Master's thesis

NTNU
Norwegian University of Science and Technology
Department of Engineering Cybernetics

Nora Graffer

Attitude Stabilization of a Quadruped Robotic System during Free-Flight

Master's thesis in Department of Engineering Cybernetics
Supervisor: Konstantinos Alexis
Co-supervisor: Jørgen Anker Olsen
June 2022

NTNU
Norwegian University of
Science and Technology

Nora Graffer

Attitude Stabilization of a Quadruped Robotic System during Free-Flight

Master’s thesis in Department of Engineering Cybernetics
Supervisor: Konstantinos Alexis
Co-supervisor: Jørgen Anker Olsen
June 2022

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Kunnskap for en bedre verden

DEPARTMENT OF ENGINEERING CYBERNETICS

MASTER THESIS

Attitude Stabilization of a Quadruped Robotic System during Free-Flight

By: **Nora Graffer**
Supervisor: Konstantinos Alexis
Co-supervisor: Jørgen Anker Olsen
June, 2022

Preface

This master thesis is submitted as part of the mandatory requirements for the master degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology. The thesis and all the work behind has been aided and supervised by Prof. Dr. Konstantinos Alexis and PhD candidate Jørgen Anker Olsen at the Department of Engineering Cybernetics, NTNU.

This thesis is partially a continuation of a specialization project conducted during the autumn of 2021 in the sense that the robot in question is equal. However, as the topic of this thesis is very different from the topic of the specialization project, there will be no need to restate neither the background theory, nor the methods that were used.

As the master thesis project is purely a theoretical and simulation based project, there has been no need for special equipment or physical set-up. It has on the other hand been indispensable to have had Matlab, Simulink and the MPC Toolbox extension. Matlab has been the main tool, besides pen and paper, for solving the mathematical equations during the project. All the simulation throughout the project has been done in Matlab Simulink, along with the MPC Toolbox extension for developing the MPC algorithms. The access to these Softwares has been provided by my university, NTNU.

Unless otherwise stated, all figures and illustrations have been created by the author.

I would like to thank my supervisor Konstantinos Alexis for the insight and guidance throughout the project, and for correcting when I made mistakes. I would also like to thank my co-supervisor Jørgen Anker Olsen for answering every question I had during the project, whenever they presented themselves, and for making sure my thesis was going as smoothly as possible.

Lastly, I want to thank my classmates for always lightening up the mood during an otherwise challenging project process.

Nora Graffer
Trondheim, June 2022

Abstract

With rapid progress in the field of legged robots and robots than can jump and land, there will be a growing need for attitude stabilization of falling objects. Attitude stabilization is a control problem of vast interest across many fields, but the main solutions to this date are bases on having aerial flaps, wings or thrusters. These solutions are not well adapted to the four-legged robots in free-flight, and there is a need to develop new solutions. The solution proposed throughout this thesis aims to stabilize the attitude by rotating the robot legs, and thus utilizing the centrifugal force. The system dynamics were modeled in Longitudinal and Lateral dynamics, followed by the process of implementing a control scheme fit to stabilize the attitudes. The attitude stabilization problem was solved with an MPC control architecture that makes the predictions based on the linearized system dynamics models, with the system plant outputs as the correcting input signal. A rigorous tuning process with emphasis on promoting short rise and settling times ensured a control scheme which succeeded in rapid stabilization of the attitude angles. Simulation results for the MPC control task of stabilizing the attitude angles showed that it was able to robustly track a reference signal of $0°$ when the initial attitude angle was between $-30°$ and $30°$, with rise and settling times of < 0.75 and < 3 seconds.

Sammendrag

Med rask fremgang innen roboter med fire ben og roboter som kan hoppe og lande, vil det være et økende behov for attitudestabilisering av fallende gjenstander. Attitudestabilisering er et kontrollproblem av stor interesse på tvers av mange felt, men hovedløsningene frem til idag er basert på å ha luftklaffer, vinger eller thrustere. Disse løsningene er dårlig tilpasset de firbeinte robotene i fritt fall, og det er behov for å utvikle nye løsninger. Løsningen som foreslås gjennom denne oppgaven har som mål å stabilisere attituden ved å rotere robotbena, og dermed utnytte sentrifugalkraften. Systemdynamikken ble modellert i longitudinell og lateral dynamikk, etterfulgt av prosessen med å implementere en kontrollarkitektur tilpasset for å stabilisere attitudene. Attitudestabiliseringsproblemet ble løst med en MPC-kontrollarkitektur som gjør prediksjoner basert på de lineariserte systemdynamikkmodellene, med systemets tilstandsverdiene som korrigerende inngangssignal. En nøye tuningsprosess med vekt på å fremme korte rise- og settling-tider sørget for en kontrollstruktur som lyktes i rask stabilisering av attitudevinklene. Simuleringsresultater for MPC-kontrollprosessen viste at den var i stand til å tracke et referansesignal på 0°, på en robust måte, når den opprinnelige attitudevinkelen var mellom $-30°$ og $30°$, med rise- og settling-tider på $< 0,75$ og < 3 sekunder.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Attitude stabilization is an essential part of control for dynamic systems like water surface, underwater, and aerial vehicles. For some dynamical systems, it is used to track a specific path and remain on course. This is the case for surface vehicles, for example. Other systems perform attitude stabilization because it is crucial for the welfare and security of the system to, for example, not fall over or have a skewed attitude. This is particularly important for aerial vehicles, where deviations in the intended attitude can lead to significant consequences.

For some dynamical systems that fall towards the ground, attitude stabilization must be used to prevent a skewed fall. Such a fall might damage or break the system's mechanical parts. This thesis's project revolves around the four-legged quadruped robot, which is meant to be able to both jump and fall. The falling process is particularly crucial because the robot parts could break if the robot does not land correctly. The quadruped robot has no air compression actuators or wings to smooth the landing and accordingly depends on a correct attitude placement to not fall to ground side-ways for example.

1.1 Purpose

Attitude stabilization as a concept is relatively old. However, it is a rather new and experimental control problem for falling objects without the use of air dynamics. The attitude of an aircraft is stabilized by moving several flaps of the aircraft and with thrusters. Both methods utilize the dynamics of the air in order to change the attitude [1]. A four-legged robot in free flight has to stabilize the attitude without such methods, as the thrusters add unnecessary weight, and the time spent in the air is too short to use moving flaps. Since quadruped robots and jumping robots are relatively new concepts, there has not yet been experimented much with the attitude control problem. As quadruped and jumping robots are becoming increasingly popular, there is a growing need to figure out how to solve this control and stabilization problem.

1.2 Problem Description

The quadruped robotic system will have neither flaps, tails, or thrusters. Each leg is equipped with three actuators. One is placed at the top of the leg to imitate the hip abduction/adduction motion. The second actuator is positioned at the same place as the first and is responsible for the hip flexion/extension motion. The last actuator is placed at the robot's knee and is responsible for the leg's knee flexion/extension.

The hypothesis is that the robotic system can stabilize the attitude solely by the motion of the legs. Each actuator will have a mass, and there will also be a mass placed at the robot's feet. Combined with a rotation, these masses will create a force extending along the attached arm. These forces will impact the main body of the robot and hence the attitude angles of the robot. It is unknown if this method is sufficient for stabilizing the attitude, nor to what extent the attitude can be stabilized solely by the motion of the legs. These questions are nonetheless the main topic of this thesis.

To stabilize the robot's attitude by rotating the leg actuators, there is first a need to model the system dynamics. Following the modeling, a control architecture that can handle the inevitable nonlinear system dynamics is needed. Several mathematical models can represent the system dynamics, but they should be modeled as accurately as possible. As for the control architecture, the complexity of the system dynamics will set demands and limitations to the type of controller scheme. Both PID and MPC controllers will be evaluated and tested before concluding with the optimal choice based on the results.

The specific problem that this thesis aims to solve is to control the attitudes ϕ and θ in two separate controllers for initial angular rotations ranging between $-30°$ and $30°$. With attitude control, the threshold for steady-state error is set at $1°$. The response time will have to be faster than for most process control systems, as the control happens mid-air, and the response time threshold is set at 1 second. Still, the control can not be faster than what can be handled robustly, implying that the attitude rotations and the actuator rotations will have to be stabilized and converge to some value. Angular oscillations, besides an overshoot, are not allowed for the control problem to be deemed sufficiently robust. Stability proofs will not be considered, but signs of instability and robustness will be discussed.

1.3 Related Work

There are few, if any, public papers on attitude stabilization of legged robots during free flight. There are instead examples of attitude stabilization of dynamical systems using the principle of rotating masses. Trung-Dung Chu and Chih-Keng Chen have implemented an MPC controller scheme to stabilize a gyroscopic inverted pendulum [2]. Although not entirely coincident with the attitude stabilization process focused on in this project, they have generated stabilizing torque from the flying gimbals, using the principle of centrifugal force. The same principle will be used to control the attitude rotations during this project. However, instead of gimbals, the weighted knee actuators and weights on the feet will create a centrifugal force when rotated.

There are a wide variety of papers dedicated to the explicit MPC and its features. Ø. Hegrenæs, J. T. Gravdahl, and P. Tøndel evaluate its use for controlling the attitude of a micro-satellite in the paper [3]. The micro-satellite will be modeled and then linearized around some equilibrium point to use the linearized models for the MPC. This modeling, linearizing, and control implementation method will also be used for the robotic system in this project. The results showed great potential, especially if the system was subjected to constraints.

Another piece of important work has been conducted by S. J. Qin, and T. A. Badgwell in the paper [4]. Their work focuses on exploring the different varieties of MPC controllers and describing how they work and for what use they work best. The paper gives insight into how the MPC scheme reacts to different environmental features and how the system, in turn, reacts to different parameter settings and tuning. This paper do not bring up attitude stabilization specifically, but it gives important knowledge as to how a successful MPC controller scheme can be set up and tuned.

1.4 Outline

The dynamical system has to be modeled before diving into control theory and methods for developing an attitude stabilizing control scheme. First, the system will be split into two 2D frames describing longitudinal and lateral dynamics. The method of Lagrange will be used for both of the 2D systems to establish the nonlinear models. Secondly, these nonlinear models will be evaluated and validated through a simulation process to determine if they accurately can describe the real-life system. Following the validation, two different attitude control strategies will be proposed, the PID and the MPC, to solve the attitude control problem. Both of the strategies will be implemented and simulated in Matlab before discussing their results. Finally, the outcome of the attitude control problem process in this thesis will be evaluated, along with possible imperfections and mistakes with the method made along the way, before reflecting on the way forward for this control problem.

Lastly, Table 1.1 below includes the main symbols used throughout this thesis, along with a description and numeric value, if any.

