# AME 556 - Robot Dynamics and Control
# Project Report


| Ardalan Aryashad | Dakota Mercer | Ibrahim K. Ozaslan |
|---|---|---|
| 7266195974 | 5683762982 | 2056611851 |
| aryashad@usc.edu | drmercer@usc.edu | ozaslan@usc.edu |


## Contents



## Code Explanation

In this section, we provide a brief explanation of the code structure we used for our project. MATLAB codes and Simscape diagrams are available at the link.

We implemented our project in a way that controls everything from a single MATLAB script file. We define constants, constraints, initial conditions, desired trajectories, MPC and PD parameters–basically everything–in this script. The simscape diagram is called from this script and all the figures and data are automatically saved. Below, are two sections from this script file. The first section defines desired trajectories for the running task and the second section sets gait scheduling parameters.

```
%% desired
body_pace_x     = 1.1;
body_pos_y      = 0.45;
walking_start   = 50; %time index at time dt*walking_start
swing_start     = 48;
walking_end     = 300;

x_desired       = [zeros(1,walking_start-1), ...
    0:body_pace_x*dt:body_pace_x*dt*(walking_end-1)];
xd_desired      = [zeros(1,walking_start), body_pace_x*ones(1,walking_end-1)];
desired_N       = length(x_desired);
theta_desired   = 0*ones(1,desired_N);
thetad_desired  = zeros(1,desired_N);
y_desired       = body_pos_y*ones(1,desired_N);
yd_desired      = zeros(1,desired_N);

%% gait schedule
params.gait_schedule = [1 1 1 1 1 0 0 0 0 0];

params.swing_vel_x      = 1;
params.swing_height     = 0.06;
params.K_gait_vel       = -0.012;
params.K_swing_p        = diag([10.72, 14]);
params.K_swing_d        = diag([.3, 2]);
```

We define two indices: *walking_start* which is the time index at which desired trajectories starts moving, and the *swing_start* which is the time index at which the agent starts taking a step, i.e., beginning of gait scheduling. By differentiating between body and leg movements in this way, we aim to establish the balance easier.

The main part of our implementation is the optimization-based controller that combines MPC with the gait scheduling. In what follows, we go through its main steps.

We first discretize the time, our controller is triggered at every 40ms:

```
    %% find time index
    k = floor(t/params.dt);
```

Next, if there is contact between at least one foot and ground, we initiate the controller by computing the position of the end effector through homogeneous transformations:

```
    if sum(contact) > 0
    %% position of end efector
    r1 = rot2D(theta)*(rot2D(q(1))*(rot2D(q(2))*[0;-params.link_l] + ...
        [0;-params.link_l]) + [0;-params.body_a/2]);
    r2 = rot2D(theta)*(rot2D(q(3))*(rot2D(q(4))*[0;-params.link_l] + ...
        [0;-params.link_l]) + [0;-params.body_a/2]);
```

If swing stage has started, i.e., *swing_start* is smaller than current time index $k$, we update gait schedule

```
%% gait schedule
if k ≥ params.swing_start
gait = circshift(params.gait_schedule, -k+params.swing_start);
sig1 = gait(1);
sig2 = 1 - sig1;
swing_dur = sum(gait,2)*params.dt;
swing_t = (sum(gait,2)-sum(gait(1:5),2))*params.dt;
end
```

Then, we compute desired foot location and associated forces using the formulas given in the notes.

```
%% desired foot
swing_p = zeros(2,1);
swing_p(1) = x + 0.5*swing_dur*xd - params.K_gait_vel*(params.swing_vel_x - xd);
swing_p(2) = params.swing_height*sin( swing_t/swing_dur*pi );

p = sig2*(r1 + [x;y]) + sig1*(r2 + [x;y]);
pd= sig2*(jacobian_q(q(1),q(2), theta)'*w(1:2)) + sig1*(jacobian_q(q(3),q(4), ...
    theta)'*w(3:4));
F_swing = params.K_swing_p*(swing_p - p) - params.K_swing_d*pd;
```

