AME 556 Project Report - Group 11

Prashanth Ravichandar
pr39760@usc.edu
8607319309

Athang Gupte
aagupte@usc.edu
7105760667

Rahul Umapathy
umapathy@usc.edu
9729386618

Abstract

This project focuses on creating controllers for the Unitree A1 robot to do a variety of activities, including walking, running, climbing stairs, and navigating an obstacle course, using MATLAB Simscape. The objective is to develop controllers that allow the robot to carry out these activities successfully and efficiently while conforming to a set of specifications and limits. Controllers were designed for walking, turning, stair climbing and obstacle avoidance. The walking activities include walking forward, backward and sideways with speed ≥ 0.5 m/s and turning in place with a yaw speed of ≥ 0.5 rad/s. Three other controllers were developed for running, stair climbing and obstacle avoidance. The robot was able to perform well in all three environments.

1 Introduction

1.1 Unitree A1 Robot

A versatile and sophisticated quadrupedal robot, the Unitree A1 was created by Unitree Robotics. With an emphasis on agility, stability, and adaptation, it is made to resemble the propensities for locomotion of animals, particularly quadrupeds. The A1 robot has four adjustable legs with several degrees of freedom, enabling it to move precisely and intricately over a variety of surfaces. With the aid of sensors like cameras and IMUs, the robot can detect its surroundings and determine the best course of action for avoiding obstacles. The A1 robot is a useful tool for researching robot dynamics and control schemes since it provides a platform for engineers and researchers to experiment with and create cutting-edge control algorithms. The Unitree A1 robot offers a significant leap in the world of robotics with its amazing locomotion skills and innovation potential and holds tremendous promise for a variety of applications, including search and rescue, exploration, and human-robot interaction.

Figure 1: UnitreeA1 Quadraped robot

1.2 Unitree A1 Robot in MATLAB

For control and analysis purposes, MATLAB may be successfully connected with the Unitree A1 robot. Designing, modeling, and putting into practice control algorithms for the A1 robot's movement

and behavior is made possible by the robust platform offered by MATLAB. Engineers and researchers may easily create and fine-tune controllers to accomplish precise and dynamic movements by utilizing the robotics and control system toolbox in MATLAB.

Simscape Multibody, a multidomain physical modeling tool, can be used with MATLAB to provide a simulation environment for the Unitree A1 robot. Simscape Multibody enables the construction of a virtual robot model that includes the mechanical framework, joints, and actuators of the actual robot. Insights into the robot's behavior and performance can be gained by accurately simulating and analyzing its dynamics and kinematics.

To create effective and reliable controllers for the A1 robot, MATLAB also provides a variety of control design and optimization tools. To accomplish particular locomotor goals or behaviors, engineers might build a variety of control systems, such as proportional-integral-derivative (PID) control, model predictive control (MPC), or adaptive control. The controller parameters can be adjusted to achieve the best performance and stability using MATLAB's optimization methods.

Additionally, the A1 robot's movements and behavior may be represented through dynamic and immersive graphics thanks to MATLAB's visualization capabilities. Engineers can see the robot's gait patterns, joint angles, and trajectory data, which helps with control algorithm analysis and troubleshooting. Additionally, MATLAB offers data logging and analysis capabilities that make it easier to evaluate performance and contrast various control strategies.

All things considered, MATLAB provides a thorough and effective environment for the creation, simulation, and evaluation of control algorithms for the Unitree A1 robot. For scientists and engineers working with the A1 robot, MATLAB is a priceless tool because of its integration with the robot's dynamics and its extensive collection of control and analytic capabilities.

