MODELING AND OPTIMAL CONTROL OF THE OPEN TORQUE-CONTROLLED QUADRUPED ROBOT SOLO-12

Niederberger Adi

Idiap-Com-02-2022

JULY 2022

Modeling and Optimal Control of the Open Torque-Controlled Quadruped Robot Solo-12

Master Thesis

Master in Artificial Intelligence

Author	: Adolf Kilian Niederberger
Student number	: 17-653-288
Project supervisor	: Sylvain Calinon
Company supervisor	: Philip Abbet

Abstract

Despite recent advances in controlling legged robots, modern quadruped robots are still not nearly as powerful as their biological counterparts. The biggest gaps exist in terms of robustness and versatility. The first is mainly caused by simplification in modeling the dynamics. On the other hand, the second point is forced by the complexity of the dynamics respectively its resulting challenging, and high computational optimization problems. Since these two requirements are in conflict with each other, a compromise must be found between the two.

Depending on the task and the requirements, it is more appropriate to neglect less or more in modeling. This work aims to test and apply different controllers on the quadruped robot Solo 12. The goal is to gain knowledge about which controllers are suitable for which tasks. Additionally, it should also be reported why a controller did fail and what could be improved to achieve a better result.

The robot is modeled based on centroidal dynamics or a simplified version of it to construct a whole-body controller. In addition, a method is used to plan feasible dynamical motions with the robot. Finally, there is also a Model Predictive Control (MPC) architecture tested.

The robot successfully performed a trotting walk, whereby the trajectory was generated with a whole-body dynamics planner. It was also shown how large the deviation from the simulation to reality is, although complex, non-linear models were used for the dynamics.

Acknowledgments

I would like to express my very great appreciation to the Idiap Research Institute for giving me the possibility to carry out my project here. Thereby I learned a lot of new, exciting things.

I would like to thank Dr. Sylvain Calinon for his excellent supervision and advice throughout this project.

I am grateful for the assistance given by Mr. Philip Abbet. I also want to thank all the robot learning and interaction group members, particularly Teguh Santoso Lembono, since he supported me a lot in my learning process.

Thanks to all the teachers and assistants at Idiap for allowing me to improve my knowledge and acquire new skills. Finally, my thanks go to all the people around me that have helped me in one way or another during this master's.

Martigny, June 24, 2022

List of Abbreviations

CoM	Center of Mass
FD	Forward Dynamics
FK	Forward Kinematics
ID	Inverse Dynamics
IK	Inverse Kinematics
LQR	Linear Quadratic Regulator
LQT	Linear Quadratic Tracking
MPC	Model Predictive Control
PID	Proportional–Integral–Derivative
QP	Quadratic Programming

List of Symbols

$\boldsymbol{I}_n$	Idendity Matrix $n \times n$
$\boldsymbol{g}$	gravitational acceleration vector
g	gravitational acceleration term

Contents

1. Introduction

Humanoids and other legged robots are considered a very exciting research topic in modern robotics, but it is also a very challenging subfield. Unlike manipulators with fixed bases, legged robots can move around, encountering surfaces or obstacles that are unknown to them. Therefore, a controller for such robotic systems should be adaptable and robust.

However, the challenge starts even earlier. Since legged robots have fewer actuators than degrees of freedom, they are underactuated. Furthermore, the nonlinear dynamics make modeling legged locomotion additionally difficult. These facts lead to complex physical behavior and cause significant computational challenges. Common approaches to address or overcome these problems are simplifying dynamic models, partitioning and splitting of problems, or a combination thereof. However, ignoring certain dynamic aspects to obtain a simpler model often has the disadvantage that capability decreases or the stability of the controller deteriorate.

This raises the question of what is a good compromise between model complexity and computational effort. Different control systems are examined in this work to investigate this issue, or at least to gain knowledge under which circumstances it makes more sense to deviate in one direction.

These controllers are combined with a quadruped robot to control it to perform trotting and jump motions.

Although the research of quadruped robots is largely based on the principle of bionics [1] and thus proposes cushioned legs [2], the robot Solo 12 [3] is used without suspension. Due to its low weight, the robot is not susceptible to vibration and exhibits leg stiffness similar to that of a human.

1.1. Structure of the Thesis

This thesis is composed of four more chapters.

Chapter 2 describes the robot and its technical properties, and a second part introduces basic concepts of robotics used in this work.

Chapter 3 delineates the methods used to generate motion planning as well the different control-architectures for the applied controllers.

Chapter 4 consist of the description of how the controllers are used and what results were obtained in the simulation and also in reality.

Chapter 5 sums up the insights and findings of this work and suggests and suggests possible changes for improvements.

2. Theoretical Background

The theoretical background is spitted into two main parts. First the robot and its technical properties are described.

The second part describes general robotics concepts that are used to control legged robots.

2.1. Description of the robot

In this section the robot used for the experiments is described. The technical properties of the robot are introduced and the software that is employed, as well as the calibration procedure is explained.

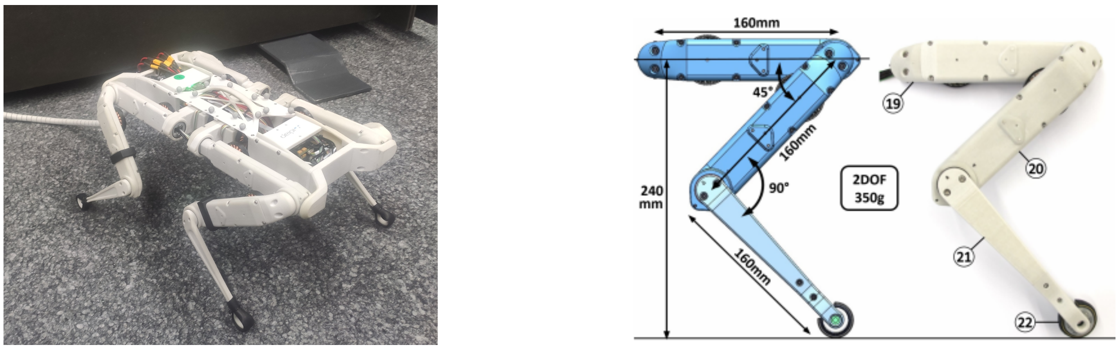
2.1.1. Fundamentals of the Robot

A four-legged mobile robot, a so-called quadruped robot, is used for the experiments to gain knowledge and thus implement the project. The robot, named Solo 8, was developed by a team from the NYU Tandon School of Engineering and an institute of the Max Planck Society[1]. Under the supervision of the Open Dynamic Robot Initiative[2], the robot was further developed in the areas of software and control. The initiative promoted the robot and made all research results and software components available as open source.

[1]`http://www.is.mpg.de/` (05.2022)

Meanwhile, a successor model of the robot has been developed, the Solo 12 robot (Figure 2.3), which is equipped with more joints and can also walk sideways compared to the Solo 8 (Figure 2.1a). However, since hardly any experiments and research results are available, findings from the Solo 8 will be used for the time being, and this robot will also be presented. However, it should be kept in mind that the results should also be transferable to the Solo 12. Due to the additional degrees of freedom, more possibilities are available, but the optimizations become more complex.

2.1.2. Technical Basics of the Robot



(a) Quadruped Solo 12 (b) 2-DOF leg

Figure 2.1.: **(a)** Quadruped robot Solo 12, it is the robot which has lateral hip joints, in comparison to Solo 8, and can therefore move sideways. **(b)** Illustration of a 2-DOF leg with hip, corresponding to Solo 8 ⑲, upper-hip ⑳, and lower leg module ㉑, and foot contact switch ㉒

The centerpiece of the robot is the leg architecture with the ability for sensors to interact, allowing for dynamic impedance and force control. The robot achieves spring-like behavior by using torque-controlled motors for leg manipulations, which corresponds to similar characteristics to muscles and elastic tendons of animal legs.

One leg of the quadruped (Figure 2.1b) consists of a hip section ⑲, a thigh ⑳ and a lower leg ㉑, with two identical brushless actuator modules used to control the hip and

[2]`https://open-dynamic-robot-initiative.github.io/` (06.2022)

knee joints. At the end of the lower leg, the foot (22) is distally attached as a foot contact sensor.

All joints are multiturn capable but limited to about three rotations due to cable routing. The robot is symmetrically constructed, which does not matter which part of the robot is on top and represents the back. Likewise, it allows the robot to perform leg manipulations with its back lying on the floor. In the bottom row of Figure 2.2, the sequence of positions is shown, which is made possible by the symmetry of the robot. The figure also shows various possible joint configurations.

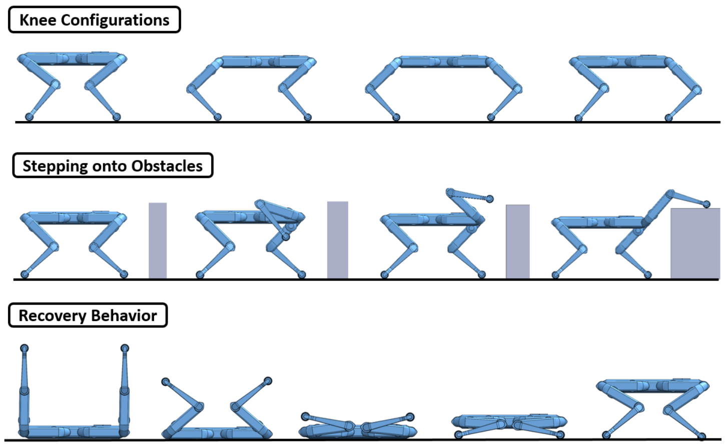


Figure 2.2.: Illustration of possible leg and knee configurations of the Quadruped Solo 12.

The foot contact sensor covers a detection range of 270°. The sensor, constructed with the help of a light-emitting diode and a light sensor, can thus sense different and complex environments. The sensor has a low sensitivity, which can also be adjusted mechanically and is characterized by a low response time. Furthermore, it is very robustly built, and thus withstands even high unpredictable shock loads.

The robot is powered by cable. The complete control of the elements is combined by a lightweight main-board control computer and can be controlled with Wifi or an Ethernet cable. The total weight of the robot is specified at 2.2 kg. From its standing height of 24 cm, the robot can jump up to 65 cm [3].

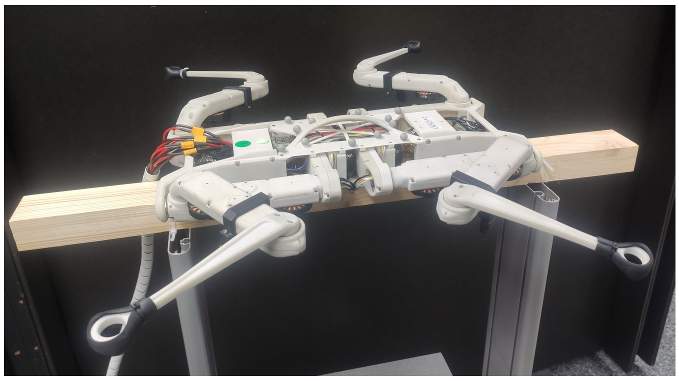


Figure 2.3.: Quadruped robot Solo 12 laid down on a trestle. Through the laid-out legs, it is visible how the hip joints work. Compared to the Solo 8, the robot has a lot of more possibilities to move.

2.1.3. Joints of the Robot

As mentioned in subsection 2.1.2, each leg of the robot consists of 3 joints. To control the robot in later tasks, it is necessary to define the order of the legs and so too for the joints. The leg names, their abbreviations, and the joint labels are illustrated in Figure 2.4.

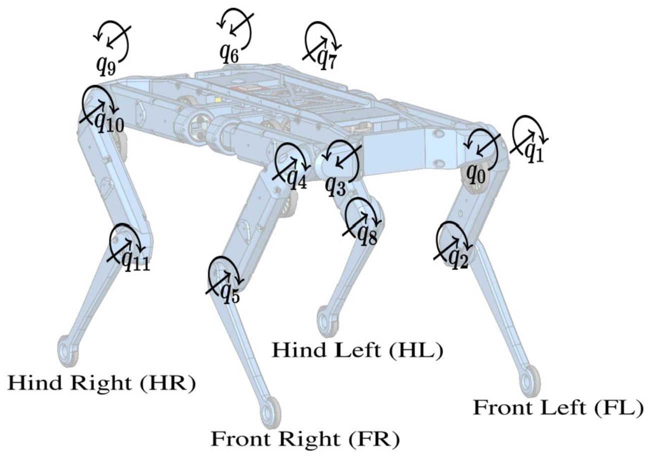


Figure 2.4.: Definition of the legs and joints of Solo 12.

The order will always be the same to control the robot in simulation or real environments. The sequence of the legs is ordered as in (2.1). The first joint for each leg is always the hip joint for hip adduction and abduction. The next joint applies hip extension and flexion, and the third joint executes extension and flexion to the lower leg.

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_{\mathrm{FL}} & \boldsymbol{q}_{\mathrm{FR}} & \boldsymbol{q}_{\mathrm{HL}} & \boldsymbol{q}_{\mathrm{HR}} \end{bmatrix} = \begin{bmatrix} q_0 & \dots & q_{11} \end{bmatrix} \tag{2.1}$$

As mentioned, the robot is controlled by torque. To control a joint, its motor must be activated by the current. For a single joint, the desired torque to be applied can be converted into motor current by the relationship:

$$\tau_i = I_i \, K_T \, N$$

where,

- I_i is the motor current denoted in *Ampere* [A]
- N is the gear reduction, here $N = 9$
- K_T is the torque constant and defines the torque-current relationship of the motor, here $K_T = 0.025\,\mathrm{N\,m/A}$

which leads to:

$$I_i = \frac{\tau_i}{K_T \, N} = \frac{\tau_i}{0.225\,\mathrm{N\,m/A}} \tag{2.2}$$

2.1.4. Communication and Control Software of the Robot

As mentioned, all communication between the robot's motor drivers and the off-board control computer is combined on a master board. The control of the robot with a PC is done via the network interfaces with an Ethernet cable or wirelessly via WiFi. When controlling WiFi, acknowledgment (ACK) should be disabled, and multicast should be used to ensure deterministic transmission time and prioritize new data packets.

The quadruped can be remotely controlled in real-time from a PC running Linux and using Preempt RT patch. The sampling control loop is performed at 1 kHz. Through a C++ software package, an API is provided to control the motor boards. The API can also be controlled with Python bindings, allowing fast prototyping [3].

2.1.5. Calibration of the Robot

Before the robot can be used, the ground truth between the joint position and the index position of the corresponding motor must be determined for all joints. This is ensured by performing a calibration of the robot. The calibration has to be executed repeatedly when the robot has been disconnected from the power supply. The reason for this is that the encoders are incremental, so due to the gear ratio (see 2.1.3), the encoder detects nine index pulses per joint revolution.

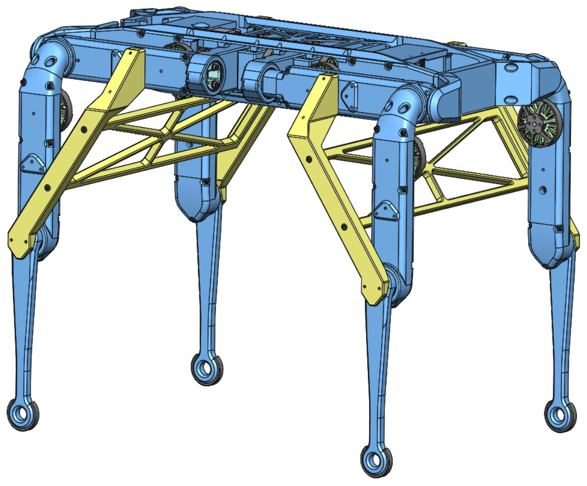


Figure 2.5.: Calibration tool attached to the robot in order to keep all joints in zero position and perform accurate calibration[3].

To conduct a calibration procedure, all robot joints must be brought to zero position, as demonstrated in Figure 2.5, before the robot is connected to the power supply. In order to precisely set the joints to the initial position, the calibration tool should be used.

Once power is applied to the robot, the software searches by moving the joints for the nearest index for each motor. When the index is detected, the offset between the zero configuration and the index position is saved and used as a compensation term so that the actual zero position can always be found uniquely. A problem can be caused when the zero position and the index on the encoder are close together. In that case, the index cannot be correctly detected. The issue becomes noticeable when a joint does not adopt the desired starting position after calibration. It can be solved by disconnecting the robot from the power supply, rotating the joint causing the problem, and rerunning the calibration.

2.2. State of the Robot and its Frames

Unlike manipulators such as an industrial robot arm whose base is rigidly attached to a fixed support, the base of quadrupeds moves because they can locomote. They have a floating base, whereas a manipulator is a fixed base robot.

In the case of a robot with a floating base, there is a need to express its position and orientation with respect to a reference frame. Generally, this frame is called an inertial frame. In robotics, it is often denoted as the global frame or world frame. Further, the base frame is often called the body frame.

Since the base frame is relative to the robot, it is a local frame. It is often convenient to define other local frames to keep some sub-tasks simple and avoid unnecessary transformations. In terms of an application with a solo, for example, for a manipulation involving only one leg, a local frame is defined at the shoulder of the corresponding leg. For that task, it may be interesting to describe the end position of the leg, also often denoted as end-effector, with respect to the local frame. In Figure 2.6 for the front left leg a local frame l_{A} is defined as an example.