The next step is to define MPC parameters. Since it is similar to our homework, we skip the details. For example, parameters for inequality constraints are determined compactly using the kroncker products as follows.

```
%% inequality constraints: force constriants
A_ineq_k = [0  1    0   0;
0 -1    0   0
0  0    0   1;
0  0    0  -1;
0 -1    0   0
0  0    0  -1;
1 -0.7  0   0;
-1 -0.7  0   0;
0  0    1  -0.7;
0  0   -1  -0.7];

b_ineq_k = ...
    [250;250;250;250;-10;-10;0;0;0;0].*[sig1;sig1;sig2;sig2;sig1;sig2;sig1;sig1;sig2;sig2];

A_ineq = [kron(zeros(params.N), zeros(size(A_ineq_k,1), n)), kron(eye(params.N), ...
    A_ineq_k)];
b_ineq = [kron(ones(params.N,1),b_ineq_k)];
```

Here, *sig* denotes gait signals. After we read desired trajectory from the script file,

```
f = params.mpc_desired(:
,k:k+params.N-1);
```

```
f = [f(:); repmat(params.F_desired, params.N,1)];
f = -params.H*f;
```

we run quadratic program solver for MPC

```
mpc = quadprog(params.H,f,A_ineq,b_ineq,A_eq,b_eq,[],[],mpc0, options);
%output variable
F = mpc(params.N*n+1:params.N*n+4);
```

Finally, we determine torque based on the gait signals

```
Torque = [jacobian_q(q(1), q(2), theta)'*( -sig1*F(1:2) + sig2*F_swing );...
jacobian_q(q(3), q(4),theta)'*( -sig2*F(3:4) + sig1*F_swing )];
```

To perform the project tasks, we made appropriate modifications on this core. Modifications are mostly consisted of changing desired trajectories or the gait scheduling parameters.

# Task 1: Simulation and Physical Constraints

To enforce constraints, we created an error block in simscape diagram next to the MPC module. It is a ordinary MATLAB function that check constraints through if-else statements and returns error if the associated inequalities are violated.In the case of constraint violation, the termination of the simulation for each constraint is demonstrated in the following links to the simulation videos:

1. **Joint angles:** Hip joint (q1), Knee joint (q2), Hip joint (q3), Knee joint (q4).
2. **Joints Velocities:** Hip joint (w1), Knee joint (w2), Hip joint (w3), Knee joint (w4).
3. **Constraints on torque** are enforced through saturation blocks connected to revolute joints and hence they cannot be violated.

