AME 556
Team 8 Final Project Report

University of Southern California

Fall 2024

Team Members:

Tianpai Le (tianpail@usc.edu, 6290177943)

Haibo Ma (haiboma@usc.edu, 2551570578)

Shuai Zhao (szhao321@usc.edu, 7722927131)

Team 8
Tianpai Le, Haibo Ma, Shuai Zhao

Table of Contents

Team 8
Tianpai Le, Haibo Ma, Shuai Zhao

Task 1: Simulation and physical constraints

This task aims to satisfy the given constraints from the problem statement, including the joint angles, velocities, and torques.

Constraint #1: Joint Angle

To set limits for the joint angles, we checked the limit boxes when setting up revolute joints representing hips and knees (Figure 1.1.1) so the legs wouldn't extend over the angle limits.

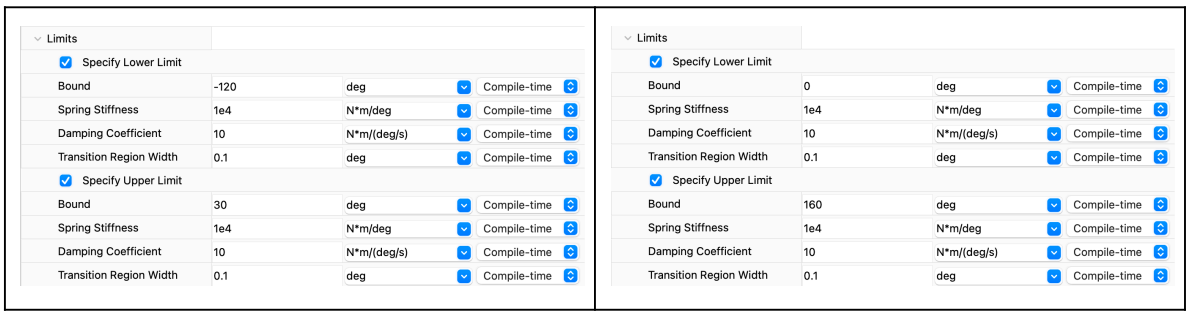


Figure 1.1.1: Joint Angle Limits (left: hip, right: knee)

In addition, a "MATLAB Function" block was set up to read the joint angles from the sensors (Figure 1.1.2). If the constraint is violated, the simulation will be terminated with an error message displayed.

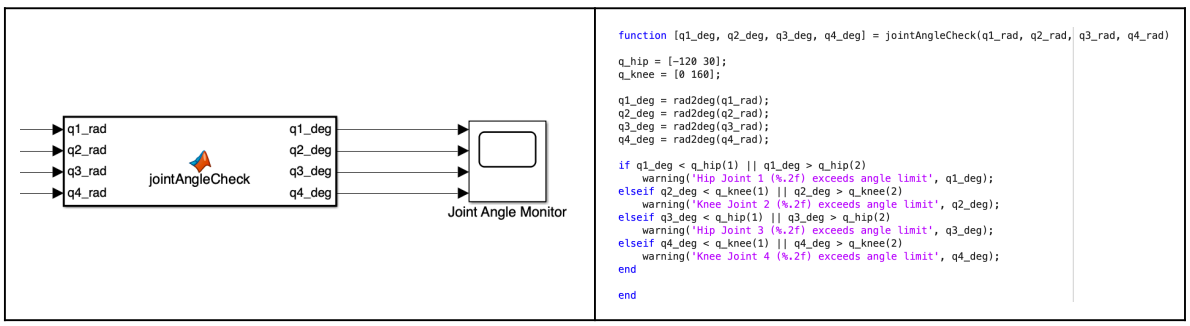
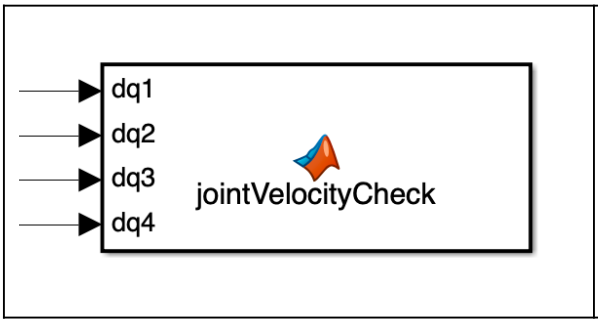


Figure 1.1.2: Error Function - Joint angle

Constraint #2: Joint Velocity

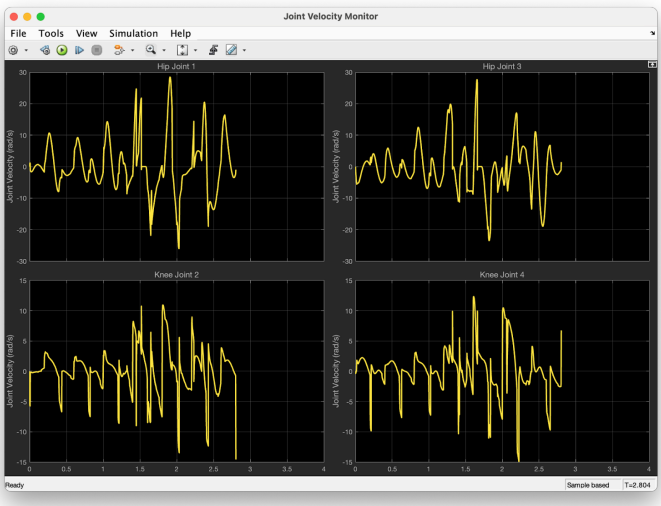
Since no saturation method is available for the joint velocities, we could only set up a "MATLAB Function" block similar to joint angles (Figure 1.2.1).




```
function jointVelocityCheck(dq1, dq2, dq3, dq4)

dq_hip = [-30 30];
dq_knee = [-15 15];

if dq1 < dq_hip(1) || dq1 > dq_hip(2)
    error('Hip Joint 1 (%.2f) exceeds velocity limit', dq1);
elseif dq2 < dq_knee(1) || dq2 > dq_knee(2)
    error('Knee Joint 2 (%.2f) exceeds velocity limit', dq2);
elseif dq3 < dq_hip(1) || dq3 > dq_hip(2)
    error('Hip Joint 3 (%.2f) exceeds velocity limit', dq3);
elseif dq4 < dq_knee(1) || dq4 > dq_knee(2)
    error('Knee Joint 4 (%.2f) exceeds velocity limit', dq4);
end

end
```

Figure 1.2.1: Error Function - Joint velocity

Below is an example of if any of the constraints is violated (Figure 1.2.2): when the velocity of knee joint 2 exceeds -15 rad/s, the simulation stops and displays an error flag. The same situation will also apply to the other two constraints.

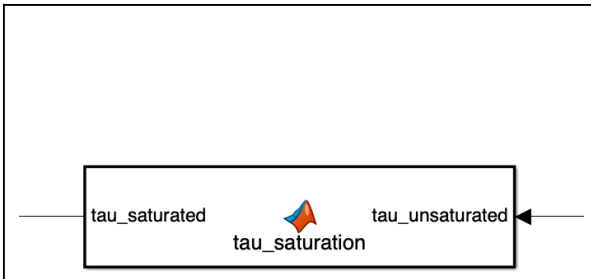



```
An error occurred during simulation and the simulation was terminated
Caused by:
  • Knee Joint 2 (-15.00) exceeds velocity limit
          Error in 'AME556_Project_Task4_v5/2D Biped Robot /MATLAB Function' (line 9)
                  error('Knee Joint 2 (%.2f) exceeds velocity limit', dq2);
Component: Simulink | Category: Block error
```

Figure 1.2.2: An Error Condition - Joint velocity

Constraint #3: Joint Torque

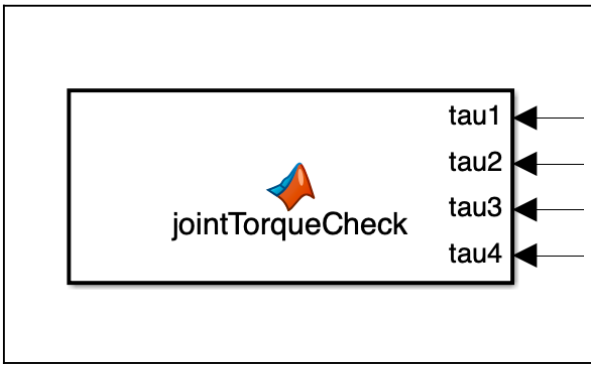
The joint torques are determined by the forces from single rigid body dynamics and the Jacobian function, which makes them susceptible to exceeding limits. Therefore, a saturation function would be needed between the calculated torque and the input port (Figure 1.3.1). When torques exceed limits, they will be saturated before being applied to the corresponding joint.




```
function tau_saturated = tau_saturation(tau_unsaturated)

tau_limits = [30, 60, 30, 60];
tau_saturated = tau_unsaturated;

for i = 1:length(tau_unsaturated)
    if tau_unsaturated(i) > tau_limits(i)
        tau_saturated(i) = tau_limits(i);
        warning('Tau %d (%.2f) is saturated to %d',...
            i, tau_unsaturated(i), tau_limits(i));
    elseif tau_unsaturated(i) < -tau_limits(i)
        tau_saturated(i) = -tau_limits(i);
        warning('Tau %d (%.2f) is saturated to -%d',...
            i, tau_unsaturated(i), tau_limits(i));
    end
end

end
```