Finally this allows to express the robots pose in the world coordinate frame as follows:

[3] `https://github.com/open-dynamic-robot-initiative/open_robot_actuator_hardware/blob/master/mechanics/general/robot_calibration.md#robot-calibration` (06.2022)

$$^{o}\boldsymbol{\mathcal{P}} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^{\mathsf{T}}$$

Where x, y and z are expressed in Cartesian coordinates and the orientation is declared with Trait-Bryan Euler angles. In which ϕ stands for roll, θ for pitch and ψ for yaw. Sometimes the orientation of the base is also expressed with quaternions. In that case ϕ, θ, ψ would be replaced by ϵ_0, ϵ_1, ϵ_2, ϵ_3, so that $^{o}\boldsymbol{\mathcal{P}} \in \mathbb{R}^7$

In order to express the state of the robot not only the pose $^{o}\boldsymbol{\mathcal{P}}$ but also the joint configurations $\boldsymbol{q}$ (2.1) are necessary. By concatenating them, the state of the robot at a specific time t can be described as follows:

$$^{o}\boldsymbol{x}_t = \begin{bmatrix} ^{o}\boldsymbol{\mathcal{P}} \\ \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} x & y & z & \phi & \theta & \psi & q_0 & \dots & q_{11} \end{bmatrix}^{\mathsf{T}} \tag{2.3}$$

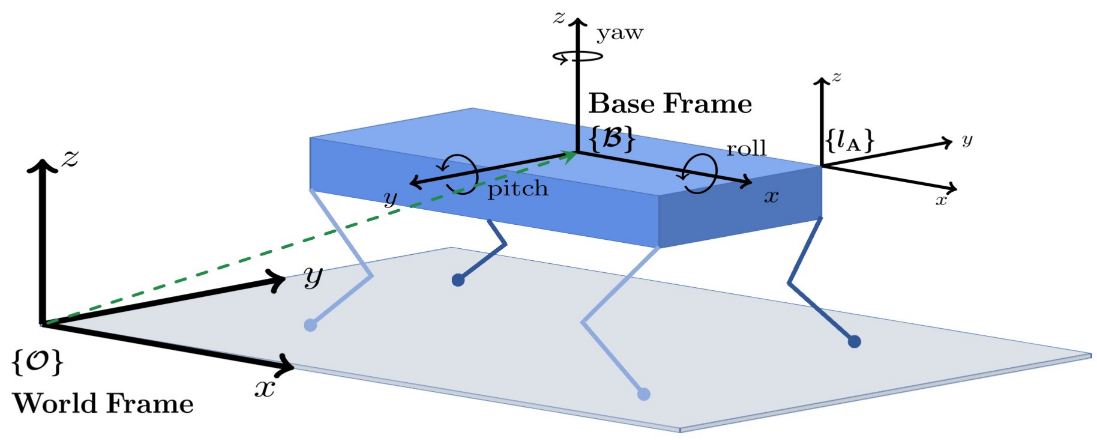


Figure 2.6.: Illustration of possible leg and knee configurations of the Quadruped Solo 12.

2.3. General Robotic Concepts

In this section, general basic concepts used in robotics are introduced. For simplification, often a fixed robot arm is often assumed to derive a method. Most of the time, these concepts can be either directly or with some constraints adapted to legged robots. Whereas including such constraints can cause very complex behavior in the model, not all concepts are introduced specifically for legged robots here.

The information in this chapter has been compiled from these sources [5, 4].

2.3.1. Robot Kinematics

Generally, robot kinematics relies on the study of the movement of robotic systems based on multi-degree of freedom kinematic chains. It is assumed that the robot's links are rigid bodies, and all joints apply either pure rotation or translation. The main goal is to have a concept that describes the relationship between the robot's joint coordinates and its spatial layout. These two-state spaces are often denoted as joint space and task space. Sometimes the term workspace is used as a synonym for task space, but mostly it is referred to as the set of points and orientation configurations the robot's end-effector can reach.

Kinematics plays a crucial role in many tasks in robotics. For example, to describe a path and its joint configurations, move a manipulator's robot's end-effector from position A to B, or check a given sequence of joint configurations if its execution would cause a collision with obstacles.

For legged robots, the concepts of kinematics are mainly used for movements of the legs, such as for locomotion, during a foot is in the swing phase. However, also in combination with robotic dynamics, it is essential to have kinematics.

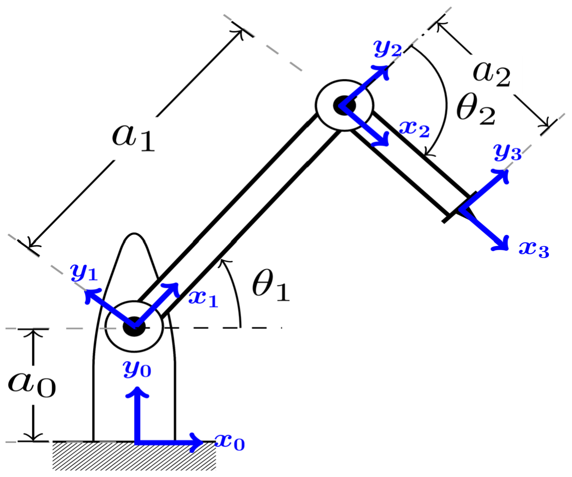


Figure 2.7.: Forward Kinematics (FK) of a two-link planar manipulator, with the attached reference frame on its links. The z-axes all point out of the page.

2.3.1.1. Forward Kinematics

FK refers to the computation of resulting motions respectively position $\boldsymbol{x}(t)$ in task frame from a joint-angle motion or configuration $\boldsymbol{q}(t)$.

Mathematically, it can be written as a function FD such that:

$$\begin{aligned} &\text{FK}: \mathbb{R}^n \rightarrow \mathbb{R}^6, \\ &\boldsymbol{x}(t) = \text{FK}(\boldsymbol{q}(t)) \end{aligned} \tag{2.4}$$

A detailed description about FK and how to solve it, is available in the appendix A.3.

2.3.1.2. Inverse Kinematics

In many robotics applications, an action is defined in task space, and the robot should perform it by applying a control command. That may be moving the end-effector along a specified path or, in the case of a legged robot, lifting a leg to reach the desired position.

Since a robot applies commands with its actuators and their positions are represented in joint space, a method is necessary to translate task space coordinates to the joint space. The concept for that is called Inverse Kinematics (IK) which carries out the opposite mapping of FK such as:

$$\begin{aligned} \mathrm{FK}^{-1}&\colon \mathbb{R}^6 \to \mathbb{R}^n, \\ \boldsymbol{q}(t) &= \mathrm{FK}^{-1}(\boldsymbol{x}(t)) \end{aligned} \tag{2.5}$$

In general, unlike FK, IK cannot be solved in a closed-form manner. There may exist multiple solutions, an infinite number of solutions, or even no possible solution is available to fulfill an IK problem.

A common way to solve the IK problem is described in detail and presented in the appendix A.4.

2.3.2. Robot Dynamics

In the previous section about robot kinematics, it was described how things move in a robotic system. On the other hand, robot dynamics is concerned with the forces and torques which are responsible for the motion. In robotics, there are the two main problems related to dynamics:

- **Forward Dynamics (FD):** given the forces, compute the accelerations (see 2.3.2.3).
- **Inverse Dynamics (ID):** given the accelerations, compute the forces (see 2.3.2.4).

To understand why and how something in the system is moving, we look at the equation of motion of a general physical system as well as for a robot mechanism. In the case of a physical system, it describes the motion as a function of time and external optional control inputs as follows:

$$\boldsymbol{F}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t), \boldsymbol{u}(t), t) = \boldsymbol{0}\,, \tag{2.6}$$

where:

- t is the variable indicating the time,
- $\boldsymbol{q}$ is the vector for generalized coordinates of a system, e.g. the vector of joint-angles for a manipulator,
- $\dot{\boldsymbol{q}}$ is the velocity (first time-derivative) of $\boldsymbol{q}$,
- $\ddot{\boldsymbol{q}}$ is the acceleration (second time-derivative) of $\boldsymbol{q}$,
- $\boldsymbol{u}$ is the vector of control inputs.

2.3.2.1. Case of Manipulators

For some controlling tasks, it makes sense to see a single leg of the quadruped as a separate fixed-base robot arm. We assume that the robot's whole body is fixed to the environment, and the constraint is strong enough to hold back any force applied to it. With this supposition, the equation of motion for a manipulator can be written as follows:

$$\boldsymbol{\tau}_q = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_{\text{ext}} \, , \tag{2.7}$$

where, $n = \dim(\boldsymbol{q})$ denotes the degree of freedom of the system,

- $\boldsymbol{\tau}_q \in \mathbb{R}^n$ is the vector of actuated joint torques,
- $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is the inertia matrix (symmetric positive-definite),
- $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^n$ is the vector with the Coriolis and centrifugal terms,
  with $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^{\intercal}\boldsymbol{\Gamma}(\boldsymbol{q})\dot{\boldsymbol{q}}$,
  where $\boldsymbol{\Gamma}(\boldsymbol{q}) \in \mathbb{R}^{n\times n\times n}$ represents the Coriolis tensor,
- $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ is the vector of joint torques caused by gravity,
- $\boldsymbol{\tau}_{\text{ext}} \in \mathbb{R}^n$ is the vector with external torque caused by extrinsic force from the environment.

Compared to the general form of the equation of motion mentioned above, the formula for the manipulator is time-invariant. That means based on the current state of $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and the applied torque $\boldsymbol{\tau}$ it is possible to compute directly the resulting joint accelerations $(\ddot{\boldsymbol{q}})$. So the system has no memory, independently of how the system reached a configuration, based on the current state and the given torque, the system will always accelerate in the same manner.

2.3.2.2. Case of Legged Robots

Legged robots such as humanoids or quadrupeds are not fixed to the environment and are denoted as floating base systems. Further, their legs can have or break contact with the environment. In order to describe its state, besides the vector of the joint configurations, the position and orientation of the floating base frame with respect to the global frame are also necessary.

Thus, to describe the robots state with n actuated joints, leads to $\boldsymbol{q} \in \mathbb{R}^{\boldsymbol{n+6}}$. Since the vector of actuated torques $\boldsymbol{\tau}$ has just n entries, fewer degrees of freedom than the system itself, the system is underacted. It is impossible to control all degrees of freedom directly.

Contact with environment

Kinematically, a contact of a leg with the environment can be expressed as equality as follows:

$$\boldsymbol{p}_{C_i}(\boldsymbol{q}) = \boldsymbol{p}_{D_i},$$

where C_i is the position of a robot's point in the inertial frame that is in contact with the environment (i.e., a leg is in contact with the ground) and D_i expresses a fixed point in the same frame. The point fixes position even so velocities and accelerations:

$$\dot{\boldsymbol{p}}_C = 0 \;\Rightarrow\; \boldsymbol{J}_C(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$
$$\ddot{\boldsymbol{p}}_C = 0 \;\Rightarrow\; \boldsymbol{J}_C(\boldsymbol{q})\ddot{\boldsymbol{p}} + \dot{\boldsymbol{q}}^{\mathsf{T}}\boldsymbol{H}_C(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

$\boldsymbol{J}_C$ is the Jacobian of the contact constraint, and $\boldsymbol{H}_C$ its Hessian.

If the robot would have n_c such contacts with the environment, there would be k constraints and the equation of motion becomes to [7]:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{S}^{\mathsf{T}}\boldsymbol{\tau} + \sum_{i=1}^{n_c} \boldsymbol{J}_{C_i}^{\mathsf{T}}\boldsymbol{f}_i \tag{2.8}$$

where, $n_q = n + 6$,

- $\boldsymbol{S} \in \mathbb{R}^{n \times n_q}$ is the selection matrix of actuated joints.

  In case the six floating base coordinates are the first entries of $\boldsymbol{q}$,

  then $\boldsymbol{S} = [\boldsymbol{0}_{6\times n}\ \boldsymbol{I}_n]$,
- $\boldsymbol{J}_{C_i} \in \mathbb{R}^{3 \times n_q}$ is the Jacobian of the location C_i of the position vector $\boldsymbol{p}_{C_i}$,
- $\boldsymbol{f}_i \in \mathbb{R}^3$ is the vector of the external forces acting on the robot at the contact point C_i.

The contact points C_i are located at the interface between the robot's limbs and the environment. Suppose there is a surface contact and the robot's surface is in full contact with an environmental surface. In that case, it is sufficient to take only the contact points C_i at the vertices of the contact polygon instead of the whole surface [6].

2.3.2.3. Forward Dynamics

FD also known as direct dynamics, is the general term for calculating resulting motion from forces. Based on a robot system with a given configuration $\boldsymbol{q}$, generalized velocity $\dot{\boldsymbol{q}}$, applied joint torques $\boldsymbol{\tau}$ and contact force $\boldsymbol{f}$, the resultant joint accelerations $\ddot{\boldsymbol{q}}$ is computed so that the constrained equations of motion are satisfied.

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \boldsymbol{S}^{\mathsf{T}}\boldsymbol{\tau} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_{\text{ext}} + \boldsymbol{J}_C(\boldsymbol{q})^{\mathsf{T}}\boldsymbol{f} \\ \boldsymbol{J}_C(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^{\mathsf{T}}\boldsymbol{H}_C(\boldsymbol{q})\dot{\boldsymbol{q}} &= \boldsymbol{0} \end{aligned} \tag{2.9}$$

It can be mathematically written as function FD such that:

$$\ddot{\boldsymbol{q}} = \text{FD}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{f})$$

FD is mainly used for simulation-related tasks. For example, the motion of a robot in a simulation environment should be represented and tracked, depending on a given applied torque input that is applied on its actuators. Generally, forward dynamics problems are characterized by high computational complexity.

2.3.2.4. Inverse Dynamics

ID is concerned with the calculation of forces from motions. By satisfying the constrained equation of motion (2.9), the joint torques $\boldsymbol{\tau}$ and the external force $\boldsymbol{f}$ is worked out based on a robot system with a given configuration $\boldsymbol{q}$, generalized velocity $\dot{\boldsymbol{q}}$ as well the generalized acceleration $\ddot{\boldsymbol{q}}$.

It can be mathematically written as function FD such that:

$$(\boldsymbol{\tau}, \boldsymbol{f}) = \text{ID}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$$

In case of the linear optimization problem is fully determined, the expression is well-defined. For a fixed robot arm, this is often satisfied, but so mostly not for legged robots. A common approach, in that case, is to shift the external force component, for example, to a contact model, where the force is given. Thus the joint torques can be calculated. Mathematically, the function can now be formulated such that:

$$(\boldsymbol{\tau}) = \text{ID}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{f})$$

2.3.2.5. Algorithms to Solve Dynamics Problems

In robotics dynamics application, the most commonly used main algorithms are the following three [8, 9] :

- **RNEA:** The recursive Newton-Euler algorithm solves the inverse dynamics problem and grows with a complexity $\mathcal{O}(n)$. It allows to calculate $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q})$ from (2.8), respectively by modification this terms separately.
- **ABA:** The articulated-body algorithm, is used to calculate the unconstrained forward dynamics. It has a computational complexity of $\mathcal{O}(n)$.
- **CRBA:** The composite-rigid-body algorithm can be used to compute the joint space inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ based on the given joint configuration. The computational complexity is of $\mathcal{O}(n^2)$. It can also be used for FD in combination with RNEA by calculating $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q})$ first, and solve then $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{\tau} - (\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_{\text{ext}})$ with respect to $\ddot{\boldsymbol{q}}$ (i.e. based on (2.7)). This leads to a computational complexity of $\mathcal{O}(n^3)$.

Libraries Based on Dynamics Algorithms

The rigid body dynamics library Pinocchio, which is used for some controlling tasks of Solo, relies on these three algorithms depending on the optimization task. The simulation environment that is used, PyBullet with its dynamic simulator Bullet, uses ABA. In comparison, the inverse dynamics computation is based on RNEA. The library TSID, which is used for solving inverse dynamics-related problems in task space, is based on RNEA too.

2.3.2.6. Dynamics in the Task Space

An alternative form to represent the equation of motion is the operational space formulation. For some applications, it is more convenient to express the dynamics in the task space, e.g., the end-effector coordinate system, instead of in the space of joint motions, respectively, joint forces and torques. The following system assumes a fixed base robot arm, wherein the task space position and orientation is considered. Therefore in the equation, wrenches (force, torque) are equalized.

To describe the end-effector dynamics in that way, $\boldsymbol{J}(\boldsymbol{q})$ must be invertible to ensure a unique mapping between joint velocities and twists of the end-effector. The operational space dynamics can then be expressed such as:

$$\boldsymbol{\Lambda}(\boldsymbol{q})\dot{\boldsymbol{\mathcal{V}}} + \boldsymbol{\mu}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\rho}(\boldsymbol{q}) = \boldsymbol{\mathcal{F}}\,,$$

with:

$$\begin{aligned} \boldsymbol{\Lambda}(\boldsymbol{q}) &= \left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{J}(\boldsymbol{q})^{\mathsf{T}}\right)^{-1}, \\ \boldsymbol{\mu}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \boldsymbol{\Lambda}(\boldsymbol{q})\left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}\right), \\ \boldsymbol{\rho}(\boldsymbol{q}) &= \boldsymbol{\Lambda}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{g}(\boldsymbol{q})\,, \end{aligned}$$

where:

- $\dot{\boldsymbol{\mathcal{V}}} \in \mathbb{R}^6$ is the time derivative of the Twist of the end-effector with, $\dot{\boldsymbol{\mathcal{V}}} = [\boldsymbol{a}^{\mathsf{T}}, \dot{\boldsymbol{\omega}}^{\mathsf{T}}]^{\mathsf{T}}$ with linear acceleration and angular acceleration,
- $\boldsymbol{\Lambda}(\boldsymbol{q}) \in \mathbb{R}^{6\times6}$ represents the mass inertia matrix for the end-effector,
- $\boldsymbol{\mu}(\boldsymbol{q}) \in \mathbb{R}^6$ is the vector that contains Coriolis and centrifugal force terms in the task space,
- $\boldsymbol{\rho}(\boldsymbol{q}) \in \mathbb{R}^6$ is the vector with gravity terms,
- $\boldsymbol{\mathcal{F}} \in \mathbb{R}^6$ is the vector representing the end-effector's wrench.