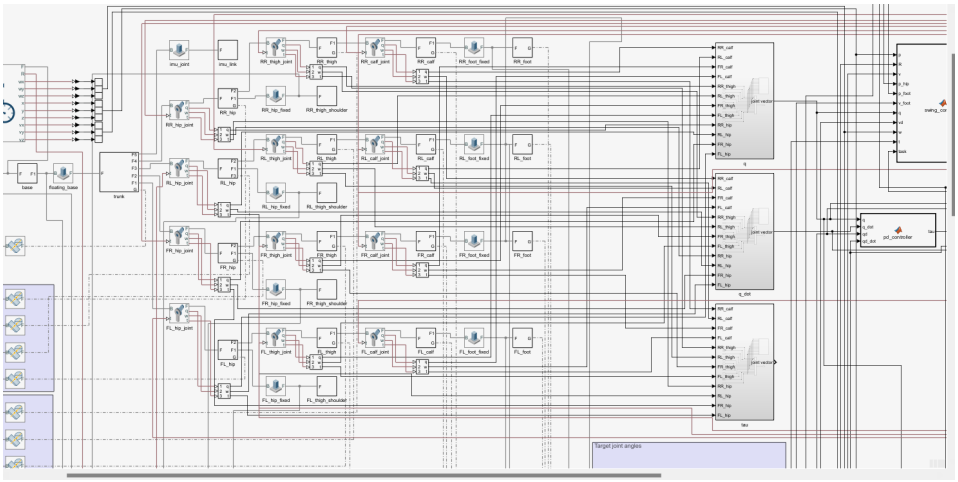

Figure 2: UnitreeA1 setup in MATLAB Simulink

2 Development of the Unitree A1 model in Simulink

2.1 Simulink model

The Simulink model of the Unitree A1 Robot was developed using Simulink. A world frame, solver and mechanism configuration were added. The blocks and the joints in the Simscape module in the library were used to develop the model of the quadruped robot. Contact force modules were added wherever there was contact. An infinite plane was added as the ground. Scopes were added at the required places to visualize the results.

2.2 Functions

Different functions were developed in setting up the robot. The common functions used in all the controllers are described in detail below.

`Vec2Skewmat`: takes a 3-dimensional vector as an input and returns a skew-symmetric matrix S as output.

`foot_jacobian`: calculates the Jacobian matrix for a robotic leg given the joint angles `q` and the leg number.

`get_current_phase`: determines the current phase and phase start time based on the given time `t` and `gait_length` (length of a gait cycle).

`qp_soln`: formulates the quadratic programming (QP) problem to calculate the ground reaction forces (`F`) for a robotic system. The `quadprog` function is called to solve the QP problem and calculate the optimal control input `u` based on the provided cost, constraints, and bounds.

`qp_caller`: calculates torques (`tau`) and ground reaction forces (`F`) based on the given parameters and inputs. Uses the `qp_soln` function and converts the output ground reaction forces to the joint torques using the `foot_jacobian` function.

`mpc_soln`: implements a model predictive control (MPC) algorithm to compute the ground reaction forces based on the current state of the system and specified parameters. The function takes as inputs the time `t`, joint positions `q`, foot positions `r1, r2, r3, r4`, desired state `xd`, MPC gain matrices `Q_mpc` and `R_mpc`, time step `dt`, prediction horizon `N`, gait length `gait_length`, and `gaitname`. The function constructs the MPC optimization problem by defining the objective function (cost) matrix `H` and vector `f`, the equality constraints matrix `Aeq` and vector `Beq`, and the inequality constraints matrix `A_ineq` and vector `b_ineq`. The equality constraints are built based upon the linearization of the system dynamics for the current system state. This uses the gait table for the current timestep using the `gait` function to define which legs are in contact and will thus apply a ground reaction force. The MPC optimization problem thus defined is solved using `quadprog` to obtain the ground reaction forces for the feet that are in contact in the current gait phase. The MPC implementation used is similar to the problem setup given in HW4 and lectures.

`mpc_<task>_caller`: returns the ground reaction forces for the stance phase of the legs using MPC optimization. Defines the MPC gains for the particular task and calls `mpc_soln` to solve the MPC optimization problem.