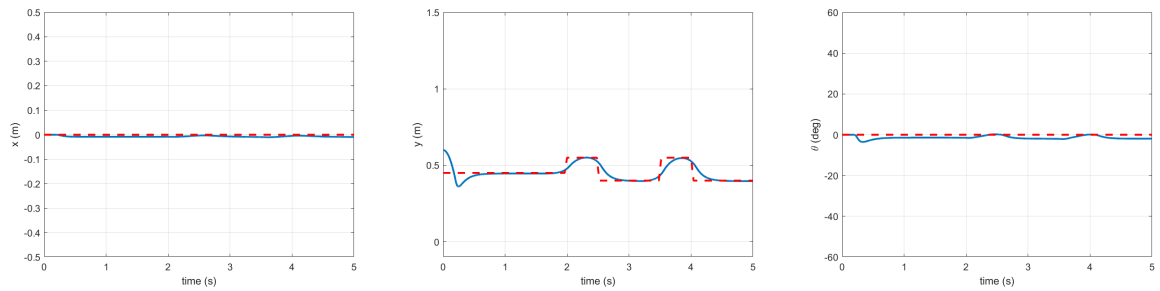
# Task 2: Standing and Walking

## Standing Up and Down

Using the following MPC parameters,

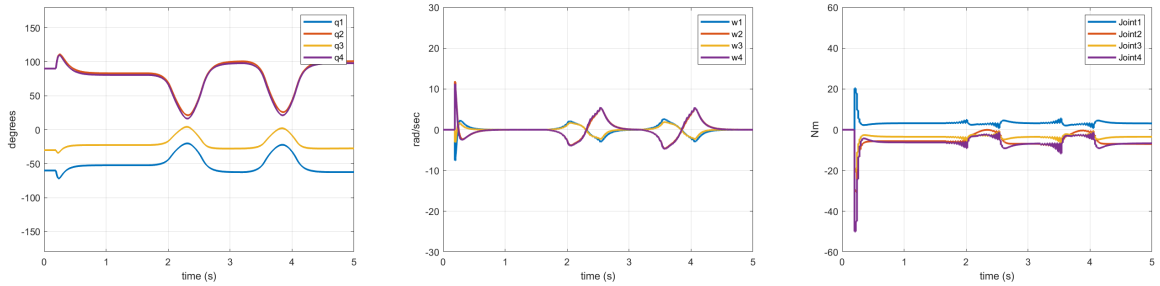
$$Q = \text{diag}(1000, 5000, 3000, 100, 100, 100, 1) \quad R = \text{diag}(10, 5, 10, 5) \times 10^{-4}$$

we performed the task. We changed desired $y\_position$ to the desired values during the given time intervals. The state trajectory of body movement and joint movements are given in Figure 1 and Figure 2. The simulation video can be accessed through the the link (moving up and down)




(a) $x$-position (b) $y$-position (c) body angle

Figure 1: Task 2 Moving Up and Down: The trajectories of the body states. The agent starts moving at $t = 2$ with the height 0.45m. First, it raises the height to 0.5m in $0.5s$ and then lowers the height to 0.4m in 1s and repeat this up-down movement one more time. Dashed lines show desired body trajectories.




(a) joint angle (b) angular velocity (c) torque

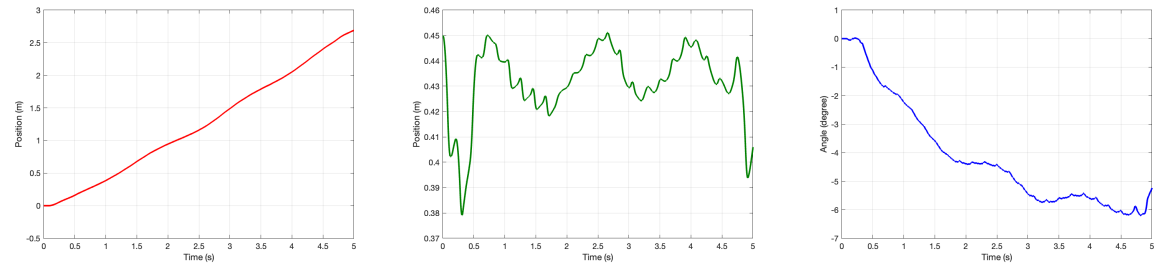
Figure 2: Task 2 Moving Up and Down: The trajectories of the joint states. The agent starts the movement at $t = 2$. All physical constraints are satisfied. As desired, toque applied to joints are within feasible interval.

## Walking Forward

Using the following parameters, we implemented gait scheduling

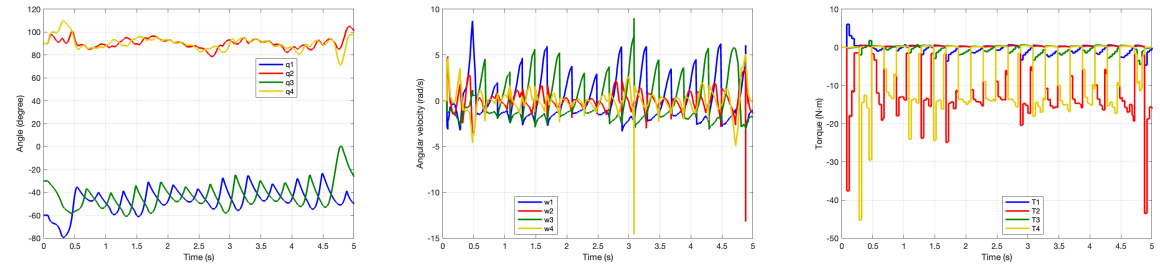
$$k_v = 0.005, \quad h_{foot} = 0.06, \quad K_p = \text{diag}(11, 10), \quad K_d = \text{diag}(0, 0)$$

we set desired foot velocity to 1m/s but the desired body velocity to 0.6m/s. The state trajectory of body movement and joint movements are given in Figure 3 and Figure 4. The simulation video can be accessed through the the link (walking forward)




(a) $x$-position (b) $y$-position (c) body angle

Figure 3: Task 2 Walking Forward: The trajectories of the body states. The agent starts running at $t_i = 0.1$ and reaches $x = 2.65$m at time $t_f = 5$s, i.e., walking with the average speed of $\bar{V} = 0.53$m/s.