Further, the components $\boldsymbol{\Lambda}$, $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ are expressed in terms of the joint configuration and not the end-effector configuration because there may be several joint configurations to fulfill the given end-effector configuration. The wrench acting on the end-effector can be translated into joint space torque as follows:

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathsf{T}}(\boldsymbol{q})\boldsymbol{\mathcal{F}}$$

2.4. Control Theory in Robotics

In the previous sections, the basics of robotics systems were presented. This allows us to describe how a robot reacts based on the current state and given input. Mainly the goal is to control the robot to apply the desired task.

There is maybe the question of why is there a need for a controller because, as described in 2.3.1.2, IK is already a tool that allows, for example, to reach a robot arm the desired position in task space, if joint positions control the robot. This thought might be reasonable, but ordinarily, a task should be executed with controlled movements and in an optimal way. In addition, sometimes, a tool is required to respond to a perturbation so that the robot can still comply with its task. The controller usually performs this part in a robot control system. Various kinds of controllers mainly differ by complexity and capacity.

This section introduces the most common controllers in robotics, revealing their main properties. Furthermore, it is shown how to construct a controller based on a linear dynamical system and different ways to obtain the optimal control commands.

2.4.1. Linear Dynamical Systems

When a robot is in action, the state changes continuously over time. A dynamical system is often set up to track or model a robot. The state can include different terms depending on the model or the system. In the case of a quadruped, this could be the joint configuration of one leg and its velocities, the pose and its velocity, or even the whole state vector as described in (2.3) including its velocity. The velocity term doesn't have to be included, but in most application it is considered, since it allows also to express the behavior in terms of its momentum.

The standard representation for modeling a dynamical system is an equation associating the state and control input with the derivation of the state. It is a representation of the equations of motion (2.6) and can be written:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \,,$$

where:

- $\boldsymbol{x} \in \mathbb{R}^n$ is the state variable,
- $\boldsymbol{u} \in \mathbb{R}^m$ indicates the control input variable.

In general, the function f can be nonlinear with respect to its arguments. Preferably, the dynamics were modeled linearly so that the dynamical system can be described as a linear, time-invariant (LTI) system. Unfortunately, in practice, this is often not the case. A common approach to handle this problem is to simplify the system so that the main characteristics are still preserved, and some other effects can be neglected. In state-space representation with time-varying dynamics, it takes the form of:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}_t^c \boldsymbol{x}(t) + \boldsymbol{B}_t^c \boldsymbol{u}(t)\,, \tag{2.10}$$

where:

- $\boldsymbol{A}_t^c \in \mathbb{R}^{n \times n}$ is the system matrix,
- $\boldsymbol{B}_t^c \in \mathbb{R}^{n \times m}$ is the input matrix.

To transfer the system to the trajectory level, in order to describing the evolution of the system in a vector form after, the dynamics can be discretized:

$$\boldsymbol{A}_t = \boldsymbol{I} + \boldsymbol{A}_t^c \Delta t, \quad \boldsymbol{B}_t = \boldsymbol{B}_t^c \Delta t, \quad \forall t \in \{1, \ldots, T\},$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t\, \boldsymbol{x}_t + \boldsymbol{B}_t\, \boldsymbol{u}_t \tag{2.11}$$

2.4.1.1. Open-Loop and Closed-Loop Control

In general, a distinction is made between two primary control architectures. The main difference between the two is feedback. On the one hand, open-loop control systems don't get feedback and operate just with inputs (Figure 2.8). On the other hand closed-loop control systems get feedback from the output-state (Figure 2.10) [10].

In the block diagrams to represent the different control architectures, the following notations are used:

- $\boldsymbol{x}_d(t)$: desired signal or reference, in literature often denoted as $\boldsymbol{r}(t)$
- $\boldsymbol{x}(t)$: output of the process, often referred as $\boldsymbol{y}(t)$
- $\boldsymbol{x}_{\text{est}}(t)$: feedback term, could be i.e. measurement or estimation of the signal
- $\boldsymbol{e}(t)$ error term i.e. $\boldsymbol{e}(t) = \boldsymbol{x}_d(t) - \boldsymbol{x}(t)$
- $\boldsymbol{u}(t)$ control input to the system
- $\boldsymbol{d}(t)$ disturbance term acting on the system

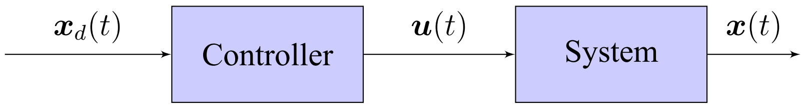


Figure 2.8.: Schematic diagram of the open-loop control system, which is also denoted as a non-feedback system.

Because open-loop controllers often require some human intervention, they are rarely used in robotics as stand-alone controllers. In the category of non-feedback systems, there exists a subcategory called feed-forward control systems (Figure 2.9). This control design can adjust its input based on known disturbances. Often the disturbances are measured or modeled and passed to the controller, which can adjust the original control command to compensate for the perturbation acting on the system.

Although open-loop controllers do not provide feedback, they are an essential tool in robotic control systems. Mainly they were used in combination with closed-loop systems, as later shown and applied, and represent a powerful utility tool. Likewise, it plays an essential role as a planning tool; it is often easier to compute an open-loop trajectory.

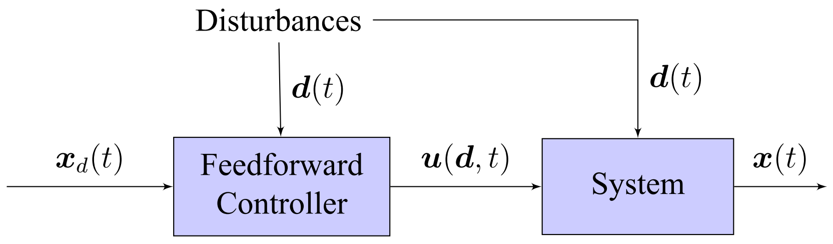


Figure 2.9.: Feed-forward control architecture, since this controller does not get any feedback, it belongs to the category of open-loop controllers.

The main feature of closed-loop control is that the control function can observe the system's state by feedback, for example, in the form of sensor output. This property allows the robot to react to disturbance and change the control input accordingly to achieve the desired task. The controller is a fully automatic control system since its control action depends on the output.

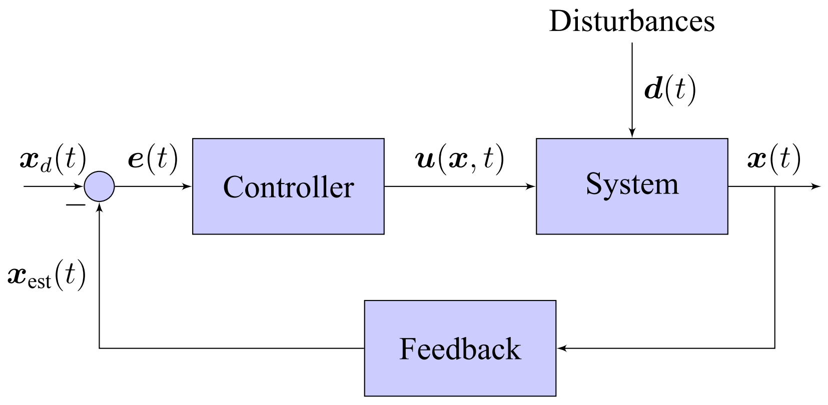


Figure 2.10.: Control architecture of a closed-loop controller.

Despite the previously mentioned advantages, the closed-loop controller also has its limitations. Depending on how feedback is provided, a delay, or the noise-term of a sensor measurement, can cause problems. So the feedback may lead to an oscillatory response or event to a loss of stability. Two closed-loop control systems are introduced below.

2.4.2. PID Controller

The Proportional–Integral–Derivative (PID) controller is a feedback control loop mechanism, consisting of proportional, integral, and derivative terms (Figure 2.11). The PID controller can be written in the general continuous form:

$$\boldsymbol{u}(t) = \boldsymbol{K}_p\,\boldsymbol{e}(t) + \boldsymbol{K}_i \int_0^t \boldsymbol{e}(t')\,\mathrm{d}t' - \boldsymbol{K}_d \frac{\mathrm{d}\boldsymbol{e}(t)}{\mathrm{d}t}, \tag{2.12}$$

where $\boldsymbol{K}_p,\ \boldsymbol{K}_i$ and $\boldsymbol{K}_d \in \mathbb{R}^{n\times n}$ are positive-definite gain matrices. The physical meaning of the three consisting parts can be seen as follows:

- **P:** implies the present effort to transform a present state into a desired state.
- **I:** describes the accumulated effort taking into account the experience information of the previous states.
- **D:** reflects the prediction effort, which is the information about trends in future states.

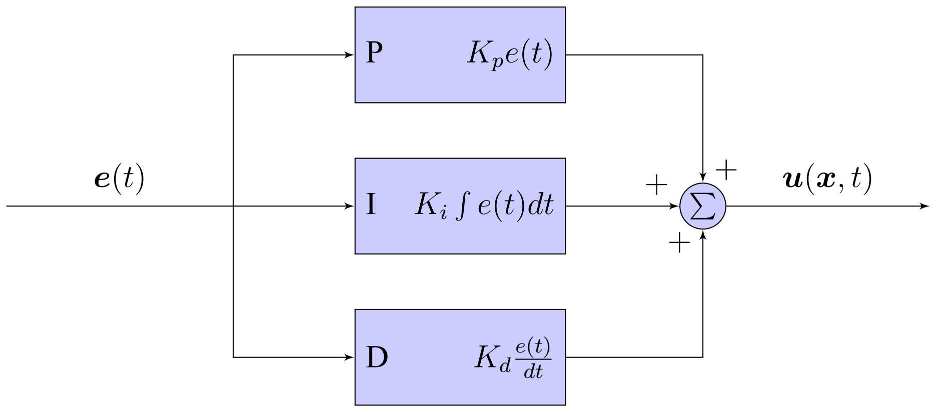


Figure 2.11.: Block diagram of a PID controller. To represent a system with PID control, the whole diagram can be substituted with the *Controller* block in Figure 2.10.

The PID controller is widely used for robotic applications, mainly because of its simplicity and clear physical meaning. A typical control method in robotics, which is also used in this work, is the usage of PD control in joint space with gravity compensation

[11]. This means just the proportional and derivative terms were used. The controller can be described:

$$\boldsymbol{\tau} = \boldsymbol{K}_p(\boldsymbol{q}_d - \boldsymbol{q}) - \boldsymbol{K}_d\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) \tag{2.13}$$

By converting this controller into the system (2.8) with assumption of no external force, the closed-loop controller is obtained:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{S}^{\mathsf{T}}\left(\boldsymbol{K}_p(\boldsymbol{q}_d - \boldsymbol{q}) - \boldsymbol{K}_d\,\dot{\boldsymbol{q}}\right) = \boldsymbol{0}, \tag{2.14}$$

where its equilibrium state is complied with $[(\boldsymbol{q}_d - \boldsymbol{q})^{\mathsf{T}}, \dot{\boldsymbol{q}}^{\mathsf{T}}]^{\mathsf{T}} = \boldsymbol{0}$. However, it should be taken into account that PD respectively PID control does not guarantee optimal control.

2.4.3. Linear Quadratic Regulator

Just like PID, Linear Quadratic Regulator (LQR) is also a closed-loop controller and works with feedback. It is a widely used method for computing the control signal by modeling it as an optimization problem. The approach tries to minimize a cost function to get optimally controlled feedback gains to construct a stable and high-performance closed-loop system.

The LQR problem [12] can be formulated as the minimization of the cost, by considering the underlying linear discrete-time system introduced in subsection 2.4.1 and initial state $\boldsymbol{x}_1$.

$$\begin{aligned} c(\boldsymbol{x}_1, \boldsymbol{u}) &= c_T(\boldsymbol{x}_T) + \sum_{t=1}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) && (2.15)\\ &= \boldsymbol{x}_T^{\mathsf{T}}\boldsymbol{Q}_T\boldsymbol{x}_T + \sum_{t=1}^{T-1}\Big(\boldsymbol{x}_t^{\mathsf{T}}\boldsymbol{Q}_t\boldsymbol{x}_t + \boldsymbol{u}_t^{\mathsf{T}}\boldsymbol{R}_t\,\boldsymbol{u}_t\Big), \\ \text{s.t.}\quad \boldsymbol{x}_{t+1} &= \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t, \quad \forall t \in \{1, \ldots, T\}, && (2.16)\end{aligned}$$

where $\boldsymbol{Q}_t \in \mathbb{R}^{n\times n}$ and $\boldsymbol{R}_t \in \mathbb{R}^{m\times m}$ are positive definite diagonal matrices. $\boldsymbol{Q}_t$ is the precision matrix and $\boldsymbol{R}_t$ is the control weight matrix, to determine the penalizing cost on high control commands.

The state feedback gains $\boldsymbol{K}_t$ can be obtained with the help of solving the discrete-time algebraic Riccati tion, by using the dynamic programming principle. First the matrices $\boldsymbol{V}_t$ have to be calculated by performing a backward recursion. The backward pass is initialized by setting the terminal condition $\boldsymbol{V}_T = \boldsymbol{Q}_T$, in order to compute $\boldsymbol{V}_t$:

$$\boldsymbol{V}_t = \boldsymbol{Q}_{\boldsymbol{xx},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^{\mathsf{T}} \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \boldsymbol{Q}_{\boldsymbol{ux},t}. \tag{2.17}$$

$$\begin{aligned} \text{with} \quad \boldsymbol{Q}_{\boldsymbol{xx},t} &= \boldsymbol{A}_t^{\mathsf{T}} \boldsymbol{V}_{t+1} \boldsymbol{A}_t + \boldsymbol{Q}_t, \\ \boldsymbol{Q}_{\boldsymbol{uu},t} &= \boldsymbol{B}_t^{\mathsf{T}} \boldsymbol{V}_{t+1} \boldsymbol{B}_t + \boldsymbol{R}_t, \\ \boldsymbol{Q}_{\boldsymbol{ux},t} &= \boldsymbol{B}_t^{\mathsf{T}} \boldsymbol{V}_{t+1} \boldsymbol{A}_t. \end{aligned} \tag{2.18}$$

The control commands $\boldsymbol{u}_t$ can be retrieved by using the forward integration:

$$\boldsymbol{u}_t = -\boldsymbol{K}_t \boldsymbol{x}_t \quad , \tag{2.19}$$

with the feedback gain

$$\boldsymbol{K}_t = \left(\boldsymbol{B}_t^{\mathsf{T}} \boldsymbol{V}_t \boldsymbol{B}_t + \boldsymbol{R}_t\right)^{-1} \boldsymbol{B}_t^{\mathsf{T}} \boldsymbol{V}_t \boldsymbol{A}_t \tag{2.20}$$

and the computation of the evolution of the state, by using the linear system (3.13) starting from its initial state $\boldsymbol{x}_1$.

2.4.4. Linear Quadratic Tracking

The LQR controller can also be extended to a closed-loop controller with an additional feed-forward part. In that case, the controller can be formulated as a Linear Quadratic Tracking (LQT) problem so that a discrete-time finite-horizon LQR is computed between each state. This can be set up with the help of expressing the tracking problem to a regulation problem by redefining the state definition to an augmented state. The augmented dynamical system is expressed:

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{t+1} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_{t+1}} = \underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}}_{\tilde{\boldsymbol{A}}} \underbrace{\begin{bmatrix} \boldsymbol{x}_t \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_t} + \underbrace{\begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix}}_{\tilde{\boldsymbol{B}}} \boldsymbol{u}_t, \tag{2.21}$$

$$\tilde{\boldsymbol{Q}}_t = \begin{bmatrix} \boldsymbol{Q}_t^{-1} + \boldsymbol{\mu}_t \boldsymbol{\mu}_t^{\top} & \boldsymbol{\mu}_t \\ \boldsymbol{\mu}_t^{\top} & 1 \end{bmatrix}^{-1} \qquad \forall t \in \{1, \dots, T\},$$

where $\tilde{\boldsymbol{Q}}_t$ is the augmented tracking weight matrix and $\boldsymbol{\mu}_t$ is the target signal to be tracked.

The tracking problem can be solved in the same manner as for the standard LQR, just by redefining the cost function (2.15), based on the variables defined in the augmented representation. After the backward recursion, the system can be decomposed. This leads finally to a control policy with an error term, handled by the feedback-gain, and feed-forward term $\boldsymbol{u}_t^{\text{ff}}$, that also depends on the solution of the Riccati equation.

$$\begin{aligned} \hat{\boldsymbol{u}}_t &= -\tilde{\boldsymbol{K}}_t \, \tilde{\boldsymbol{x}}_t & \text{with } \tilde{\boldsymbol{K}}_t = \left[\boldsymbol{K}_t, \boldsymbol{k}_t\right] \\ &= \boldsymbol{K}_t \left(\boldsymbol{\mu}_t - \boldsymbol{x}_t\right) - \underbrace{\left(\boldsymbol{k}_t + \boldsymbol{K}_t \boldsymbol{\mu}_t\right)}_{\boldsymbol{u}_t^{\text{ff}}}, \end{aligned} \tag{2.22}$$

where $\tilde{\boldsymbol{K}}_t$ is obtained just as in (2.20), but again by using of the augmented variables introduced in (2.21).