Figure 1.3.1: Saturation Function - Joint torque

The error function is similar to the previous two constraints (Figure 1.3.2).




```
function jointTorqueCheck(tau1, tau2, tau3, tau4)

tau_hip = [-30 30];
tau_knee = [-60 60];

if tau1 < tau_hip(1) || tau1 > tau_hip(2)
    error('Hip Joint 1 (%.2f) exceeds torque limit', tau1);
elseif tau2 < tau_knee(1) || tau2 > tau_knee(2)
    error('Knee Joint 2 (%.2f) exceeds torque limit', tau2);
elseif tau3 < tau_hip(1) || tau3 > tau_hip(2)
    error('Hip Joint 3 (%.2f) exceeds torque limit', tau3);
elseif tau4 < tau_knee(1) || tau4 > tau_knee(2)
    error('Knee Joint 4 (%.2f) exceeds torque limit', tau4);
end

end
```

Figure 1.3.2: Error Function - Joint torque

Task 2: Standing and walking

The goal of this task includes three robot motions:

- Standing up and down: Control y-position of robot COM from 0.45m to 0.55m and then from 0.55m to 0.4m
- Walking forward with a speed of ≥ 0.5m/s for 5 seconds
- Walking backward with a speed of ≥ 0.5m/s for 5 seconds

Standing up and down

System Setup

The system applies a regular MPC controller, with 10 horizons and a sampling time of 0.04 seconds (Table 2.1.1).

Table 2.1.1: Control Algorithm Setup - Task 2 Standing up and down

Controller	MPC
Number of horizons, N	10
Sampling time, dt	0.04 s

The program's total simulation time is 1.5 seconds. From the start to 0.5 seconds, the robot will stand up from 0.45 m to 0.55 m; from 0.5 seconds to the end, it will crouch from 0.55 m to 0.4 m.

Simulation Result

The y-direction displacement of the robot is shown in Figure 2.1.1. It can be seen the robot reaches the desired heights within the specified time above.

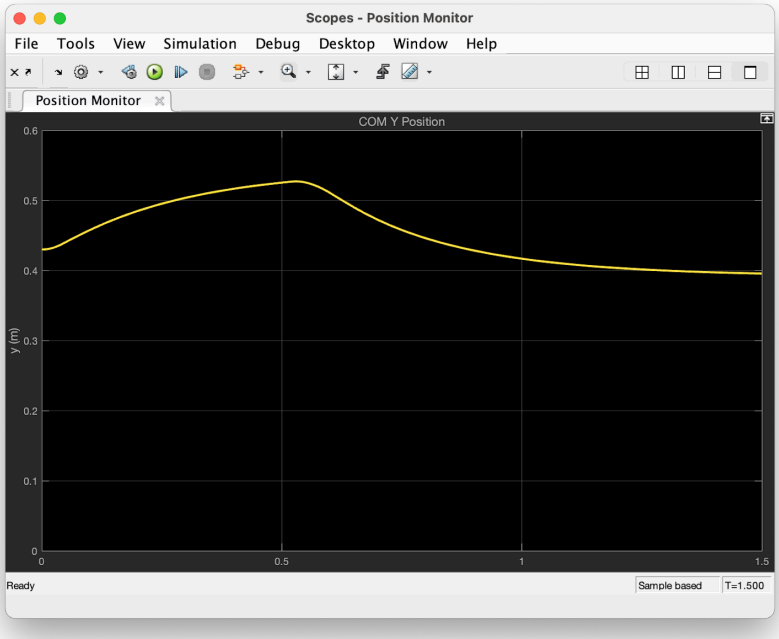


Figure 2.1.1: Position Graph - Task 2 Standing up and down

Animation

Animation link: [Task 2 Standing up and down]

Physical Constraints

Figure 2.1.2 below includes the joint angle, torque, and velocity graphs. All physical constraints are satisfied, and no error flags appeared during the simulation.

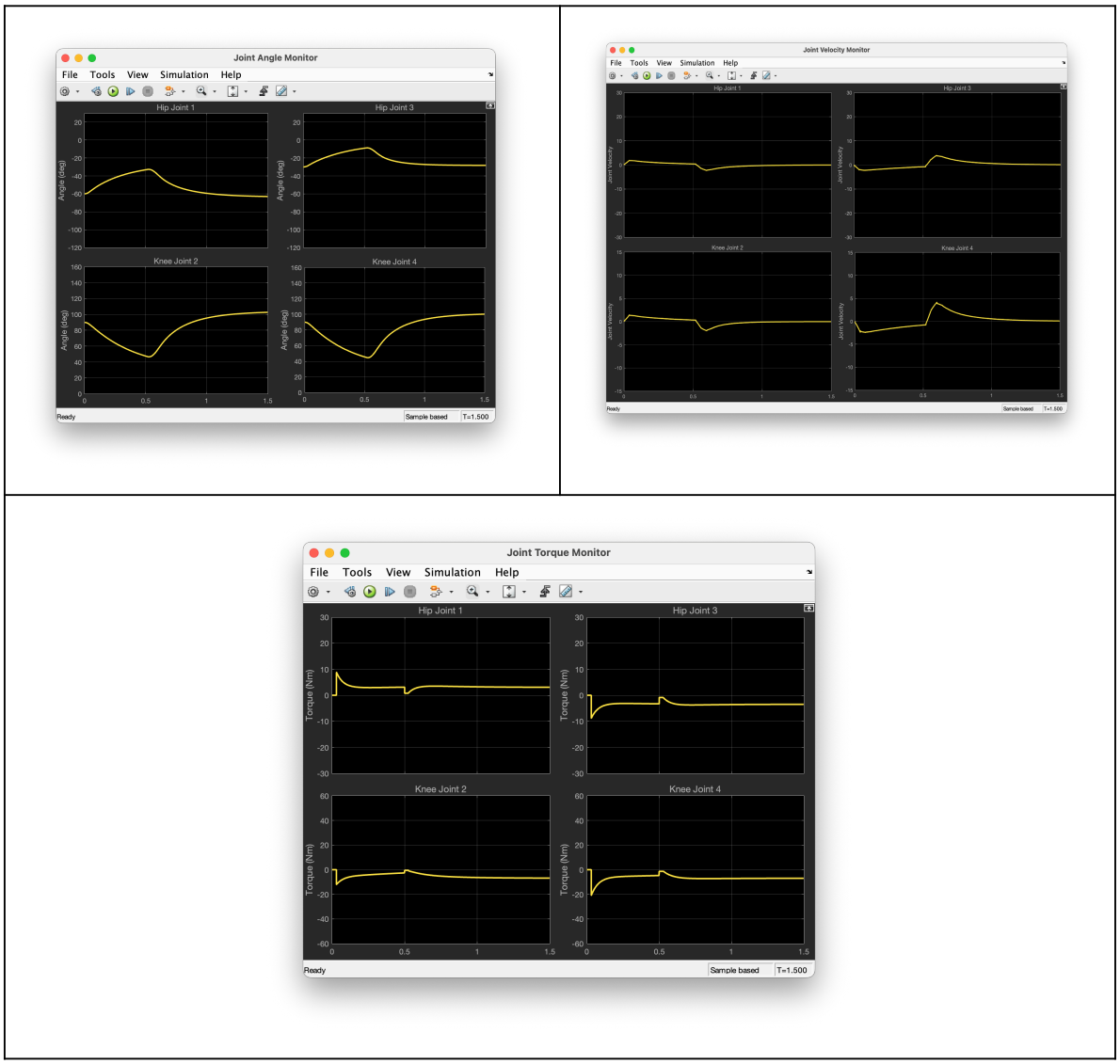


Figure 2.1.2: Physical Constraints - Task 2 Standing up and down

Walking forward

System Setup

The system applies condensed MPC as the control algorithm with a walking gait schedule and regular swing foot policy. See Table 2.2.1 below for more details.