Symbol	**Description**	**Value**
θ	Pitch - attitude angle about y-axis	$(-90°, 90°)$
ϕ	Roll - attitude angle about x-axis	$(-90°, 90°)$
g	Gravity constant	$9.81 m/s^2$
m	Mass of main body with hip actuators	8 kg
$m2_{l/r}$	Left/Right side knee-actuator mass	0.4 kg
$m3_{l/r}$	Left/Right side foot mass	0.3 kg
h	Height of the robot's main body	0.15 m
l	Length of the robot's main body	0.3 m
w	Width of the robot's main body	0.3 m
l_1	Upper leg link	0.15 m
l_2	Lower leg link	0.15 m
$q1_l$	Left-side hip abduction/adduction actuator rotation angle	$(-90°, 90°)$
$q1_r$	Right-side hip abduction/adduction actuator rotation angle	$(-90°, 90°)$
$q2_l$	Left-side hip flexion/extension actuator rotation angle	$(-90°, 90°)$
$q2_r$	Right-side hip flexion/extension actuator rotation angle	$(-90°, 90°)$
$q3_l$	Left-side knee flexion/extension actuator rotation angle	$(-90°, 90°)$
$q3_r$	Right-side knee flexion/extension actuator rotation angle	$(-90°, 90°)$
$\boldsymbol{q}$	System state vector	Specified when used

Table 1.1: Table of the symbols used throughout the thesis for the links and rigid parts of the robot, along with attitude and actuator angular rotations.

Chapter 2

Theory

2.1 Rigid Body Kinematics

Rigid body kinematics describes the kinematics of systems where the objects do not change shape due to the environment. This system is an idealization based on the idea that some bodies do not deform. This approximation works well as long as the changes in shape are minimal compared to the general motion of the body [5].

The assumption that rigid bodies do not deform implies that the distance between each particle-pair of the body remains the same. The body consisting of these particles must, for that reason, move in uniform. If the position of one particle in the body is known, the positions of all other particles are consequently known as well. This simplifies the kinematic calculations as the velocity, acceleration, and force vectors for all the particles in the rigid body can be reduced to only one set of vectors for the entire rigid body.

There follows specific characteristics regarding rigid body motion from the statements above. The first states that all rigid body lines, a line being the vector from one point to another in the body, have the same angular velocity ω and angular acceleration α. The other states that all rigid body motion can be decomposed into the translation of a single point and the rotation about that single point. Regarding the kinematic calculations, these characteristics imply that it is only necessary to evaluate one single point of the body. In addition, the motion is easily calculated as long as the translation and rotation of said point are known.

A vector can be expressed in several coordinate frames, and it is often helpful in robotics to have several coordinate frames for the system. Two of the most used coordinate frames are the Inertial-frame and the BODY frame. The inertial frame stands still regarding the dynamic system, whereas the BODY coordinate frame follows a point in the dynamical system. In other words, the BODY frame follows the same rotations as the rigid body, whereas the Inertial frame does not. [6].

A vector $\boldsymbol{v^a}$ in frame a can be expressed in frame b by means of a rotation matrix from a to b, written R_a^b, as seen in Equation 2.1.

$$\boldsymbol{v^b} = R_a^b \boldsymbol{v^a} \tag{2.1}$$

This rotation matrix, R_a^b, consists of the elements $r_{ij} = \boldsymbol{b_i a_j}$, which are called direction cosines. These rotation matrices can transform all the vectors in one coordinate frame to any arbitrarily coordinate frame, as long as the angular rotations between the two coordinate frames are known. It is also possible to go the opposite direction if that is needed, because the inverse of the matrix in 2.1 is rotation matrix from frame b to a, as seen in Equation 2.2 below.

$$R_a^b = (R_b^a)^{-1} = (R_b^a)^T \tag{2.2}$$

The rotation matrix addresses the rotation from one frame to another. It does not specify any translation from one point to another, which might be helpful in some cases. In order to include both a change in rotation and position, the Homogeneous Transformation Matrix can be utilized. The Transformation matrix presents both the rotation and position of a coordinate frame with respect to a reference frame. This is particularly helpful in robotics and control systems in general. This is because each rigid body of the system can be expressed in terms of the preceding rigid body by switching the coordinate frame using the transformation matrix. For example, in the case of robotic manipulators, a coordinate frame is fixed to each manipulator link with respect to the precedent link. The difference in rotation between the rigid bodies is addressed, as well as the displacement vector. The matrix consists of the rotation matrix from the reference coordinate system to the new coordinate system and the displacement vector, $\boldsymbol{r^a_{ab}}$, which describes the vector from the origin of the reference frame to the origin of the new coordinate frame, relative to the reference frame. Equation 2.3 shows how the matrix is set up.

$$T_a^b = \begin{bmatrix} \boldsymbol{R_a^b} & \boldsymbol{r_{ab}^b} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.3}$$

The homogeneous transformation matrices can be used in combination, such that a transformation from frame a to b, and then from b to c will give the transformation from a to c, as derived in Equation 2.4, retrieved from the book [6].

$$\begin{aligned} T_a^b T_b^c &= \begin{bmatrix} \boldsymbol{R_a^b} & \boldsymbol{r_{ab}^b} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R_b^c} & \boldsymbol{r_{bc}^c} \\ \boldsymbol{0} & 1 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{R_a^c} & \boldsymbol{r_{ac}^c} \\ \boldsymbol{0} & 1 \end{bmatrix} \\ &= T_a^c \end{aligned} \tag{2.4}$$

The Equation 2.4 shows how to move between the rigid bodies in the system by multiplying the transformation matrices. The orientation and position of the frame of rigid body k can be found by the transformation matrix T^{k}_{k-1} with respect to the precedent rigid body, and so forth. The position and rotation of k with respect to the original reference frame are then found by multiplying all the transformation matrices between the frame of k and the original reference frame.

A 3D coordinate system defined by axes x, y, and z has three rotational angles. The angle ϕ about the x-axis, θ about the y-axis, and ψ about the z-axis. In rigid-body motion, these angles are commonly used and referred to as the angles roll, pitch, and yaw, respectively. The resulting rotation matrix R^{b}_{a} from frame a to b is given by Equation 2.5.

$$R^{a}_{b} = R_z(\psi)R_y(\theta)R_x(\phi) \tag{2.5}$$

The rotation matrix describes the rotation about the axes z, y, and x and is a combination of the individual rotation matrices with respect to these rotational angles. Knowing the position and velocity vector in one frame and the transformation matrix between that frame and another means the position and velocity can be expressed in the frame, as seen by Equation 2.1. This is valid for all position and velocity vectors, and is helpful in further calculations.

The velocity of a point is defined as the derivative of the position with respect to time. Hence, the velocity vector is the position vector derivative. Given a position vector in frame a, $\boldsymbol{p^a}$, the position in frame b is again given as:

$$\boldsymbol{p^b} = R^{b}_{a}\boldsymbol{p^a} \tag{2.6}$$

Using the expression above to find the velocity vector in frame b, gives:

$$\boldsymbol{v^b} = \dot{\boldsymbol{p^b}} = R^{b}_{a}\dot{\boldsymbol{p^a}} + \dot{R^{b}_{a}}\boldsymbol{p^a} \tag{2.7}$$

Equation 2.7 shows how to find the velocity vectors in a new frame, given the original position vector and the rotation matrix [6].

The acceleration vectors become increasingly complex as more "links" are involved in the equation. Both linear and angular accelerations might be present in the system and need to be derived from linear velocities and angular velocities, respectively. Depending on the method used for modeling the system, there might not be a need to derive the accelerations. This is one of the benefits of using the method of Lagrange in order to derive the Equations of Motion, as opposed to the traditional Newton-Euler method.

The equations of motion is a widely used term for the motion equations of a rigid-body system. It is a set of differential equations for the velocity and angular velocity of the system. The foundation of the Newton-Euler method are Newton's laws and the theories of rotational dynamics, often credited to Euler [6]. The different position and motion vectors and forces, moments, and torques are differentiated or manipulated to reach the desired resulting differential equations of motion.

Depending on the system, these calculations can be very complex, long, and difficult. Another option for finding the equations of motion is using the Lagrangian method.

2.2 Lagrangian Dynamics

The method of Lagrange is bases around the kinetic energy T and the potential energy U. It takes the difference between the two energies as given below:

$$L = T - U \tag{2.8}$$

where L is called the Lagrangian. The method relies on algebraic operations on the energy equations, using generalized coordinates [6]. Given the Lagrangian L, the next step is to find some differential expressions involving this Lagrangian, to resolve the equation below.

$$\frac{d}{dt}\left(\frac{\delta \boldsymbol{L}}{\delta \dot{\boldsymbol{q}}}\right) - \frac{\delta \boldsymbol{L}}{\delta \boldsymbol{q}} = 0 \tag{2.9}$$

The q represents the states of the system in Equation 2.9. The equation consists of the vectors L, q and $\dot{q}$, and will give as many equations as there are states in q as a result. The resulting set of equations will be equal to the one derived using Newton's law $F = ma$, but it is much simpler for many cases to use the Lagrangian method over the Newton-Euler method. After having found each partial derivative involved in Equation 2.9, in addition to the time-derivatives, it remains only to set them into the equations. In addition, the kinetic and potential energies, T and U, are scalars and usually easy to find, while finding and determining all the forces can be challenging. Especially when several variables are involved and forces point in different and varying directions. [7]. This system described using Equation 2.9 assumes the only forces involved are those conservative forces due to the potential energy, such as the forces due to gravity. However, these are rarely the only forces involved in many robotic applications. The different actuators are driven by a force τ_i, and these forces must be included in the equations as well. Thus, a complete set of equations can be found by

$$\frac{d}{dt}\left(\frac{\delta \boldsymbol{L}}{\delta \dot{\boldsymbol{q}}}\right) - \frac{\delta \boldsymbol{L}}{\delta \boldsymbol{q}} = \boldsymbol{\tau} \tag{2.10}$$

where $\boldsymbol{\tau}$ is the set of generalized actuator forces. Some of these will be zero, such as for the position states, while some will have a value, such as those states that represent the joint positions, usually actuator angles or actuator position [6].