(a) joint angle (b) angular velocity (c) torque

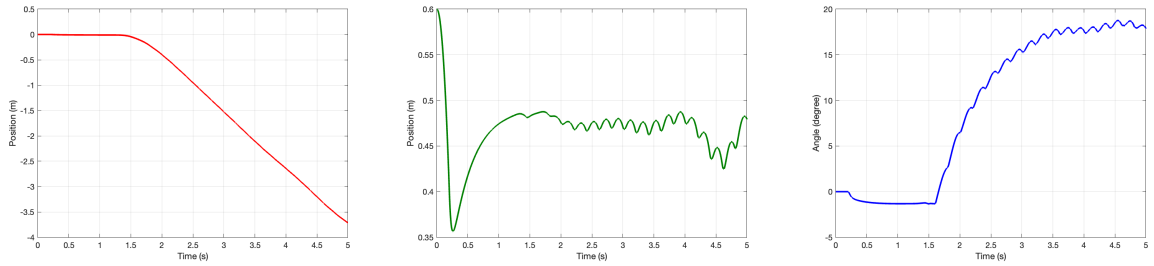
Figure 4: Task 2 Walking Forward: The trajectories of the joint states. The agent starts running at $t_i = 0.1$s. All physical constraints are satisfied. And, toque applied to joints are within feasible interval.

## Walking Backward

Using the following parameters, we implemented gait scheduling

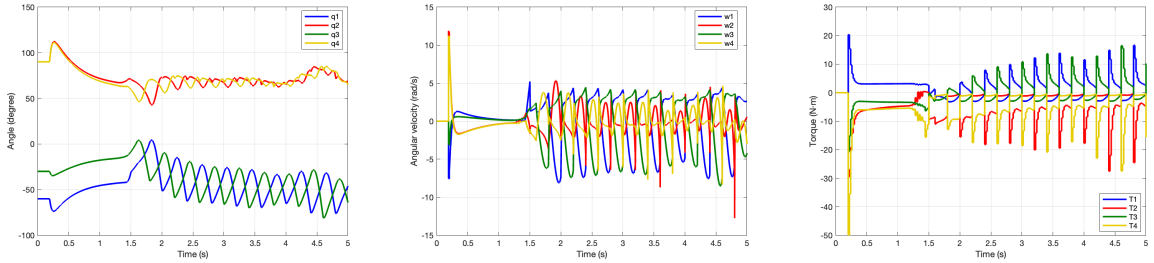
$$k_v = 0.011, \quad h_{foot} = 0.01, \quad K_p = \text{diag}(31, 10), \quad K_d = \text{diag}(0.2, 0.08)$$

we set desired foot velocity to $-0.5$m/s but the desired body velocity to $-0.5$m/s. The state trajectory of body movement and joint movements are given in Figure 5 and Figure 6. The simulation video can be accessed through the the link (walking backward)




(a) $x$-position (b) $y$-position (c) body angle

Figure 5: Task 2 Walking Backward: The trajectories of the body states. The agent starts running at $t_i = 1.5$ and reaches $x = -3.65$m at time $t_f = 5$s, i.e., Walking with the average speed of $\bar{V} = -0.73$m/s. the extra time at the start comparing to forward walking was needed to reach the robot to its steady position of standing.