`mpc_caller`: returns the joint torques for stance phase of the legs by calling the appropriate `mpc_<task>_caller` for the specific task and then converting the ground reaction

`foot_placement_<task>`: returns the joint torques for swing phase of the legs by calculating the foot trajectory for the specific task. The foot trajectories are implemented using a PD controller to allow smooth movement in swing phase.

3 Controller Design

3.1 Task 1: Walking

3.1.1 Approach

The robot starts in a sitting position, and then starts walking after a fixed period dedicated to standing and stabilization.

First, to get the robot to stand, we apply a PD controller for 1.95s. The PD controller stabilizes the robot in a stable configuration with the center of mass (COM) about 0.28 meters above the ground. The gains for PD controller are `Kp` = 1000 and `Kd` = 1000.

From 1.95s onwards an MPC and foot placement controllers start acting. The MPC used is based on Single Rigid Body Dynamics assumption. The following subsections describe details of our approach for MPC and foot placement for the walking tasks.

To solve the walking task, we first tried to perform trotting in place with only the front leg moving up and down and the other legs being static. Then once that was successful, we tried to perform trotting in place with all four legs moving. After in-place trotting was successful, we moved to walking with first a speed of 0.1 m/s and then increased the velocity gradually.

3.1.2 Gait

The "trotting" gait was used for walking with the following parameters - `N` = 20 , `dt` = 0.015 s , gait length = 0.15 s. The gait alternates between the FL-RR and FR-RL stances with no flight phase between the transitions. The following table shows the timing for "trotting" gait used -

N	1	2	3	4	5	6	7	8	9	10
dt	0.00	0.015	0.03	0.045	0.06	0.075	0.09	0.105	0.12	0.135
Fore Left (FL)	1	1	1	1	1	1	1	1	1	1
Fore Right (FR)	0	0	0	0	0	0	0	0	0	0
Rear Left (RL)	0	0	0	0	0	0	0	0	0	0
Rear Right (RR)	1	1	1	1	1	1	1	1	1	1

N	11	12	13	14	15	16	17	18	19	20
dt	0.15	0.165	0.18	0.195	0.21	0.225	0.24	0.255	0.27	0.285
Fore Left (FL)	0	0	0	0	0	0	0	0	0	0
Fore Right (FR)	1	1	1	1	1	1	1	1	1	1
Rear Left (RL)	1	1	1	1	1	1	1	1	1	1
Rear Right (RR)	0	0	0	0	0	0	0	0	0	0

3.1.3 MPC

The desired states for walking are:

$$
\begin{aligned}
p^d &= [0; 0; 0.28] \\
\dot{p}^d &= v^d_{\text{COM}} \\
\theta^d &= [0; 0; 0] \\
\omega^d &= [0; 0; 0]
\end{aligned}
$$

For the turning task we use the following desired states:

$$
\begin{aligned}
p^d &= [0; 0; 0.28] \\
\dot{p}^d &= [0; 0; 0] \\
\theta^d &= [0; 0; 0] \\
\omega^d &= [0; 0]
\end{aligned}
$$

Tuning: For walking forward and backward tasks, we set the gain for `px` to zero to allow the MPC to adjust to constantly changing `px` values. While tuning, we noticed that the robot had very low stability in the roll orientation and kept oscillating around the x axis through the COM, so we increased the gains of `theta(1)` and `wz` to very high values to achieve stability. Other cases that we tuned for involved controlling the yaw drift and displacement in the y-direction. The final gains for the MPC for walking forwards and reverse are:

px	py	pz	roll	pitch	yaw	vx	vy	vz	wx	wy	wz
0	35	80	350	10	35	100	45	4	700	10	70

Table 1: `Q_mpc` gains for walking forward and backward

`R_mpc` is set to 0.0001 times identity matrix. The values are set in MATLAB as shown in the following snippet -

```
Q_mpc = diag([ ...
     0, ...   % px
     35, ...  % py
     80, ...  % pz
     350, ... % roll
     10, ...  % pitch
     35, ...  % yaw
     100, ... % vx
     45, ...  % vy
     4, ...   % vz
     700, ... % wx
     10, ...  % wy
     70 ...   % wz
    ]);
 R_mpc = 0.0001*eye(12);
```

