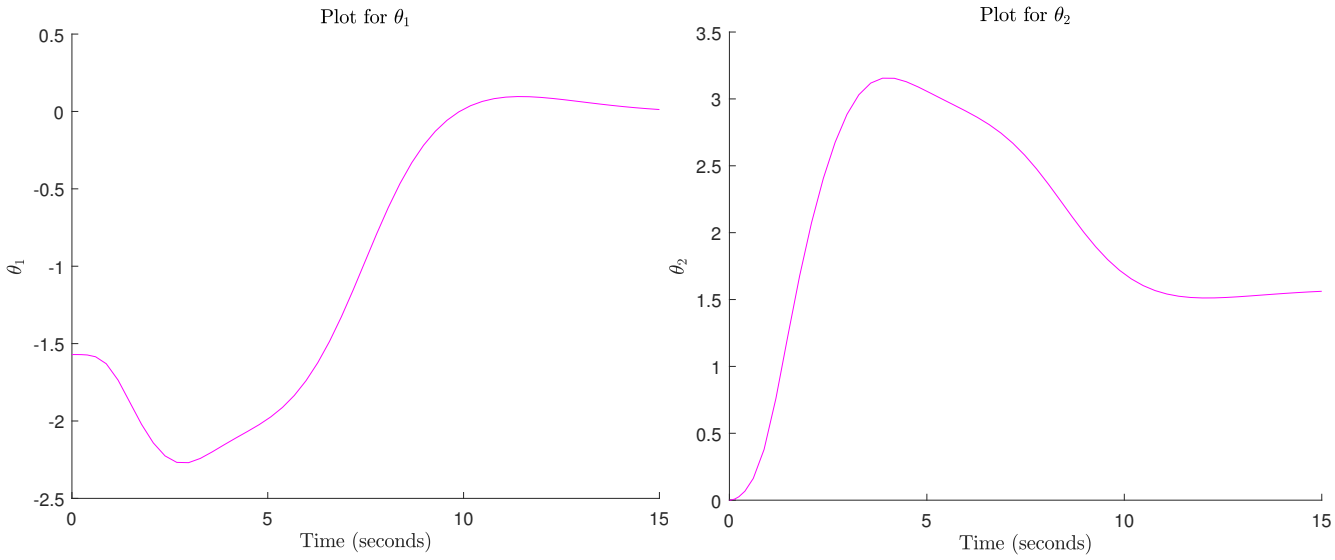

Figure 18: Trajectory for $K_p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$,