The equations resulting from solving Equation 2.10 are of a form where some parts depend on the double derivatives, $\ddot{\boldsymbol{q}}$, some are dependent on the derivatives, $\dot{\boldsymbol{q}}$, and some are only dependent on the original variables or constants. Because of these characteristics of the equations of motion, it is possible to rewrite the set of equations into the form below, which is particularly helpful when working with manipulators:

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \tau_g(\boldsymbol{q}) = \boldsymbol{\tau} \tag{2.11}$$

The matrix M in Equation 2.11 is not dependent on any derivatives and is the only matrix involved with the double derivatives. It is referred to as the inertia-matrix, or sometimes mass-matrix, and it is a positive definite matrix. This is because it originates from the inertia- or mass-part of the Kinetic energy. Kinetic energy is defined as the energy an object has because of its motion, [8], and is usually written for one state as below:

$$T = \frac{1}{2}mv^2 \tag{2.12}$$

where m is the mass of the object [9]. This equation assumes the velocity is linear. If the system is also subject to an angular velocity, the kinetic energy will include the rotational kinetic energy. Letting ω be the rotational velocity and I the moment of inertia, the rotational kinetic energy will be:

$$T_{rot} = \frac{1}{2}I\omega^2 \tag{2.13}$$

This equation is the result of the same principle used for linear kinetic energy; the kinetic energy of the object is equal to the work done on the object, known as the work-energy principle [10]. For a system with both linear and rotational kinetic energy, the total kinetic energy will be the combined sum of these two. Seen as angular velocity ω can be written using the linear velocity v if the rotational radius r is known, as $\omega = \frac{v}{r}$, both kinetic energies can be written using the linear velocity. The total kinetic energy is then simplified to

$$T = \frac{1}{2}m^* v^2 \tag{2.14}$$

where m^* represents the sum of the mass used in linear kinetic energy and the moment of inertia I divided by the radius r. This is the case for a single-variable system.

However, in multi-variable systems, the equation for kinetic energy can be written:

$$T = \frac{1}{2}\dot{\boldsymbol{q}}^T M(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.15}$$

The M-matrix in Equation 2.15 is the same mass-matrix or inertia matrix, as in Equation 2.11.

The C-matrix in Equation 2.11 is called the Coriolis matrix and deals with the Coriolis and centrifugal terms. These terms and forces are present in systems that do not only have linear velocity and acceleration but that also involve rotational changes and angular velocities and accelerations. The terms in C originate from the part $\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{q}}\right)$ in Equation 2.10 which only involves the kinetic energies. The kinetic energy of the original position will often only include one derivative variable for each state, such as for the velocity of x, y, and z that are often denoted u, v, and w. The derivative of the partial derivative from those expressions will, for that reason, only result in a double derivative. The position of the links, however, is often dependent on the position of the precedent link, and the velocity vector will, for this reason, be dependent on one or more derivative variables. The resulting time-derivative of the partial derivatives will, for this reason, not necessarily be an expression involving the double derivative but an expression involving two or more variable derivatives. This is why the C matrix often has many zeros, for the original position states and on the diagonals, and non-zeroes elsewhere [6].

The vector $\boldsymbol{\tau}_{\boldsymbol{g}}$ denotes the gravitational forces working on the system, and it originates from the partial derivatives of the potential energy; $\frac{\delta U}{\delta \boldsymbol{q}}$, which comes from the potential part of the expression $\frac{\delta U}{\delta \boldsymbol{q}}$. These are the terms and forces due to gravity working on the rigid bodies. Lastly, $\boldsymbol{\tau}$ denotes the actuator torques as described earlier.

These are general patterns the Equations of Motion for manipulators take and are a natural starting point for modeling or working with manipulators or robotic links. Writing the equations in this form also allows for extracting the double derivative of the states, $\ddot{\boldsymbol{q}}$, effectively. Since M is a positive, definite matrix, there is no complication in taking the inverse of it. Other than that, it might be cumbersome and time-consuming depending on the size [9].

2.3 PID

PID-controls are a type of feedback-controllers, as described in the section above. They are still the most common form of feedback controllers, and an industry standard for process control problems [11].

The PID-controller in itself is a simple and easy to implement type of controller. The algorithm for the controller is written as:

$$u(t) = \left(Ke(t) + T_i \int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt}\right) \tag{2.16}$$

In Equation 2.16, K, T_i and T_d are gains, while $e(t)$ is the time-dependent error between the actual variable-value and the desired reference value. The algorithm's output is then $u(t)$, which gives the control actuation. As seen in the algorithm, three parts decide the output u; the error itself, the integral of the error, and the derivative of the error, each with its own control gain. This gives the control algorithm its name, P for proportional, I for integral, and D for derivative. The three different terms can be seen as control actions based on the error's present state, past state, and future state, which will be explained more in detail.

The proportional control part takes the present value for error, $e(t)$, and decides a control action based on its size. As the K-gain is set to negative, the algorithm will give an output that opposes the error. The thought is that the control action will drive the system towards a zero-valued error. This is, however, rarely the case with only proportional control. It is proven that there will always be a steady-state error, a constant error when the system is in a steady-state, in proportional control. In addition, a higher gain K will decrease the error $e(t)$, but it will also lead to larger and quicker oscillations, as the control output $u(t)$ gets larger with larger K.

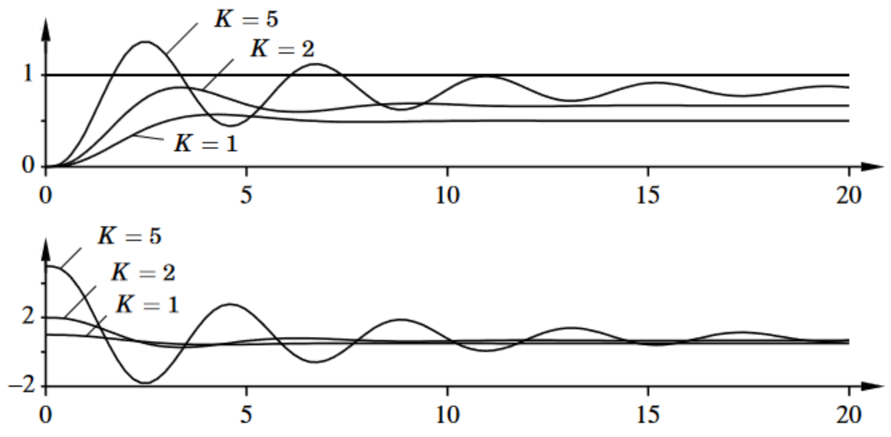


Figure 2.1: System response with a proportional control. Illustration retrieved from [11].

The Illustration 2.1 shows the response of a system using only proportional control. It corresponds accurately with the described theory, where the system response oscillates more as the K gain gets larger and how it approaches the desired value more as K gets larger. It is also clear that there will be a steady-state error with only proportional control.

The integral term works differently than the proportional term. With increasing time, the integral value over the error will increase. This is why the integral term is referred to as past state control action. The longer in time there exists an error, the higher value will the integral term give, and in turn, the higher in value will the output $u(t)$ be, as opposed to the proportional control term, which stays constant with constant error-value. With the integral term, the steady-state error can be eliminated, and the response of the control is generally more effective with an integral term. Decreasing the gain T_i will increase the integral action. However, the decrease in T_i also leads to an increase in oscillations of the system, and there is, for this reason, no benefit in having a too low integral gain.

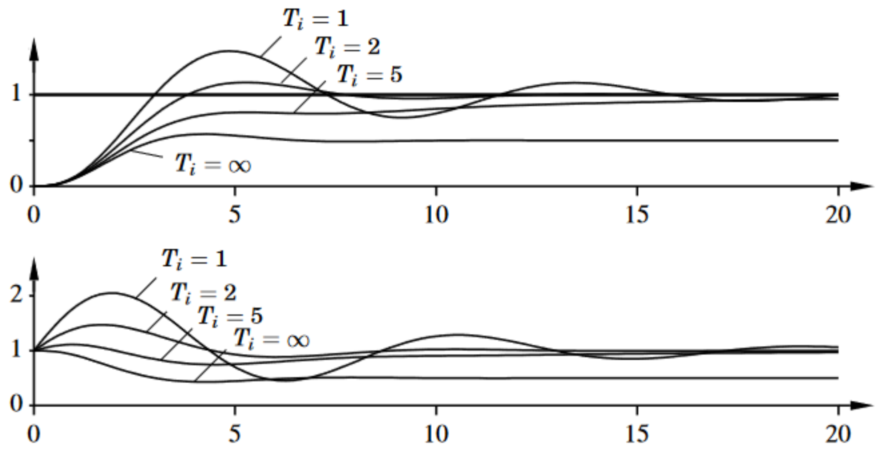


Figure 2.2: System response using proportional and integral control. K = 1. Illustration retrieved from [11].

The Illustration 2.2 shows the response of a particular system with both proportional and integral control. It is clear that a decrease in the gain T_i leads to more oscillation of the system but also a smaller steady-state error. Increasing the T_i gain towards infinity, as is shown, is equivalent to not having any integral effect at all, as the response is the same as for the Illustration 2.1 with K equal to 1.

The derivative term is also called the damper term because it has a damping effect on the system's behavior. Both increases in the proportional term and integral term lead to a system more prone to larger oscillations. The derivative term can dampen these oscillations and provide a smoother system response, with a good choice for the gain T_d. Choosing it either too small or too large will give little damping effect on the system, meaning there is usually an interval for T_d in which the effect of the derivative control term works best [11].

A common technique for achieving a good result is to set the derivative term to zero first and then choose the proportional and integral gain such that the systems response is close to the desired value, meaning the error is as close to zero as possible, perhaps with a certain margin. This will lead to an oscillatory system, and then the damper gain T_d is chosen to minimize these oscillations.

The PID control described in this section is the standard, basic PID control algorithm. There are several variations and changes possible in order for it to suit the control problem at hand better. It is, for example, possible to leave the gains non-constant and create an adapting PID controller. It is also possible to use different techniques and theories to find the best-suited control gains K, T_i and T_d. However, several control problems might be better solved with a different control algorithm than the PID controller. This tends to be the case for MIMO control problems with multiple inputs and outputs. With an increasing number of variables and an increasing complexity in the mathematical model of the system, the PID struggles to control in a satisfying manner. In addition, it might become challenging to combine the controls correctly for systems with multiple control outputs. Either way, the PID controller is a good starting point for determining what type of control algorithm is needed and observing how the system responds to different control scenarios.

2.4 MPC

The PID controller works on many control problems but might struggle as the variables and complexity increase, as mentioned above. Another option for control is the Model Predictive Control - MPC controller. The Model Predictive Controller is a much more advanced technique but also more effective for MIMO-systems, or multiple input, multiple output systems [12].