3. Methodology

3.1. Centroidal Momentum Dynamics

Legged robots locomotion is often modeled by the representation of its Center of Mass (CoM), and thus tasks are also planned with CoM trajectories. Different approaches are used in practice to model the dynamics and the contact with the environment. A common way is to represent the translational dynamics using a linear inverted pendulum and model it as point-mass-model. This simplified model allows to represent the dynamics linearly and formulate optimal control problems so that the minimizing function remains quadratic. The main drawback of this representation is that it does not characterize the rotational inertia owing to the assumption that net angular momentum is zero, even though it has important implications for gait and balanced behavior [13].

On the other hand, considering the whole body dynamics on the whole-body level leads to a large and complex optimal control problem, and a numerical solver cannot solve it directly. One method that has recently gained popularity is mapping the dynamics to the centroidal space. This means the dynamics of a legged robot are projected at its CoM. The CoM represents the location of its effective total mass and, therefore, the point at which the robot's total linear momentum and angular momentum are naturally defined[14].

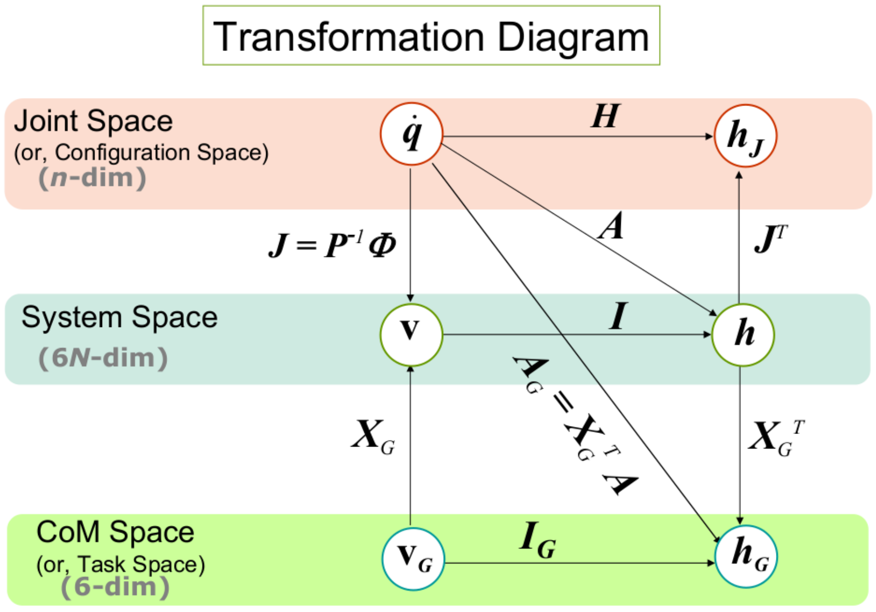


Figure 3.1.: Diagram to represent the different spaces that are used to describe robot dynamics and its relationships.[15] The matrix $\boldsymbol{H}$ expresses the inertia matrix in joint space, in this work introduced as $\boldsymbol{M}$ in Equation 2.7. The system space, also denoted as whole body space, consists of N rigid body links of the robot, always expressed with the 6 motion components. The CoM space defines a frame located at the robot's CoM.

The centroidal momentum of a floating base robot is the aggregated momentum of the individual link momenta acting on the system and can be calculated with the *Centroidal Momentum Matrix* $\boldsymbol{A}_\mathrm{G}(\boldsymbol{q}) \in \mathbb{R}^{6\times n+6}$ as follows:

$$\boldsymbol{h}_\mathrm{G} = \boldsymbol{A}(\boldsymbol{q})_\mathrm{G}\dot{\boldsymbol{q}} \tag{3.1}$$

The resulting centroidal momentum vector $\boldsymbol{h}_\mathrm{G}$ depends finally just on kinematic quantities, where the Centroidal Momentum Matrix acts as a linear matrix function.

The centroidal dynamics representation allows to model whole-body dynamics by reducing the space dimension. Nevertheless, the dynamics remain no longer linear, so optimal control problems may be non-convex. The following methods and models for planning and control problems are based entirely or in an adapted form on centroidal dynamics.

3.2. Trajectory Planning with Kino-Dynamic Motion Generation

A typical control schema to control a robot is composed of a planner and a controller part. The main purpose of the planner is motion planning by incorporating interaction forces. Withal dynamical constraints should also be fulfilled to ensure the feasibility of the motion in practice. This task often leads to a high complexity problem because different sub-problems related to kinematics and dynamics have to be solved. Therefore mostly, the path planning task is carried out offline.

The planner that is used in this work is based on the approach described in the paper *Structured contact force optimization for kino-dynamic motion generation* [16][4]. The code for the planner used in this work was implemented and provided by the New York University and the Max-Planck Institute for Intelligent Systems.

The starting situation is given a set of predefined contact points with a reference contact force profile over time, and the goal is to find an optimal trajectory that is consistent with the full robot dynamics. The optimization problem is split into two sub-problems and is solved alternately. The procedure is briefly explained here.

The equations of motions of a legged robot (2.8) can be separated into an 6 row under-actuated part (subscript u) and into an n row actuated (subscript a) dynamics part:

$$\boldsymbol{M}_{\mathrm{u}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}_{\mathrm{u}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{i=1}^{n_c} \boldsymbol{J}_{\mathrm{i,u}}^{\mathsf{T}} \boldsymbol{\lambda}_{\mathrm{i}}, \tag{3.2a}$$

$$\boldsymbol{M}_{\mathrm{a}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}_{\mathrm{a}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau}_q + \sum_{i=1}^{n_c} \boldsymbol{J}_{\mathrm{i,a}}^{\mathsf{T}} \boldsymbol{\lambda}_{\mathrm{i}}, \tag{3.2b}$$

[4]`https://github.com/machines-in-motion/kino_dynamic_opt` (06.2022)

where $\boldsymbol{M}$ is the inertia joint space matrix, the vector of non-linear terms $\boldsymbol{N}$ containing Coriolis, centrifugal, gravity and joint friction forces, $\boldsymbol{J}_i$ is the associated Jacobian of the i-th end-effector contact with the acting wrench $\boldsymbol{\lambda}_\mathrm{i}$.

The under-actuated part (3.2a) of the dynamics can be expressed with respect to the CoM via the centroidal momentum matrix (3.1), so the dynamic depends on joint configuration and velocities. On the other hand, the rate of centroidal momentum $\dot{\boldsymbol{h}}_\mathrm{G}$ can be derived from the Newton-Euler dynamics [17], which equals the total wrench generated by the external contacts and the gravitational force. This leads to the following relation:

$$\boldsymbol{A}_\mathrm{G}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{A}}_\mathrm{G}(\boldsymbol{q})\dot{\boldsymbol{q}} = \underbrace{\begin{bmatrix} m\boldsymbol{g} + \sum_{i=1}^{n_c} \boldsymbol{f}_i \\ \sum_{i=1}^{n_c} (\boldsymbol{p}_i - \boldsymbol{x}_\mathrm{CoM}) \times \boldsymbol{f}_i + \boldsymbol{\tau}_i \end{bmatrix}}_{\dot{\boldsymbol{h}}_\mathrm{G}}, \tag{3.3}$$

where $\boldsymbol{p}_i$ is the location of the contact i and $\boldsymbol{x}_\mathrm{CoM}$ the position of the CoM, both computed with FK. $\boldsymbol{f}_i$ and $\boldsymbol{\tau}_i$ are the force respectively torque sub components of the wrench vector $\boldsymbol{\lambda}_\mathrm{i}$ acting on contact i.

The problem is decomposed into two steps to solve the planning process and comply with the equations of motion for a floating base system.

The first task is to find a vector of joint trajectories $\boldsymbol{q}(t)$ in combination with contact force profiles $\boldsymbol{\lambda}(t)$ satisfying equation (3.3), assuming kinematic reachability. Secondly, the joint torques $\boldsymbol{\tau}_q(t)$ can be computed with equation (3.2b), under the assumption sufficient joint torque can be provided by the actuators.

The whole motion generation task can be formulated as an optimal control problem as follows:

$$\min_{\boldsymbol{q},\boldsymbol{h},\boldsymbol{\lambda},\boldsymbol{c}} \quad J(\boldsymbol{q},\boldsymbol{h},\boldsymbol{\lambda},\boldsymbol{c}) = \sum_t J_t(\boldsymbol{q}_t) + J'_t(\boldsymbol{h}_t,\boldsymbol{\lambda}_t,\boldsymbol{c}_t) \tag{3.4}$$

$$\text{s.t.} \qquad \boldsymbol{c}_{\mathrm{t,i}} = \boldsymbol{p}_{\mathrm{i}}(\boldsymbol{q}_t) \tag{3.5}$$

$$\boldsymbol{h}_t = \begin{bmatrix} \boldsymbol{x}_{\mathrm{CoM}}(\boldsymbol{q}_t) \\ \boldsymbol{A}_{\mathrm{G}}(\boldsymbol{q}_t)\dot{\boldsymbol{q}}_t \end{bmatrix} \tag{3.6}$$

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \Delta \boldsymbol{f}_t(\boldsymbol{h}_t,\boldsymbol{\lambda}_t) \tag{3.7}$$

$$\boldsymbol{\lambda}_t, \boldsymbol{c}_t \in \mathcal{S}_t, \quad \forall t, \tag{3.8}$$

where $J_t(\boldsymbol{q}_t)$ is the objective function related to the kinematic sub problem (i.e. move one leg from a contact via-point to another) and $J'_t(\boldsymbol{h}_t,\boldsymbol{\lambda}_t,\boldsymbol{c}_t)$ the objective function on momentum, force profile and contact locations. $\mathcal{S}$ is the set denoting all the valid force profiles, such as the contact forces lie in a feasible friction cone. The momentum equation (3.3) is defined as constraint in (3.7) with discretization of a time-step Δ and the function:

$$\dot{\boldsymbol{h}} = \begin{bmatrix} \dot{\mathrm{r}} \\ \dot{\boldsymbol{h}}_{\mathbf{G}} \end{bmatrix} = \begin{bmatrix} \dot{\mathrm{r}} \\ \dot{\mathrm{l}} \\ \dot{\mathrm{k}} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\mathrm{l} \\ m\boldsymbol{g} + \sum_{i=1}^{n_c} \boldsymbol{f}_i \\ \sum_{i=1}^{n_c}(\boldsymbol{p}_i - \boldsymbol{x}_{\mathrm{CoM}}) \times \boldsymbol{f}_i + \boldsymbol{\tau}_i \end{bmatrix} = \boldsymbol{f}(\boldsymbol{h},\boldsymbol{\lambda}), \tag{3.9}$$

where **r**, l, **k** are CoM position, the linear and angular momentum.

3.2.1. Functionality of the Algorithm

Based on the desired, predefined contact points with its reference contact force profiles and CoM positions, $\bar{\boldsymbol{\lambda}},\ \bar{\boldsymbol{h}},\ \bar{\boldsymbol{c}}$ can be initialized. After the kinemtatic related sub-problem is solved first.

Kinematic Sub-problem

The sub-problem is an unconstrained optimization problem, respectively the constraints

(3.5, 3.6) from the original problem are considered in the cost function and become soft constraints.

$$\min_{\boldsymbol{q}} \quad \sum_t J_t(\boldsymbol{q}_t) + \| \begin{bmatrix} \boldsymbol{x}_{\mathrm{CoM}}(\boldsymbol{q}_t) \\ \boldsymbol{A}_{\mathrm{G}}(\boldsymbol{q}_t)\dot{\boldsymbol{q}}_t \end{bmatrix} - \bar{\boldsymbol{h}}_t\|^2 + \|\boldsymbol{p}_i(\boldsymbol{q}_t) - \bar{\boldsymbol{c}}_{t,i}\|^2 \tag{3.10}$$

The optimization problem is solved with a Gauss-Newton method. After minimizing over $\boldsymbol{q}$, momentum and contact locations are updated:

$$\bar{\boldsymbol{h}}, \bar{\boldsymbol{c}}_i := \begin{bmatrix} \boldsymbol{x}_{\mathrm{CoM}}(\boldsymbol{q}_t) \\ \boldsymbol{A}_{\mathrm{G}}(\boldsymbol{q}_t)\dot{\boldsymbol{q}}_t \end{bmatrix}, \boldsymbol{p}_{\mathrm{i}}(\boldsymbol{q}_t)$$

The second sub-problem is then solved with the previously updated variables.

Momentum and Contact Force Related Sub-problem

In that optimization problem joint states $\boldsymbol{q}$ are ignored. Therefore contact locations $\boldsymbol{c}_{\mathrm{t,i}}$ and momentum $\boldsymbol{h}_t$ were introduced in the global optimization problem as redundant variables. This optimization problem is non-convex, but well structured and solved with a modified primal-dual interior-point method [18].

$$\min_{\boldsymbol{h},\boldsymbol{\lambda},\boldsymbol{c}} \quad \sum_t \|\boldsymbol{h}_t - \bar{\boldsymbol{h}}_t\|^2 + \sum_i^{n_{c,t}} \|\boldsymbol{\lambda}_{t,i} - \bar{\boldsymbol{\lambda}}_{t,i}\|^2 + \|\boldsymbol{c}_{t,i} - \bar{\boldsymbol{c}}_{t,i}\|^2 \tag{3.11}$$

$$\text{s.t.} \quad \boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \Delta f_t(\boldsymbol{h}_t, \boldsymbol{\lambda}_t) \tag{3.12}$$

$$\boldsymbol{\lambda}_t, \boldsymbol{c}_t \in \mathcal{S}_t, \quad \forall t \tag{3.13}$$

Afterwards the obtained momentum and contact locations are again updated:

$$\bar{\boldsymbol{h}}, \bar{\boldsymbol{c}}_i := \boldsymbol{h}, \boldsymbol{c}_i$$

Subsequently, the algorithm moves again back to the first sub-problem (3.10) by using the updated variables $\bar{\boldsymbol{h}}, \bar{\boldsymbol{c}}_i$. The two sub-problems will be optimized alternately until the solutions of the two sub-problems do not change anymore. The resulting solution is a consensus between both optimization problems, a locally optimal solution for the global full kino-dynamic plan.

3.3. Impedance Controller

A Cartesian impedance controller is used to apply some basic movements and tasks with the Solo robot, based on the approach presented in [3][5]. The physical definition of mechanical impedance is the ratio of power to velocity, which describes the resistance that a material opposes to a mechanical harmonic force, depending on the phase angle of the vibration.

When a quadruped interacts with its environment, it causes a reaction of the robot and the environment. The robot's response is like an impedance, and the environment reacts as an admittance, which is the inverse of impedance. With the help of an impedance controller, the caused motions and their contact forces can be modulated and controlled. This control mechanism relies on a mass-spring-damper system [19] to model such behavior.

To each foot a local frame is attached with its origin to its hip (i.e. Figure 2.6 $\{l_A\}$), so that each foot can be represented such as in Figure 3.2. In addition to the springs in the x and z directions, one acts along the y axis and points out of the page. A controller for one leg with respect to its hip can be formulated as follows:

$$\boldsymbol{\tau} = \boldsymbol{J}^\intercal \left(\boldsymbol{K}(\boldsymbol{x}_d - \boldsymbol{x}) - \boldsymbol{D}\dot{\boldsymbol{x}} \right), \tag{3.14}$$

where $\boldsymbol{x}_d \in \mathbb{R}^3$ represents the spring setpoint with respect to the local frame, $\boldsymbol{x} \in \mathbb{R}^3$ is the foot position, $\boldsymbol{J}$ the foot Jacobian and $\boldsymbol{K} \in \mathbb{R}^3$ respectively $\boldsymbol{D} \in \mathbb{R}^3$ are the stiffness and damping matrices. $\boldsymbol{\tau}$ is the actuation torques to apply to the motors. Finally, the desired torque to apply on a joint can be converted into motor current by the relationship described in (2.2).

[5]`https://github.com/machines-in-motion/mim_control` (05.2022)

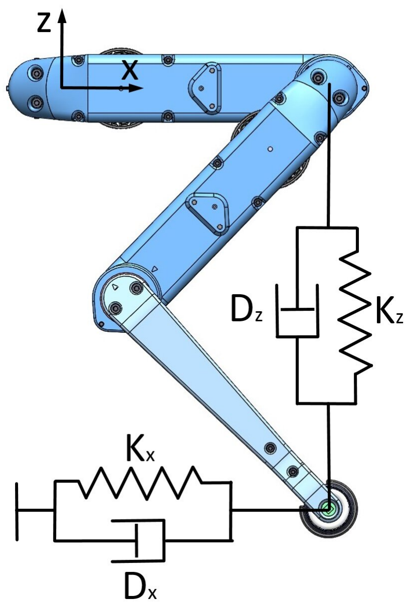


Figure 3.2.: Cartesian impedance controller, consisting of three mass-spring-damper system, acting in x, z and y direction, where the latter points out of the page.

3.4. Centroidal Motion Controller

In section 3.1 the centroidal dynamics were introduced and it was shown that this representation is suitable for modeling whole-body motion.

The following control structure is based on centroidal dynamics. It is used to control the robot Solo 12 and is described in the publication *An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research*[3]. The code for the controller used in this work was implemented and provided by the New York University and the Max-Planck Institute for Intelligent Systems.

The controller is composed of three steps. The wrench at the CoM is computed in the first phase. Secondly, the centroidal twist is distributed on the legs in contact with the ground. The resulting force to be exerted is finally transferred so that joint torque is obtained.