Table 2.2.1: Control Algorithm Setup - Task 2 walking forward

Controller	Condensed MPC
Number of horizons, N	10
Sampling time, dt	0.04 s
Gait schedule (at phase = 0)	Walking σ_L: [1 1 1 1 1 0 0 0 0 0] σ_R: [0 0 0 0 0 1 1 1 1 1]
Swing foot policy	$p_{foot,x}^{des} = p_{c,x} + \frac{1}{2}\Delta t \cdot dp_{c,x}$ $p_{foot,y}^{des} = h_{swing} sin\left(\frac{t_s}{\Delta t}\pi\right)$

For the physical setup (Figure 2.2.1), the robot starts between the red and gray zones. The gray zone indicates a 2.5-m track. The total simulation time is 6 seconds.

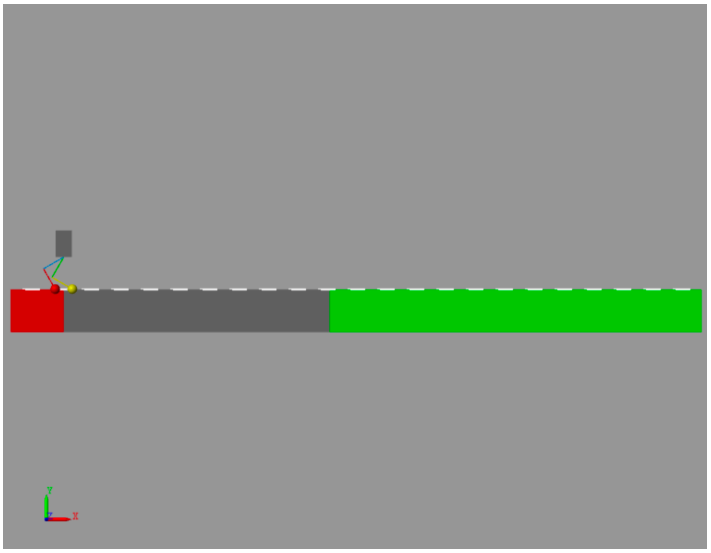


Figure 2.2.1: Physical Setup - Task 2 walking forward

Simulation Result

The graph below (Figure 2.2.2) shows the robot's x-direction displacement and velocity. It can be seen the velocity exceeds 0.5 m/s before 1 second, and stays above that until the end of the simulation (6 s), which illustrates the speed stays above 0.5 m/s for at least 5 seconds.

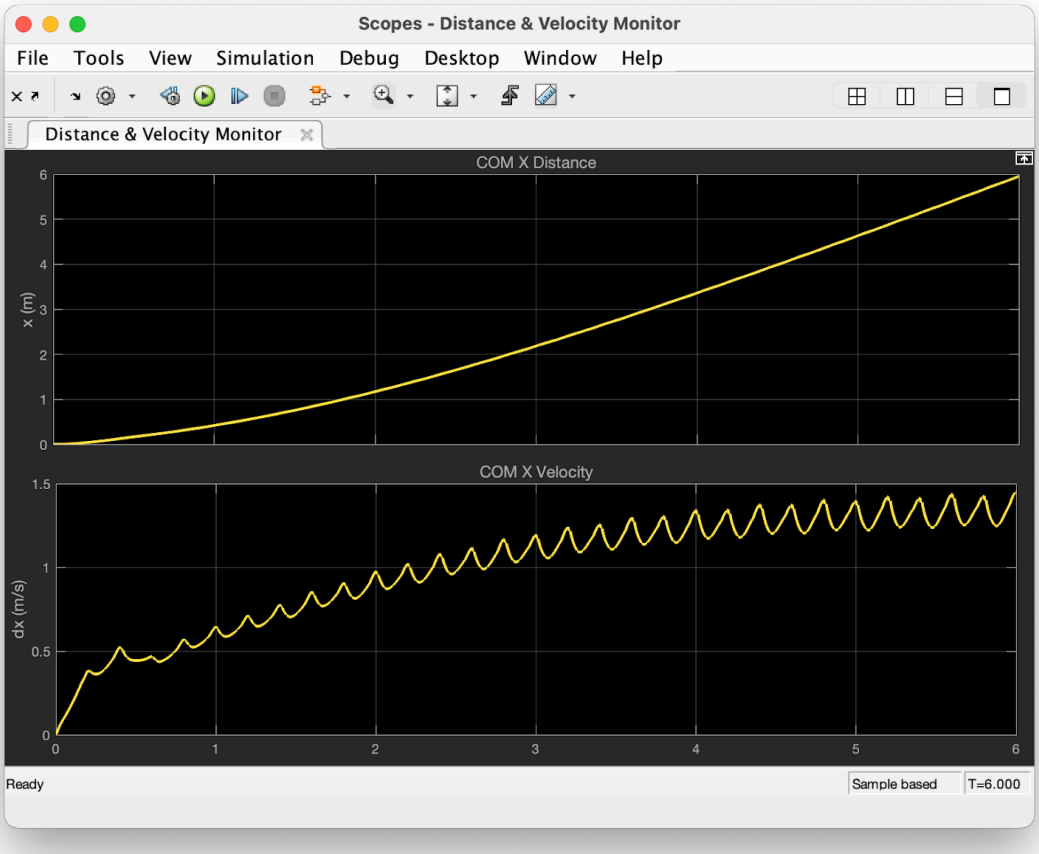


Figure 2.2.2: Distance & Velocity Graph - Task 2 walking forward

Animation

Animation link: [Task 2 walking forward]

Physical Constraints

Figure 2.2.3 below includes the joint angle, torque, and velocity graphs. All physical constraints are satisfied, and no error flags appeared during the simulation.