For walking sideways, `R_mpc` is same as walking forwards and the Q gains are as follows:

px	py	pz	roll	pitch	yaw	vx	vy	vz	wx	wy	wz
35	0	80	350	10	35	45	85	4	700	10	70

Table 2: `Q_mpc` gains for walking sideways

Finally, for turning, the Q gains were increased for angular velocity in z axis and set to zero for yaw, and `R_mpc` was set to 0.00001 times identity. The gains for the turning task are

px	py	pz	roll	pitch	yaw	vx	vy	vz	wx	wy	wz
20	20	70	300	300	0	4	4	4	300	300	100

Table 3: `Q_mpc` gains for turning

3.1.4 Foot Placement

For foot placement, a linear triangular trajectory was chosen with max height, h = 0.05 m. The stride length in the x-direction was calculated from the velocity and desired velocity of the COM using the Raibert heuristic for foot location from Raibert [1].

$$\text{stride} = \frac{t_{\text{swing}}}{2} v^d_{\text{COM}} + K_{\text{step}}(v_{\text{COM}} - v^d_{\text{COM}}) \qquad \text{(Raibert heuristic)}$$

Then the foot trajectory is given by the following equations which gives us p^d_{foot} and v^d_{foot}.

$$z^d_{\text{foot}} = \begin{cases} \frac{2h}{t_{\text{swing}}}(t - t_0) & \text{if } t \leq \frac{t_{\text{swing}}}{2} \\ h - \frac{2h}{t_{\text{swing}}}(t - t_0 - \frac{t_{\text{swing}}}{2}) & \text{if } i \geq \frac{t_{\text{swing}}}{2} \end{cases}$$

$$x^d_{\text{foot}} = \begin{cases} x_{\text{foot}} + \frac{\text{stride}}{t_{\text{swing}}}(t - t_0) & \text{if } t \leq \frac{t_{\text{swing}}}{2} \\ x_{\text{foot}} + \frac{\text{stride}}{t_{\text{swing}}}(t - t_0) & \text{if } t \leq \frac{t_{\text{swing}}}{2} \end{cases}$$

For better stability upon foot placement, mainly to avoid placing the feet too much under the body, we added a correction term to p^d_{foot} in the y-direction. We add 0.03 m and subtract 0.03 m from the y^d_{foot} for the left and right feet, respectively.

Finally, the forces required for moving the foot to the resulting p^d_{foot} and v^d_{foot} are obtained using a PD controller. After testing, we found that the values of `Kp` = 135 and `Kd` = 90 gave us the best results.

For walking sideways, we use the same trajectory but applied to the y-axis. For turning, the foot placement is set to always follow the hip and thus the target x,y coordinates of the foot are set to the hip's x,y coordinates.

3.1.5 Setting desired parameters (Simulink)

We used a ramp and saturation to set the desired COM velocity to increase smoothly using the following parameters

1. Ramp: slope $= 0.2$ (-0.2 for reverse), start time $= 1.95$s , initial output $= 0.1$ (-0.1 for reverse)
2. Saturation: upper limit $= 0.6$, lower limit $= -0.6$

We found that saturating COM velocity to 0.5 m/s maintains an average velocity of about 0.5 m/s throughout the simulation. We also tested with a desired velocity value of 0.6 m/s. While that works for walking in reverse consistently maintaining a velocity of >0.5 m/s, however for forward walking, the robot stumbles and fails to retain balance for our simulation time of 10 s. For walking sideways, the same gait, foot placement, and velocity ramping is used but in the y-direction.

3.2 Task 2: Running

3.2.1 Approach

The robot starts from sitting position same as the walking task, and then starts trotting after a period of stabilization in the standing stance.

We use the same PD controller from the walking task for standing. The time that PD controller is run is changed to 2.0 s to align with the gait timing.

