Quadruped Locomotion Through Nonlinear Model Predictive Control

Zixin Zhang, Khai Nguyen, Tianyu Ren, Saurabh Borse, Fukang Liu
Carnegie Mellon University
Pittsburgh, PA 15213
`{zixinz,xuankhan,tianyur,sborse,fukangl}@andrew.cmu.edu`

Abstract

The control of dynamic-legged locomotion poses a significant challenge due to the under-actuation of the system and the uncertainty inherent in the external environment. In this project, we implemented a control framework based on nonlinear model predictive control (NMPC) to enable the dynamic walking of a quadruped robot. The final deliverable is a controller that can control the ANYmal robot in simulation to walk steadily on flat ground with a trotting gait using the OCS2 toolbox and the sequential quadratic programming (SQP) solver. To achieve this, we solved the optimization problem using real-time iteration, filter-based line search, and constraint projection. Our results indicate that the solver is sensitive to the initial guess of states and is able to find a local minimum that is very close to the global minimum, as validated based on physics.

1 Introduction

Control of highly dynamic legged robots has been a challenging problem due to the under-actuation of the body during many gaits and constraints placed on the ground reaction forces. For example, during dynamic gaits such as trotting, the body of the robot is always under-actuated. Additionally, ground reaction forces must always remain in a friction cone to avoid slipping. With certain assumptions made on the base's orientation and angular velocity, the planning problem for legged robots can be formulated as a convex optimization problem [1], which can be solved within a Model Predictive Control (MPC) scheme [2].

The main advantage of MPC is the fact that it allows the current timeslot to be optimized while keeping future timeslots into account. MPC has been widely used in the control of quadruped robots. To generate discrete control policies, MPC solves a constrained trajectory optimization problem at each time stamp to minimize some objective function, typically a quadratic cost of tracking a reference state trajectory. The output of the trajectory optimization will be a sequence of control inputs(preview capabilities), and we take the first one of this sequence as the real control input.

The method of this project is structured similarly to Fig. 1 [3]. A typical MPC-based legged locomotion control architecture contains an MPC controller, a whole-body controller, and a state estimator. MPC finds the optimal ground reaction force based on the user input, and then the whole-body controller finds the desired joint torques based on the solution of MPC. The state estimator keeps updating the current state of the robot and feeds it to the other two parts.

The core of this architecture is the continuous solution of an NMPC problem. To accomplish this, we first discretize the original continuous-time problem into a nonlinear programming (NLP) problem, which is then solved using a sequential quadratic programming (SQP) algorithm. Our trajectory optimization problem is formulated to find the control inputs that enable a walking quadruped robot. The decision variables for this optimization problem include the trajectory of the robot's state and the control inputs in the horizon. The control inputs are set to be the ground reaction force on each foot and the velocity of the joints. Our goal is to determine the motion of the robot using the ground reaction forces, as these are the only external forces acting on the quadruped robot system. At the same time, by controlling the velocity of the foot, we can ensure a smooth swing and touchdown behavior. Furthermore, we employ a range of equality and inequality constraints to ensure that the robot's behavior satisfies the specified gait and avoids foot-slip. To ensure that the quadratic programming (QP) subproblem in each SQP iteration is convex, we use a quadratic approximation to obtain a positive semi-definite Hessian. Additionally, constraint projection is employed to eliminate equality constraints and improve computational performance. Our results indicate that the solver is sensitive to the initial guess of states and may break for a poor initial guess.

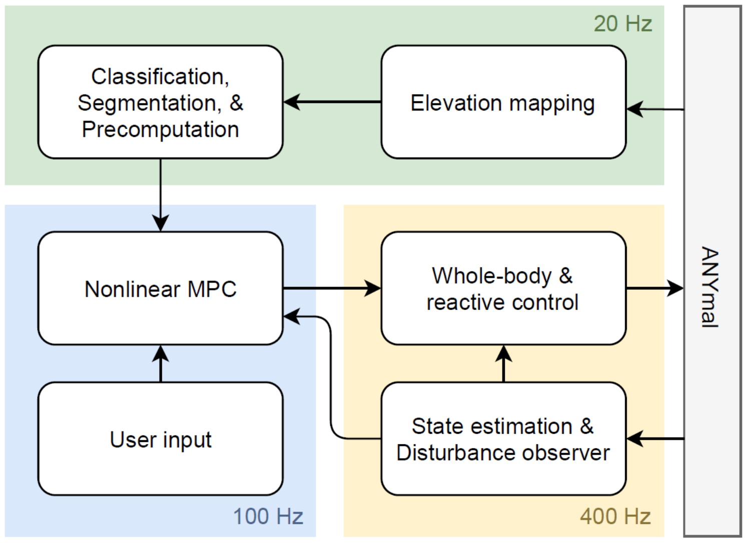
20 Hz
Classification, Segmentation, & Precomputation
Elevation mapping
Nonlinear MPC
Whole-body & reactive control
ANYmal
User input
State estimation & Disturbance observer
100 Hz
400 Hz

Figure 1: MPC-based Legged Robots Controller Architecture [3].

2 Problem Statement

Basic quadruped locomotion is driven by tracking a predefined trajectory of states and control inputs which are generated from a fixed gait schedule. Our overall optimization problem is to minimize both the state tracking error and control input tracking error of the robot. This naturally forms a multi-objective problem which is commonly handled via a weighted sum approach. In the MPC framework, at each time stamp, a constrained trajectory optimization is quickly solved to obtain the control signal (see Fig. 2). In this course project, we will mainly focus on how to solve this problem at one MPC step and analyse the results.

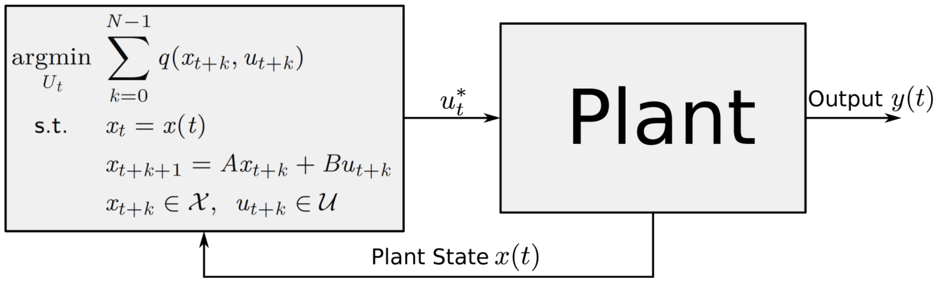
$\underset{U_t}{\text{argmin}} \sum_{k=0}^{N-1} q(x_{t+k}, u_{t+k})$
s.t. $x_t = x(t)$
$x_{t+k+1} = Ax_{t+k} + Bu_{t+k}$
$x_{t+k} \in \mathcal{X}, \quad u_{t+k} \in \mathcal{U}$
u_t^*
Plant
Output $y(t)$
Plant State $x(t)$

Figure 2: MPC framework for optimal control problem [4].

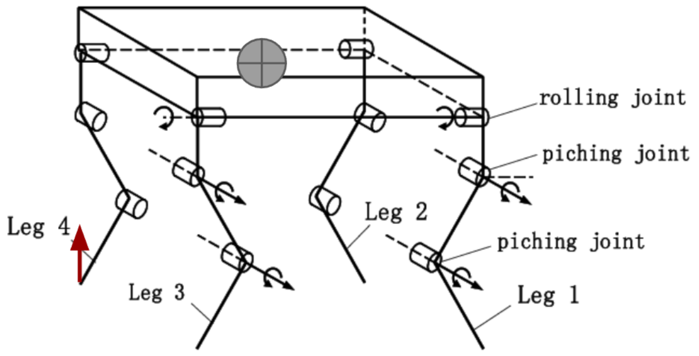

Figure 3: Simplified diagram of a quadruped robot.