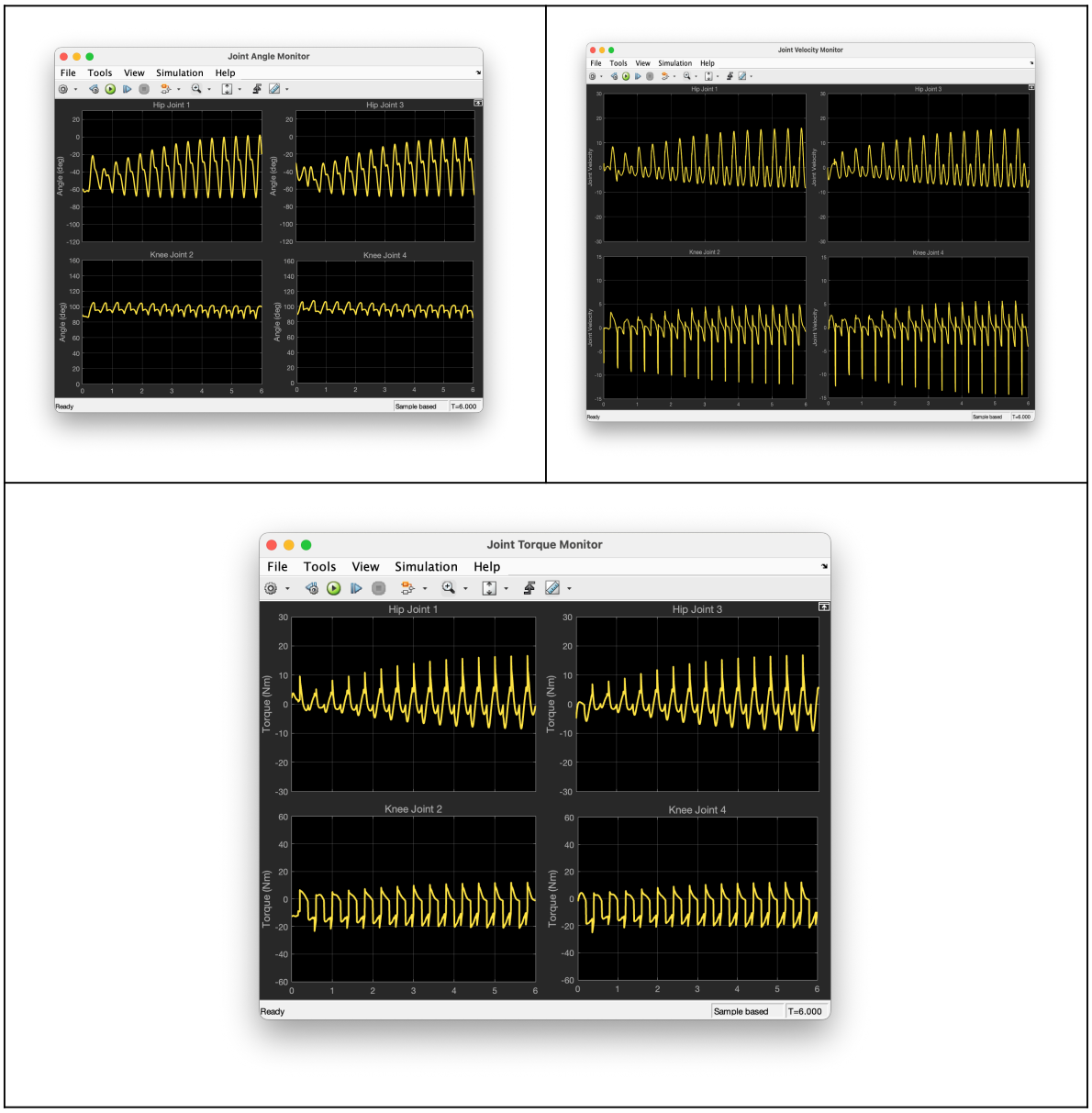


Figure 2.2.3: Physical Constraints - Task 2: Walking forward

Walking backward

System Setup

Overall, the setup is similar to the walking forward condition. However, to better balance the robot for negative x-direction walking, the gait schedule is swapped for both feet, which means the left foot will be the first to swing at the start of the simulation. See Table 2.3.1 below for more details.

Table 2.3.1 Control Algorithm Setup - Task 2 walking backward

Controller	Condensed MPC
Number of horizons, N	10
Sampling time, dt	0.04 s
Gait schedule (at phase = 0)	Walking σ_L: [0 0 0 0 0 1 1 1 1 1] σ_R: [1 1 1 1 1 0 0 0 0 0]
Swing foot policy	$p_{foot,x}^{des} = p_{c,x} + \frac{1}{2}\Delta t \cdot dp_{c,x}$ $p_{foot,y}^{des} = h_{swing} sin\left(\frac{t_s}{\Delta t}\pi\right)$

For the physical setup (Figure 2.3.1), the robot starts between the red and gray zones. The gray zone indicates a 2.5-m track. The total simulation time is 5.5 seconds.

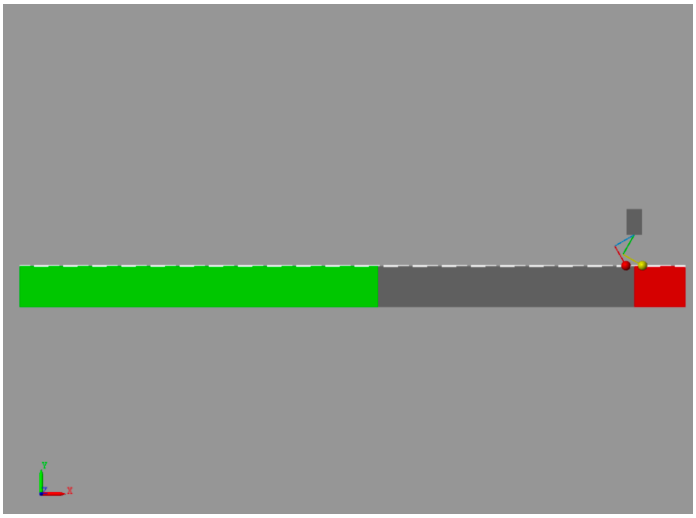


Figure 2.3.1: Physical Setup - Task 2 walking backward

Simulation Result

The distance & velocity graph (Figure 2.3.2) shows that the velocity exceeds -0.5 m/s before 0.5 seconds and stays above that until the end of the simulation (5.5 s). So, the speed is ≥0.5 m/s (in -x direction) for at least 5 seconds.

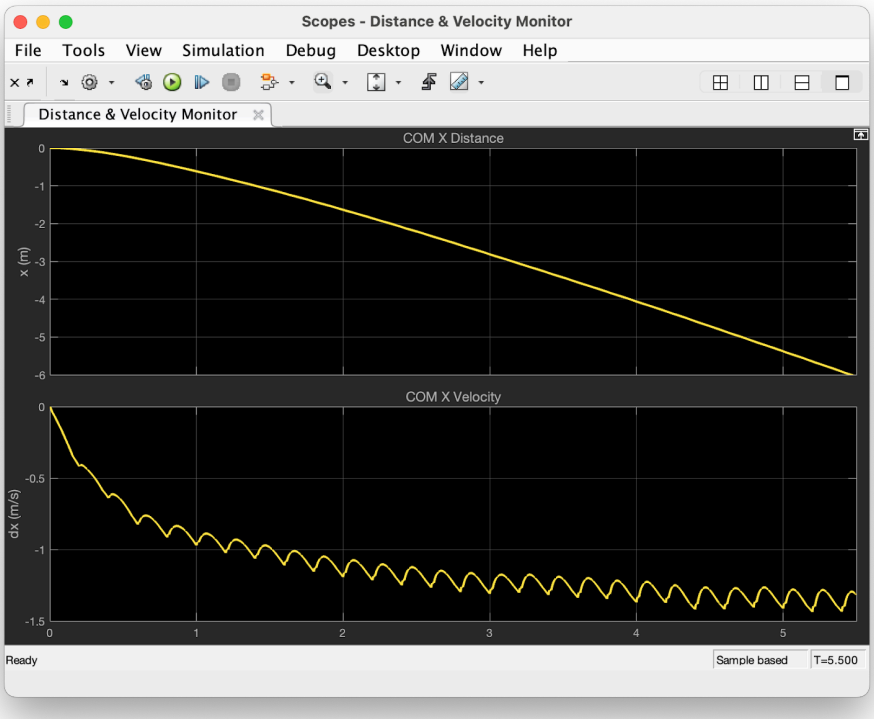


Figure 2.3.2: Distance & Velocity Graph - Task 2 walking backward

The top graph shows the robot's x-direction displacement and the bottom graph shows the robot's x-direction velocity.

Animation

Animation link: [Task 2 walking backward]

Physical Constraints

Figure 2.3.3 below includes the joint angle, torque, and velocity graphs. All physical constraints are satisfied, and no error flags appeared during the simulation.