3.4.1. Centroidal PD-Controller

The model for this controller assumes the position of the CoM fixed at the base frames' origin. The controller presumes valid dynamics trajectories as input consisting of reference centroidal wrench $\overline{\boldsymbol{W}}_{\mathrm{CoM}}$, CoM position $\bar{\boldsymbol{x}}_{\mathrm{CoM}}$ and its velocity, angular momentum $\bar{\mathbf{k}}$, and base orientation $\bar{\boldsymbol{\epsilon}}$. All variables denoted with a bar representing a reference value, all other variables expressing measured states of the robot. The desired wrench $\boldsymbol{W}_{\mathrm{CoM}}$ at the CoM is obtained as follows:

$$\boldsymbol{W}_{\mathrm{CoM}} = \overline{\boldsymbol{W}}_{\mathrm{CoM}} + \begin{bmatrix} \boldsymbol{K}_c(\bar{\boldsymbol{x}}_{\mathrm{CoM}} - \boldsymbol{x}_{\mathrm{CoM}}) + \boldsymbol{D}_c(\dot{\bar{\boldsymbol{x}}}_{\mathrm{CoM}} - \dot{\boldsymbol{x}}_{\mathrm{CoM}}) \\ \boldsymbol{K}_b(\bar{\boldsymbol{\epsilon}} \boxminus \boldsymbol{\epsilon}) + \boldsymbol{D}_b(\bar{\mathbf{k}} - \mathbf{k}) \end{bmatrix},$$

where $\boldsymbol{K}_c$, $\boldsymbol{K}_b$, $\boldsymbol{D}_c$ and $\boldsymbol{D}_b$ are positive definite gain matrices. The operator $\boxminus$ maps the orientation error with the logarithm mapping into an angular velocity vector [4]. Because orientation and angular momentum control are mixed, a constant inertia tensor is assumed to ensure consistency.

The reference wrench is either also provided from the planner or can be calculated with the Centroidal Momentum Matrix $\boldsymbol{A}_{\mathrm{G}}(\boldsymbol{q})$ and the reference generalized joint velocities and accelerations.

$$\overline{\boldsymbol{W}}_{\mathrm{CoM}} = \boldsymbol{A}_{\mathrm{G}}(\bar{\boldsymbol{q}})\ddot{\bar{\boldsymbol{q}}} + \dot{\boldsymbol{A}}_{\mathrm{G}}(\bar{\boldsymbol{q}})\dot{\bar{\boldsymbol{q}}} \tag{3.15}$$

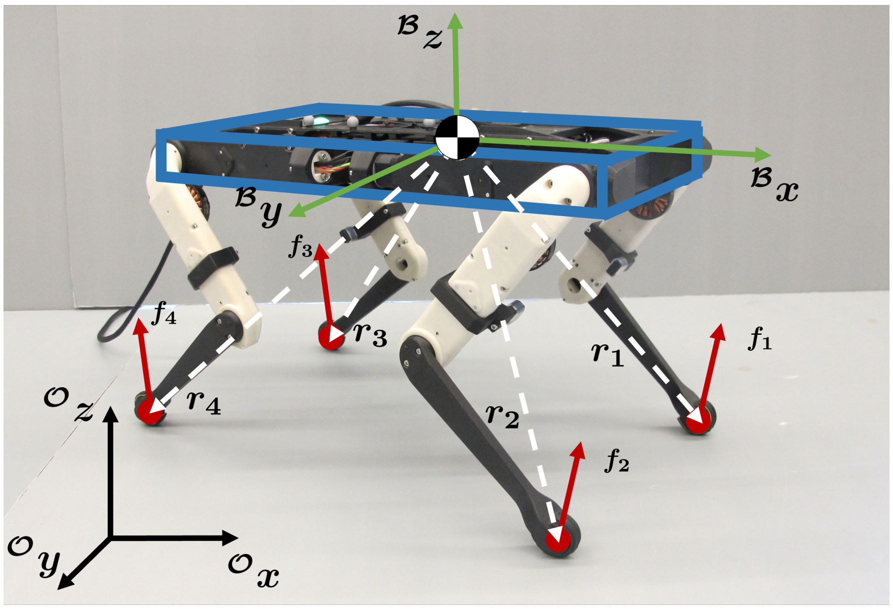


Figure 3.3.: Illustration of the rigid-body model with centroidal dynamics and its units needed to calculate the wrench acting on the robot's CoM.

The wrench is distributed to the number of foots in contact n_c at each control step by solving the following Quadratic Programming (QP) problem:

$$\begin{aligned} &\min_{\boldsymbol{f}_i,\boldsymbol{\eta},\zeta_1,\zeta_2} \sum_{i\in\mathcal{C}} \boldsymbol{f}_i^2 + \alpha(\boldsymbol{\eta}^2 + \zeta_1^2 + \zeta_2^2) \\ \text{s.t.} \quad &\boldsymbol{W}_{\mathrm{CoM}} = \sum_{i\in\mathcal{C}} \begin{bmatrix} \boldsymbol{f}_i \\ \boldsymbol{r}_i \times \boldsymbol{f}_i \end{bmatrix} + \boldsymbol{\eta} \\ &f_{i,x} < \mu f_{i,z} + \zeta_1, \quad f_{i,y} < \mu f_{i,z} + \zeta_2 \qquad \forall i \in \mathcal{C}, \end{aligned}$$

where the feet in contact are determined by an *AND* condition, the reference and the measured feedback need to indicate contact. The indexes for the feet which satisfy this condition are part of $\mathcal{C}$. The vector from the contact location to the CoM position is denoted as $\boldsymbol{r}_i$, and the force to apply at the regarding contact location is labeled as $\boldsymbol{f}_i \in \mathbb{R}^3$, as shown in Figure 3.3. The variables $\boldsymbol{\eta}$, ζ_1 and ζ_2 are introduced as slack variables in order to make the QP always solvable, with α set to a large scalar weight. μ is the friction coefficient, and z expresses the normal force direction perpendicular to the ground.

A low impedance controller (e.g. (3.14)) is included in the control law to get the actuation torques for all joints. So that the computed force and the impedance control amount can finally be translated with the help of the foot Jacobian $\boldsymbol{J}_i$. The impedance controller part allows navigating the feet not in contact with the ground, for example, to execute the swing trajectory during trotting. But it is also crucial to handle disturbance and hard impact dynamics.

$$\boldsymbol{\tau_i} = \boldsymbol{J}_i^{\mathsf{T}} \left(\boldsymbol{f}_i + \boldsymbol{K}({}^{\scriptscriptstyle B}\bar{\boldsymbol{x}}_i - {}^{\scriptscriptstyle B}\boldsymbol{x}_i) + \boldsymbol{D}({}^{\scriptscriptstyle B}\dot{\bar{\boldsymbol{x}}}_i - {}^{\scriptscriptstyle B}\dot{\boldsymbol{x}}_i) \right)$$

3.5. Control Architecture of a Reactive Controller

This controller is based on an overall whole-body controller that receives ground reaction forces from the MPC as well as the reference position of the feet in the swing phase provided from footstep planner respectively swing foot trajectory planner, and computes for all joints the actuation torques. The goal of this controller is to track the commanded reference velocity from the joystick as closely as possible.

The controller is based on the Mini Cheetah [20], but implemented as well from the *Gepetto team, LAAS-CNRS*[21][6]. The control architecture is reused and just the MPC implemented in this work.

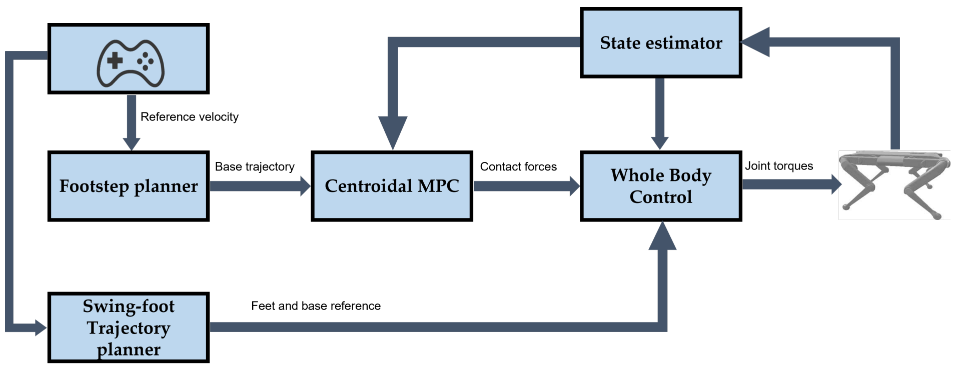


Figure 3.4.: Control architecture of the reactive walking controller with its centroidal MPC.

[6]`https://github.com/Gepetto/quadruped-reactive-walking` (05.2022)

3.5.1. MPC Formulation of the Controller

The MPC subpart of the control architecture relies on a simplified centroidal dynamic model [22]. The goal is to find the optimal contact forces of the feet in the stance phase over a given prediction horizon to follow the reference CoM trajectory as close as possible. The dynamics are modeled with second-order dynamics so that the state of the robot is:

$$\boldsymbol{x}(t) = \begin{bmatrix}\boldsymbol{\Theta}^\intercal & \boldsymbol{x}_{\mathrm{CoM}}^\intercal & \boldsymbol{\omega}^\intercal & \dot{\boldsymbol{x}}_{\mathrm{CoM}}^\intercal\end{bmatrix}^\intercal, \tag{3.16}$$

where $\boldsymbol{\Theta}$ is the robot's orientation denoted with Z-Y-X Euler angles, with (ϕ, θ, ψ) the roll, pitch and yaw angles and $\dot{\boldsymbol{\omega}} \in \mathbb{R}^3$ its angular velocity. $\boldsymbol{x}_{\mathrm{CoM}} \in \mathbb{R}^3$ is the CoM position, and $\dot{\boldsymbol{x}}_{\mathrm{CoM}}$ the CoM velocity.

The MPC problem will be modeled with the help of a linear dynamical system such as introduced in (2.10). The robot is modeled as a single rigid body with centroidal dynamics (see Figure 3.3). In order to solve the optimization problem fast, and keeping the dynamics linear, following simplifications were made:

(1) roll and pitch angels are small

(2) actual state of the robot is close to the desired next state

(3) pitch and roll velocities are small

The dynamics of the MPC in world frame can be written as follows:

$$\ddot{\boldsymbol{x}}_{\mathrm{CoM}} = \frac{\sum_{i\in\mathcal{C}} \boldsymbol{f}_i}{m} - \begin{bmatrix}0\\0\\g\end{bmatrix} \tag{3.17}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{\mathcal{I}}\boldsymbol{\omega}) = \sum_{i\in\mathcal{C}}(\boldsymbol{p}_i - \boldsymbol{x}_{\mathrm{CoM}}) \times \boldsymbol{f}_i\,, \tag{3.18}$$

where (3.17), expresses the rate of change of the linear momentum at the robot's CoM, and (3.18) the rate of change of the angular momentum at the CoM, where $\boldsymbol{\mathcal{I}}$ is the rotational inertia tensor.

Rotations of the body along the axes from body coordinates to the world frame and the time derivative of that matrix can be expressed as follows [23]:

$$\boldsymbol{R} = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) \tag{3.19}$$

$$\dot{\boldsymbol{R}} = [\boldsymbol{\omega}]_\times \boldsymbol{R}, \tag{3.20}$$

where $[\boldsymbol{\omega}]_x \in \mathbb{R}^{3\times3}$ is the skew-symmetric matrix representation of the angular velocity.

Take advantage of the Simplifications

The rate of change of the robot's orientation can be found from the inversion of the rotation matrix denoted in (3.19) and with the angular velocity of the body, expressed in the world frame. The first simplification **(1)** leads finally to relation in (3.21), so that the transformation just can be expressed with the transpose of the rotation matrix along the z-axis.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}\cos(\psi)/\cos(\theta) & \sin(\psi)/\cos(\theta) & 0\\ -\sin(\psi) & \cos(\psi) & 0\\ \cos(\psi)\tan(\theta) & \sin(\psi)\tan(\theta) & 1\end{bmatrix}\boldsymbol{\omega}$$

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \approx \begin{bmatrix}\cos(\psi) & \sin(\psi) & 0\\ -\sin(\psi) & \cos(\psi) & 0\\ 0 & 0 & 1\end{bmatrix}\boldsymbol{\omega}$$

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \approx \boldsymbol{R}_z(\psi)^\mathsf{T}\boldsymbol{\omega} \tag{3.21}$$

The rotational tensor of the body in world coordinates can then be expressed as:

$$\boldsymbol{\mathcal{I}} \approx \boldsymbol{R}_z(\psi) \cdot {}^{B}\boldsymbol{\mathcal{I}} \cdot \boldsymbol{R}_z(\psi)^\intercal$$

Based on the second simplification **(2)**, the lever arm definition is adapted, so that the predefined positions from the reference trajectory (variables denoted with a bar) are used. The definition of the moment arm becomes to $\boldsymbol{r}_i = \bar{\boldsymbol{p}}_i - \bar{\boldsymbol{x}}_{\text{CoM}}$.

The third simplification **(3)**, which additionally requires that the off-diagonal terms of the inertia tensor are also small, keeps the rate of the angular momentum linear with respect to its angular velocity. This means the effect of precession and nutation of the rotating body are neglected. To equation from (3.18) becomes to:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{\mathcal{I}}\boldsymbol{\omega}) &= \boldsymbol{\mathcal{I}}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\boldsymbol{\mathcal{I}}\boldsymbol{\omega}) \approx \boldsymbol{\mathcal{I}}\dot{\boldsymbol{\omega}} \\ &\approx \sum_{i \in \mathcal{C}} \boldsymbol{r}_i \times \boldsymbol{f}_i \,, \end{aligned} \tag{3.22}$$

3.5.1.1. LQT Formulation with Constraints as QP Problem

Since all dynamic relations needed to express the robots state (3.16) are derived, the linear dynamical system can be formulated by including the gravity term in the state as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{\Theta} \\ \boldsymbol{x}_{\text{CoM}} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{x}}_{\text{CoM}} \\ g \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{R}_z(\psi)^\intercal & \boldsymbol{0}_3 & \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{1}_3 & \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0} & \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^\intercal \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{A}^c(\psi_t)} \begin{bmatrix} \boldsymbol{\Theta} \\ \boldsymbol{x}_{\text{CoM}} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{x}}_{\text{CoM}} \\ g \end{bmatrix} + \underbrace{\begin{bmatrix} \boldsymbol{0}_3 & \cdots & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \cdots & \boldsymbol{0}_3 \\ \boldsymbol{\mathcal{I}}^{-1}[\boldsymbol{r}_1]_\times & \cdots & \boldsymbol{\mathcal{I}}^{-1}[\boldsymbol{r}_4]_\times \\ \boldsymbol{1}_3/m & \cdots & \boldsymbol{1}_3/m \\ 0 & \cdots & 0 \end{bmatrix}}_{\boldsymbol{B}^c(\boldsymbol{r}_1, \cdots, \boldsymbol{r}_4)} \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \boldsymbol{f}_4 \end{bmatrix} \,,$$

and after discretization of the system with the method described in subsection 2.4.1, the system becomes to:

$$\underbrace{\begin{bmatrix}\boldsymbol{\Theta}^{\mathsf{T}} & \boldsymbol{x}_{\mathrm{CoM}}^{\mathsf{T}} & \boldsymbol{\omega}^{\mathsf{T}} & \dot{\boldsymbol{x}}_{\mathrm{CoM}}^{\mathsf{T}} & g\end{bmatrix}^{\mathsf{T}}}_{\boldsymbol{x}_{t+1}} = \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t\,,$$

with $\boldsymbol{x}_t \in \mathbb{R}^{13\times1}$, $\boldsymbol{A}_t \in \mathbb{R}^{13\times13}$, $\boldsymbol{B}_t \in \mathbb{R}^{13\times12}$, $\boldsymbol{u}_t \in \mathbb{R}^{12\times1}$.

Batch Formulation Based on Prediction Horizon

The input to the MPC is a reference trajectory denoted as prediction horizon. It consists of all the states that should be tracked so that:

$$\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{x}_{ref,1}^{\mathsf{T}}, \boldsymbol{x}_{ref,2}^{\mathsf{T}}, \dots, \boldsymbol{x}_{ref,T}^{\mathsf{T}}\end{bmatrix}^{\mathsf{T}}, \quad \text{with: } \mathbf{x}_{ref,i} = \begin{bmatrix}\boldsymbol{\Theta}_i^{\mathsf{T}} & \boldsymbol{x}_{\mathrm{CoM,i}}^{\mathsf{T}} & \boldsymbol{\omega}_i^{\mathsf{T}} & \dot{\boldsymbol{x}}_{\mathrm{CoM,i}}^{\mathsf{T}} & g\end{bmatrix}^{\mathsf{T}},$$

where for this implementation the prediction horizon is composed of 16 states so that $T = 16$. Based on that, the cost function of the tracking problem can be defined as:

$$J = ||\boldsymbol{x} - \boldsymbol{\mu}||_{\boldsymbol{Q}}^2 + ||\boldsymbol{u}||_{\boldsymbol{R}}^2, \tag{3.23}$$

with $\boldsymbol{x} = [\boldsymbol{x}_1^{\mathsf{T}} \cdots \boldsymbol{x}_T^{\mathsf{T}}]^{\mathsf{T}}$ and $\boldsymbol{u} = \begin{bmatrix}\boldsymbol{u}_1^{\mathsf{T}} \cdots \boldsymbol{u}_{T-1}^{\mathsf{T}}\end{bmatrix}^{\mathsf{T}}$. The evolution of the system over the whole prediction horizon, given the initial state $\boldsymbol{x}_1$ is expressed with the batch formulation, such as follows:

$$\boldsymbol{x} = \boldsymbol{S}_x\boldsymbol{x}_1 + \boldsymbol{S}_u\boldsymbol{u},$$

with the transfer matrices:

$$\boldsymbol{S_x} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A}_1 \\ \boldsymbol{A}_2\boldsymbol{A}_1 \\ \vdots \\ \prod_{t=1}^{T-1} \boldsymbol{A}_{T-t} \end{bmatrix} ; \quad \boldsymbol{S_u} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{B}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{A}_2\boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \left(\prod_{t=1}^{T-2} \boldsymbol{A}_{T-t}\right)\boldsymbol{B}_1 & \left(\prod_{t=1}^{T-3} \boldsymbol{A}_{T-t}\right)\boldsymbol{B}_2 & \cdots & \boldsymbol{B}_{T-1} \end{bmatrix}$$

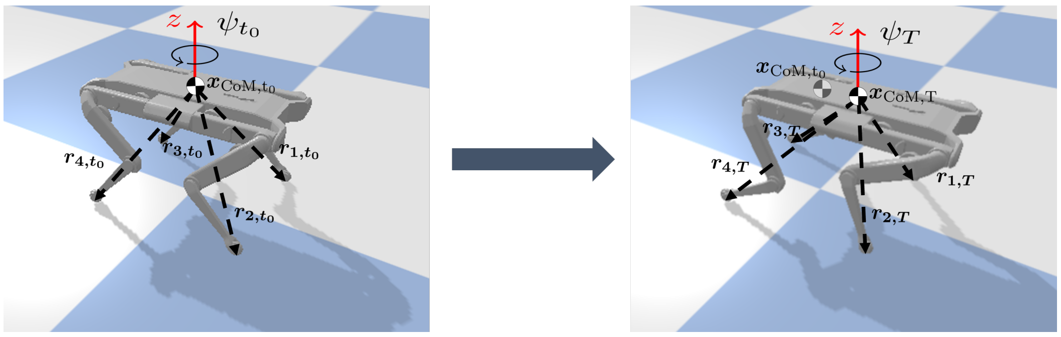


Figure 3.5.: Example of one MPC period with the task of moving the robot's CoM in x direction. The initial state of the robot is shown on the left picture, and the target position on the right.