(a) joint angle (b) angular velocity (c) torque

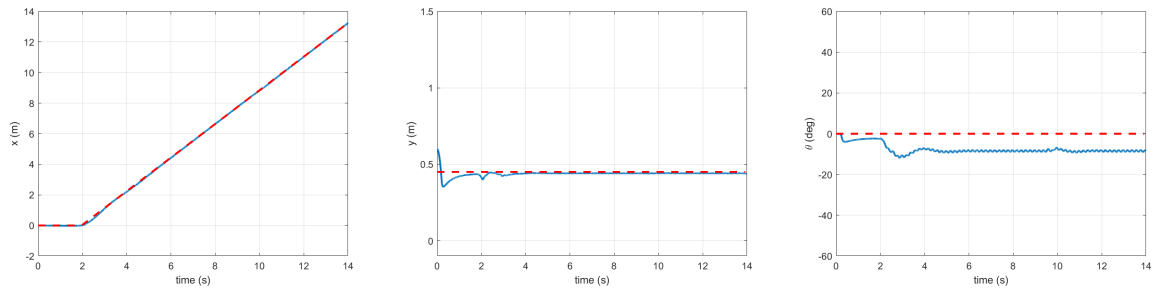
Figure 6: Task 2 Walking Backward: The trajectories of the joint states. The agent starts running at $t_i = 1.5$. All physical constraints are satisfied. And, toque applied to joints are within feasible interval.

# Task 3: Running on flat ground

Using the following parameters, we implemented gait scheduling

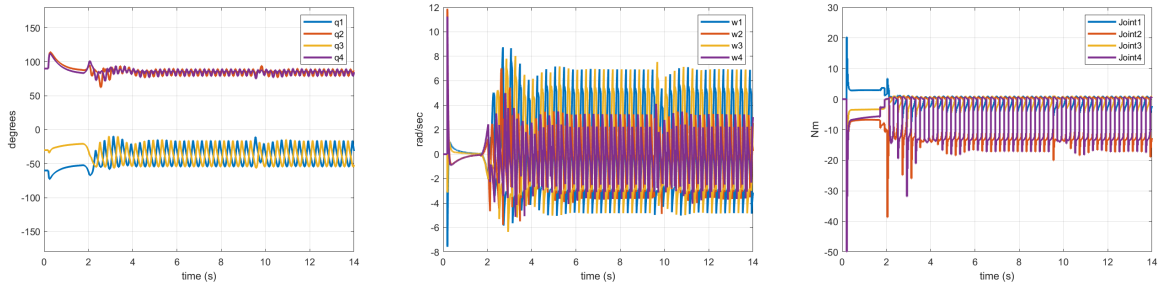
$$k_v = 0.012, \quad h_{foot} = 0.06, \quad K_p = \text{diag}(10.72, 14), \quad K_d = \text{diag}(0.3, 2)$$

we set desired foot velocity to 1m/s but the desired body velocity to 1.1m/s. The state trajectory of body movement and joint movements are given in Figure 7 and Figure 8. The simulation video can be accessed through the the video link (running)




(a) $x$-position (b) $y$-position (c) body angle

Figure 7: Task 3 Running: The trajectories of the body states. The agent starts running at $t_i = 2$ and reaches $x = 10$m at time $t_f = 11.08$s, i.e., runs 10m in $\Delta t = 9.08$s. Dashed lines show desired body trajectories.




(a) joint angle (b) angular velocity (c) torque

Figure 8: Task 3 Running: The trajectories of the joint states. The agent starts running at $t_i = 2$. All physical constraints are satisfied. As desired, toque applied to joints are within feasible interval.

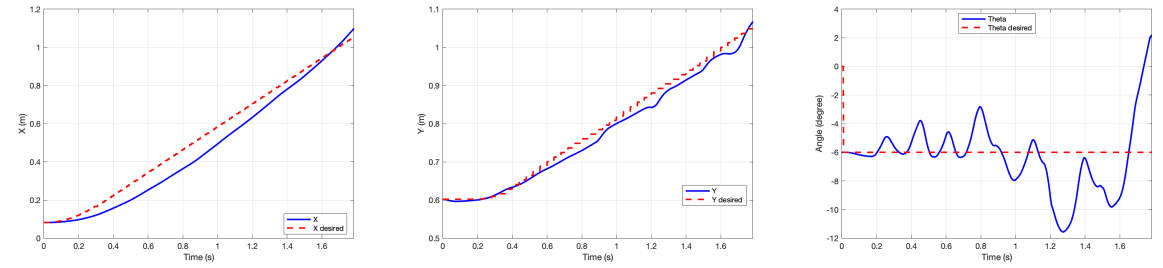
# Task 4: Stair Climbing

Using the following parameters, we implemented our gait. Note that due to non-constant velocities as the agent was accelerating, gain scheduling was used during acceleration and after. Furthermore, as one leg was continually lagging behind the other during stair climbing, different gains were tuned for each foot. Note the derivative gains on our left foot are slightly higher than the right foot, for example. This is due to an average higher error resulting from the left foot lagging the right foot. This often resulted in speed or torque violations, so a higher derivative gain was tuned to reduce this.