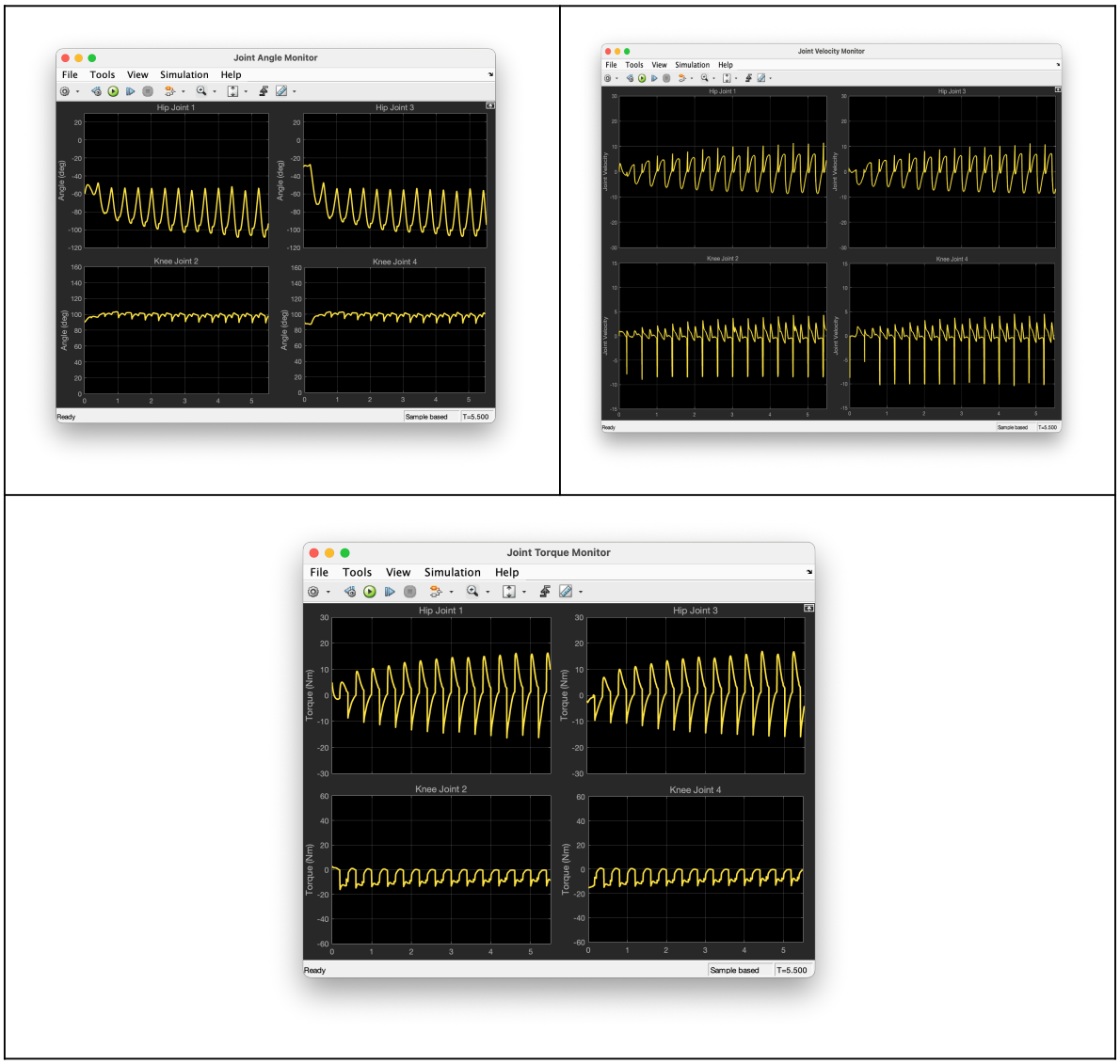


Figure 2.3.3: Physical Constraints - Task 2 walking backward

Task 3: Running on flat ground

This task aims to run on flat ground for 10 meters, with a flight phase in the gait schedule.

System Setup

The system applies a condensed MPC controller. The main feature of the control algorithm is the gait phase, which contains two phases where both feet are off the ground. Other aspects remain the same as the regular walking setup. See Table 3.1 below for more details.

Table 3.1: Control Algorithm Setup - Task 3

Controller	Condensed MPC
Number of horizons, N	10
Sampling time, dt	0.04 s
Gait schedule (at phase = 0)	Walking with a flight phase σ_L: [1 1 1 1 0 0 0 0 0 0] σ_R: [0 0 0 0 0 1 1 1 1 0]
Swing foot policy	$p_{foot,x}^{des} = p_{c,x} + \frac{1}{2}\Delta t \cdot dp_{c,x}$ $p_{foot,y}^{des} = h_{swing} sin\left(\frac{t_s}{\Delta t}\pi\right)$

For physical setup (Figure 3.1), the track is set to be 10 m long (gray zone). In addition, a “Stop Simulation” block is included in the model (Figure 3.2). When the robot reaches the finish line, the program will be stopped automatically. Therefore, the total traveling time will be the same as the program’s stop time.

Figure 3.1: Physical Setup - Task 3

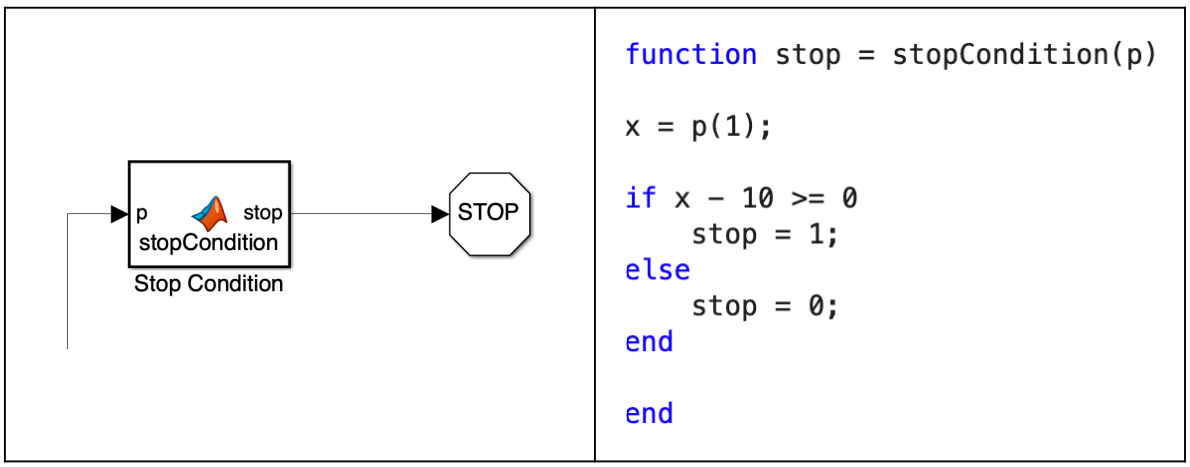


Figure 3.2: Program Stop Condition - Task 3

Simulation Result

The top graph shows the robot's x-direction displacement and the bottom graph shows the robot's x-direction velocity. We implemented a function that stops the simulation once the x-displacement reaches 10m, and the robot completes the task at **9.753 seconds**.

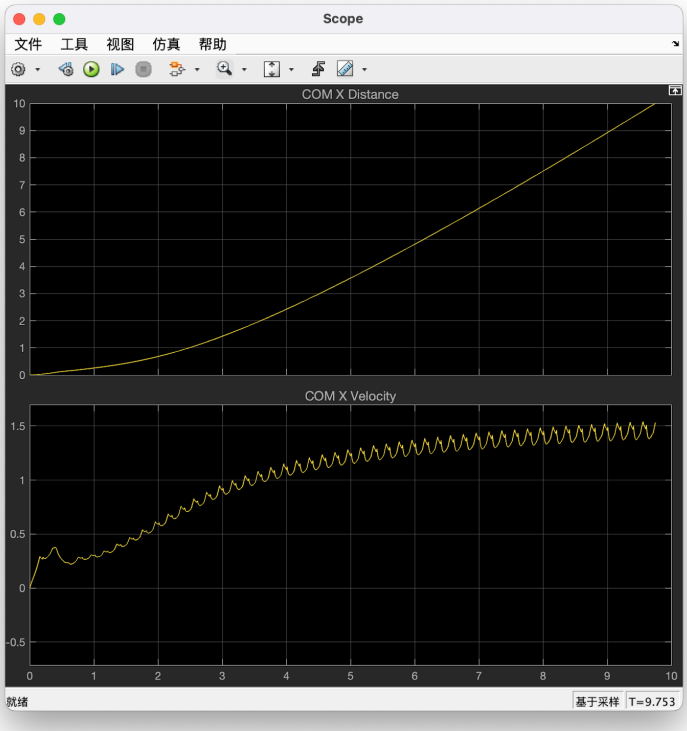


Figure 3.3: Distance & Velocity Graph - Task 3

Animation

Animation link: [Task 3 Running on flat ground]

Physical Constraints

Figure 3.4 below includes the joint angle, torque, and velocity graphs. All physical constraints are satisfied. No error flags appeared during the simulation.