The system matrices $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are constructed at each time step based on the reference trajectory since $\boldsymbol{A}_t$ depends on the yaw angle of the robot and $\boldsymbol{B}_t$ on the lever arms. Because of that, the dynamics are only accurate if the robot can follow the reference trajectory. In practice, this is often not the case. But since the controller uses not the whole control inputs but rather recomputes the MPC problem with the new initial state from the robot estimator, the system can handle this limitation.

To overcome the issue of the minimization problem from (3.24) tends to keep the control commands too small, and the robot loses at the end of the prediction horizon on height, the cost function is modified. The problem is caused by the different scales of the sub-cost values from the cost function. The control command is much higher than the cost to go from one state to another. Since weighting the cost by reducing the costs on control, commands causes a delay in the behavior, a reference control command to keep the robot's height is included. The cost can be expressed as:

$$J = ||\boldsymbol{x} - \boldsymbol{\mu}||^2_{\boldsymbol{Q}} + ||\boldsymbol{u}||^2_{\boldsymbol{R}} + ||\boldsymbol{u} - \boldsymbol{u}_{\text{ref}}||^2_{\boldsymbol{R}}, \tag{3.24}$$

with:

$$\boldsymbol{u}_{\text{ref,i,t}} = \begin{cases} \boldsymbol{g}\frac{m}{n_{c_t}}, & \text{i} \in \mathcal{C}_t \\ \left[\,0\ 0\ 0\,\right]^\intercal, & \text{otherwise} \end{cases} \quad \forall t \in \{1, \ldots, T\text{-}1\}, \forall i \in \{1, \ldots, 4\},$$

where $\mathcal{C}_t$ is the set containing the legs in contact with the ground at time step t.

The minimizing problem can be expressed as a quadratic term of the form:

$$\| \underbrace{\left(\boldsymbol{S}_{\boldsymbol{u}}^\intercal \boldsymbol{Q} \boldsymbol{S}_u + \boldsymbol{R}\right)}_{\tilde{\mathbf{A}}} \hat{\boldsymbol{u}} - \underbrace{\boldsymbol{S}_{\boldsymbol{u}}^\intercal \boldsymbol{Q}\left(\boldsymbol{\mu} - \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1\right) + \boldsymbol{R}\boldsymbol{u}_{\text{ref}}}_{\tilde{\mathbf{b}}} \|^2$$

The optimization problem can be converted into a standard QP form by setting $\mathbf{P} = 2\tilde{\mathbf{A}}^\intercal\tilde{\mathbf{A}}$ and $\mathbf{q} = -2\tilde{\mathbf{A}}^\intercal\tilde{\mathbf{b}}$ [24]:

$$\begin{aligned} \min_{\hat{\boldsymbol{u}}}. \quad & \tfrac{1}{2}\hat{\boldsymbol{u}}^\intercal \mathbf{P} \hat{\boldsymbol{u}} + \mathbf{q}^\intercal \hat{\boldsymbol{u}} \\ \text{s.t.} \quad & \boldsymbol{G}\hat{\boldsymbol{u}} \leq \boldsymbol{h} \\ & \tilde{\mathbf{A}}\hat{\boldsymbol{u}} = \tilde{\mathbf{b}}, \end{aligned}$$

where the term with the inequality constraint expresses the friction cone constraints and the fact that the normal force has to be positive since the robot cannot pull itself on the ground, these conditions apply to all legs and at all time steps and are expressed such as:

$$\mid f_x \mid \leq \mu f_z, \quad \mid f_y \mid \leq \mu f_z, \quad f_z \geq 0 \quad f_z \leq f_{max},$$

where μ is the friction coefficient, the upper limit of the force is just applied to the z component because the other constraints also limit the force along y and x directions. For one leg, it can be expressed as follows:

$$\underbrace{\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 1 \\ -1 & 0 & -\mu \\ 1 & 0 & -\mu \\ 0 & 1 & -\mu \\ 0 & -1 & -\mu \end{bmatrix}}_{\boldsymbol{L}} \underbrace{\begin{bmatrix} f_{x,i} \\ f_{y,i} \\ f_{z,i} \end{bmatrix}}_{\boldsymbol{u}_{i,t}} \leq \underbrace{\begin{bmatrix} 0 \\ f_{max} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\boldsymbol{h}_{i,t}}$$

The matrix $\boldsymbol{G}$ is a block diagonal matrix consisting of matrices $\boldsymbol{L}$, so that $\boldsymbol{G} \in \mathbb{R}^{(T-1)\cdot 24 \times 12\cdot(T-1)}$ and $\boldsymbol{h} \in \mathbb{R}^{(T-1)\cdot 24}$ is built from the concatenation of $\boldsymbol{h}_{i,t}$ vectors.

To ensure that no force is exerted on legs in contact with the environment, the corresponding entries in the $\boldsymbol{B}_t$ input matrix are set to zero at that time step.

The resulting QP problem is solved with the python library quadprog, based on the Goldfarb/Idnani dual algorithm [25].

4. Experiments and Results

4.1. Trajectory Replay

This approach aims to calculate the dynamics in advance and offline and, based on that, control the robot with a suitable controller in a simulated environment. During the behavior in the simulation, the data is recorded and stored so that it can be sent directly to the robot and its lower-level actuator controller afterward.

The schematic process of the method is shown in Figure 4.1. The planner described in section 3.2 is used to generate feasible CoM trajectory by passing desired via-points. The output is composed of sequences with joint positions $\boldsymbol{q}$ and joint velocities $\dot{\boldsymbol{q}}$, CoM potion $\boldsymbol{x}_{\mathrm{CoM}}$ and its velocity $\dot{\boldsymbol{x}}_{\mathrm{CoM}}$, positions of feet $\boldsymbol{x}$ and its speed $\dot{\boldsymbol{x}}$ in the workspace and a contact plan when which foot is in contact with the environment. The kino-dynamic motion controller introduced in section 3.4, consisting of a centroidal PD-controller and a Cartesian impedance controller, is applied in simulation to execute the planned trajectories. During the simulation, the joint configurations, the joint velocity and the applied joint torques $\boldsymbol{\tau}$ are tracked and stored. Subsequently, the recorded data is used as input to the robot controller to apply movements in real environments. The joint values act as a reference for the robot controller, and the torque components provide a feed-forward term.

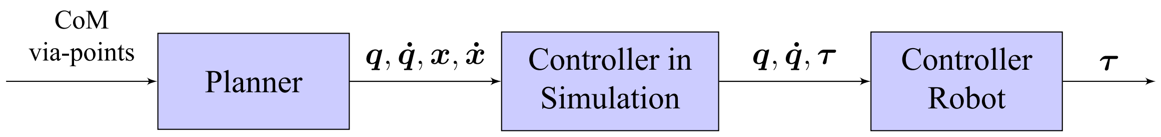


Figure 4.1.: Process structure from the desired via-points as inputs to the planner, to the desired actuation torque in order to finally execute movements with the robot.

4.1.1. Jumping

The goal of this task is to apply a jumping motion with the quadruped Solo 12. The planner (see. 3.2) is used to generate a physically executable jump trajectory. The following CoM via-points where provided to the algorithm:

- contact duration: $t \in [0; 1] \quad x_{\mathrm{CoM}}^{\mathsf{T}} = \begin{bmatrix} 0 & 0 & 0.2 \end{bmatrix}^{\mathsf{T}}$
- CoM via-point $t = 1.25 \quad x_{\mathrm{CoM}}^{\mathsf{T}} = \begin{bmatrix} 0 & 0 & 0.7 \end{bmatrix}^{\mathsf{T}}$
- contact duration: $t \in [1.5; 3] \quad x_{\mathrm{CoM}}^{\mathsf{T}} = \begin{bmatrix} 0 & 0 & 0.2 \end{bmatrix}^{\mathsf{T}}$,

where a first-order friction cone defines the contact points (pyramid)[26]. Further, in addition to a desired initial and final joint configuration (such as in Figure 3.3), another joint via-point is closed to the time when the robot is supposed to jump off ($t = 0.9$) is used. The aim is to keep the mass of the robot compact by having the legs as close as possible to the center of mass. The corresponding joint configuration is visible in Figure 4.2.

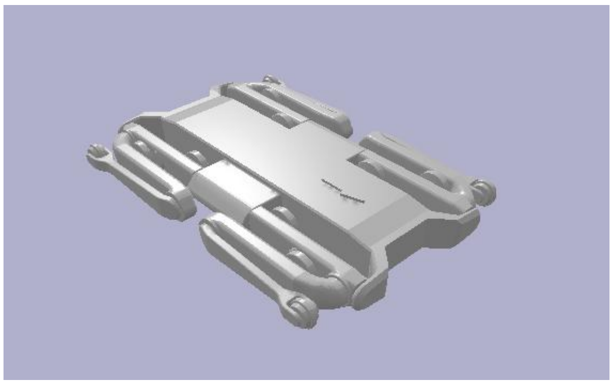


Figure 4.2.: Desired joint via-point configuration, before the robot is supposed to jump off.

Output of the Planner

The planner was executed with the via-points described, which took roughly 20 seconds. The algorithm should now have found a local optimum that satisfies both the kinematic and dynamic objectives. The obtained CoM trajectory along the z axis and the regarding contact plan of the legs is plotted in Figure 4.3.

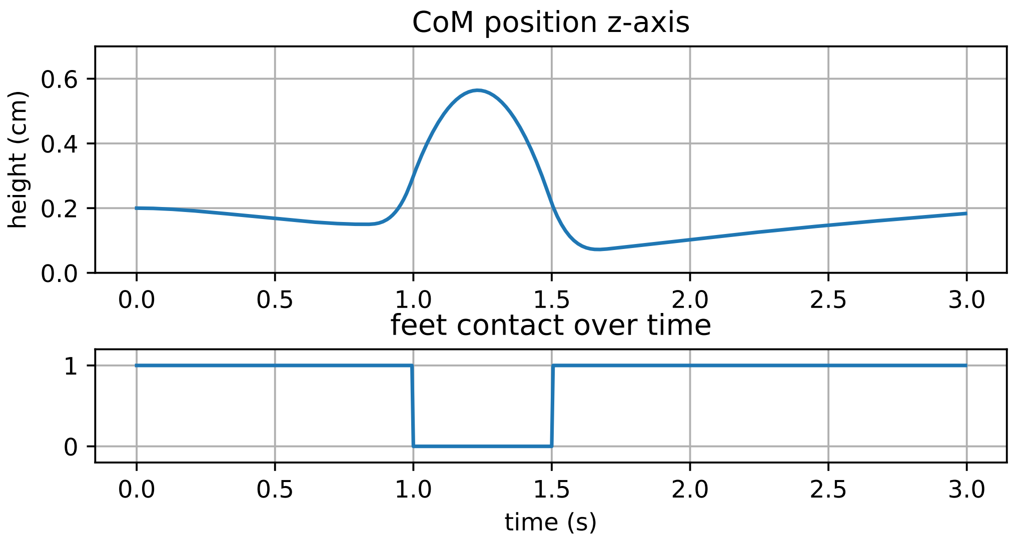


Figure 4.3.: Resulting CoM trajectory along z axis.

Comparing the planner's output of the feet contact plan with the desired feet contact periods shows that the planner can completely fulfill the desired contact conditions. On the other hand, by comparing the desired CoM height with the output of the planner, it is apparent that the planner cannot fully accomplish the jump height. Dynamical limitations most probably cause this, or the time period when the legs are not in contact with the environment is chosen too short.

Result in Simulation

By passing the planned trajectory to the controller (see. section 3.4) in simulated environment, the robot behaves such as pointed out in Figure 4.4.

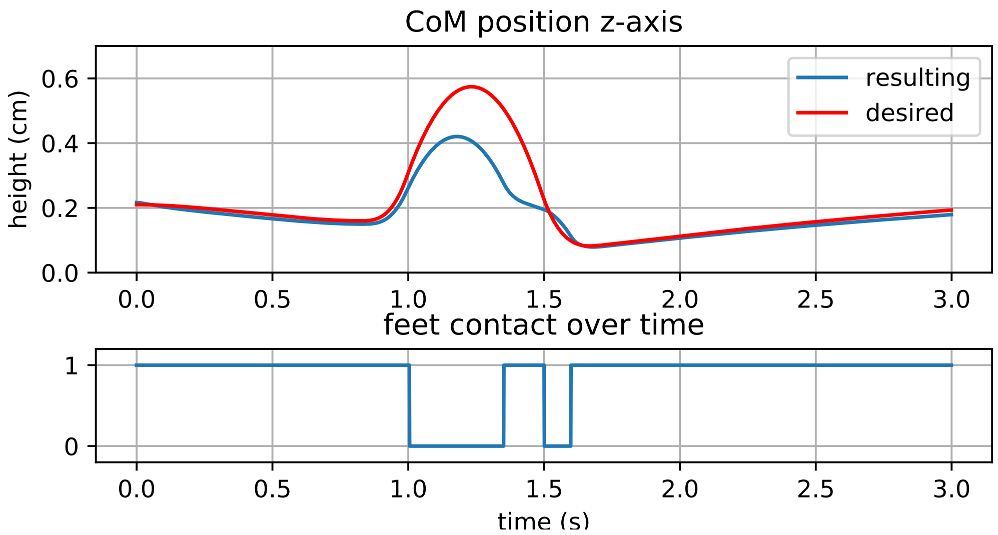


Figure 4.4.: Resulting CoM trajectory along z axis.

It turns out that the robot jumps significantly less high than planned, even though the robot follows up to 1 second almost exactly the planned CoM position. Even by looking at the velocity respectively at the acceleration of the CoM and comparing it with the planned values, there's no significant deviance. The cause could be a friction loss that is not considered in the planning approach or that the gain coefficients of the centroidal PD-controller are not well-tuned.

Behavior in Reality

The stored joint configurations, joint velocity and joint torques obtained from the simulation are finally applied to the real robot. On the first try, the robot switched to error mode when the jumping motion started. Even by holding the robot so that no force acts on the feet, the robot couldn't execute the trajectory.

After testing some different PD-gains on the robot's low-level controller, the robot managed to execute the motion, while holding it. Putting the robot back on the ground the time till the robot switched to error mode was longer, but the jumping still not successful.

4.1.2. Trotting

This task aims to move the robot by walking at a trot. Contact time sections of the feet, such as in Figure 4.5, were defined over a time period of 10 seconds. Additionally, 6 CoM via points were defined to reach in an average speed of $0.047\ m/s$ and move the robot in x direction.

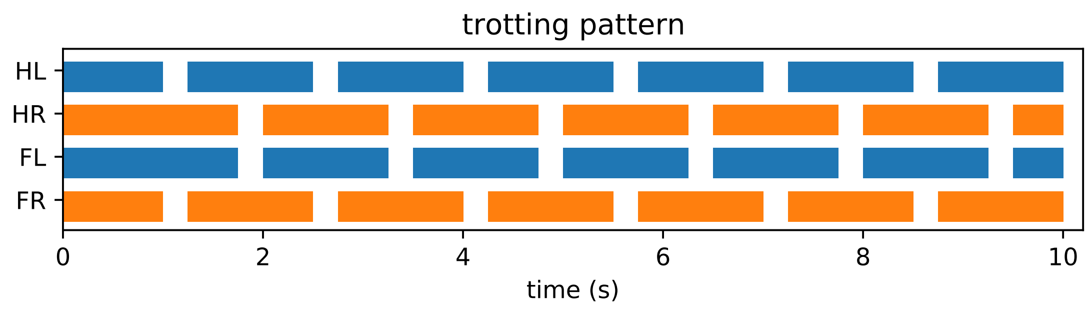
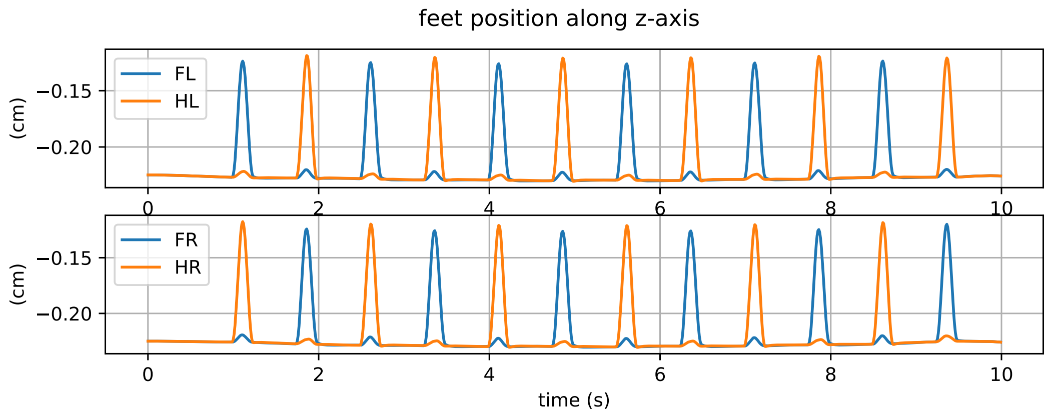


Figure 4.5.: Desired trotting pattern, that is passed to the planner, where colored sections indicating contact with the environment of the leg.