The general formulation of the optimal control problem can be given by:

$$
\begin{aligned}
&\text{minimize} \quad \Phi(\mathbf{s}(T)) + \int_0^T l(\mathbf{s}(\tau), \mathbf{u}(\tau), \tau) d\tau && \text{tracking cost} \\
&\text{with respect to} \\
&\qquad \mathbf{s}(t) : \mathbb{R} \to \mathbb{R}^{24} && \text{robot state} \\
&\qquad \mathbf{u}(t) : \mathbb{R} \to \mathbb{R}^{24} && \text{control input} \\
&\text{subject to} \\
&\qquad \mathbf{s}(0) - \hat{\mathbf{s}} = \mathbf{0} && \text{initial condition} \\
&\qquad \dot{\mathbf{s}}(t) - \mathbf{f}\left(\mathbf{s}(t), \mathbf{u}(t)\right) = 0 && \text{robot dynamics} \\
&\qquad \mathbf{h}(\mathbf{s}, \mathbf{u}) = \mathbf{0} && \text{foot constraint} \\
&\qquad \mathbf{g}_i^j(\mathbf{s}, \mathbf{u}) \leq \mathbf{0}, \quad i = 1, \ldots, 4 && \text{bounds on decision variables} \\
&\qquad \mathbf{g}_i^F(\mathbf{u}) \leq \mathbf{0}, \quad i \in \mathcal{C} && \text{friction constraint} \\
&\text{where} \\
&\qquad \text{described in details in next sections}
\end{aligned}
\tag{1}
$$

2.1 Robot Dynamics

The quadruped robot can be modeled as a set of four fully actuated limbs attached to an unactuated 3D rigid body (as in Fig. 3) Each limb has three joints that can be controlled. The following equations can represent the dynamics of this system: [5]

$$
\begin{aligned}
\mathbf{M}_u(\mathbf{q}_j)\dot{\mathbf{v}}_j + \mathbf{b}_u(\mathbf{q}_j, \mathbf{v}_j) &= \mathbf{J}_{c_u}^T(\mathbf{q}_j)\mathbf{F}_c \\
\mathbf{M}_a(\mathbf{q}_j)\dot{\mathbf{v}}_j + \mathbf{b}_a(\mathbf{q}_j, \mathbf{v}_j) &= \boldsymbol{\tau}_j + \mathbf{J}_{c_a}^T(\mathbf{q}_j)\mathbf{F}_c
\end{aligned}
\tag{2}
$$

with $\mathbf{q}_j \in \mathbb{R}^{12}$ and $\mathbf{v}_j \in \mathbb{R}^{12}$ are generalized (joint) coordinates and velocities respectively. $\mathbf{M}$ is the generalized mass matrix, $\mathbf{b}$ represents the nonlinear components including Coriolis, centrifugal, and gravitational terms, and $\boldsymbol{\tau}_j$ is the vector of joint torques. $\mathbf{J}_c$ is a matrix of stacked contact Jacobians, while $\mathbf{F}_c$ is a vector of stacked contact wrenches. The subscripts u and a are associated with the un-actuated and actuated parts of the defined quantities.

From the above equation, we may assume that we have sufficient control authority in the robot's joints. With proper transformation, we can derive the centroidal dynamics as:

$$
\dot{\mathbf{h}}_{com} = \begin{bmatrix} \sum_{i=1}^4 \mathbf{f}_{c_i} + m\mathbf{g} \\ \sum_{i=1}^4 \mathbf{r}_{com,c_i} \times \mathbf{f}_{c_i} + \boldsymbol{\tau}_{c_i} \end{bmatrix}
\tag{3}
$$

where $\mathbf{h}_{com} \in \mathbb{R}^6$ is a combination of linear momentum and angular momentum of the center of mass. $\mathbf{r}_{com,c_i}$ represents the position of the contact point c_i (used interchangeably as i) with respect to the center of mass. Contact forces and torques are denoted as $\mathbf{f}_{c_i} \in \mathbb{R}^3$ and $\boldsymbol{\tau}_{c_i} \in \mathbb{R}^3$.

The base pose movement with respect to inertia frame could be achieved through centroidal momentum matrix (CMM) [6]:

$$\dot{\mathbf{q}}_b = \mathbf{A}_b^{-1}\left(\mathbf{h}_{com} - \mathbf{A}_j\dot{\mathbf{q}}_j\right) \tag{4}$$

where $\mathbf{q}_b \in \mathbb{R}^6$ is the base pose (position and orientation) with respect to the inertial frame. Since we consider point contact instead of patch contact, contact torques can be neglected here. Therefore we can represent our system with **state vector** $\mathbf{s} = [\mathbf{h}_{com}^\intercal, \mathbf{q}_b^\intercal, \mathbf{q}_j^\intercal]^\intercal \in \mathbb{R}^{24}$ and **input vector** $\mathbf{u} = [\mathbf{f}_{c_1}^\intercal, \mathbf{f}_{c_2}^\intercal, \mathbf{f}_{c_3}^\intercal, \mathbf{f}_{c_4}^\intercal, \mathbf{v}_j^\intercal]^\intercal \in \mathbb{R}^{24}$, where $\dot{\mathbf{q}}_j = \mathbf{v}_j$.

The **initial condition** in (1) constrains the state at time stamp zero of the optimization problem as the current state feedback of the robot.

Finally, **full system dynamics** in (1) can be set up by collecting (3) and (4) together:

$$\dot{\mathbf{s}}(t) - \mathbf{f}\left(\mathbf{s}(t), \mathbf{u}(t)\right) = \mathbf{0} \tag{5}$$

In fact, we did not use these derivations for implementation, instead, we used Pinocchio [7] to generate fast forward and inverse dynamics from robot configuration. However, the above derivations provide a better understanding on the robot configuration and variable definition.

2.2 Objective Function

The modes and their switching times are assumed to be fixed in our project, hence the notations dependent on them can be simplified. From this, the reference state and input vector can be denoted as $\mathbf{s}^{REF}(t)$ and $\mathbf{u}^{REF}(t)$, leading to:

$$\Delta\mathbf{s} = \mathbf{s}^{REF} - \mathbf{s}$$
$$\Delta\mathbf{u} = \mathbf{u}^{REF} - \mathbf{u}$$

To form the objective function, we use a weighted sum of multiple cost functions which are the quadratic tracking errors of states and inputs. They are evaluated within the whole MPC horizon T as

$$l(\mathbf{s}(t), \mathbf{u}(t), t) = \frac{1}{2}\Delta\mathbf{s}(t)^\intercal\mathbf{Q}\Delta\mathbf{s}(t) + \frac{1}{2}\Delta\mathbf{u}(t)^\intercal\mathbf{R}\Delta\mathbf{u}(t) \tag{6}$$

$$\Phi(\mathbf{s}) = \frac{1}{2}\Delta\mathbf{s}(T)^\intercal\mathbf{Q}\Delta\mathbf{s}(T) \tag{7}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are positive definite weighting matrices. These quadratic functions are convex. In the quadruped robot walking problem, we can make the terminal cost and the stage cost the same. Because in walking MPC, the final state is nothing special and we do not really have a final goal.