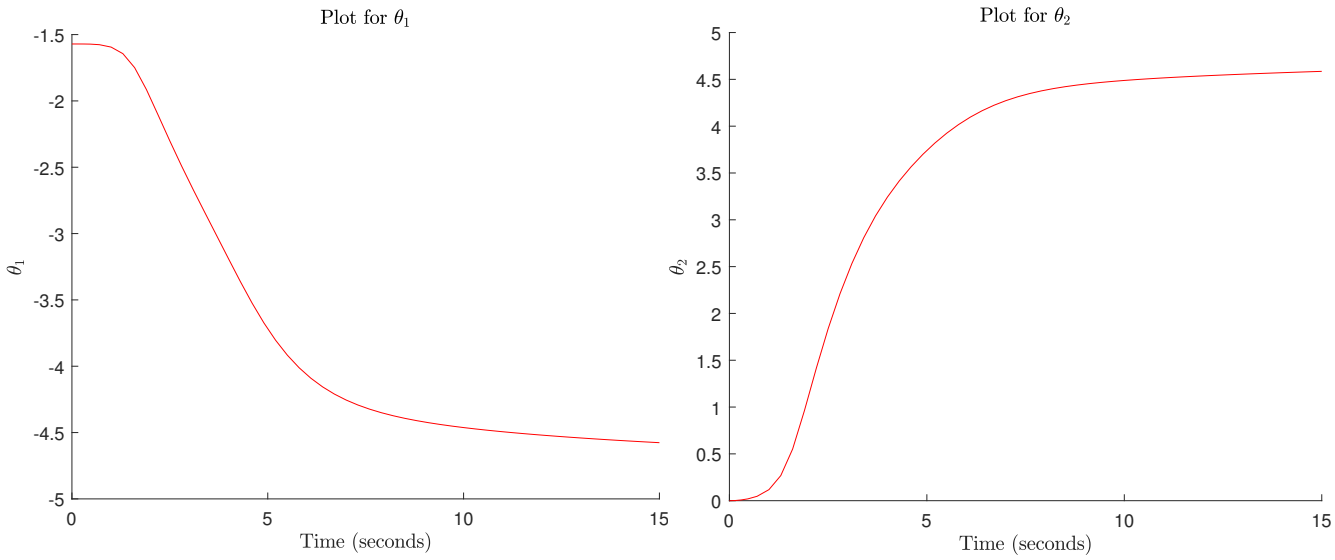

Figure 19: Trajectory for $K_p = \begin{bmatrix} 0.2 & 0 \\ 0 & 1 \end{bmatrix}$

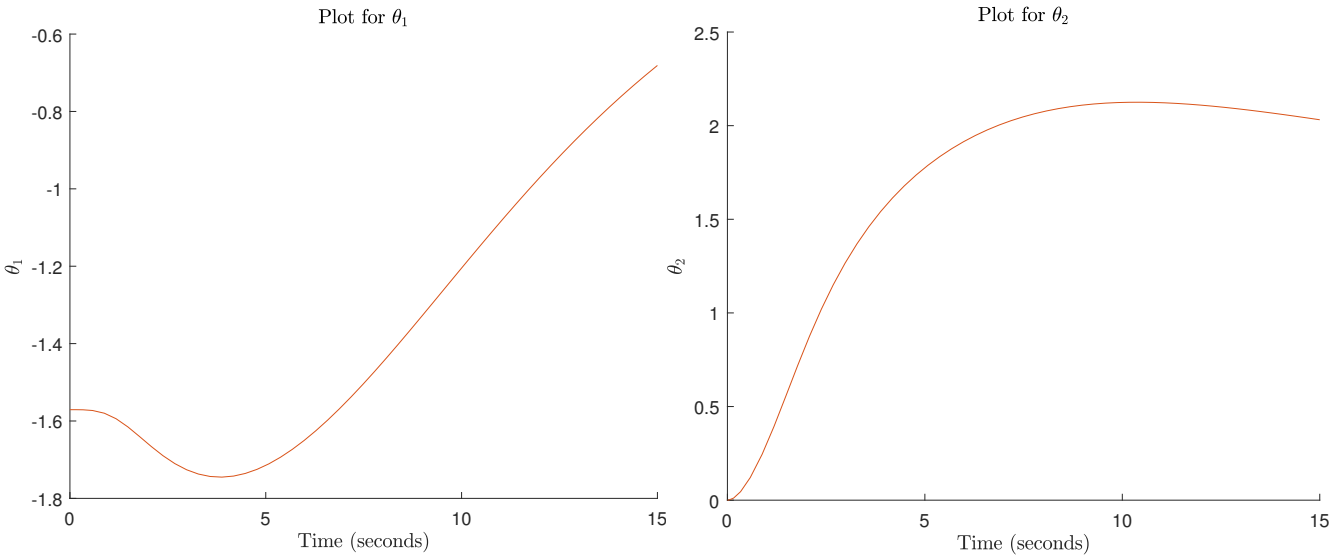

Figure 20: Trajectory for $K_p = \begin{bmatrix} 1 & 0 \\ 0 & 0.2 \end{bmatrix}$

2.2.4 Comments on Simulation Results

The graphs for Kp1 return a similar result as the PD Gravity Controller, this makes sense as K_{P_1} is the 1×1 identity matrix.

The graphs for K_{P_2} show that θ_1 is decreasing over time, and θ_2 shows a critically damped response.

The graphs for K_{P_3} shows θ_1 increasing over time, and θ_2 reaching a peak at around 9 seconds before slowly decreasing. This shows the difference in joint angle velocities based on the emulated springs and dampers from the different K_P gain matrices.