From 2.0 s, we use the MPC and foot placement controller to control the robot to run.

3.2.2 Gait

For this task, the "trotting" gait was used with a flight phase between each stance change. The following parameters were adopted - `N` = 40 , `dt` = 0.01 s , gait length = 0.2 s. The gait alternates between the FL-RR and FR-RL stances with a flight phase of 2 `dt`, i.e. 0.4s between the transitions.

3.2.3 MPC

The desired states for running are same as that for walking forwards. The formulation of the MPC is also similar to the walking task in all aspects except the change in the gait sequence which now includes 2 timesteps with no legs exerting any force.

3.2.4 Foot Placement

For foot placement, we adopted the trajectory by Chen et al. [2]. This trajectory is a cycloid and reduces the impact impulse that cause high instability. This trajectory is also differentiable and the velocity at the contact is zero so it reduces slipping during impact. We noticed a significant improvement in performance after this change.

$$
\begin{aligned}
x &= S\left(\frac{t}{T_{swing}} - \frac{1}{2\pi}\sin\left(\frac{2\pi t}{T_{swing}}\right)\right) - \frac{S}{2} \\
v_x &= S\left(\frac{1}{T_{swing}} - \frac{1}{T_m}\cos\left(\frac{2\pi t}{T_{swing}}\right)\right) \\
z &= \begin{cases} 2H\left(\frac{t}{T_{swing}} - \frac{1}{4\pi}\sin\left(\frac{4\pi t}{T_{swing}}\right)\right) & \text{if } 0 \le t \le \frac{t_{\text{swing}}}{2} \\ 2H\left(1 - \frac{t}{T_{swing}} + \frac{1}{4\pi}\sin\left(\frac{4\pi t}{T_{swing}}\right)\right) & \text{if } \frac{t_{\text{swing}}}{2} \le t \le T_{swing} \end{cases} \\
v_z &= \begin{cases} 2H\left(\frac{1}{T_{swing}} - \frac{1}{T_{swing}}\cos\left(\frac{4\pi t}{T_{swing}}\right)\right) & \text{if } 0 \le t \le \frac{t_{\text{swing}}}{2} \\ 2H\left(-\frac{1}{T_{swing}} + \frac{1}{T_{swing}}\cos\left(\frac{4\pi t}{T_{swing}}\right)\right) & \text{if } \frac{t_{\text{swing}}}{2} \le t \le T_{swing} \end{cases}
\end{aligned}
$$

The following figure 3 shows the foot trajectory employed in our solution -

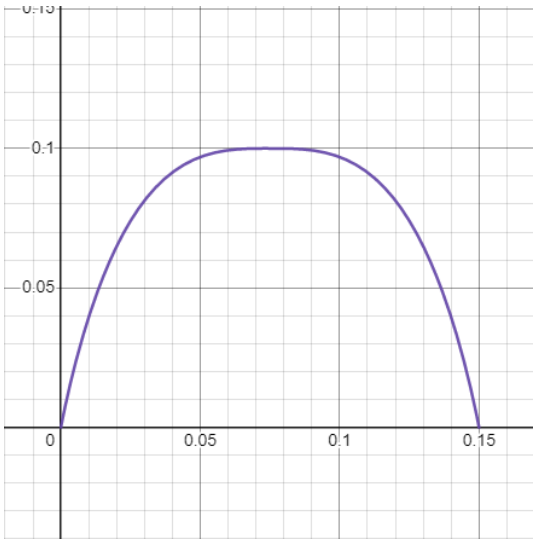

Figure 3: Running Trajectory (z vs x)

Due to lack of time this task was only partially completed. After using the modified foot trajectories we had better results in running but the max speed achievable was still limited. With more tuning of the MPC gains, we expect to get faster speeds and more stability in running.