$$\text{Left Leg:} \quad k_{vl} = 0.001, \quad K_{pl} = \text{diag}(50, 50), \quad K_{dl} = \text{diag}(1.8, 2.2),$$

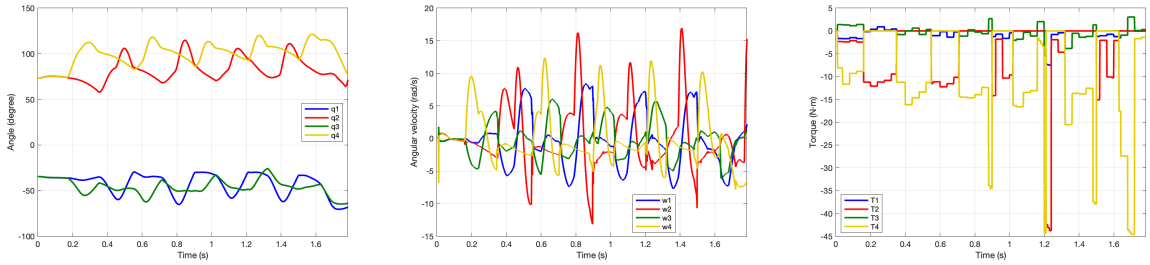
$$\text{Right Leg:} \quad k_{vr} = 0.008, \quad K_{pr} = \text{diag}(50, 50), \quad K_{dr} = \text{diag}(1, 2)$$

A desired constant x-velocity was set to .6m/s, along with a corresponding constant desired y-velocity to account for vertical change in position as stairs are climbed. The simulation video can be accessed through the the video link (stair climbing).




(a) $x$-position (b) $y$-position (c) body angle

Figure 9: Task 4 Stair Climbing: The trajectories of the body states. The agent starts the climb at $t_i = 0$s and $x_i = .082$m and reaches $x = 1.082$m at time $t_f = 1.77$s, i.e., runs 1m in $\Delta t = 1.77$s. Dotted red lines denote desired trajectories of the states.




(a) joint angle (b) angular velocity (c) torque

Figure 10: Task 4 Stair Climbing: Hip Joint Positions and Velocities. Knee joint positions and velocities along with joint torques.

Several methods were explored to achieve this task, including the use of "torque-sensing" as an alternative to gait scheduling. To achieve this, **and** and **or** gate logic was implemented in our Simulink model that sensed when each stair was in contact with a foot. This method of leg placement had pros and cons. One pro is that tuning the gait schedule to match the foot placement timing was no longer needed; this was a significant issue when attempting to stair climb as x-velocity was difficult to control. However, a significant issue was that foot placement was no longer in sync with the MPC;something that could easily be accomplished with a gait schedule. This resulted in the foot hitting the ground and having to wait a maximum of 40ms before the MPC turned back on and commanded a new force, which lent itself to an increased chance of foot slippage.

A simple square wave for y-desired foot placement yielded the best result to avoid foot/stair collision. We also implemented tunable gains that scaled the desired x-foot placement while the desired y-foot placement was not high enough. This resulted in a foot placement that went in the negative x-direction slightly while raising the leg which was very effective at avoiding stair collision.

Controlling x-velocity sufficiently played a key role in proper stair climbing as it resulted in a steady-state foot placement at each step. X and Y velocity state inequalities were tuned in the MPC to help achieve "constant" velocities along with our Q-gains. On a similar note, the PD gait controller was turned off for a specified amount of time before stair climbing to allow for a build-up of velocity before the first step was taken. A faster sampling rate of 10 ms was implemented on our PD gait controller relative to the MPC (40ms) as well. A faster rate was chosen as the 40 ms delay resulted in a laggy PD response and subsequent stair collision of the legs; this was also the reason a square wave was chosen for desired y-foot placement instead of a sine wave.