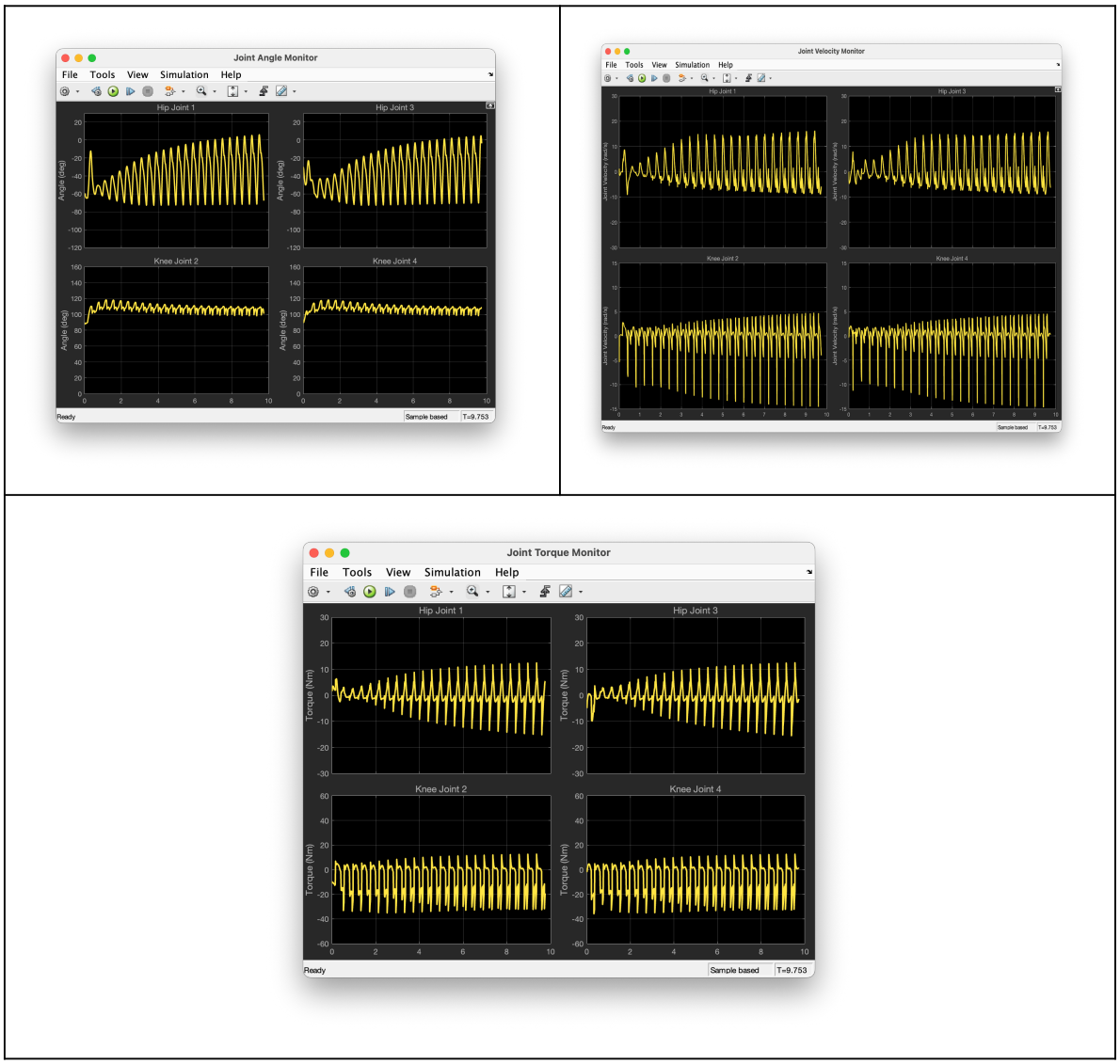


Figure 3.4: Physical Constraints - Task 3

Task 4: Stair climbing

The goal of this task is to control the robot to walk on 5 consecutive stairs, with each stair having a rise of 10 cm and a run of 20 cm.

System Setup

While the gait schedule stays the same as walking, the main feature of this setup is the swing leg control. When climbing stairs, a velocity on the y-direction will be added to the swing leg, driving it to go upwards. In addition, the periodic sine function is modified to make it look closer to a square wave, which would make the robot's non-contact area further from the stairs. See Table 4.1 below for more details.

Table 4.1: Control Algorithm Setup - Task 4

Controller	Condensed MPC
Number of horizons, N	10
Sampling time, dt	0.04 s
Gait schedule (at phase = 0)	Walking σ_L: [1 1 1 1 1 0 0 0 0 0] σ_R: [0 0 0 0 0 1 1 1 1 1]
Swing foot policy	When walking on flat ground: $p_{foot,x}^{des} = p_{c,x} + \frac{1}{2}\Delta t \cdot dp_{c,x}$ $p_{foot,y}^{des} = h_{swing} sin\left(\frac{t_s}{\Delta t}\pi\right)$ When climbing stairs: $p_{foot,x}^{des} = p_{c,x} + \frac{1}{2}\Delta t \cdot dp_{c,x}$ $p_{foot,y}^{des} = p_{foot,y}^{support} + \frac{1}{2}\Delta t \cdot dp_{c,y} + coeff * h_{swing} \sqrt[3]{sin\left(\frac{t_s}{\Delta t}\pi\right)}$ (1)
Note: (1) $coeff$ And h_{swing} might vary from different stages of climbing stairs	

For physical setup (Figure 4.1), the stairs are set up with the robot starting with an offset distance to it, which allows for the robot to have some acceleration time.

For time benefit, we created a “Time Recorder” module (Figure 4.2), which will capture the current time readings when robot COM reaches the start and end of the stairs. The difference between the two readings would be the robot’s actual running time on stairs.

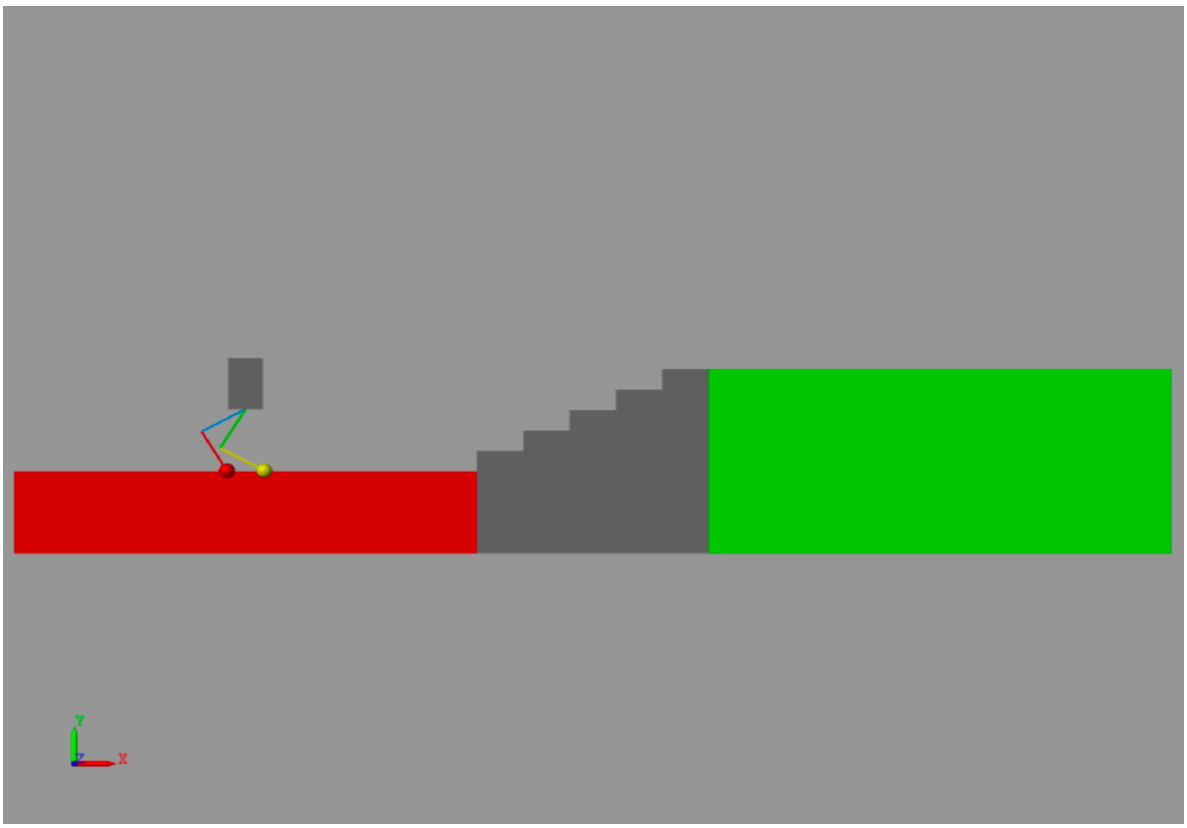


Figure 4.1: Physical Setup - Task 4

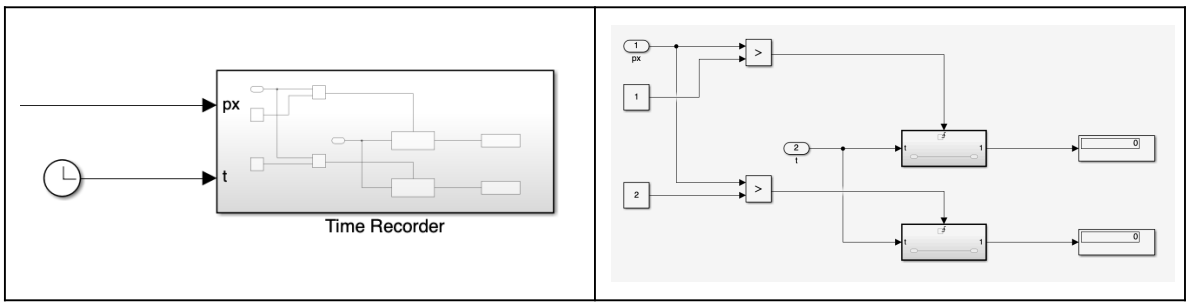


Figure 4.2: Time Recorder Module - Task 4

Simulation Result

The graph below (Figure 4.3) shows the robot's COM in both x and y directions. The climbing stairs motion happens approximately between 1.5 and 2.5 seconds (the exact time can be seen in Figure 4.4).