The MPC controller uses a model of the dynamic system in addition to noise and disturbance models, and/or current state measurements, to estimate the controller state and predict future output values of the plant. These predicted outputs are then subjected to a quadratic programming optimization problem in order to determine the control moves [12], [13]. In other words, the MPC-algorithm analyses the input-output relationship and dynamics of the particular system to determine the control input variables. The MPC algorithm allows for multi-input, multi-output, and inequality constraints on the input and output variables, which is one of the benefits of using it for multi-variable control problems. The algorithm is highly dependent on having an accurate system model since this model is the basis for the predictions. If the predictions are off, the control algorithm will struggle to perform as good predictions, and the system will not behave as wanted. This makes the MPC-control much more complex and prone to errors compared to, for example, the PID-controller. However, in return, it has proven to perform much better for many multi-variable problems if the process model is accurate enough [12]. The objectives of the MPC algorithm have been summarized by Qin and Badgwell, and go as follows:

1. Prevent violations of input and output constraints.
2. Drive some output variables to their optimal set points, while maintaining other outputs within specified ranges.
3. Prevent excessive movement of the input variables.
4. Control as many process variables as possible when a sensor or actuator is unavailable. ([12])

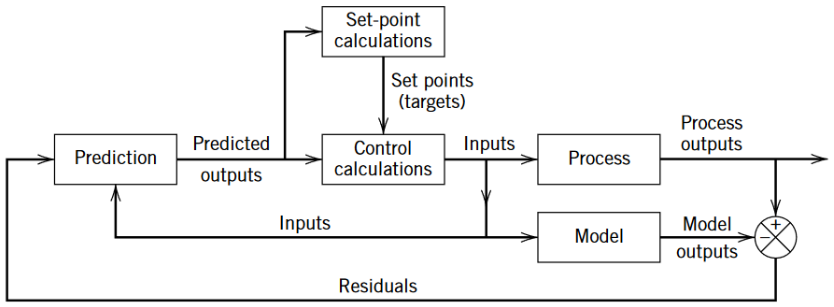


Figure 2.3: Block diagram illustrating the model predictive control - MPC. Retrieved from [12, p. 415]

Figure 2.3 shows the composition of the MPC algorithm. The bottom term called Residuals denotes the difference between the model outputs and the process outputs. With the previous control inputs, these residuals are used in the prediction step, which predicts the next system outputs based on the model. The set-point calculations block is used for calculating the desired state outputs. This includes an optimization scheme in order to determine the control set points. The optimization objective might be to maximize a profit function or minimize a cost function. This block will be discussed more in detail further down. The inequality constraints on the control inputs and output variables can also be included in this block or in the block denoted Control Calculations. As seen from the illustration, the residuals act as the feedback signal of the loop, whereas the error $e(t)$ acts as a feedback signal for the PID controller. It is still possible to look at the feedback signal Residuals as the error, as it is the error between expected or predicted output and actual output. The objective of the MPC in using these various blocks is to find a sequence of control moves or control inputs in order for the predicted outputs to move towards the desired set points in the most efficient way while satisfying the constraints set on the input and output variables [12].

2.4.1 Prediction Step

The Prediction block and the MPC algorithm is dependent on having a system model of a form where the output y is the present system state output given the model. The model can be represented in several different way, for example as a discrete-time state space system:

$$\begin{aligned} \boldsymbol{x}(k) &= A\boldsymbol{x}(k-1) + B\boldsymbol{u}(k-1) \\ y(k) &= C\boldsymbol{x}(k) \end{aligned} \tag{2.17}$$

It can also be represented as a transfer matrix, or a convolutional type model [14]. These are all linear representations and assume the process model can be written linearly. The next step is rewriting the discrete-time process model into a step response model. Continuing with the state space representation from Equation 2.17, the output $\boldsymbol{y}$ can be rewritten as:

$$
\begin{gathered}
Y = G\boldsymbol{x} + HU + F\boldsymbol{u} \\
Y = \begin{bmatrix} \boldsymbol{y}(t+1) \\ . \\ . \\ . \\ \boldsymbol{y}(t+N) \end{bmatrix}
\end{gathered}
\tag{2.18}
$$

The vector $\boldsymbol{Y}$ in Equation 2.18 is the predicted output for all steps from one step forward and until a set value N. In the same way, the vector $\boldsymbol{U}$ denotes all future control inputs from the next time step and until time step N.

$$
U = \begin{bmatrix} \boldsymbol{u}(t+1) \\ . \\ . \\ . \\ \boldsymbol{u}(t+N) \end{bmatrix}
\tag{2.19}
$$

The vectors $\boldsymbol{x}$ and $\boldsymbol{u}$ are the current states and control inputs, $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$. The matrices G, H and F are the model matrices, made out of the state-space system [15]. They are written below as:

$$
\begin{gathered}
G = \begin{bmatrix} CA \\ . \\ . \\ . \\ CA^n \end{bmatrix}, H = \begin{bmatrix} 0 & 0 & ... & 0 \\ CB & 0 & ... & 0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ CA^{N-2}B & CA^{N-3}B & ... & 0 \end{bmatrix} = \begin{bmatrix} h(1) & 0 & ... & 0 \\ h(2) & h(1) & ... & 0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ h(N) & h(N-1) & ... & h(1) \end{bmatrix} \\
F = \begin{bmatrix} h(2) \\ h(3) \\ . \\ . \\ . \\ h(N+1) \end{bmatrix}
\end{gathered}
\tag{2.20}
$$

The matrices in Equation 2.20 are made up of the matrix A, B and C retrieved from the state-space model. The output $\boldsymbol{Y}$ of these calculations now represents the predicted outputs for several steps, from the next step to step number N. This MPC algorithm analyzes the behavior of the process over a horizon in time and is then, hopefully, able to choose the control variables in a manner such that the predicted response has the qualities and characteristics that are wanted [14], [15].

2.4.2 Optimization Function

The block in Figure 2.3 denoted Set-point calculations include an optimization function. The objective of this block is to produce a sequence of set points or targets for the states of the process to follow. These are a set of optimum targets, meaning they have been found using an optimization algorithm in order for the process to have an optimal behavior. They are calculated so that they either maximize or minimize an economic objective function [12]. The cost function to minimize, if this is the optimization problem goal, is usually a quadratic problem function on the form written below:

$$
\begin{aligned}
min_{\Delta U} \sum_{k=0}^{N-1} & (\boldsymbol{y}(k+1)-\boldsymbol{r}(t))^T \boldsymbol{Q}(\boldsymbol{y}(k+1)-\boldsymbol{r}(t)) + (\boldsymbol{u}(k)-\boldsymbol{u}(k-1))^T \boldsymbol{R}(\boldsymbol{u}(k)-\boldsymbol{u}(k-1)) \\
& s.t.: \boldsymbol{u}_{min} \leq \boldsymbol{u}(k) \leq \boldsymbol{u}_{max}, k = 0, ..., N-1 \\
& \Delta \boldsymbol{u}_{min} \leq \boldsymbol{u}(k) - \boldsymbol{u}(k-1) \leq \Delta \boldsymbol{u}_{max}, k = 0, ..., N-1 \\
& \boldsymbol{y}_{min} \leq \boldsymbol{y}(k) \leq \boldsymbol{y}_{max}, k = 1, ..., N
\end{aligned}
\tag{2.21}
$$

The problem involves minimizing the first line in Equation 2.21 above for the control inputs $\boldsymbol{u}$ and state outputs $\boldsymbol{y}$, while not violating the constraints set at the lines two, three and four. The constraints can include control input, control input rate and state output constraints. The matrices Q and R contain the weights set for the control input rates and state outputs, to prioritize which reference values to follow for the state outputs, and penalize control input rate moves.

Chapter 3

Mathematical Modeling

In order to both observe and control the attitude of the robot's main body, there is a need to derive the mathematical models describing the system. The system in this project consists of the robot torso and the four robot legs, each equipped with three degrees of freedom. The system is a highly complex and nonlinear 3D system, for which mathematical modeling can be rather challenging. The resulting equations of motion will also be coupled, further complicating the solution.

There are several options for tackling the problem of finding good, representative mathematical models. Since the system exists in 3D, the most accurate models would also be 3D models. This is, however, more complex and time-consuming than modeling in 2D. It is also possible to split the system into several 2D systems, develop the mathematical models for each of the 2D cases and evaluate them independently. This is a considerable simplification but will nonetheless help understand and develop the equations. The 2D models are, for that reason, a good starting point, regardless of which modeling solution is to be used in the end.

The attitude angles roll, pitch and yaw, respectively $[\phi, \theta, \psi]$ describe the rotation around the axes x, y and z. The mathematical modeling will continue using Euler angles, because that it is the most familiar angular representation. However, they suffer from singularities from some sets of angles, when the middle angle of the roll, pitch, yaw sequence is $+/-90°$. Since the system is split into 2D-frames, this singularity will not be an issue. Additionally, if the switch to a 3D-system was made later, the assumption is that the attitude angles lie below the $+/-90°$ interval anyway.

The robot composition can then be split into three different 2D scenes with one attitude angle present in each scene. The roll rotation around the x-axis is present in the 2D plane $y-z$, the pitch rotation around the y-axis is present in the $x-z$ plane, while the yaw rotation around the z-axis is in the $x-y$ plane. The motions in the $x-z$ plane are usually referred to as the longitudinal motions, with the x-axis named the longitudinal axis. The motions in the $y-z$ plane are referred to as the lateral motions, with the lateral axis y. These are the primary 2D scenes and motions to be evaluated, as they relate to the roll angle ϕ and pitch angle θ, which are the most important attitude angles for the landing of the robot. The movement in these two 2D-scenes are illustrated for longitudinal dynamics in Figure 3.1, and for the lateral dynamics in Figure 3.2.

3.1 Longitudinal

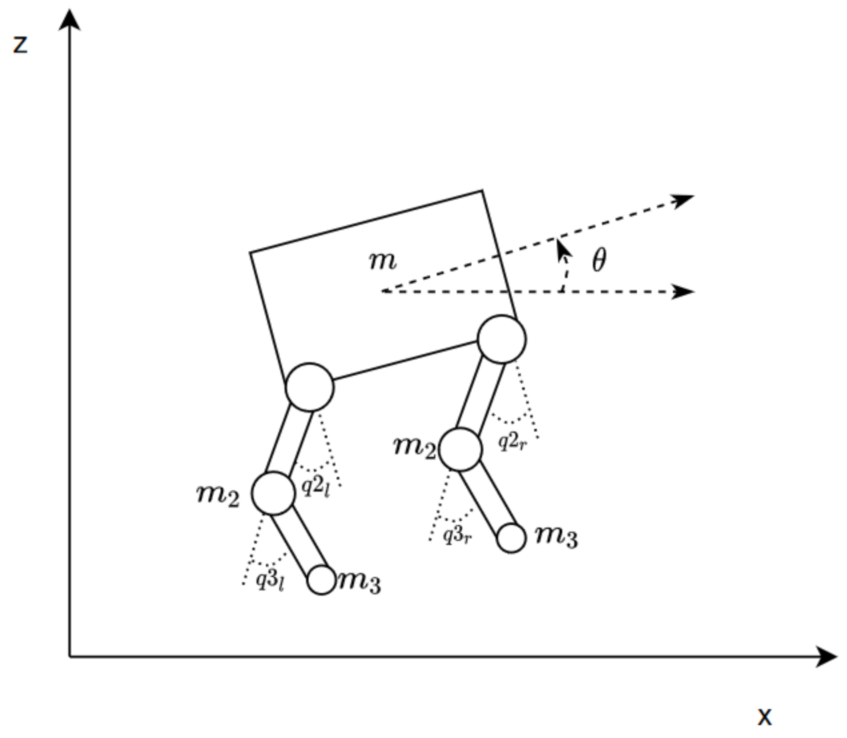


Figure 3.1: Longitudinal sketch of robotic system, with the attitude angle θ about the y-axis.