2.3 Foot Equality Constraints

In our problem, we assume that a particular gait cycle is predefined within a set of contact switching times. We want to ensure that the expected touch-down takes place according to the switching schedule. Therefore, all equality constraints are posed at potential contact points, which could be active or inactive. Let us define the set of all active foot contacts by $\mathcal{C}$. At a fixed mode instance $\mathcal{C}_j$ which lasts from time t_j^0 to t_j^f, the corresponding **(foot) state-input equality constraints** $\mathbf{h}_j(\mathbf{s}, \mathbf{u}, t)$ can be expressed as below:

$$\forall t \in [t_j^0, t_j^f] \text{ and } \forall i \in \{1, \ldots, 4\}$$

$$\begin{cases} \mathbf{v}_{c_i} = \mathbf{0} & \text{if } c_i \in \mathcal{C}_j \quad (8a) \\ \mathbf{v}_{c_i} \cdot \hat{\mathbf{n}} - \mathbf{v}^{REF}(t) = \mathbf{0} & \text{if } c_i \in \overline{\mathcal{C}}_j \quad (8b) \\ \mathbf{f}_{c_i} = \mathbf{0} & \text{if } c_i \in \overline{\mathcal{C}}_j, \quad (8c) \end{cases}$$

where $\mathbf{v}_{c_i}$ is the linear velocity of contact point c_i expressed in the inertial frame and $\mathbf{v}_{c_i} = \mathbf{J}_{j,i}\mathbf{v}_j$. This set of constraints (8) has the following meanings:

- The foot of a stance leg should stay fixed on the ground (8a);

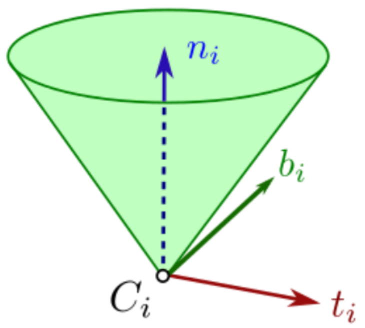

Figure 4: Illustration of a friction cone.

- All swing legs should track a reference trajectory $\mathbf{v}^{REF}(t)$ along the surface normal $\hat{\mathbf{n}}$ (8b), leaving the remaining orthogonal directions – which determine the stride-length – free for the planner to resolve.
- For each foot in swing phase, the contact forces are zero (8c).

Equation (8b) implies that the foot lifts off and touches down with a desired velocity while the planner has the freedom of foot placement (stride length) in the tangential direction.

2.4 Inequality Constraints

To guarantee that our planner produces dynamically feasible trajectories and forces that would also respect the system's inherent operating limits, the following **friction constraint** is introduced (see Fig. 4):

$$\forall t \in [t_j^0, t_j^f] \text{ and } \forall i \in \{1, \dots, 4\}$$

$$\mathbf{g}_i^F(\mathbf{u}) = \sqrt{{f_{c_i}^X}^2 + {f_{c_i}^Y}^2 + \epsilon} - \mu_c f_{c_i}^Z \leq \mathbf{0}, \quad c_i \in \mathcal{C}_j \tag{9}$$

where $\mathbf{f}_{c_i} = [f_{c_i}^X, f_{c_i}^Y, f_{c_i}^Z]^\intercal$. This inequality guarantees the foot contact forces remain inside the friction cone with friction coefficient μ_s and $\epsilon \neq 0$ a parameter that ensures a continuous derivative at $\mathbf{f}_{c_i} = 0$, and at the same time provides a safety margin [8].

Additionally, we have **simple bounds for decision variables**, shown in Table 1. These boundaries ensure that the robot's joints do not exceed the limits of position, velocity and torque, while ensuring that the robot is not broken by excessive ground reaction forces.

$$\mathbf{g}_i^j(\mathbf{x}) = \begin{bmatrix} \mathbf{q}_j - \mathbf{q}_j^{UB} \\ \mathbf{q}_j^{LB} - \mathbf{q}_j \\ \mathbf{v}_j - \mathbf{v}_j^{UB} \\ \mathbf{v}_j^{LB} - \mathbf{v}_j \\ \mathbf{f}_{c_i} - \mathbf{f}_{c_i}^{UB} \\ \mathbf{f}_{c_i}^{LB} - \mathbf{f}_{c_i} \end{bmatrix} \leq \mathbf{0} \tag{10}$$

2.5 Whole-body Controller

This section serves as the description of the low-level architecture for the quadruped locomotion, not for the optimization problem.

The whole-body controller computes the desired joint torques based on the solution from whole-body planner. In our project, we plan to divide the whole-body control into ground force control and swing leg control.

During ground force control, joint torques are computed with

$$\boldsymbol{\tau}_i = \mathbf{J}_{j,i}^T \mathbf{R}_i^T \mathbf{f}_{c_i} \tag{11}$$

Table 1: Bounds on decision variables.

Description	Value	Unit
ground reaction force max	10mg	N
ground reaction force min	0	N
hip joint position max	46	deg
hip joint position min	-46	deg
thigh joint position max	240	deg
thigh joint position min	-60	deg
calf joint position max	-52.5	deg
calf joint position min	-154.5	deg
joint velocity max	21	deg/s
joint torque max	33.5	N.m

where $\mathbf{R}$ is the rotation matrix which transforms from body to world coordinates, $\mathbf{J} \in \mathbb{R}^{3\times 3}$ is the foot contact Jacobian, and $\mathbf{f}_{c_i}$ is the vector of ground reaction forces calculated from the model predictive controller described above. The whole-body controller will be implemented as a separate thread to ensure stable control.

We also propose to use a PD controller to compute joint torques for tracking the optimized swing trajectory in the NMPC solution for each foot. The full controller for tracking swing trajectories is

$$\boldsymbol{\tau}_i = \mathbf{K}_p(\mathbf{q}_j^{MPC} - \mathbf{q}_j) + \mathbf{K}_d(\mathbf{v}_j^{MPC} - \mathbf{v}_j) \tag{12}$$

where $\mathbf{K}_p$ and $\mathbf{K}_d$ are diagonal matrices of proportional and derivative gains, $\mathbf{q}_j^{MPC}$ and $\mathbf{v}_j^{MPC}$ are joints position and velocity in the NMPC solution, and $\mathbf{q}_j$ and $\mathbf{v}_j$ are current joint state.

3 Analysis of Problem Statement

The original continuous-time problem is discretized to get a solvable NLP in next sections, in which the decision variables contain the state trajectory $\mathbf{X} \in \mathbb{R}^{24N}$ and control input trajectory $\mathbf{U} \in \mathbb{R}^{24(N-1)}$ of the robot, where N is the MPC horizon length. The objective function is quadratic, which is continuous and smooth. We also use a fixed time step to discretize the original problem. These makes solving the problem much easier.

As we stated in the previous sections, we have a total of three equality constraints, and one inequality constraint for each knot point in the trajectory. The inequality constraint can make sure that the optimal ground reaction forces stay in the nonlinear friction cone, and the ϵ used to smooth the function make it a practical constraint. The other three constraints result from the walking gait requirement, which are practical constraints. The standing feet should have no motion, and the swing feet shouldn't have ground reaction forces. We also want the z-axis position of the swinging foot to satisfy the generated swing curve. All these constraints are continuous and smooth.