3.3 Task 3: Stair Climbing

The robot starts from a standing position, and we use QP to stabilize the robot till 1.95 seconds. Stair climbing happens whenever the leg is within 0.1m from a step. Since the front legs are usually faster than the rear legs, they are made to trot in place if the rear legs are greater than 0.5m from a step. All the other components and controllers are the same as the walking task setup.

3.4 MPC

The MPC set-up is similar to the walking setup. The desired states are:

$$
\begin{aligned}
p^d &= [0; 0; 0.28] \\
\dot{p^d} &= [0.2; 0; 0;] \\
\theta^d &= [0; -\frac{\pi}{6}; 0] \\
\omega^d &= [0; 0; 0.28]
\end{aligned}
$$

While climbing the stairs, we need to increase the height of the robot. We assume the walking surface to be a plane and estimate it similar to Bledt et al. [3].

$$
\begin{aligned}
z(x, y) &= a_0 + a_1 x + a_2 y \\
\boldsymbol{a} &= (\boldsymbol{W}^T \boldsymbol{W})^\dagger \boldsymbol{W}^T \boldsymbol{p}^z \\
\boldsymbol{W} &= [\mathbf{1}\ \boldsymbol{p}^x\ \boldsymbol{p}^y]_{4\times 3}
\end{aligned}
$$

Here † represents the Moore-Penrose pseudoinverse, $\boldsymbol{p}$ refers to the foot positions and (x, y) are the COM coordinates. Then the new COM position is found by

$$
z_{\text{COM}} = p_z^d \cos(\theta) + z - 0.02
$$

0.02 is subtracted to account for the spheres at the end of the calves. and θ is the angle between the above plane and the ground. The gains for the MPC are:

```
Q_mpc = diag([ ...
     0, ...   % px
     80, ...  % py
     150, ... % pz
     50, ...  % roll
     150, ... % pitch
     50, ...  % yaw
     100, ... % vx
     80, ...  % vy
     4, ...   % vz
     50, ...  % wx
     10, ...  % wy
     50 ...   % wz
    ]);
R_mpc = 0.00001*eye(12);
```

3.4.1 Foot Placement

We began by using the linear trajectories for walking. For climbing, we used to linear equations. But this didn't give us good results. So, we use two different functions depending on whether we are walking or climbing. For walking, we use the same trajectory as in running. Additionally, we add an extra 0.1 to desired location of the hind legs, in order to keep up with the front legs. This later gets compensated while climbing up the steps. For climbing, we use the following from Zamani et al. [4]:

$$x = \frac{6x_f}{T_{sw}^5}t^5 - \frac{15x_f}{T_{sw}^4}t^4 + \frac{10x_f}{T_{sw}^3}t^3 \qquad v_x = \frac{30x_f}{T_{sw}^5}t^4 - \frac{60x_f}{T_{sw}^4}t^3 + \frac{30x_f}{T_{sw}^3}t^2$$

where x_f and T_sw are the final position in the x-direction and swing direction, respectively.

For the z direction,

$$z = \begin{cases} \frac{6(h_s+\Delta h)}{t_s^5}t^5 - \frac{15(h_s+\Delta h)}{t_s^4}t^4 + \frac{10(h_s+\Delta h)}{t_s^3}t^3, & \text{if } t \le ts \\ \frac{6(z_f-(h_s+\Delta h))}{(T_sw-t_s)^5}(t-t_s)^5 - \frac{15(z_f-(h_s+\Delta h))}{(T_sw-t_s)^4}(t-t_s)^4 + \\ \frac{10(z_f-(h_s+\Delta h))}{(T_sw-t_s)^3}(t-t_s)^3 + (h_s+\Delta h) & \text{if } i > ts \end{cases}$$

$$v_z = \begin{cases} \frac{30(h_s+\Delta h)}{t_s^5}t^4 - \frac{60(h_s+\Delta h)}{t_s^4}t^3 + \frac{30(h_s+\Delta h)}{t_s^3}t^2, & \text{if } t \le ts \\ \frac{30(z_f-(h_s+\Delta h))}{(T_sw-t_s)^5}(t-t_s)^4 - \frac{60(z_f-(h_s+\Delta h))}{(T_sw-t_s)^4}(t-t_s)^3 + \\ \frac{30(z_f-(h_s+\Delta h))}{(T_sw-t_s)^3}(t-t_s)^2 & \text{if } i > ts \end{cases}$$