Figure 3.1 illustrates the quadruped robot in 2D when the x and z axis are regarded. The axis x, y, and z make up the inertial coordinate frame. This coordinate frame is independent of the robot's motion and thus remains constant no matter how the robot moves. In this case, the z-axis points vertically upwards from the ground and is naturally always pointed in the opposite direction of the gravity constant g, which points downwards to the ground. The ground, in this case, is the x-axis. As it is only 2D, the y-axis is not considered, but the x- and y-axis make up the ground plane. In Figure 3.1, the y-axis stands orthogonal to both the x- and z-axis and points directly inwards to the screen, following the right-hand rule [16]. The angle θ shown in the figure follows the common notation described in Section 2.1, where θ is the rotational angle about the y-axis and is usually referred to as the pitch angle for rigid-body kinematics. The angle of θ is positive when the rotation is counter-clockwise and negative when there is a clockwise rotation. This is again a common notation. This rotation makes up a part of the robot's attitude, which is wished to control. As there is no direct force or actuator to control the rotation of θ, it is necessary to control the robot legs such that the rotational angle changes. This is only possible because the legs have inertia and hence contribute to a force on the robot's main body.

As the scene is in 2D, only two robot legs are shown, and they are differed by denoting the leg closest to the z-axis as the left leg and the leg furthest from the z-axis as the right leg. The variable m represents the mass of the main body of the robot and the masses of all the hip actuators. It is simplified because all these masses are assumed to make up a point mass placed in the exact, geometric middle of the robot body. The masses m_2 are the mass of the knee actuators, and the masses m_3 are the masses of the feet. The mass on each foot is necessary for the bottom part of the leg to have sufficient moment of inertia to help contribute to the changes in the rotation of the robot body. The robot-leg links are assumed to be massless, such that there is only mass at the bottom of each link, and these are again assumed to be point masses.

The angles $q2_l$ and $q2_r$ represent the angles of the hip flexion-extension actuators placed in the hips of the robot legs. $q2_l$ denotes the left leg's hip flexion-extension angle, while $q2_r$ denotes the angle on the right legs. The upper leg is assumed to have zero rotation in $q2_l$ and $q2_r$ if the leg is parallel to the height vector of the main robot body. If the rotation θ is zero, the vector from the hip actuator shown in the figure to the knee actuator will be parallel to the z-axis. The angles $q2_l$ and $q2_r$ are also defined such that a positive angle rotates counter-clockwise while a negative angle rotates clockwise. Looking at Figure 3.1, this implies that both of the angles $q2_l$ and $q2_r$, in this case, are negative angles.

The angles $q3_l$ and $q3_r$ represent the angles of the knee flexion/extension actuator joints. As with the angles $q2_l$ and $q2_r$, each leg will have one knee flexion/extension actuator joint. The two legs in this 2D-frame closest to the z-axis are denoted as left legs, and consequently, the knee flexion/extension angle is denoted $q3_l$. The other two legs furthest from the z-axis are the right legs and have a knee flexion/extension angle denoted $q3_r$. Similarly, as with the $q2_l$ and $q2_r$ angles, the legs behind are disregarded in the figure, and only the two legs in front are illustrated.

In order to make Figure 3.1 less crowded and confusing, a few variables have been omitted from the illustration. The gravity constant g points directly downwards, parallel to the z-axis since this frame is an Inertial frame. The sizes and lengths are also missing from the illustration. The robot's main body has a length l that is parallel to the x-axis when θ is zero. This is the scalar length from the robot's left side closest to the z-axis to the right side of the robot furthest from the z-axis. The height of the robot's main body, h, denotes the scalar length from the bottom of the robot's body, the horizontal side closest to the x-axis, and the upper side of the robot body, the side furthest from the x-axis. The four robot legs have equal geometric composition. The length of the upper leg, from the hip actuators to the knee actuators, is denoted l_1, while the length of the bottom leg, from the knee actuator to the foot, is denoted l_2. Because it is assumed that all the weight of the legs is placed in the actuators and the bottom part of the feet, there is no need to know the geometric composition of the leg links l_1 and l_2, except for the length.

The dynamics of the two legs not shown will in this 2D calculation be equivalent to those shown, where the leg behind the left leg has equal dynamics as the left leg and the same regarding the leg behind the right leg. The reason for this is the exclusion of the y-axis, meaning the position of the legs behind will not depend on any other variables than the same variables for the position of the legs in front. To simplify the 2D calculations, they are assumed to have equal motions in this longitudinal 2D frame. Using that they have equal motions in this longitudinal 2D frame, there is no need to model both front and behind legs. The legs behind are hence 'merged' with the legs in front by adding the masses from the legs behind with the legs in front. For the left legs, this implies that the mass of the knee flexion/extension actuator joint of the front left leg now has twice the original mass. The same goes for the foot mass and then similarly for the right legs. This is quite a significant simplification, but the control results should not change too much because of it.

The dynamics of the robot in this 2D-frame can be found by using either the Newton-Euler method or the method of Lagrange, described in Section 2.1 and Section 2.2, respectively. In order to choose the preferred method, it might be beneficial first to decide the state vector. The state vector for the Longitudinal dynamics includes the 2D-position, $[x, z]$, and the rotational angle θ. These states determine the position of the robot's main body. Next, to determine the legs' positions, it is only necessary to include the actuator joint angles. The state vector now includes the variables; $[x, z, \theta, q2_l, q2_r, q3_l, q3_r]$. The positions of the legs behind are assumed to be equal to the positions of the legs in front and are thus excluded. Since all velocities are assumed to be direct derivatives of the main body and angular positions, there is no need to include velocities in the state vector. The final state vector consists of seven variables, a somewhat large number of states. The forces working on the robot are also quite complex, as the robot rotates with the θ angle, and the main body of the robot is subject to forces caused by the rotation of the actuator joints. These qualities of the problem favor the method of Lagrange over the Newton-Euler method. It is possible to derive the Equations of Motion using Newton-Euler. However, it will most likely be more difficult and time-consuming than using Lagrange and a bit unnecessary when it has already been proven that the two methods produce the same result.

The method of Lagrange is described in Section 2.2 and involves finding the kinetic and potential energy of the system. Finding the kinetic energy involves the velocities of the system or the derivative of the positions. As the system rotates around the y-axis with the angle θ, it might be helpful to first find the rotational matrices from the middle of the robot body in the Inertia frame to the different masses of the system. The rotational matrices can then be used to derive the transfer matrices, which will simplify the calculations later. The first step is then to find the different rotational and transfer matrices.

3.1.1 Left Legs

The left-side legs will have a different θ-based rotational matrix than the right-side legs. This is easily seen following the rotational matrix convention found in [6, p. 221] in equations 6.98-6.100.

The rotational matrix about θ can be found using the convention found in [6, p. 221]. Following the same procedure of vector multiplication as in equations 6.98-6.100 on that same page, the matrix elements can be found. First, the Inertial frame is set as the frame a, with unit vectors $\boldsymbol{a_1}$, $\boldsymbol{a_2}$ and $\boldsymbol{a_3}$. The BODY-frame, where the x-axis follows the length of the robot's main body is now called frame b, with unit vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$ and $\boldsymbol{b_3}$. As it is a rotation about the y-axis, all vectors in the x-z plane multiplied with the y-vector will be equal to zero. The y-vector in frame a multiplied with the y-vector in frame b, $(\boldsymbol{a_2} \cdot \boldsymbol{b_2})$, will be equal to 1. The remaining elements will be as follows:

$$\begin{aligned}
\boldsymbol{a_1} \cdot \boldsymbol{b_1} &= [1,0,0] \cdot [\cos\theta, 0, \sin\theta]^T = \cos\theta \\
\boldsymbol{a_1} \cdot \boldsymbol{b_3} &= [1,0,0] \cdot [-\sin\theta, 0, \cos\theta]^T = -\sin\theta \\
\boldsymbol{a_3} \cdot \boldsymbol{b_1} &= [0,0,1] \cdot [\cos\theta, 0, \sin\theta]^T = \sin\theta \\
\boldsymbol{a_3} \cdot \boldsymbol{b_3} &= [0,0,1] \cdot [-\sin\theta, 0, \cos\theta]^T = \cos\theta
\end{aligned} \tag{3.1}$$

The resulting rotation matrix is then the matrix with the elements found above, and will look like below:

$$R_\theta = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.2}$$

Since this frame is a 2D-frame, and the plan is to perform the longitudinal calculations only in 2D for this part, it is not necessary to include the y-row of the rotational matrix. It can then be simplified to a 2x2 matrix:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.3}$$

The matrix above is used in a transformation matrix to give the transformation from the origin of the Inertial frame to the origin of the BODY frame. The origin of the Inertial frame has the coordinates $[x, z] = [0, 0]$. The origin of the BODY frame is the point about which the θ angle rotates. This is where the robot's main body mass is placed in this 2D frame. This point has the coordinates $[x, z]$ in the Inertial frame, and $[0, 0]$ in the BODY frame. Following the convention for Transformation matrices explained in Section 2.1, this implies that the vectorial displacement between the two points is the vector $[x, y]$. The Transformation matrix, which consists of the rotational matrix and the vector displacement, then becomes:

$$T_I^B = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & z \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The next step is finding the Transformation matrix from this point in the middle of the robot's body in the BODY-frame and to the point where the left hip actuators are placed, as illustrated with the white circle in Figure 3.1. For simplicity, the size of the actuator is neglected so that the point is placed precisely at the left bottom corner of the square robot body. This point is also regarded as in the BODY frame, and there is no rotation between the two points involved in the Transformation. The rotation matrix then becomes:

$$R_1^2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.5}$$

The rotation matrix above is an identity matrix of the size of the position vector (2x2). An identity matrix multiplied with a vector is equal to the same vector, such that $I\boldsymbol{v} = \boldsymbol{v}$. This means that this rotation matrix will not change the rotation of the position vector, which is correct. The displacement from the middle of the robot's body and to the bottom left corner, when both points are regarded in the BODY frame, is the vector given below:

$$\begin{bmatrix} -\frac{l}{2} \\ -\frac{h}{2} \end{bmatrix} \tag{3.6}$$

The variables l and h are as described before the length of the main body and the height of the main body, respectively. The resulting Transformation matrix based on this rotation matrix and displacement vector is given below:

$$T_0^1 = \begin{bmatrix} 1 & 0 & -\frac{l}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

The transformation from the origin of the Inertial frame to the bottom left corner of the robot in the BODY frame is then given by the matrix multiplication of the two transformation matrices, T_I^B and T_0^1. This is not necessary for further calculations here, but it might give an idea as to whether the transformations are correct or not. The resulting transformation matrix is given below:

$$T_I^1 = T_I^B T_0^1 = \begin{bmatrix} \cos\theta & -\sin\theta & x - \frac{l}{2}\cos\theta + \frac{h}{2}\sin\theta \\ \sin\theta & \cos\theta & z - \frac{l}{2}\sin\theta - \frac{h}{2}\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The resulting transformation matrix illustrates how the rotations are multiplied when multiplying several transformation matrices, while the displacements are added together but multiplied with the previous rotation.