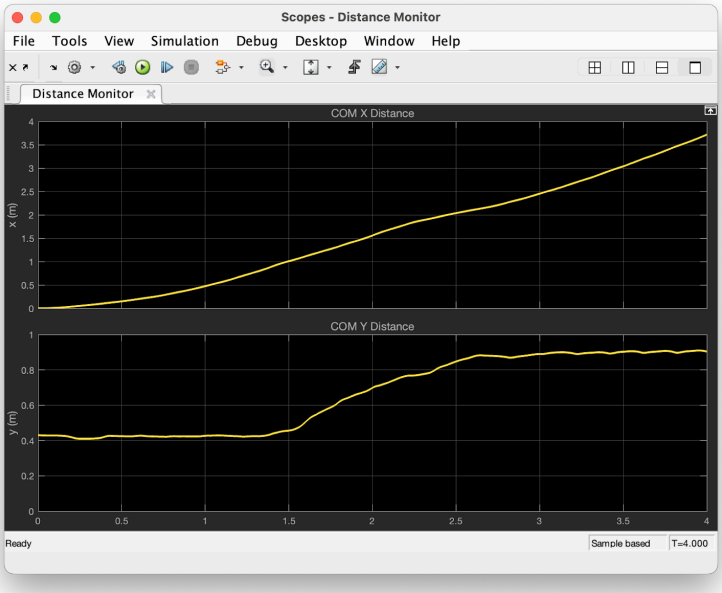


Figure 4.3: Distance Graph - Task 4

According to the time difference between the two displays (Figure 4.4), the total time of climbing stairs is **0.956 seconds**.

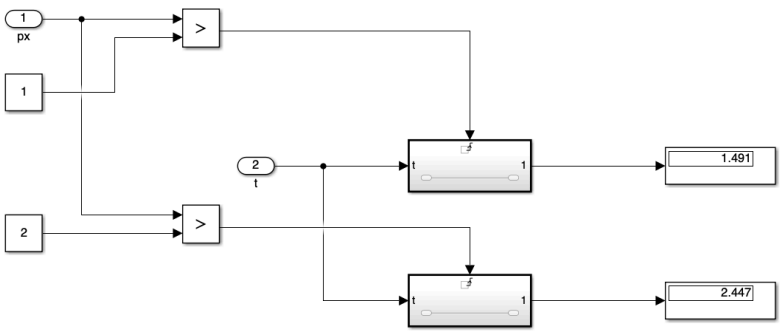


Figure 4.4: Time Display - Task 4

Animation

Animation link: [Task 4 Stair climbing]

Physical Constraints

Figure 4.5 below includes the joint angle, torque, and velocity graphs. It turned out the hip joints might exceed angle limits occasionally despite saturation. So, we set the hip joint upper limits a little bit lower than 30 rad/s (29.7 and 29.6 for 1 and 3). After adjustments, all physical constraints are satisfied without affecting much performance. No error flags appeared during the simulation.

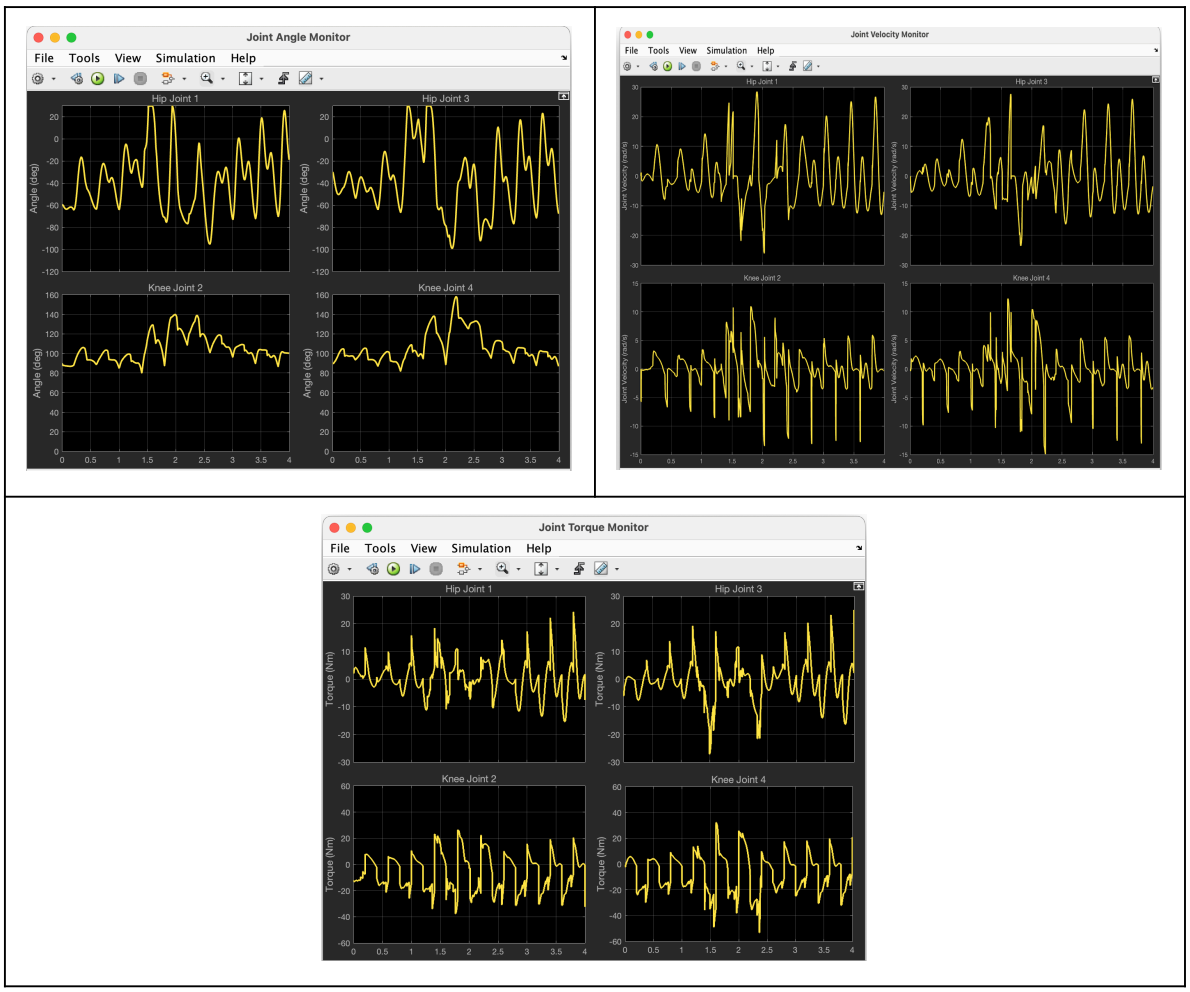


Figure 4.5: Physical Constraints - Task 4

Task 5: Obstacle course

This task aims to control the robot to overcome the obstacle course as shown in Figure 5.1.

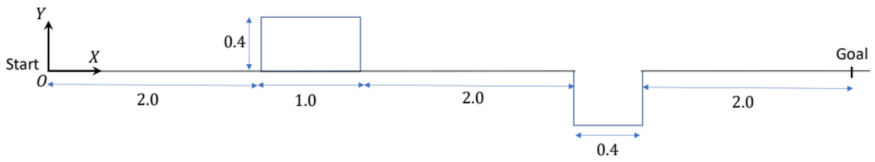


Figure 5.1: Obstacle course setup - Task 5

System Setup

The control algorithm of the system is based on walking, and the main difference is the addition of a jumping gait, which includes the standing/landing phase and flight phase. During a standing/landing phase, both robot's feet would be in contact with the ground. During a flight phase, both feet would be off the ground. See Table 5.1 below for more details.

Table 5.1: Control Algorithm Setup - Task 5

Controller	Condensed MPC
Number of horizons, N	10
Sampling time, dt	0.04 s
Gait schedule (at phase = 0)	Walking: σ_L: [1 1 1 1 1 0 0 0 0 0] σ_R: [0 0 0 0 0 1 1 1 1 1] Jumping (standing/landing): σ_L: [1 1 1 1 1 1 1 1 1 1] σ_R: [1 1 1 1 1 1 1 1 1 1] Jumping (flight): σ_L: [0 0 0 0 0 0 0 0 0 0] σ_R: [0 0 0 0 0 0 0 0 0 0]
Swing foot policy	$p^{des}_{foot,x} = p_{c,x} + coeff * \frac{1}{2}\Delta t \cdot dp_{c,x}$ (2) $p^{des}_{foot,y} = h_{swing} sin\left(\frac{t_s}{\Delta t}\pi\right)$
Note: (2) $coeff$ might vary from different time or distance conditions	

For physical setup, we replicated the course (Figure 5.2) so that the robot could walk on it.