Here t_s is the root of the equation:

$$\frac{6x_f}{T_{sw}^5}t_s^5 - \frac{15x_f}{T_{sw}^4}t_s^4 + \frac{10x_f}{T_{sw}^3}t_s^3 = 0$$

For our implementation, we use $T_sw = 0.15, x_f = 0.12, t_s = 0.1, d_s = 0.1, h_s = 0.1, \Delta h = 0.02$, and $z_f = 0.1$. The figure below shows the trajectory:

3.5 Task 4: Obstacle Course

The obstacle course task was partially completed. Our approach combined our solutions for stair climbing for clearing the first and last obstacles, and walking with a lower target `z_COM` for clearing the floating obstacle.

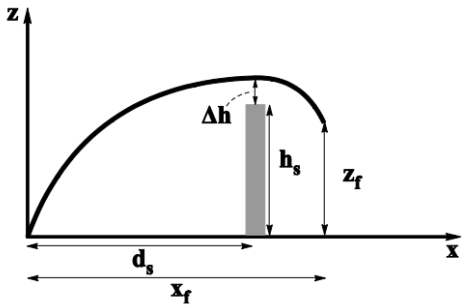

Figure 4: Stair Climbing Trajectory

Due to lack of time this task was only partially completed. As can be seen in the video, the robot could already climb the first obstacle (front legs) using the basic stair climbing solution. Some more fine tuning and conditioning to the environment would allow the robot to clear all the obstacles.

4 Results

The submission videos and plots can be found on this drive link - https://drive.google.com/drive/folders/1cieX9QgovDjhuURTOqxIOvmO-gdPelYA?usp=share_link

Our code is available on GitHub, on this link - github.com/athanggupte/AME556-Robotics-Project

The final commit for each task is in a separate dedicated branch for the specific task.

4.1 Task 1: Walking

Score: 35

4.2 Task 2: Running

Score: partial

4.3 Task 3: Stair Climbing

Score: 30

4.4 Task 4: Obstacle Course

Score: partial

5 Conclusion

The Unitree A1 robot was successfully given four different tasks in this capstone project: walking, running on flat ground, climbing stairs, and negotiating an obstacle course. Speed, agility, safety, and collision avoidance needs were taken into account when designing the controllers in MATLAB Simscape. The robot performed wonderfully on each task, which was assessed according to a set of criteria. By successfully completing these tasks, the Unitree A1 robot and its developed controllers are shown to be both effective and capable. This study emphasizes the significance of creating sophisticated control plans and utilizing MATLAB's capabilities for robotic dynamics and control, paving the way for further improvements in robot behavior and mobility.

6 Acknowledgements

We thank Prof. Quan Nguyen, TA Mohsen Sombolestan, and other teams – who helped us gain a holistic view of the various techniques and controllers implemented herein this project and

understand the concepts with clarity.

References

[1] Marc H Raibert. Dynamically stable legged locomotion: progress report: October 1982-october 1983. 1983.

[2] Mingfang Chen, Qi Li, Sen Wang, Kaixiang Zhang, Hao Chen, and Yongxia Zhang. Single-leg structural design and foot trajectory planning for a novel bioinspired quadruped robot. *Complexity*, 2021:1–17, 01 2021. doi: 10.1155/2021/6627043.

[3] Gerardo Bledt, Matthew J. Powell, Benjamin Katz, Jared Di Carlo, Patrick M. Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252, 2018. doi: 10.1109/IROS.2018.8593885.

[4] Ali Zamani, Mahdi Khorram, and S Ali A Moosavian. Stable stair-climbing of a quadruped robot. *arXiv preprint arXiv:1809.02891*, 2018.