The next significant point is the position of the knee actuator on the left leg. In figure 3.1, this actuator is also illustrated as a white circle. The situation is simplified by assuming the mass of the leg limb, l_1, has no mass and that all the mass of the actuator, m_2, is placed in a single point at the bottom of the leg limb. This means the scalar length between the previous point, the hip actuator at the bottom left corner of the robot, and the point of the knee mass is precisely the length of the leg limb l_1. There is no rotation between the frames at these points since they are both in the same frame; BODY. The BODY frame only follows the position of the robot's main body, and the changes in the legs do not affect this BODY frame. The rotation matrix between the points is equal to the rotation matrix, R_1^2, which is just the 2x2 Identity matrix. The knee-actuator point's displacement is dependent on the angle $q2_l$, which follows the counter-clockwise direction when positive. The leg limb l_1 and the stippled line in Figure 3.1 given by the angle $q2_l$ form a right triangle. This right triangle gives the basis for computing the displacements in x and z directions. The displacement vector is given below:

$$\begin{bmatrix} l_1 \sin(q2_l) \\ -l_1 \cos(q2_l) \end{bmatrix} \tag{3.9}$$

The rotation matrix with the displacement vector gives the Transformation matrix, T_1^2.

$$T_1^2 = \begin{bmatrix} 1 & 0 & l_1 \sin(q2_l) \\ 0 & 1 & -l_1 \cos(q2_l) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

The last point to evaluate is the feet of the robot leg, as illustrated as a small white circle in Figure 3.1 at the bottom of the robot legs. The mass m_3 is placed here and again assumed to be a point mass at the very bottom of the leg limb called l_2. The leg limb is assumed to have no mass. Similarly, as with the transformation from the bottom left corner of the robot body to the knee actuator, there is no rotation, and the rotation matrix is the 2x2 Identity matrix. The displacement of the point mass m_3 from the point mass m_2 is dependent on both the angle $q2_l$ and $q3_l$. This is more easily seen if the angle θ is zero. The displacement is found with the help of the right triangle that is formed from the angle $q2_l + q3_l$, and the leg limb l_2. The resulting displacement vector is given below:

$$\begin{bmatrix} l_2 \sin(q2_l + q3_l) \\ -l_2 \cos(q2_l + q3_l) \end{bmatrix} \tag{3.11}$$

Inserting this displacement vector and rotation matrix into the transformation matrix gives the resulting transformation matrix from the knee actuator to the foot of the robot legs.

$$T_2^3 = \begin{bmatrix} 1 & 0 & l_2 \sin(q2_l + q3_l) \\ 0 & 1 & -l_2 \cos(q2_l + q3_l) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

The transformation matrices T_I^B, T_0^1, T_1^2 and T_2^3 give the transformations from the origin of the Inertia frame, $[x, y] = [0, 0]$, to the different points on the left side of the robot, (left, bottom corner of the robots main body, knee actuator, and foot). In order to find a specific transformation, the transformation matrices between that point and the origin are multiplied to give the resulting transformation matrix from the origin and to the point. The right-most or third column of the transformation matrices represents the vectorial displacement seen from the original frame. This implies that a matrix multiplication of the matrices between the origin in the Inertial frame and an arbitrarily point b in the BODY-frame gives a transformation matrix $T_{I_o}^{b_{BODY}}$ of which the third column gives the position of point b in Inertial frame. The Transformation matrices give a method for finding the different positions of the robot parts when multiplied together.

In order to derive the kinetic energy of the system, the velocities of the different rigid bodies are needed. The velocity is needed for all parts with mass, which with the simplifications made, the robot's main body, the knee actuators, and the feet are all modeled as point masses. The velocities can be found by taking the time-derivative of the positions. The next step is to retrieve the positions of the left side knee actuator and the left side foot from the transformation matrices and then take the time-derivative of these positions.

The position of the robots main body is already known to be the position vector:

$$\begin{bmatrix} x \\ z \end{bmatrix} \tag{3.13}$$

The time derivative of this position is just the angular velocities along the x-axis and z-axis, such that the velocity vector can be written as:

$$v_m = \begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} u \\ w \end{bmatrix} \tag{3.14}$$

The notations u and w are common for representing the linear velocities, but the notations with $\dot{x}$ and $\dot{z}$ will be used forward in order to keep the relationship between the position $[x, z]$ and the derivatives clear.

The position of the mass m_2 or knee-actuator illustrated as the white circle in Figure 3.1 is found by multiplying the transformation matrices T_I^B, T_0^1, T_1^2, as they are the transformation matrices between the origin in Inertial frame and the position of m_2 in BODY frame. The resulting transformation matrix T_I^2 is given below as:

$$T_I^2 = \begin{bmatrix} \cos\theta & -\sin\theta & x + \cos\theta l_1 \sin(q2_l) + \sin\theta l_1 \cos(q2_l) - \frac{l}{2}\cos\theta + \frac{h}{2}\sin\theta \\ \sin\theta & \cos\theta & z + \sin\theta l_1 \sin(q2_l) - \cos\theta l_1 \cos(q2_l) - \frac{l}{2}\sin\theta - \frac{h}{2}\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

The position of the left side mass m_2 in the Inertial frame is then found by extracting the two first elements of the third column, such that the position is given as the vector below:

$$p_{m2_l} = \begin{bmatrix} x + \cos\theta l_1 \sin(q2_l) + \sin\theta l_1 \cos(q2_l) - \frac{l}{2}\cos\theta + \frac{h}{2}\sin\theta \\ z + \sin\theta l_1 \sin(q2_l) - \cos\theta l_1 \cos(q2_l) - \frac{l}{2}\sin\theta - \frac{h}{2}\cos\theta \end{bmatrix} \tag{3.16}$$

The derivative and velocity of the position p_{m2_l} are found by taking the time derivative of the position vector given above in Equation 3.16. This can be done using the Symbolic Variables in Matlab and defining the different variables as symbols. In addition to the variables x, z, $theta$ and $q2_l$, the derivatives are defined as well, as dx, dz, $dtheta$ and $dq2_l$. Matlab has a function called `diff`, which allows for finding the differentiation of an expression. The function `diff(f, x)` returns the partial derivative of the expression `f` with respect to the symbolic variable `x`. It does, however, not take into account the time derivative of `x`. If the expression is changed to `diff(f, x)`·`dx`, it will give the partial derivative of `f` with respect to `x`, which again is differentiated with respect to time. In order to differentiate the complete expression with respect to time, the different partial derivatives are added together. The final expression for velocity along the x-axis can then be found using Matlab symbolic, and the function below:

```
v_x = diff(f_x, x)dx + diff(f_x,z)dz + diff(f_x, theta)dtheta +
diff(f_x, q2_l)dq2_l
```

The same can be done for the velocity along the z-axis, and the two expressions make up the velocity vector of the mass m_2, v_{m2_l}.

The feet masses, m_3, are positioned at the bottom of the legs, and are assumed to be a point masses. The length from m_2 and m_3, with the assumption that they are both point masses, is then the length of the bottom leg limb, l_2. The position in Inertial frame, as an expression of the variables x, z, l_1, l_2, $q2_l$, $q3_l$ and θ, is found using the same method as with the position of m_2. The matrices multiplied above give the transformation matrix from the origin in the inertial frame to the mass m_2. If the transformation matrix from the mass m_2 to m_3, T_2^3 from Equation 3.12, is included in the matrix multiplication, the resulting transformation matrix T_I^3 will give the transformation from the origin in inertial frame to the mass m_3 in inertial frame. Hence, it is sufficient to multiply the matrix T_I^2, which is already calculated, with the transformation matrix T_2^3. This resulting transformation matrix T_I^3 is given below:

$$T_I^3 = \begin{bmatrix} \cos\theta & -\sin\theta & p_{xm2} + \cos\theta l_2 \sin(q2_l + q3_l) + \sin\theta l_2 \cos(q2_l + q3_l) \\ \sin\theta & \cos\theta & p_{zm2} + \sin\theta l_2 \sin(q2_l + q3_l) - \cos\theta l_2 \cos(q2_l + q3_l) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

The transformation matrix in Equation 3.17 is simplified by interchanging the expressions for the position of m_2 by the variables p_{xm2} and p_{zm2}. The remaining expressions of the third column of the transformations matrix give the change in position from the mass m_2 to m_3 in the Inertial frame. These expressions are the result of the rotation about θ multiplied with the translation vector from mass m_2 to m_3.

The position of the point mass m_3 is found in the same way as with m_2 by extracting the two first elements of the third column of the transformation matrix T_I^3. The position vector $\boldsymbol{p_{3ml}}$ is then given as below:

$$\boldsymbol{p_{3ml}} = \begin{bmatrix} p_{xm2} + \cos\theta l_2 \sin(q2_l + q3_l) + \sin\theta l_2 \cos(q2_l + q3_l) \\ p_{zm2} + \sin\theta l_2 \sin(q2_l + q3_l) - \cos\theta l_2 \cos(q2_l + q3_l) \end{bmatrix} \tag{3.18}$$

The velocity of the point mass m_3 is found by taking the time derivative of the position vector p_{3ml} in Equation 3.18. In the same manner as with m_2, this can be solved using Symbolic variables in Matlab, with the function `diff(f, x)`. The partial derivative is taken for all time dependant variables in the position vector and then added together using Matlab. As opposed to with m_2, the time-dependant variable $q3_l$ has to be included in this expression. The vector that results from this differentiation will be the velocity vector of the point mass m_3 and is here denoted $\boldsymbol{v_{m3_l}}$.

The velocity vectors of the masses on the left hand side of the robot, m, m_2 and m_3, are calculated and denoted $\boldsymbol{v_m}$, $\boldsymbol{v_{m2_l}}$ and $\boldsymbol{v_{m3_l}}$. The first velocity vector gives the velocity of the center of mass of the robot, while the vectors $\boldsymbol{v}_{m2_l}$ and $\boldsymbol{v}_{m3_l}$ give those of the point masses m_2 and m_3, respectively.