Output of the Planner

To generate the whole trajectory, the planner took this time around 90 seconds running time. Some feet configurations are plotted from the output trajectory in Figure 4.6.



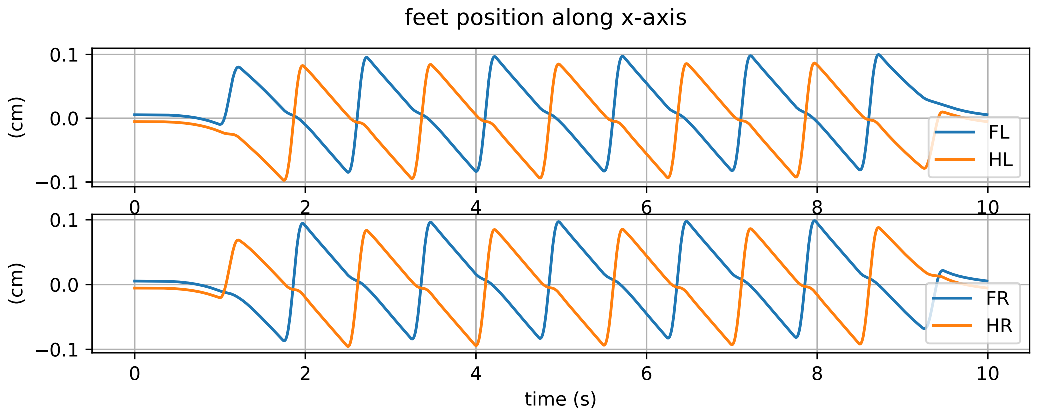


Figure 4.6.: Feet trajectory in task space with respect to the CoM of the robot, obtained from the planner.

The values along the z dimension are negative at the beginning because they are denoted in the robot's base frame. It turns out that the planner has scheduled feet to swing phases, where the highest distance to the floor is about 5 centimeters. By looking at the position of the feet in the x direction, it is noticeable when a foot is in the swing phase, its x value increases because the corresponding moves closer to the robots CoM.

Result in Simulation

By using the centroidal controller (see. section 3.4) in simulation by passing the planned trotting trajectory, the quadruped robot was able to perform trotting walking. The resulting position of the front left foot can be seen in Figure 4.7 as well its desired, planned position.

It can be observed that the resulting position deviates slightly from the reference over time. This seems to be caused by the foot sliding slightly in the direction of motion while it is on the ground.

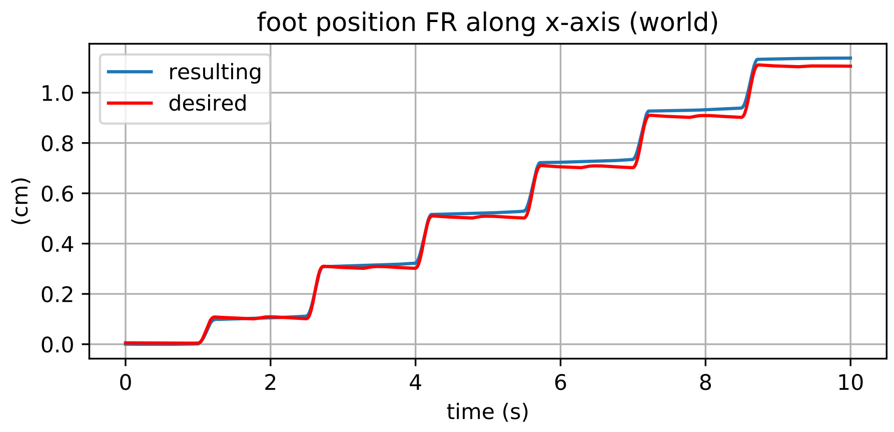


Figure 4.7.: Position of the front left foot over time in x direction. When there is almost no increase of the value, the foot is in contact with the environment.

Behavior in Reality

The collected data from the simulation, consisting of joint configuration, joint velocity and joint torques, is afterward applied to the real robot. Similar to the case of the jump example, the robot switched to error mode at the beginning. It also appeared that the choice of the PD-gains has a crucial impact on the performance. By adjusting them, the robot could finally carry out a trotting motion.

In addition, the speed of the CoM was also changed by up- or down-sampling of the whole trajectory. The robot could thus execute different speeds of the trotting trajectory. It turned out that the gain values were related to the speed since they also had to be readjusted depending on the velocity.

4.2. Controller with MPC

The reactive walking controller introduced in section 3.5, is based on an overall whole-body controller that receives ground reaction forces from the MPC as well as the reference position of the feet in the swing phase and computes for all joints the actuation torques. The overall goal is to track the commanded reference velocity as closely as possible.

Result in Simulation

By commanding with the joystick velocity commands to the robot, it was possible to control the robot in simulation. The robot responded reactively to commands and command changes. It was possible to walk with the robot in a straight line, to walk sideways or to perform turns in place.

In comparison to the original implementation [21], the robot did not behave quite as smoothly visually. The main reason for that seems to be, that their implementation is about 8 times faster. So in average in the original implementation, it took about one sixteenth and in our case the half prediction horizon, till the next MPC window was recalculated.

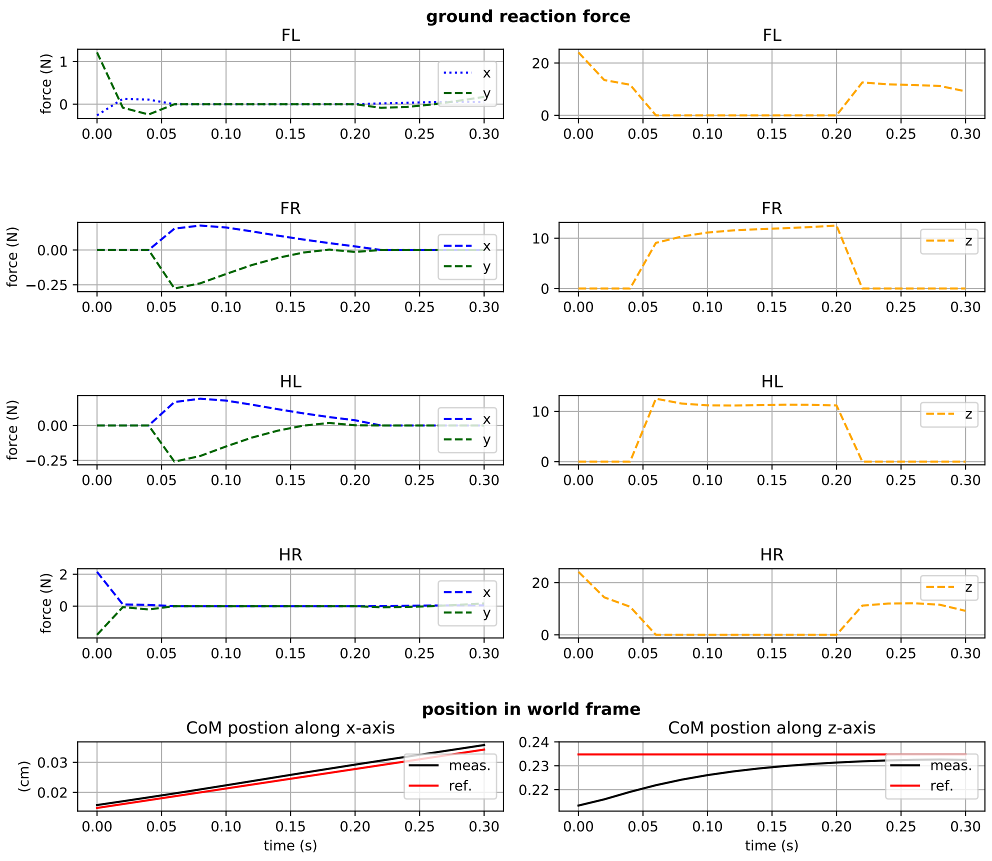


Figure 4.8.: Visualization of one MPC prediction horizon with the calculated optimal ground reaction force and the resulting CoM-position by applying it under the model assumption.

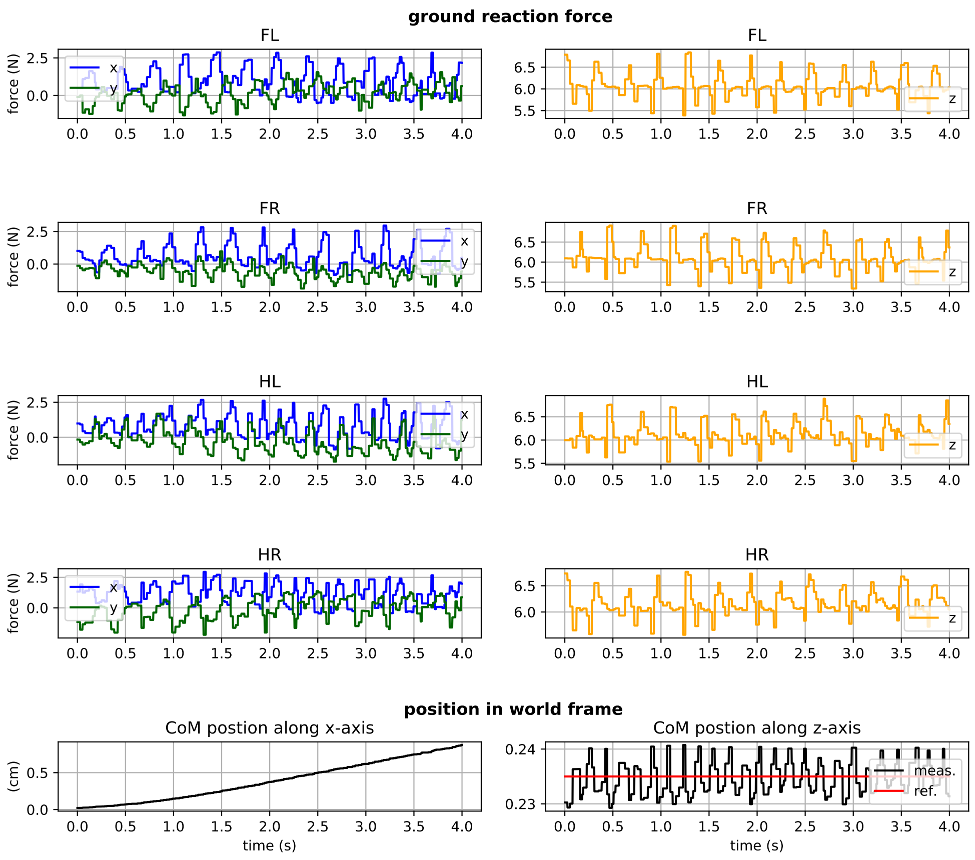


Figure 4.9.: Resulting aggregated ground reaction forces of multiple MPC predictions horizons, where the robot is commanded with a constant velocity in x-direction..

Behavior in Reality

Unfortunately, the controller did not perform well in reality. It was possible to run the controller so that the robot was executing the default trotting pattern. However, even without commanding the robot using the joystick, the robot lost stability after some time.

It is not clear what is causing the problem. It has been observed that the time taken for the robot to lose stability is partly dependent on the calibration. Sometimes the robot was calibrated before usage without the calibration tool.

It is not clear what is causing the problem. It has been observed that the time taken for the robot to lose stability is partly dependent on the calibration. Sometimes the robot was calibrated before use without the calibration tool, and it was found that it turned out to fall after less time. For other movements, such as trotting by trajectory replying, the robot was stable even when calibrated without the tool.

5. Conclusion

Using the planner described in section 3.2 it was possible to generate whole-body motion. So just CoM via-points and reference time steps, when a specific leg is in contact with the environment, have to be passed as input to the planner algorithm. The obtained trajectory consisting of CoM position and velocity, as well as the robots, reference joint positions and velocities, is a compromise between the kinematics and dynamics-related goals. The algorithm is built to ensure inevitable dynamical constraints are valid.

Trajectories generated with this planner are very valuable; they allow to test other controllers on the robot to perform whole-body motions. This way, previously complex tasks can be tried in a simulated environment, improved and finally executed with the robot. Both movements could be executed with the controller in the simulation. The joint configurations, their speed and actuation torque were recorded. The stored sequence was then replayed on the real robot.

The desired jumping motion could not be successfully executed with the robot. The trotting example, on the other hand, could be executed, even at different speeds. Through these experiments is has been observed that the robot is very sensitive to the settings of the PD gains and that these must be varied depending on the task.

The more complex controlling architecture of the reactive walking controller is introduced in section 3.5, gave pleasing results in a simulated environment. The robot could be controlled with the joystick. The robot behaved stably and reactively to the joystick's command changes.

Unfortunately, in a real environment, the robot of the reactive controller with MPC was not stable. The balance was usually lost when no reference speed was sent to the robot.

The most important finding is that the results of the experiments are not entirely consistent. Sometimes it worked quite well to develop something in simulation and then transfer it to reality. However, since some experiments failed, it seems that this effect needs to be studied in more detail in the future. Nevertheless, it is encouraging that a relatively simple approach could be used to generate a rather complicated trajectory and successfully transfer it to the real robot.

6. Lists

Bibliography

[1] Mantian Li et al. “System Design of a Cheetah Robot Toward Ultra-high Speed”. In: *International Journal of Advanced Robotic Systems* (May 2014), p. 1. DOI: 10.5772/58563.

[2] Zhong Yuhai et al. “Analysis and research of quadruped robot’s legs: A comprehensive review”. In: *International Journal of Advanced Robotic Systems* 16 (May 2019), p. 172988141984414. DOI: 10.1177/1729881419844148.

[3] F. Grimminger et al. “An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3650–3657. DOI: 10.1109/LRA.2020.2976639.

[4] ETH Zurich Robotic Systems Lab. *Robot Dynamics: Lecture notes*. ETHZ Press, 2017. URL: https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/documents/RobotDynamics2017/RD_HS2017script.pdf.

[5] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley select coursepack. Wiley, 2005. ISBN: 9780471765790. URL: https://books.google.ch/books?id=muCMAAAACAAJ.

[6] Stéphane Caron, Quang Cuong Pham, and Yoshihiko Nakamura. “Stability of Surface Contacts for Humanoid Robots: Closed-Form Formulae of the Contact Wrench Cone for Rectangular Support Areas”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015 (Jan. 2015). DOI: 10.1109/ICRA.2015.7139910.

[7] Gianluca Garofalo et al. “On the inertially decoupled structure of the floating base robot dynamics”. In: *IFAC-PapersOnLine* 48 (Dec. 2015), pp. 322–327. DOI: `10.1016/j.ifacol.2015.05.189`.

[8] R. Featherstone and D. Orin. “Robot dynamics: equations and algorithms”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 826–834 vol.1. DOI: `10.1109/ROBOT.2000.844153`.

[9] Justin Carpentier et al. “The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives”. In: *SII 2019 - International Symposium on System Integrations*. Paris, France, Jan. 2019. URL: `https://hal.laas.fr/hal-01866228`.

[10] A. Hopgood and an O’Reilly Media Company Safari. *Intelligent Systems for Engineers and Scientists, Third Edition, 3rd Edition*. CRC Press, 2016. URL: `https://books.google.ch/books?id=6mxRzQEACAAJ`.

[11] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 169–170. ISBN: 354023957X.

[12] S. Calinon and D. Lee. “Learning Control”. In: *Humanoid Robotics: a Reference*. Ed. by P. Vadakkepat and A. Goswami. Springer, 2019, pp. 1261–1312. DOI: `10.1007/978-94-007-6046-2_68`.

[13] Sanyal Amit and Ambarish Goswami. “Dynamics and Control of the Reaction Mass Pendulum (RMP) as a 3D Multibody System: Application to Humanoid Modeling”. In: vol. 1. Jan. 2011. DOI: `10.1115/DSCC2011-6086`.

[14] David Orin, Ambarish Goswami, and Sung-Hee Lee. “Centroidal dynamics of a humanoid robot”. In: *Autonomous Robots* 35 (Oct. 2013). DOI: `10.1007/s10514-013-9341-4`.

[15] D.E. Orin and Ambarish Goswami. “Centroidal Momentum Matrix of a Humanoid Robot: Structure and Properties”. In: Oct. 2008, pp. 653–659. DOI: `10.1109/IROS.2008.4650772`.

[16] Alexander Herzog, Stefan Schaal, and Ludovic Righetti. “Structured contact force optimization for kino-dynamic motion generation”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016. DOI: 10.1109/iros.2016.7759420. URL: https://doi.org/10.1109%2Firos.2016.7759420.

[17] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. “Whole-body motion planning with centroidal dynamics and full kinematics”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 295–302. DOI: 10.1109/HUMANOIDS.2014.7041375.