Our discrete NMPC problem is an NLP problem, due to nonlinear objective function and constraints. However, the problem is not convex because equation (8b) is nonlinear. This implies that there may be multiple local optima. In practice, however, we do not really care much whether we can find the global optimum. First, the behavior of the robot will not become outrageous under these constraints. Then, we will make the solution as close as possible to the previous one by warm-starting it, that is using the last solution as the initial guess, to ensure that the change of control input is not too radical.

In NMPC, the horizon length N, and the weight matrices $\mathbf{Q}$ and $\mathbf{R}$ are very important. They can significantly affect the final solution. Here, we will determine the parameters mainly by manual tuning.

4 Optimization Study

4.1 Numerical Method

To convert the continuous optimum control issue into a finite-dimensional nonlinear program, we take a direct multiple-shooting strategy (NLP). The subsequent occurrences of the (1) are similar because MPC computes control inputs across a receding horizon, and when using a SQP technique, the prior solution may be shifted to effectively warm-start the problem. We replicate the end state of the prior solution and initialize the inputs with the references generated for new portions of the shifted horizon for which there is no initial guess.

As an overview of the approach described in the following sections, a pseudo-code is provided in the Algorithm below, referring to the relevant equations used at each MPC step. The QP is solved using HPIPM, which will be concisely described later.

Algorithm:

```
1. Given: previous solution $\mathbf{x}_i$
2. Discretize the continuous problem
3. Compute the linear quadratic approximation
4. Compute the equality constraint projection
5. $\delta\tilde{\mathbf{x}} \leftarrow$ Solve the projected QP subproblem
6. $\delta\mathbf{x} \leftarrow \mathbf{P}\delta\tilde{\mathbf{x}} + \mathbf{p}$, back substitution
7. $\mathbf{x}_{i+1} \leftarrow$ Line-Search$(\mathbf{x}_i, \delta\mathbf{x})$,
```

4.1.1 Discretization

The general nonlinear MPC problem presented below can be formulated by defining and evaluating a cost function and constraints on the grid of nodes.

The continuous signal $\mathbf{s}(t)$ and $\mathbf{u}(t)$ is parameterized over sub-intervals of the prediction horizon $[t, t+T]$ to obtain a finite-dimensional decision problem. This creates a grid of nodes $k \in \{0, ..., N\}$ defining control times t_k separated by intervals of constant duration $dt \approx T/(N-1)$.

Denoting $\mathbf{s}_k = \mathbf{s}(t_k)$ and integrating the continuous dynamics (5) over and interval leads to a discrete time representation of the dynamics:

$$\mathbf{s}_{k+1} = \mathbf{s}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}(\mathbf{s}(t), \mathbf{u}(t))dt, \quad \mathbf{s}(t_k) = \mathbf{s}_k. \tag{13}$$

This equation can be numerically approximated through RK4 [9]. Given 5, 13, $\mathbf{u}_k$ and dt, we get:

$$\begin{aligned}
&\dot{\mathbf{s}}(t) = \mathbf{f}\left(\mathbf{s}(t), \mathbf{u}(t)\right)\\
&\mathbf{k}_1 = \mathbf{f}\left(\mathbf{s}_k, \mathbf{u}_k\right)\\
&\mathbf{k}_2 = \mathbf{f}\left(\mathbf{s}_k + \frac{\mathbf{k}_1}{2}dt, \mathbf{u}_k\right)\\
&\mathbf{k}_3 = \mathbf{f}\left(\mathbf{s}_k + \frac{\mathbf{k}_2}{2}dt, \mathbf{u}_k\right)\\
&\mathbf{k}_4 = \mathbf{f}\left(\mathbf{s}_k + \mathbf{k}_3 dt, \mathbf{u}_k\right)\\
\Rightarrow &\mathbf{s}_{k+1} = \mathbf{f}_k^d(\mathbf{s}_k, \mathbf{u}_k) = \mathbf{s}_k + \frac{dt}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)
\end{aligned} \tag{14}$$

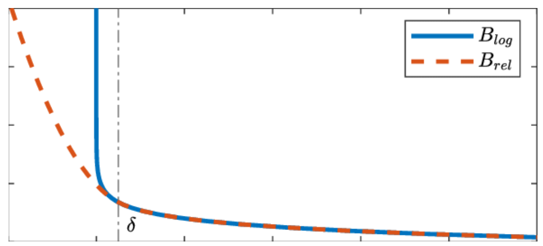

Figure 5: Plot of a relaxed barrier function.

RK4 is applied at each time step for the dynamics constraint function. The other variables, objective function, and constraints are simply discretized into discrete counterparts. We then obtain the following discretized problem:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{2}(\mathbf{s}_N - \mathbf{s}_N^{REF})^\intercal \mathbf{Q}(\mathbf{s}_N - \mathbf{s}_N^{REF}) + \sum_{k=0}^{N-1} l^d(\mathbf{s}_k, \mathbf{u}_k) \\
\text{w.r.t} \quad & \mathbf{x} = [\mathbf{s}_0^\intercal, \mathbf{s}_1^\intercal, \dots, \mathbf{s}_N^\intercal, \mathbf{u}_0^\intercal, \mathbf{u}_1^\intercal, \dots, \mathbf{u}_{N-1}^\intercal]^\intercal \\
\text{s.t.} \quad & \mathbf{g}_i^j(\mathbf{x}) \leq \mathbf{0}, \quad i = 1, \dots, 4 \\
& \mathbf{g}_i^F(\mathbf{x}) \leq \mathbf{0}, \quad i \in \mathcal{C} \\
& \mathbf{s}_0 = \hat{\mathbf{s}} \\
& \mathbf{s}_{k+1} = \mathbf{f}_k^d(\mathbf{s}_k, \mathbf{u}_k) \\
& \mathbf{h}_k(\mathbf{s}_k, \mathbf{u}_k) = \mathbf{0} \\
\text{where} \quad & l^d(\mathbf{s}_k, \mathbf{u}_k) = \frac{1}{2}(\mathbf{s}_k - \mathbf{s}_k^{REF})^\intercal \mathbf{Q}(\mathbf{s}_k - \mathbf{s}_k^{REF}) + \frac{1}{2}(\mathbf{u}_k - \mathbf{u}_k^{REF})^\intercal \mathbf{R}(\mathbf{u}_k - \mathbf{u}_k^{REF})
\end{aligned} \tag{15}$$

4.1.2 Relaxed Barrier Functions

This problem is highly dimensional, at each time step we have to calculate a batch of constraints, and using the active set method to deal with inequality constraints is complicated. Moreover, it is possible to prove the stability of the MPC framework with the relaxed barrier function [10]. Therefore, all inequality constraints were handled through the penalty cost. We used relaxed barrier functions in this project (see Fig. 5). This penalty function was defined as a log-barrier on the interior of the feasible space and switches to a quadratic function at a distance δ from the constraint boundary

$$B(z) = \begin{cases} -\mu \ln(z), & z \geq \delta \\ \frac{\mu}{2}\left(\left(\frac{z - 2\delta}{\delta}\right)^2 - 1\right) - \mu \ln(\delta), & z < \delta \end{cases} \tag{16}$$

where $\mu > 0$, $0 < \delta \leq 1$. Both functions are strictly convex and strictly decreasing on their domains. Thus, if $g : \mathbb{R} \to \mathbb{R}$ is a strictly convex proper function, the function $B(-g(x, u))$ is also a strictly convex proper function on $\mathbb{R}$.