Figure 5.2: Model Setup - Task 5

Simulation Result

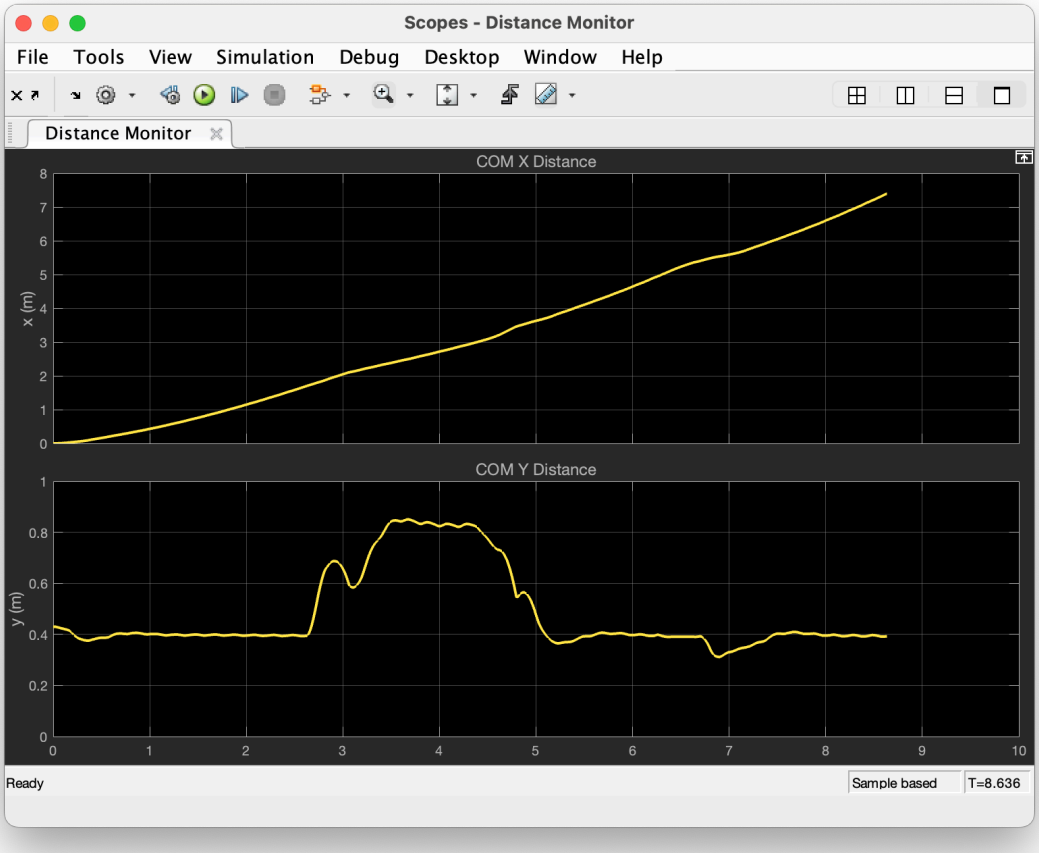


Figure 5.3: Distance Graph - Task 5

The graph above (Figure 5.3) shows the robot's COM in y directions. The first obstacle overcome happens approximately between 3 and 5 seconds, and the robot jumps over the second obstacle (dip) between 6.7 and 7.3 seconds. The total simulation time is **8.636 seconds**.

Animation

Animation link: [Task 5 Obstacle course]

Physical Constraints

In this task, we found it difficult to complete the obstacle course and control the physical constraints at the same time. Therefore, we had our best possible solution that prioritizes finishing the course, despite leaving some points exceeding the physical constraints (Figure 5.4).

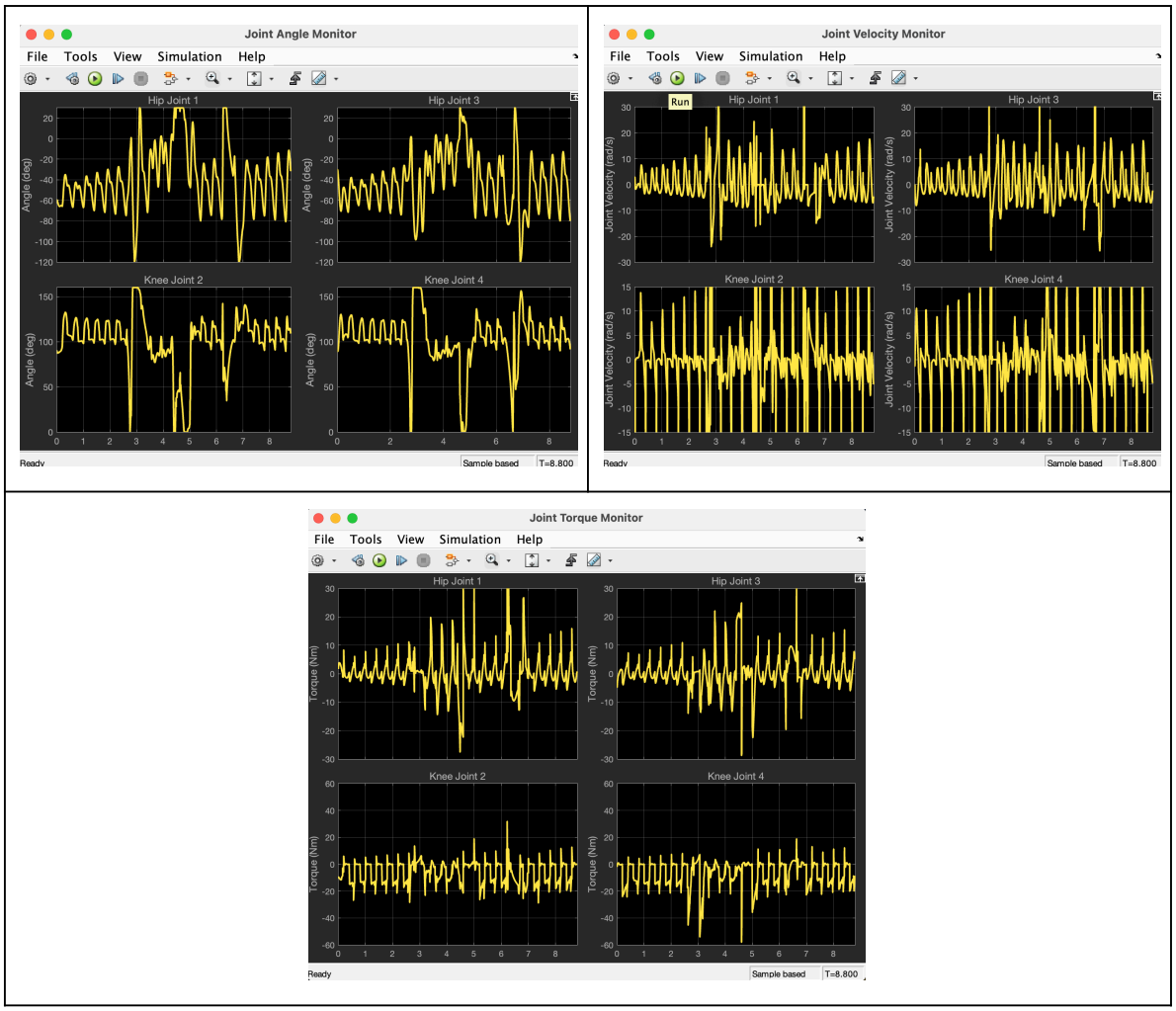


Figure 5.4: Physical Constraints - Task 5

Scoreboard

Below (Table 6.1) are the scores of our team:

Table 6.1: Team 8 Score Table

Task	Time (s)	Scoring Criteria	Score
Task 1	Not applicable	20	20
Task 2	Not applicable	20	20
Task 3	9.753	200/Time	20.506
Task 4	0.956	20/Time	20.920
Task 5	8.636	200/Time	23.159
Total Score			**104.585**

Appendix: Supplemental Files

Below are all our file links. The animation and plots are in Google Drive, and GitHub links contain all our code works.

Github: https://github.com/lewistple/AME-556-Final-Project-Team-8/tree/main

Google Drive:
https://drive.google.com/drive/folders/1X5afuaTzF0ysiH4mFXYE15aORsdP0oDZ?usp=sharing