3.1.2 Right Legs

The legs on the right side of Figure 3.1 have very similar dynamics as the ones on the left side. The figure of the Longitudinal plane shows that the relationship between the point mass m_2 on the right side, and the bottom right corner of the robot's body, where the right hip actuators are placed, is equal to the relationship between the point mass m_2 on the left side and the bottom left corner of the robot. The angle $q2_r$ is different from $q2_l$, but the dependency between the angle and the point mass m_2 is equal on the left and right sides. In addition, the relationship between the point masses m_2 and m_3 on the right side is equal to the relationship between the two masses on the left side, except for an angle interchange, from $q3_l$ to $q3_r$. This simplifies the calculations because the transformation matrices from the bottom right corner of the robot to the mass m_2 on the right side will be equal to the transformation matrix T_1^2 from the bottom left corner to the left side mass m_2, except with the angle $q2_r$ instead of $q2_l$. The same applies to the transformation matrix from the right side mass m_2 to m_3, where $q2_l$ and $q3_l$ are switched for $q2_l$ and $q3_l$ in the transformation matrix T_2^3 given in Equation 3.12. The rotation matrix about the y-axis with rotation θ is the same as before, R_θ in Equation 3.2. Additionally, the transformation matrix from the origin in the Inertial frame to the center of mass in the BODY frame is the same as before, T_I^B.

The transformation matrix from the center of mass in the BODY frame to the bottom right corner of the robot body in the BODY frame is the only transformation matrix that is not similar to the ones for the left side. The rotation matrix in this transformation matrix will be the same 2x2 Identity matrix, but the displacement vector will be a vector from the middle of the robot's body and to the right instead of to the left. However, only the displacement along the x-axis is different. It will have a positive sign as the vector follows the positive direction of the x-axis instead of a negative sign as in the left-hand side transformation matrix. The transformation matrix from the middle of the robot's main body to the bottom right corner is given below:

$$T_0^1 = \begin{bmatrix} 1 & 0 & \frac{l}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

The transformation matrix from the bottom right corner of the robot body to the right knee actuators is equal to the one for the left side, except with a change in angle:

$$T_1^2 = \begin{bmatrix} 1 & 0 & l_1 \sin(q2_r) \\ 0 & 1 & -l_1 \cos(q2_r) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

The last transformation matrix from the right side mass m_2 to m_3 is equal to the one for the left side, except with the angles $q2_r$ and $q3_r$ instead of $q2_l$ and $q3_l$.

$$T_2^3 = \begin{bmatrix} 1 & 0 & l_2 \sin(q2_r + q3_r) \\ 0 & 1 & -l_2 \cos(q2_r + q3_r) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

Since the transformation matrix, T_I^B from the origin in the Inertial frame to the center of mass in the BODY frame is equal to before, all the transformation matrices are found.

The position vectors of the right side masses m_2 and m_3 are found similarly as for the left side masses by multiplying the transformation matrices between the origin in the Inertial frame and the point mass. The two first elements of the third column of the resulting transformation matrices give the position vectors, $\boldsymbol{p}_{\boldsymbol{m}2_r}$ and $\boldsymbol{p}_{\boldsymbol{m}3_r}$. The velocity vectors $\boldsymbol{v}_{\boldsymbol{m}2_r}$ and $\boldsymbol{v}_{\boldsymbol{m}3_r}$ are found by taking the time derivative of the position vectors, while the velocity vector $\boldsymbol{v}_{\boldsymbol{m}}$ is equal to the one derived for the left side, and given in Equation 3.14.

3.1.3 LaGrange calculation

The method of Lagrange or Lagrangian mechanics is described in Section 2.2. It is a different formulation from the Newton-Euler mechanics but tries to solve the same problem; formulating the dynamics of a system. The Lagrangian, $\boldsymbol{L}$, is the function that results from taking the difference between the kinetic energy T and the potential energy U. The kinetic energy T is the sum of all kinetic energies of the particles in the system. With the assumption of point masses for the robot system, this includes the center of mass of the robot and the points where the masses m_2 and m_3 are placed. Another assumption is that all forces involved are conservative, such as the force due to gravity. The potential energy, U, will, in this scenario, be the summation of all potential energies of the masses, which will be functions of the position vectors.

The total kinetic energy T is the sum of all kinetic energies, where each kinetic energy T_i can be written as in Equation 2.14 from Section 2.1. In this 2D case, the velocity v will be a vector, and the expression will then be as written below:

$$T_i = \frac{1}{2} \boldsymbol{v}^T m_i \boldsymbol{v} \tag{3.22}$$

The vector $\boldsymbol{v}$ in the expression above will be the time derivatives of the position of the masses, which were calculated in the previous section, while the mass m will be the mass of particle i. The total kinetic energy of the system is then the summation of all these kinetic energies with the velocities as expressed in the previous section.

$$\begin{aligned} T &= \frac{1}{2}\boldsymbol{v}_{\boldsymbol{m}}^{T} m \boldsymbol{v}_{\boldsymbol{m}} + \frac{1}{2}\boldsymbol{v}_{\boldsymbol{m2}_l}^{T} m_2 \boldsymbol{v}_{\boldsymbol{m2}_l} + \frac{1}{2}\boldsymbol{v}_{\boldsymbol{m3}_l}^{T} m_3 \boldsymbol{v}_{\boldsymbol{m3}_l} + \frac{1}{2}\boldsymbol{v}_{\boldsymbol{m2}_r}^{T} m_2 \boldsymbol{v}_{\boldsymbol{m2}_r} + \frac{1}{2}\boldsymbol{v}_{\boldsymbol{m3}_r}^{T} m_3 \boldsymbol{v}_{\boldsymbol{m3}_r} \\ &= \frac{1}{2}\Big(m||\boldsymbol{v}_{\boldsymbol{m}}||^2 + m_2\Big(||\boldsymbol{v}_{\boldsymbol{m2}_l}||^2 + ||\boldsymbol{v}_{\boldsymbol{m2}_r}||^2 \Big) + m_3\Big(||\boldsymbol{v}_{\boldsymbol{m3}_l}||^2 + ||\boldsymbol{v}_{\boldsymbol{m3}_r}||^2 \Big)\Big) \end{aligned} \tag{3.23}$$

Equation 3.23 above gives the total kinetic energy of the longitudinal system. The potential energy U is the sum of the potential energies of each particle, or in this system, each point mass. The only force that can contribute to potential energy in this system is gravity. The Inertial coordinate frame is defined such that the gravity constant g points directly downwards and is parallel to the z-axis. Because of this, the position in z for each particle, with the mass of the particle, is sufficient for calculating the total potential energy. With the notation p_{m_z} for the position of the center of mass m in z-direction, and similarly for the masses m_2 and m_3, the total potential energy U can be expressed as below:

$$U = mgp_{m_z} + m_2 g p_{m2l_z} + m_2 g p_{m2r_z} + m_3 g p_{m3l_z} + m_3 g p_{m3r_z} \tag{3.24}$$

The Lagrangian L is then calculated as below, using the total kinetic energy T and total potential energy U.

$$\boldsymbol{L} = T - U \tag{3.25}$$

The state vector was set to be the vector $[x, z, \theta, q2_l, q2_r, q3_l, q3_r]^T$. The Lagrangian L is then a one-dimensional function of these seven states and the constants for length, mass, and gravity. Following the Lagrangian method described in Section 2.2, the next step is to find a set of derivatives of this Lagrangian L. The final object with the Lagrangian method is to solve the equation given below,

$$\frac{d}{dt}\left(\frac{\delta \boldsymbol{L}}{\delta \dot{\boldsymbol{q}}}\right) - \frac{\delta \boldsymbol{L}}{\delta \boldsymbol{q}} = \boldsymbol{\tau} \tag{3.26}$$

where $\boldsymbol{q}$ represents the state vector consisting of seven states, and $\dot{\boldsymbol{q}}$ represents the derivatives of these states. When the partial derivatives are taken over these seven states and derivatives, the final expression from Equation 3.26 on the left-hand side will be a vector with the same number of rows as there are states in the state vector. The actuator torque vector is the variable $\boldsymbol{\tau}$ on the right-hand side. The actuator angles will have a torque variable τ_i, which includes the states $[q2_l, q2_r, q3_l, q3_r]$, while the other states that are not directly subjected to such actuator input will have a zero-valued τ_i-variable. These are the states $[x, z, \theta]$. The partial and time derivatives can be calculated by hand or by a program using symbolic variables. Considering the system's complexity and the number of states, using a computer program to derive these expressions might be less challenging. The calculations will be less prone to errors that easily could occur when they are performed by hand. Again for this task, Matlab has the built-in Symbolic toolbox package that can perform the computations. The variables in the state vector:

$$\begin{bmatrix} x \\ z \\ \theta \\ q2_l \\ q2_r \\ q3_l \\ q3_r \end{bmatrix} \tag{3.27}$$

are defined with `syms` in Matlab, as well as the derivatives of the states:

$$\begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\theta} \\ q\dot{2}_l \\ q\dot{2}_r \\ q\dot{3}_l \\ q\dot{3}_r \end{bmatrix} \tag{3.28}$$

The Lagrangian $\boldsymbol{L}$ resulting from the computed kinetic and potential energy is defined in Matlab, with the variables defined with `syms`. Following this, all the partial derivatives $\frac{\delta L}{\delta \dot{q}_i}$ and $\frac{\delta L}{\delta q_i}$ are computed for each state q_i and each state derivative q_i. This is done with the built-in Matlab function `diff(f,x)`, by setting the lagrangian $\boldsymbol{L}$ for the function f, and the state q_i or the state derivative q_i for x. The time derivative of the state derivative partial derivatives, $\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{q}_i}\right)$, can be computed by the `diff`-function. This time the f-function in `diff` is replaced with the partial derivative $\frac{\delta L}{\delta \dot{q}_i}$, while the differentiation parameter will be dt. This is the same as taking the partial derivative with respect to the time-dependent variables, multiplied with the partial derivative of the time-dependant variables with respect to time, as seen below:

$$\frac{dU}{dt} = \sum \left(\frac{\delta U}{\delta q_i}\frac{dq_i}{dt} + \frac{\delta U}{\delta \dot{q}_i}\frac{d\dot{q}_i}{dt} \right) \tag{3.29}$$

Replacing U in the equation above with $\frac{\delta L}{\delta \dot{q}_i}$ and computing will give the needed time derivatives of the state derivative partial derivatives.

Since the variables τ_i are control variables, all the expressions needed for the lagrangian computations are found. The final computed lagrangian equation will then be as written below:

$$\begin{bmatrix} \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{x}}\right) - \frac{\delta L}{\delta x} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{z}}\right) - \frac{\delta L}{\delta z} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\theta}}\right) - \frac{\delta L}{\delta \theta} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta q\dot{2}_l}\right) - \frac{\delta L}{\delta q2_l} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta q\dot{2}_r}\right) - \frac{\delta L}{\delta q2_r} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta q\dot{3}_l}\right) - \frac{\delta L}{\delta q3_l} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta q\dot{3}_r}\right) - \frac{\delta L}{\delta q3_r} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \tau_{2l} \\ \tau_{2r} \\ \tau_{3l} \\ \tau_{3r} \end{bmatrix} \tag{3.30}$$

Equation 3.30 above gives the symbolic result of the Lagrangian equations. Each row in the equation answers to one of the states. The three first states, x, z and θ, have zero torque input, while the angular states of the leg actuators, $q2_l$, $q2_r$, $q3_l$ and $q3_r$, each have a torque input τ_i.