The related penalty is taken element-wise for vector-valued inequality constraints. The sum of all penalty functions are added to the running cost to form a new objective function:

$$\phi(\mathbf{x}) = \frac{1}{2}(\mathbf{s}_N - \mathbf{s}_N^{REF})^\intercal \mathbf{Q}(\mathbf{s}_N - \mathbf{s}_N^{REF}) + \sum_{k=0}^{N-1}\Big(l^d(\mathbf{s}_k, \mathbf{u}_k) + \sum_{i=1}^{4} B_j(-\mathbf{g}_i^j) + \sum_{i\in\mathcal{C}} B_F(-\mathbf{g}_i^F)\Big) \tag{17}$$

One potential limitation of the proposed objective function is the possibility of violating inequality constraints. As demonstrated in Figure 6, the optimal joint velocities for the knee joints of each leg were found to be greater than zero. However, selecting an appropriate value for the hyperparameter μ can help mitigate this issue. Empirically, a value of $\mu = 1$ has been shown to be effective in minimizing the impact of this potential limitation.

4.1.3 Sequential Quadratic Program

The new decision variable is a collection of all state and input within the horizon, $\mathbf{x}$, problem (15) can be written as a general NLP:

$$\min_{\mathbf{x}} \quad \phi(\mathbf{x}), \tag{18a}$$
$$\text{s.t.} \quad \mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{18b}$$
$$\mathbf{H}(\mathbf{x}) = \mathbf{0} \tag{18c}$$

where $\phi(\mathbf{x})$ is the objective function, $\mathbf{F}(\mathbf{x})$ includes initial state and dynamics constraints, and $\mathbf{H}(\mathbf{x})$ includes all foot equality constraints for all horizon steps.

From the Karush-Kuhn-Tucker (KKT) optimality conditions, SQP-based methods apply Newton-type iterations to solve that linear KKT equations while assuming some regularity conditions on the constraints. Let's define the Lagrangian of the NLP in (18) as:

$$\mathcal{L}\left(\mathbf{x}, \boldsymbol{\lambda}_{\mathbf{F}}, \boldsymbol{\lambda}_{\mathbf{H}}\right)=\phi(\mathbf{x})+\boldsymbol{\lambda}_{\mathbf{F}}^{\top} \mathbf{F}(\mathbf{x})+\boldsymbol{\lambda}_{\mathbf{H}}^{\top} \mathbf{H}(\mathbf{x}) \tag{19}$$

with Lagrange multipliers $\boldsymbol{\lambda}_{\mathbf{F}}$ and $\boldsymbol{\lambda}_{\mathbf{H}}$, corresponding to the dynamics and equality constraints. Solving the KKT equations is equivalent to solving the following potentially non-convex QP:

$$\min_{\delta \mathbf{x}} \quad \nabla_{\mathbf{x}} \phi\left(\mathbf{x}_i\right)^{\top} \delta \mathbf{x}+\frac{1}{2} \delta \mathbf{x}^{\top} \mathbf{B}_i \delta \mathbf{x} \tag{20a}$$

$$\text{s.t} \quad \mathbf{F}\left(\mathbf{x}_i\right)+\nabla_{\mathbf{x}} \mathbf{F}\left(\mathbf{x}_i\right)^{\top} \delta \mathbf{x}=\mathbf{0} \tag{20b}$$

$$\mathbf{H}\left(\mathbf{x}_i\right)+\nabla_{\mathbf{x}} \mathbf{H}\left(\mathbf{x}_i\right)^{\top} \delta \mathbf{x}=\mathbf{0} \tag{20c}$$

where the decision variable, $\delta \mathbf{x}=\mathbf{x}-\mathbf{x}_i$, determines the update step relative to the current iteration $\mathbf{x}_i$, and the Hessian $\mathbf{B}_i=\nabla_{\mathbf{x}}^2 \mathcal{L}\left(\mathbf{x}_i, \boldsymbol{\lambda}_{\mathbf{F}}, \boldsymbol{\lambda}_{\mathbf{H}}\right)$. The solution to (20) results in a candidate decision variable update, $\delta \mathbf{x}_i$, and updated Lagrange multipliers.

4.1.4 Quadratic Approximation

As we seek to deploy MPC on dynamic robotic platforms, we do not want the optimization problem provide any difficulty to the numerical solver. In this case, we want $\mathbf{B}_i$ to be semi-definite to make the resulting QP convex. Hence, we use an approximation of Hessian instead. For tracking error of each foot, initially we have:

$$\boldsymbol{\epsilon}_i=\left[\begin{array}{c}\mathbf{q}_i-\mathbf{q}_i^{REF} \\ \dot{\mathbf{q}}_i-\dot{\mathbf{q}}_i^{REF} \\ \mathbf{p}_i-\mathbf{p}_i^{REF} \\ \mathbf{v}_i-\mathbf{v}_i^{REF} \\ \mathbf{f}_i-\mathbf{f}_i^{REF}\end{array}\right] \tag{21}$$

as $\mathbf{p}_i$ and $\mathbf{v}_i$ are the foot position and velocity in world frame.

With Generalized Gauss-Newton approximation [11] ($\mathbf{X}_i$ is a weight matrix):

$$\nabla_{\mathbf{x}}^2\left(\frac{1}{2} \boldsymbol{\epsilon}_i(\mathbf{x})^{\top} \mathbf{X}_i \boldsymbol{\epsilon}_i(\mathbf{x})\right) \approx \nabla_{\mathbf{x}} \boldsymbol{\epsilon}_i(\mathbf{x})^{\top} \mathbf{X}_i \nabla_{\mathbf{x}} \boldsymbol{\epsilon}_i(\mathbf{x}) \tag{22}$$

Similarly, by exploiting the convexity for penalty function:

$$\nabla_{\mathbf{x}}^2(B(\mathbf{g}(\mathbf{x}))) \approx \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x})^{\top} \nabla_{\mathbf{g}}^2 B(\mathbf{g}(\mathbf{x})) \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \tag{23}$$

where $\nabla_{\mathbf{g}}^2 B(\mathbf{g}(\mathbf{x}))$ maintains the curvature information of the convex penalty function.

4.1.5 Constraint Projection

The equality constraints in section 2.3 were chosen carefully to have a full-row rank with respect to control inputs. This will allow us to eliminate equality constraints, before solving the QP through a change of variables.

$$\delta \mathbf{x}=\mathbf{P} \delta \tilde{\mathbf{x}}+\mathbf{p} \tag{24}$$

for which linear transformation satisfies,

$$\nabla_{\mathbf{x}} \mathbf{H}\left(\mathbf{x}_i\right)^{\top} \mathbf{P}=\mathbf{0}, \quad \nabla_{\mathbf{x}} \mathbf{H}\left(\mathbf{x}_i\right)^{\top} \mathbf{p}=-\mathbf{H}\left(\mathbf{x}_i\right) \tag{25}$$

Hence, after substituting (24) in (20), the following QP is solved w.r.t. $\delta \tilde{\mathbf{x}}$

$$\min_{\delta \tilde{\mathbf{x}}} \quad \nabla_{\tilde{\mathbf{x}}} \tilde{\phi}\left(\mathbf{x}_i\right)^{\top} \delta \tilde{\mathbf{x}}+\frac{1}{2} \delta \tilde{\mathbf{x}}^{\top} \tilde{\mathbf{B}}_i \delta \tilde{\mathbf{x}} \tag{26a}$$

$$\text{s.t} \quad \tilde{\mathbf{F}}\left(\tilde{\mathbf{x}}_i\right)+\nabla_{\tilde{\mathbf{x}}} \tilde{\mathbf{F}}\left(\mathbf{x}_i\right)^{\top} \delta \tilde{\mathbf{x}}=\mathbf{0} \tag{26b}$$

This projected problem now only contains costs and system dynamics but has stage-wise structure, hence solving the QP only requires Ricatti iteration via HPIPM, the Riccati-based interior-point method solver we use, which has been tailored to exploit the particular structure of the obtained QP.