[18] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006, pp. 566–576.

[19] Fares J. Abu-Dakka and Matteo Saveriano. “Variable Impedance Control and Learning—A Review”. In: *Frontiers in Robotics and AI* 7 (2020). ISSN: 2296-9144. DOI: 10.3389/frobt.2020.590681. URL: https://www.frontiersin.org/article/10.3389/frobt.2020.590681.

[20] Donghyun Kim et al. “Highly Dynamic Quadruped Locomotion via Whole-Body Impulse Control and Model Predictive Control”. In: *CoRR* abs/1909.06586 (2019). arXiv: 1909.06586. URL: http://arxiv.org/abs/1909.06586.

[21] Pierre-Alexandre Léziart et al. “Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5007–5013. DOI: 10.1109/ICRA48506.2021.9561559.

[22] Jared Carlo et al. “Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control”. In: Oct. 2018, pp. 1–9. DOI: 10.1109/IROS.2018.8594448.

[23] Shiyu Zhao. “Time Derivative of Rotation Matrices: A Tutorial”. In: *CoRR* abs/1609.06088 (2016). arXiv: 1609.06088. URL: http://arxiv.org/abs/1609.06088.

[24] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006, pp. 267–275.

[25] Donald Goldfarb and Ashok U. Idnani. “A numerically stable dual method for solving strictly convex quadratic programs”. In: *Mathematical Programming* 27 (1983), pp. 1–33.

[26] Milutin Nikolić, Branislav Borovac, and Mirko Raković. “Dynamic balance preservation and prevention of sliding for humanoid robots in the presence of multiple spatial contacts”. In: *Multibody System Dynamics* 42 (Feb. 2018). DOI: `10.1007/s11044-017-9572-9`.

[27] Yannick Millot and Pascal P. Man. “Active and passive rotations with Euler angles in NMR”. In: *Concepts in Magnetic Resonance Part A* 40A.5 (2012), pp. 215–252. DOI: `https://doi.org/10.1002/cmr.a.21242`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/cmr.a.21242`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cmr.a.21242`.

[28] A. Reddy. “DIFFERENCE BETWEEN DENAVIT - HARTENBERG (D-H) CLASSICAL AND MODIFIED CONVENTIONS FOR FORWARD KINEMATICS OF ROBOTS WITH CASE STUDY”. In: Dec. 2014. DOI: `10.13140/2.1.2012.9607`.

[29] *Multibody dynamics notation (version 2)*. English. Dept. of Mechanical Engineering. Report locator DC 2019.100. Technische Universiteit Eindhoven, Nov. 2019.

[30] Tomomichi Sugihara. “Solvability-Unconcerned Inverse Kinematics by the Levenberg–Marquardt Method”. In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 984–991. DOI: `10.1109/TRO.2011.2148230`.

[31] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016–2021.

List of Figures

List of Tables

A. Appendix

A.1. Rotations

To describe a robot in work space, it is mostly represented by its position and orientation. This introduces the need of rotation matrices to express its body frame in the world frame. A rotation in three-dimensional space can be seen as three constitutive elementary rotations on each axis span the coordinate system. Further, rotations can have different interpretations, the so-called active and passive rotations. In robotics it is common to represent a rotation as a mutual orientation between two coordinate systems, which is the definition of a passive rotation. [27]

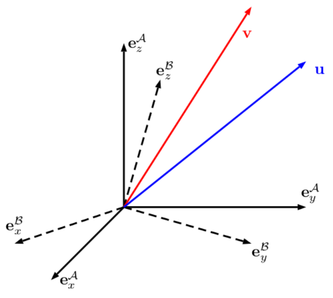

$\mathbf{e}_z^{\mathcal{A}}$
$\mathbf{e}_z^{\mathcal{B}}$
$\mathbf{v}$
$\mathbf{u}$
$\mathbf{e}_y^{\mathcal{A}}$
$\mathbf{e}_x^{\mathcal{B}}$
$\mathbf{e}_y^{\mathcal{B}}$
$\mathbf{e}_x^{\mathcal{A}}$

Figure A.1.: A passive rotation can be seen as a rotation of the coordinate frame, whereas an active rotation as the mapping of an object u to v in the same frame.

- **Passive Rotation:**

  $\boldsymbol{R}_{\mathcal{AB}}$ maps the vector u from frame $\mathcal{B}$ to frame $\mathcal{A}$:
  $\boldsymbol{R}_{\mathcal{AB}}\,{}^{\mathcal{A}}\boldsymbol{u} = \boldsymbol{R}_{\mathcal{AB}}\,{}^{\mathcal{B}}\boldsymbol{u}$

- **Active Rotation:**

  $\boldsymbol{R}$ represents a matrix operator that rotates a vector ${}^{\mathcal{A}}\boldsymbol{u}$ to a vector ${}^{\mathcal{A}}\boldsymbol{v}$ in the reference frame $\mathcal{A}$:
  ${}^{\mathcal{A}}\boldsymbol{v} = \boldsymbol{R}\,{}^{\mathcal{A}}\boldsymbol{u}$

A.2. Rigid Body Dynamics

In robotics it is often not possible to represent and describe a system and its dynamics by all its single bodies separately. Instead the system is mapped by an interconnection of bodies, under the assumption the body is rigid and do not deform by applying external forces. The study of the dynamics of this physical science subdomain is called rigid body dynamics. In practice the previous assumption is never strictly true, as all bodies deform when they move, but often such deformations are negligible in comparison to the overall motion of the body.

To describe the motion of a single particle, its position, velocity and acceleration is needed. For the determination of a rigid body's state, the same information are needed, but also its orientation and rotational motion is necessary. In order to track the state and dynamics under the influence of known external forces, the laws of kinematics, Newton's law, or its derivative, and Lagrangian mechanics are needed.

In the following, some basic formulas for rigid body dynamics are provided.

Linear Motion		**Rotational Motion**	
Position	x	θ	Angular position
Velocity	v	ω	Angular velocity
Acceleration	a	α	Angular acceleration
Motion equation	$x = vt$	$\theta = \omega t$	Motion equation
Mass (linear inertia)	m	$\mathcal{I}$	Moment of inertia
Newton's second law	$F = ma$	$\tau = \mathcal{I}\alpha$	Newton's second law
Momentum	$p = mv$	$L = \mathcal{I}\omega$	Angular momentum
Work	$F\Delta x$	$\tau\Delta\theta$	Work
Kinetic energy	$\frac{1}{2}mv^2$	$\frac{1}{2}\mathcal{I}\omega^2$	Kinetic energy
Power	Fv	$\tau\omega$	Power

A.3. Forward Kinematics

A common systematic approach to calculate forward kinematics is to attach to each link a reference frame with the goal to express the translation from one reference frame to the next by a product of a homogeneous transformation. The Denavit-Hartenberg (DH) convention is a widely used representation for the forward kinematics for 3D serial robot kinematics.

To express a relative displacement between two frames, six independent parameters are used. Namely three for the position translation and three Euler angles to specify the rotation. The DH representation, allows by a clever choice of the origin and the coordinate axes, to reduce the number of parameters to four.

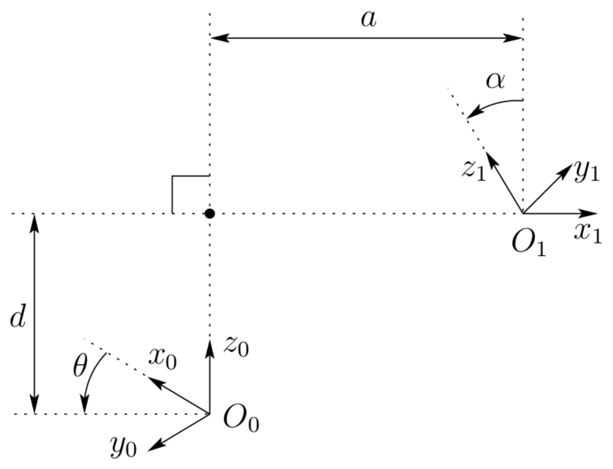


Figure A.2.: Translation between two frames according to DH parameters [5]

For a serial chain robot with n joints, the link frames are sequentially labeled from $0, \cdots, n+1$, where frame 0 is the fixed frame, the frame i is attached to link i and joint i, and $n+1$ represents the end-effector frame (Modified DH parameters [28]). The axes of the joints are always aligned along the z-axis of each child link's reference frame and the direction of a positive rotation about it is determined by the right-hand rule. The x-axis is defined by the cross product of the z-axis from the current link with the z-axis of its child link.

Each homogeneous transformation $\boldsymbol{T}_{i-1,i} \in SE(3)$ (position of frame i with respect to $i-1$) is represented by a product of four basic transformations, consisting of two rotations and two translations with [5]:

$$\boldsymbol{T}_{i-1,i}(\theta_i) = \text{Rot}_{x_{i-1}}(\alpha_{i-1})\ \text{Trans}_{x_{i-1}}(a_{i-1})\ \text{Trans}_{z_i}(d_i)\ \text{Rot}_{z_i}(\theta_i)$$

where,

- θ_i is the joint angle, from x_{i-1} to x_i about the z_i-axis,
- d_i is the link offset, along z_i from the origin of of frame i to the intersection with the axis x_{i-1},

- a_{i-1} is the link length (not necessary the actual physical length of the robot's link), along the x_{i-1}-axis perpendicular to the intersection with z_i,
- α_{i-1} is the link twist, denoted as the angle from z_{i-1} to z_i about the x_{i-1}-axis.

This leads finally to the matrix of the modified DH parameters:

$$T_{i\text{-}1,i} = \left[\begin{array}{ccc|c} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -d_i\sin(\alpha_{i-1}) \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & d_i\cos(\alpha_{i-1}) \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} & \boldsymbol{R} & & \boldsymbol{T} \\ \hline 0 & 0 & 0 & 1 \end{array}\right],$$

where $\boldsymbol{R}$ is the submatrix describing rotation and $\boldsymbol{T}$ is the submatrix representing the translation part.

The forward kinematics of a serial chain with n links based on a joint vector $\boldsymbol{q} \in \mathbb{R}^n$ consisting of the joint angles θ can finally be expressed as:

$$\boldsymbol{T}_{0,n+1}(\boldsymbol{q}) = \prod_{i=i}^{n} \boldsymbol{T}_{i-1,i}(\theta_i)\ \boldsymbol{T}_{n,n+1}$$

$\boldsymbol{T}_{n,n+1}$ stands for the configuration of the fixed end-effector frame n.

As already mentioned, FK refers also to the study of resulting motions in task space, caused by motions in the joint space. The velocity of the end-effector can be expressed by $\dot{\boldsymbol{x}} = \frac{\mathrm{d}x}{\mathrm{d}t}$. As in (2.4), the output of the function for FK depends on time, and when the function is derived with respect to time, the chain rule has to be applied. This leads then to:

$$\begin{aligned}\dot{\boldsymbol{x}} &= \frac{\partial \mathrm{FK}(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{\mathrm{d}\boldsymbol{q}(t)}{\mathrm{d}t} = \frac{\partial \mathrm{FK}(\boldsymbol{q})}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} \\ &= \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \end{aligned} \tag{A.1}$$

In case the FK function maps the position of the end-effector in task space, the matrix $\boldsymbol{J}$, called Jacobian, expresses the linear sensitivity of the end-effector velocity. The Jacobian matrix is defined of partial derivatives as:

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} \frac{\partial x_1}{\partial q_1} & \cdots & \frac{\partial x_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_6}{\partial q_1} & \cdots & \frac{\partial x_6}{\partial q_n} \end{bmatrix}, \text{ with: } \mathrm{FK}(\boldsymbol{q}) = \begin{bmatrix} x_1 \\ \vdots \\ x_6 \end{bmatrix},$$

where its dimension is of $6 \times n$. Each of the rows of the Jacobian correspond to each of the component of the FK functions and each column corresponds to the derivative of a single joint component. Further it is important to note that the Jacobian changes in dependence of the configuration $\boldsymbol{q}$.

It is important to note that according to (A.1), the Jacobian maps the joint velocities to time-derivatives of the end-effector in task space, which means the resulting vector represents a spatial velocity, and not containing the linear and angular velocity. Nevertheless this velocity vector is often named as velocity twist (i.e. as per Park & Lynch .cite modern robotics), whereas sometimes it is denoted as end-effector velocity (i.e Spong). [29] To express the end-effector motion as Twist $\boldsymbol{\mathcal{V}} \in \mathbb{R}^6$, consisting of a linear velocity and angular velocity, the geometric Jacobian is used. In literature when $\boldsymbol{J}$ is written, it mostly refers to the geometric Jacobian.

$$\boldsymbol{\mathcal{V}} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{J}_v(\boldsymbol{q}) \\ \boldsymbol{J}_\omega(\boldsymbol{q}) \end{bmatrix} \dot{\boldsymbol{q}}$$

Depending on the task, it is not always necessary to consider the velocity of the full task space, as only position or orientation is required. Therefore the Jacobian can be spitted into a position and rotation Jacobian. $\boldsymbol{J}_v$, also denoted as transnational Jacobian maps just the linear velocity components and $\boldsymbol{J}_\omega$ influences only the orientation by mapping the angular velocity.

A.4. Inverse Kinematics

There exist two distinct methods to solve the inverse kinematics problem. The analytical approach, which tries to express the equation (2.5) into a closed form mathematical expression, so that finally the whole system can be solved at once. On the other hand by using the numerical method, the solution is approximatively obtained by updating the system iteratively starting from an initial guess $\boldsymbol{q}_0$. Both methods have their advantages and drawbacks. So the analytical method is generally faster and it computes all possible IK solutions, once the equations are deployed. In contrast it is often difficult to derive it mathematically and its usage is not very flexible, as it depends on different kinematic structures as well just applicable to non-redundant robots. Therefore many robotics libraries and simulation environments rely on numerical methods to perpetuate flexibility. A common way to solve IK is (A.2). We can see the task such as we want to bring a point $\boldsymbol{x}$ of the robot (i.e. the end-effector) to a desired position $\boldsymbol{x}_d$. The residual dependents on the robots joint configuration as:

$$\boldsymbol{r}(\boldsymbol{q}) = \boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q}) \tag{A.2}$$

To follow the task, the residual should be brought to zero if possible. By using the Jacobian (A.1) from FK, it is possible to express the velocity of the end-point given the joint velocity $\dot{\boldsymbol{q}}$. Assuming over a small duration of δt a velocity $\dot{\boldsymbol{q}}$ is applied. After this time step, the new residual would be $\boldsymbol{r}' = \boldsymbol{r} - \dot{\boldsymbol{x}}\delta t$. The goal is to cancel it to zero:

$$\boldsymbol{r}' = \boldsymbol{0} \leftrightarrow \dot{\boldsymbol{x}}\delta t = \boldsymbol{r}$$

This brings up the definition of the velocity residual:

$$\boldsymbol{v}(\boldsymbol{q}, \delta t) \overset{\text{def}}{=} \frac{\boldsymbol{r}(\boldsymbol{q})}{\delta t} = \frac{\boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q})}{\delta t}$$

The optimal choice for the joint velocity is then:

$$\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}} = \boldsymbol{v}(\boldsymbol{q}, \delta t)$$

In case the Jacobian would be invertible, the optimal joint velocity would be $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\boldsymbol{v}$. Since this is usually not fulfilled, the system can be solved according to the principle of least squares:

$$\min_{\dot{\boldsymbol{q}}}. \|\boldsymbol{J}\dot{\boldsymbol{q}} - \boldsymbol{v}\|^2,$$

with the best solution by the pseudo-inverse $\boldsymbol{J}^\dagger$ of the Jacobian. By rewriting it as $\dot{\boldsymbol{q}} = (\boldsymbol{J}^\intercal\boldsymbol{J})^{-1}\boldsymbol{J}^\intercal\boldsymbol{v}$, it's noticeable this is the Gauss-Newton Approach. If the FK would be linear in $\boldsymbol{q}$, and at least a solution exists, it would be directly solvable with a given initial guess $\boldsymbol{q}_0$ by:

$$\boldsymbol{q}_1 = \boldsymbol{q}_0 + \Delta\boldsymbol{q}, \text{ where} \tag{A.3}$$

$$\Delta\boldsymbol{q} = \boldsymbol{J}^\dagger(\boldsymbol{q}_0)\Big(\boldsymbol{x}_d - f(\boldsymbol{q}_0)\Big) \tag{A.4}$$

is the change that has to be applied in the joint configuration to reach the target position in task space. However, usually FK is not linear in $\boldsymbol{q}$, but the updated guess from (A.3) should still be closer to the root, and by updating (A.4) with new guess, the solution can so iteratively approached.

In comparison to the Newtons Method, the Newton-Gauss algorithm isn't considering the second-order derivatives of the Hessian, as it just based on a first-order Taylor approximation to locally linearize the gradient difference. This often leads, when the residual term is to large, to instabilities or overshooting of the solution. A simple way to overcome this problem is by scaling each update step:

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i + \alpha_i\Delta\boldsymbol{q}_i, \;\; \alpha \in [0, 1]$$

In case of singularity configuration or generally to increase the numerical stability of IK, regularization methods can be used. A common way for that is by applying Levenberg-Marquardt damping [30], so that the pseudo-inverse is defined such as in (A.5). The dumping matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ is set up proportional to the task error. It is a diagonal matrix with the corresponding dumping factors on its diagonal.

Pybullet that is used in that project for simulated related task, uses in his physics engine to solve IK the Damped Least Squares method, which just another name for the Levenberg-Marquardt method [31].

$$\boldsymbol{J}^{\dagger} = (\boldsymbol{J}^{\intercal}\boldsymbol{J} + \boldsymbol{W})^{-1}\boldsymbol{J}^{\intercal} \tag{A.5}$$