After obtaining the solution to (26), we can back-substitute that to (24) to achieve the original variable.

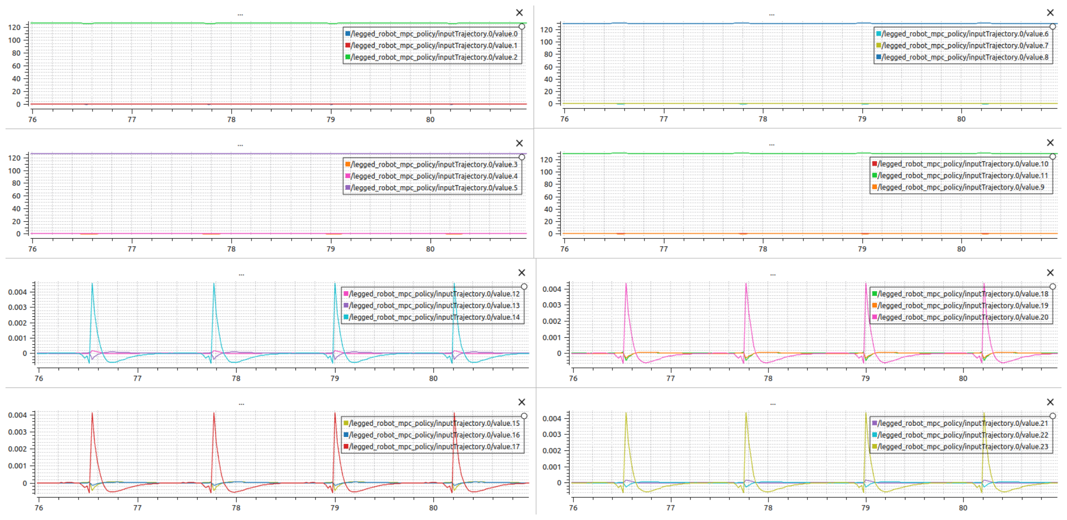
Figure 6: The solutions in stance mode

4.1.6 HPIPM

HPIPM is a high-performance framework for quadratic programming (QP) that provides the building blocks necessary to effectively and consistently handle model predictive control problems. Currently, HPIPM supports three different QP types and offers (partial) condensing procedures and interior point method (IPM) solvers. It should be noted that other equality constraints (such as the constraint on the initial state value $x(0) = \hat{x}$) must be restated as inequality constraints with equal upper and lower limits because the existing formulation does not accept equality constraints other than the dynamics equations. The IPM version used in HPIPM can effectively handle such reformulation. It deals with inequality constraints by using slack variables. This solver deeply exploits the Riccati-based recursion, which is perfectly aligned with our problem (26).

4.1.7 Line Search

The line-search of our project is based on the filter line-search used in IPOPT [12]. Each update either improves the constraint satisfaction or the cost function. The constraint satisfaction $\theta(\mathbf{x})$ is measured by taking the norm of all constraints scaled by the time discretization.

$$\theta(\mathbf{x}) = \delta t \left\| \left[\mathbf{F}(\mathbf{x})^{\mathrm{T}}, \mathbf{H}(\mathbf{x})^{\mathrm{T}}\right]^{\mathrm{T}} \right\|_2 . \tag{27}$$

The focus changes to decreasing the constraints when the constraint is violated beyond a set threshold. The focus changes to minimizing costs when constraint violation is below a minimum threshold.

Algorithm 1 shows the line-search method of this project. There are three cases that a step can be accepted. First, when the behavior is at high constraint violation, a step will be rejected if the new constraint violation is above the threshold and worse than the current violation. Second, when both new and old constraint violations are lower than the threshold, and the current step is in a descent direction, we require that the cost decrease satisfies the Armijo condition. Finally, the primary acceptance condition is given where either a cost or constraint decrease is requested.

4.2 Analysis

In this study, we focus on the standing mode of a robot because its state is more constant and easier to analyze than when it is walking. Our results, shown in Fig. 6, reveal the required forces on each foot and the required joint velocities for this mode.

To solve the NLP, we employed the Sequential Quadratic Programming (SQP) method. When the change in the Lagrangian between two consecutive iterations was less than the specified tolerance

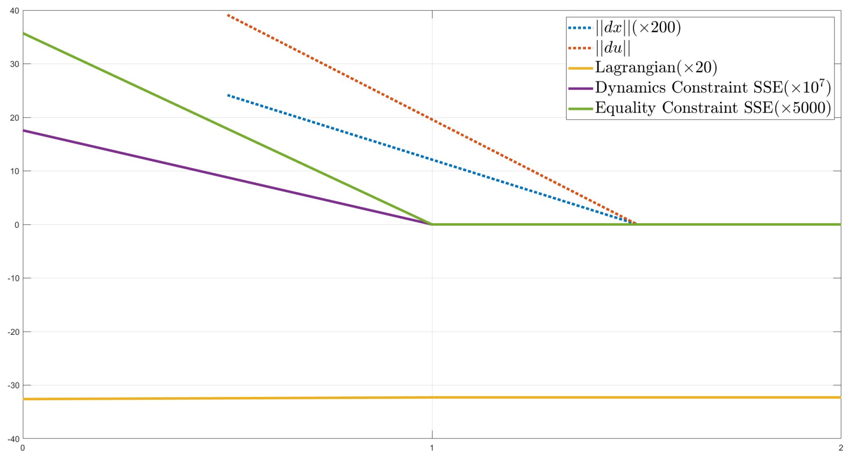

Figure 7: The performance of the solver with a good initial guess

(1e-4), the solver was terminated and the solutions were output. This indicates that the Karush-Kuhn-Tucker (KKT) condition was satisfied within the specified tolerance, and a local minimum was found.

However, due to the non-convexity of our problem, we cannot guarantee that the solver found the global minimum. Nevertheless, our solution is likely to be very close to the global minimum based on physical principles. Ideally, the control input should be $\left[0, 0, \frac{1}{4}mg, 0, 0, 0\right]$, which means that the ground reaction force on each foot should be a quarter of the robot's weight, be perpendicular to the ground, and all feet should have zero velocities. As shown in Fig. 6, the ground reaction forces on each foot are nearly equal, with each force around 126 Newtons and the robot's weight approximately 504 Newtons, which is just four times the force on each foot. The velocities of each joint were not exactly zero, but they were very close to that value.

The initial guess is an important factor in the optimization of high-dimensional, nonlinear problems such as this one. Through various tests, we have found that a good initial guess is essential in obtaining the results shown in Fig. 6. In this case, the guess for the state trajectory was the default standing state of the robot, while the guess for the control input trajectory was the most ideal input as previously mentioned, i.e., every knot point on the trajectory was $\left[0, 0, \frac{1}{4}mg, 0, 0, 0\right]$. Using this initial guess, the SQP algorithm was able to find the solution in just two iterations. Fig. 7 shows the scaled results of the change in various variables during the optimization process. After the first iteration, the Lagrangian and constraint violations became extremely small, and in the second iteration, the step length obtained from solving the quadratic programming problem was also very small. The whole SQP was completed in an average of 8 ms.

In addition to using a good initial guess, we also tested some very poor initial guesses. For instance, setting the initial guess to be a zero vector resulted in the solver needing five iterations to find a solution (as shown in Fig. 8). If we set all decision variables to 1, the solver was unable to find a solution and was terminated because the step size obtained from the line search became too small (<1e-4), while the equality constraints violation remained very large. This emphasizes the importance of a good initial guess in the optimization of high-dimensional, nonlinear problems.

We also conducted a detailed analysis of the effects of different initial guesses for the state and control inputs on the solution through various tests. First, we combined a good guess of the state trajectory with a guess of multiple input trajectories. We simply used a number multiplied by the optimal initial guess of the input trajectory and recorded the results (as shown in Fig. 9). In the left side of the figure, the multiplier is negative and the initial guess is infeasible because it is impossible to produce a negative ground reaction force. We can see that these significantly infeasible guesses had a worse effect on the solver's performance.

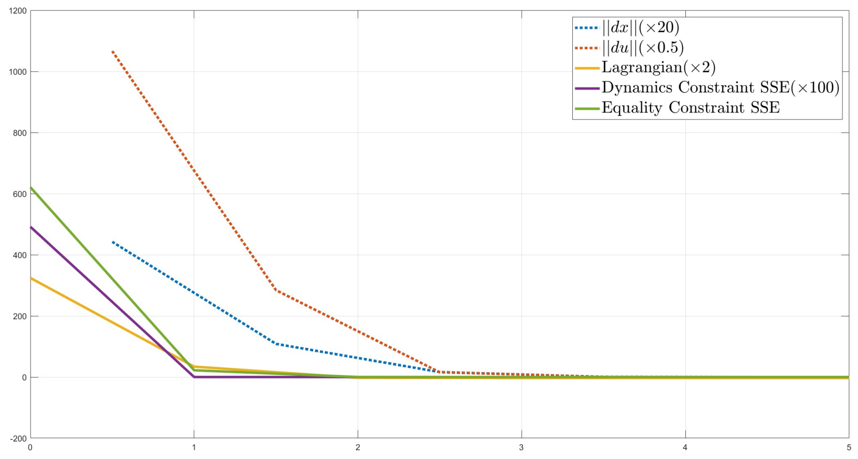

Figure 8: The performance of the solver with all zero vector initial guess

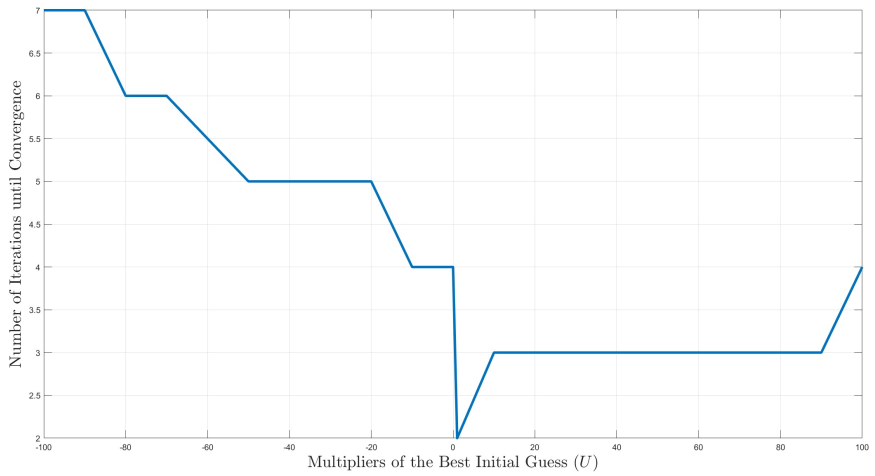

Figure 9: The performance of the solver with different control inputs guesses

Table 2: Solver's performance with different initial guesses of states

Initial Guess	$\leq -10S_{best}$	0	S_{best}	$10S_{best}$	$\geq 20S_{best}$
Number of Iterations until Convergence	Fail	5	2	6	Fail

Finally, we used the same method to investigate the sensitivity of the solver to initial guesses for the state. The results (Table 2) show that the solver is much more sensitive to guesses of the state than guesses of the control inputs. A poor initial guess for the state can quickly cause the solver to fail. This highlights the importance of carefully choosing the initial guess for the state in the optimization of this problem.

We have also conducted some studies on the Q and R parameters in the optimization problem. The parameters that we used and that have been tested to actually work are placed in Tables 3 - 8. It is challenging to carry out extensive parameter studies due to the large number of parameters involved. In our attempt to explore this issue, we implemented simple variations of the parameters in the Q matrix. One such variation involved setting the value of a specific parameter, such as p_base_x in Table 5, to 0. This resulted in the user's command to control the robot in the x-direction becoming ineffective, as expected. This can be understood by considering that a parameter of 0 will prevent the objective function from penalizing the tracking error for the corresponding state.

5 Summary

In this project, we focused on the control of dynamic walking for a quadruped robot. To achieve this, we formulated a trajectory optimization problem and utilized a nonlinear model predictive control scheme to determine the appropriate control inputs for a walking quadruped robot. The continuous-time problem was discretized into a NLP problem, which was then solved using a sequential quadratic programming algorithm. To ensure that the quadratic programming subproblem in each SQP iteration was convex, we employed a quadratic approximation to obtain a positive semi-definite Hessian. Additionally, constraint projection was used to eliminate equality constraints and accelerate the solution process. Through the use of the OCS2 Toolbox and the Robot Operating System (ROS), we were able to control the robot to stand and walk stably in simulation. Furthermore, we conducted various tests to analyze the properties of local and global optimality, initial guesses, and parameters. Our results indicate that the solver is able to find the local optimum and is likely to be very close to the global optimum. Additionally, we found that the solver is highly sensitive to initial guesses of the state trajectory. A range of feasible parameter values was obtained through manual tuning.

6 Contributions

- **Zixin Zhang:** Organize team meetings. Disceretization method. Part of the proposal and the final presentation. Implementation of the algroithm. Results analysis. Part of the final report (analysis of problem statement, analysis, summary).
- **Khai Nguyen:** Proposal (part of presentation, cost function, constraints). Report (part of presentation, problem statement, SQP formulation, QP solver)
- **Fukang Liu:** Responsible for the introduction part in the proposal, and line-search part in the final report.
- **Saurabh Borse:** Robot dynamics and constraints projection. Part of the proposal and final presentation. Part of the final report (abstract, introduction).
- **Tianyu Ren:** (Proposal) General NMPC formulation and centroidal dynamics model. (Final) Quadratic approximation. And corresponding part in the presentation.

7 Appendix

The code is available at GitHub: https://github.com/Atom990/24785-NMPC

Table 3: General parameters

Time Step	0.015 s
Maximum SQP Iterations	10
Friction Coefficient	0.1
ϵ	5.0
MPC Horizon	1.0 s
Tolerance of the Lagrangian gradient in the KKT condition	1e-4

Table 4: State weight matrix Q: centroidal momentum

Index	Value	Related State
(0,0)	15.0	vcom_x
(1,1)	15.0	vcom_y
(2,2)	30.0	vcom_z
(3,3)	5.0	L_x / robotMass
(4,4)	10.0	L_y / robotMass
(5,5)	10.0	L_z / robotMass

Table 5: State weight matrix Q: base pose

Index	Value	Related State
(6,6)	500.0	p_base_x
(7,7)	500.0	p_base_y
(8,8)	500.0	p_base_z
(9,9)	100.0	theta_base_z
(10,10)	200.0	theta_base_y
(11,11)	200.0	theta_base_x

Table 6: State weight matrix Q: joint positions

Index	Value	Related State
(12,12)	20.0	LF_HIP
(13,13)	20.0	LF_THIGH
(14,14)	20.0	LF_KNEE
(15,15)	20.0	LH_HIP
(16,16)	20.0	LH_THIGH
(17,17)	20.0	LH_KNEE
(18,18)	20.0	RF_HIP
(19,19)	20.0	RF_THIGH
(20,20)	20.0	RF_KNEE
(21,21)	20.0	RH_HIP
(22,22)	20.0	RH_THIGH
(23,23)	20.0	RH_KNEE

Table 7: Control weight matrix R: feet contact forces

Index	Value	Related State
(0,0)	1.0	left_front_force
(1,1)	1.0	left_front_force
(2,2)	1.0	left_front_force
(3,3)	1.0	right_front_force
(4,4)	1.0	right_front_force
(5,5)	1.0	right_front_force
(6,6)	1.0	left_hind_force
(7,7)	1.0	left_hind_force
(8,8)	1.0	left_hind_force
(9,9)	1.0	right_hind_force
(10,10)	1.0	right_hind_force
(11,11)	1.0	right_hind_force

Table 8: Control weight matrix R: foot velocity

Index	Value	Related State
(12,12)	5000.0	LF_x
(13,13)	5000.0	LF_y
(14,14)	5000.0	LF_z
(15,15)	5000.0	LH_x
(16,16)	5000.0	LH_y
(17,17)	5000.0	LH_z
(18,18)	5000.0	RF_x
(19,19)	5000.0	RF_y
(20,20)	5000.0	RF_z
(21,21)	5000.0	RH_x
(22,22)	5000.0	RH_y
(23,23)	5000.0	RH_z

Algorithm 1 Backtracking Line-Search

```
1: Hyperparameters: $\alpha_{\text{min}} = 10^{-4}$, $\theta_{\text{max}} = 10^{-2}$, $\theta_{\text{min}} = 10^{-6}$, $\eta = 10^{-4}$, $\gamma_\phi = 10^{-6}$, $\gamma_\alpha = 0.5$
2: $\alpha \leftarrow 1.0$
3: $\theta_k \leftarrow \theta(\mathbf{x}_i)$
4: $\phi_k \leftarrow \phi(\mathbf{x}_i)$
5: Accept ← False
6: while Not Accepted and$\alpha \geq \alpha_{\text{min}}$ do
7:     $\theta_{i+1} \leftarrow \theta(\mathbf{x}_i + \alpha\delta\mathbf{x})$
8:     $\phi_{i+1} \leftarrow \phi(\mathbf{x}_i + \alpha\delta\mathbf{x})$
9:     if $\theta_{i+1} > \theta_{\text{max}}$ then
10:        if $\theta_{i+1} < (1 - \gamma_\theta)\theta_i$ then
11:            Accept ← True
12:        end if
13:    else if $\max(\theta_{i+1}, \theta_i) < \theta_{\text{min}}$ and $\nabla\phi(\text{x}_i)^{\text{T}}\delta\text{x} < 0$ then
14:        if $\phi_{i+1} < \phi_i + \eta\alpha\nabla\phi(\text{x}_i)^{\text{T}}\delta\text{x}$ then
15:            Accept ← True
16:        end if
17:    else
18:        if $\phi_{i+1} < \phi_i - \gamma_\phi\theta_i$ or $\theta_{i+1} < (1 - \gamma_\theta)\theta_i$ then
19:            Accept ← True
20:        end if
21:    end if
22:    if Not Accepted then
23:        $\alpha \leftarrow \gamma_\alpha\alpha$
24:    end if
25: end while
26: if Accepted then
27:     $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \alpha\delta\mathbf{x}$
28: else
29:     $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i$
30: end if
```

References

[1] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters*, 6(3):4688–4695, 2021.

[2] Melanie N Zeilinger, Davide M Raimondo, Alexander Domahidi, Manfred Morari, and Colin N Jones. On real-time robust model predictive control. *Automatica*, 50(3):683–694, 2014.

[3] Ruben Grandia, Fabian Jenelten, Shaohui Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model predictive control. *arXiv preprint arXiv:2208.08373*, 2022.

[4] C. Jones, F. Borrelli, and M. Morari. Lecture notes in model predictive control, 2015.

[5] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters*, 6(3):4688–4695, 2021.

[6] David E Orin, Ambarish Goswami, and Sung-Hee Lee. Centroidal dynamics of a humanoid robot. *Autonomous robots*, 35(2):161–176, 2013.

[7] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619. IEEE, 2019.

[8] Ruben Grandia, Farbod Farshidian, René Ranftl, and Marco Hutter. Feedback mpc for torque-controlled legged robots. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4730–4737, 2019.

[9] Wikipedia. Runge–Kutta methods — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Runge%E2%80%93Kutta%20methods&oldid=1126399865`, 2022. [Online; accessed 13-December-2022].

[10] Christian Feller and Christian Ebenbauer. A stabilizing iteration scheme for model predictive control based on relaxed barrier functions. *Automatica*, 80:328–339, 2017. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2017.02.001. URL `https://www.sciencedirect.com/science/article/pii/S0005109817300559`.

[11] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range. *Automatica*, 47(10):2279–2285, 2011.

[12] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106 (1):25–57, 2006.

Nomenclature

$\boldsymbol{\lambda}$	Lagrange multipliers
$\boldsymbol{\tau}$	Joint torques
$\delta\mathbf{x}$	Optimal SQP step
δ	Relaxed barrier function parameter
ϵ	Friction cone smooth parameter
$\mathbf{B}$	Relaxed barrier function
$\mathbf{b}$	Nonlinear components
$\mathbf{F}(\mathbf{x})$	NLP initial and dynamic constraints
$\mathbf{F}_c$	Vector of stacked contact wrenches
$\mathbf{f}_c$	Ground reaction forces

$\mathbf{H}(\mathbf{x})$	NLP general equality constraints
$\mathbf{h}_{com}$	Centroidal momentum
$\mathbf{J}_c$	Matrix of stacked Jacobians
$\mathbf{M}$	Generalized Mass Matrix
$\mathbf{P}$	Constraint projection Variable
$\mathbf{p}$	Constraint projection Variable
$\mathbf{Q}$	State weight matrix
$\mathbf{q}_b$	Robot's body pose
$\mathbf{q}_j$	Joints position
$\mathbf{R}$	Control weight matrix
$\mathbf{r}_{com}$	Position of the contact point w.r.t the COM
$\mathbf{s}$	Robot states
$\mathbf{s}^{REF}$	Reference robot states
$\mathbf{u}$	Control inputs
$\mathbf{u}^{REF}$	Reference control inputs
$\mathbf{v}$	Joints velocity
$\mathbf{x}$	Decision variables
μ	Relaxed barrier function parameter
μ_c	Friction coefficient
ϕ	NLP objective function
g	Gravitational acceleration
m	Robot's mass
N	Horizon length
T	Horizon time