The system of equations in 3.30 follows the properties of the Lagrangian dynamical systems described in Section 2.2. Some of the terms in the equation depend only on the double derivative $\ddot{\boldsymbol{q}}$ and the state $\boldsymbol{q}$, while others are dependent on the states q and the state derivatives $\dot{\boldsymbol{q}}$. Lastly, some terms of the equation depend only on the states $\boldsymbol{q}$ or on none of the states. The system of equations can be written in the form of Equation 2.11:

$$M(\boldsymbol{q})\ddot{q} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{q} + \boldsymbol{\tau}_g(\boldsymbol{q}) = B\boldsymbol{u} \tag{3.31}$$

This is done by reordering the system of equations and extracting the correct terms into the matrices M and C, and the vector tau_g. The matrix M^{nxn}, referred to as the inertia matrix with n being the number of states, consists of the terms related to the acceleration of the states. The Coriolis matrix C^{nxn} is non-zero for systems that are subjected to rotational changes and angular velocities and accelerations. The vector $\boldsymbol{\tau}_g^{nx1}$ originates from the partial derivatives $\frac{\delta U}{\delta q_i}$, with U being the potential energy of the system. Lastly, the vector $\boldsymbol{\tau}^{nx1}$ is equal to before, with zero for the three first states and a torque value τ_i for the four last states. The vector $\boldsymbol{u}$ consists of the torque values from $\boldsymbol{\tau}$, while the B-matrix maps the control inputs from $\boldsymbol{u}$.

Extracting the different terms into the matrices and vectors M, C and $\boldsymbol{\tau}_g$ can easily be done in Matlab, using the built-in functions `collect(f,x)` and `coeffs`.

The first function rearranges the function f so that the terms involving the variable x are gathered in x, x^2, x^3, etc. The second function returns the coefficients of a function with respect to set of chosen symbolic variables.

When the system of equations from Equation 3.30 is rewritten in the form of 3.31 with the matrices and vectors M, C and $\boldsymbol{\tau}_g$, the vector consisting of the double derivatives of the states, $\ddot{\boldsymbol{q}}$ is only present once in the equation. These are the state accelerations of the system and can be extracted from the equation by moving around on matrices and vectors. The inertia matrix M is a positive definite matrix, and it is possible to take the inverse of M. The equation for finding $\ddot{\boldsymbol{q}}$ will be as written below:

$$\ddot{\boldsymbol{q}} = M(\boldsymbol{q})^{-1}\Big(B\boldsymbol{u} - C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{\tau}_g(\boldsymbol{q})\Big) \tag{3.32}$$

3.2 Lateral

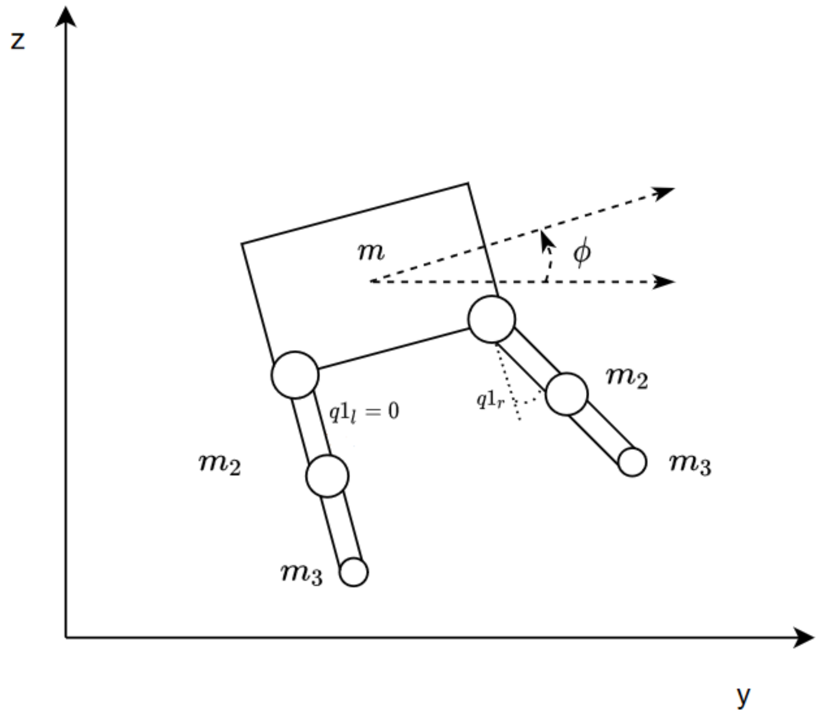


Figure 3.2: Lateral sketch of robotic system, with the attitude angle ϕ about the x-axis.

Figure 3.2 illustrates the quadruped robot in the lateral 2D frame. The robot's view in this frame is directly from the front, as opposed to viewing the robot's side in the longitudinal frame. The y-axis follows the horizontal line while the

z-axis remains the upward axis, similarly to the longitudinal frame. This coordinate frame disregards the dynamics along the x-axis. The gravity constant g points downwards and is parallel to the z-axis. The angle ϕ rotates around the x-axis with the positive angle rotation rotating counterclockwise, as shown in the figure. The ϕ rotation is often referred to as the roll angle, and its derivative is referred to as the roll rate p. ϕ makes up the second of the robot's attitude angles.

The robot legs shown in this illustration are divided into left legs and right legs as in the longitudinal frame. The legs behind are not shown but assumed in this frame to have the same motion as the legs in front, such that the dynamics can be simplified to a robot with two legs that each have the mass of two legs. The masses m_2 are placed at the knee actuators of the legs and are assumed to be point masses. The masses m_3 are placed at the bottom of the legs and assumed to be point masses. The robot links are assumed to be massless.

The angles $q1_l$ and $q1_r$ represent the hip abduction/adduction angles. This is the angle for the hip swing-up, as illustrated in the figure. The angle $q1_l$ represents the left angle, while $q1_r$ represents the right angle. They are defined such that a positive angle rotates anti-clockwise and vice versa. For Figure 3.2, this means that the angle $q1_r$ is positive, while the angle $q1_l$ is zero.

Some variables are omitted from the figure to leave it less crowded. The robot's main body has the same height h as before, while the width, being the horizontal angle following the x-axis, is denoted w. The lengths of the legs remain as before, with l_1 as the upper leg and l_2 the length of the lower leg.

The dynamics of the robot in the lateral 2D-frame can be found using the method of Lagrange, described in Section 2.2. The state vector includes the 2D-position, $[y,z]$, and the rotational angle ϕ. Additionally, the states of the angles $q1_l$ and $q1_r$ need to be included. The state vector becomes: $[y,z,\phi,q1_l,q1_r]$ and has five states. In order to use the method of Lagrange, the kinetic and potential energies of the system are needed, and the dynamical system is again split into a left and right side to determine the positions of the masses, followed by the velocities of the masses.

3.2.1 Left Legs

The position of the masses on the left side can be found using the same method as in the longitudinal frame. The Inertial frame is defined with the origin at the point $[y,z]=[0,0]$. The origin of the BODY frame is the position of the center of the mass of the robot's main body, with the y-axis parallel to the horizontal sides of the robot body (following the stippled line projected by the angle ϕ). The first step in computing the Lagrangian equations is to find the transformation matrices between the origin in the Inertial frame and the different masses.

The transformation matrix from the origin in the Inertial frame to the center of mass in the BODY frame is dependent on the rotational matrix describing the rotation about the x-axis, ϕ. By observation of the Figures 3.1 and 3.2, it is evident that the rotational matrix will look somewhat similar to the one for the angle θ, except where the cosines and sines are placed in the 3x3 matrix. Since this is a rotation about the x-axis, the first row in the matrix will look like: $[100]$, while all other elements regarding x will be zero. The last four elements in the bottom right corner will be the cosine/sine expression, and the final rotational matrix will look like below:

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3.33}$$

The frame is a 2D-frame with the x-axis excluded, and the matrix above can then be simplified to a 2x2 matrix that only includes the cosine and sine expressions:

$$R_\phi = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \tag{3.34}$$

The vectorial translation from the origin in the Inertial frame to the origin in the BODY frame is given by the vector $[y, z]$. Following the convention for transformation matrices from Section 2.1, the transformation matrix given the rotation matrix in Equation 3.34 and the translation vector will be as written below:

$$T_I^B = \begin{bmatrix} \cos\phi & -\sin\phi & y \\ \sin\phi & \cos\phi & z \\ 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$

The next step is to find the transformation matrix from the origin in the BODY frame to the point where the left hip actuators are placed. This will be at the bottom left corner of the robot's main body, given that the mass of the actuator is a point mass. This point is taken in the same BODY frame as the previous point. Hence there is no rotation between the points. The rotational matrix between these points simply becomes the 2x2 identity matrix. The displacement between these points, when they are both regarded in the same frame, is the vector given below:

$$\begin{bmatrix} -w/2 \\ -h/2 \end{bmatrix} \tag{3.36}$$

The variables w and h are the width and height of the robot's body, as described earlier. The transformation matrix based on this rotation matrix and displacement vector is given below as:

$$T_0^1 = \begin{bmatrix} 1 & 0 & -\frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

The next position that is needed is where the knee actuator is placed. In Figure 3.2, this is illustrated as a white circle in the middle of the robot's leg, but it is again assumed to be a point mass. The scalar length between this point and the hip actuator is the length of the upper leg, l_1. This point is in the same BODY frame as before, and again there is no rotation between this point and the last. The rotation matrix is the 2x2 identity matrix, while the displacement in y and z depends on the hip actuator rotation $q1_l$. Following the rules for right triangles, it is found that the displacement along the BODY frames y-axis will be $l_1 \sin(q1_l)$, and the displacement along with the frames z-axis will be $-l_1 \cos(q1_l)$. The transformation matrix T_1^2 will then be as written below:

$$T_1^2 = \begin{bmatrix} 1 & 0 & l_1 \sin(q1_l) \\ 0 & 1 & -l_1 \cos(q1_l) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

The last position needed is the point of the foot, at the very bottom of the robot's leg. This is illustrated as a white circle in Figure 3.2 and is assumed to be a point mass. The scalar length between this point and the mass placed at the knee will be the length of the bottom leg part, l_2. There is no rotation between these two points, as they are still considered in the BODY frame, and the rotation matrix will be the 2x2 identity matrix. The displacement is only dependent on the length of the bottom leg part and the angle $q1_l$. The displacement vector is computed to be: $[l_2 \sin(q1_l), -l_2 \cos(q1_l)]$. The transformation matrix, T_2^3, will be as written below:

$$T_1^2 = \begin{bmatrix} 1 & 0 & l_2 \sin(q1_l) \\ 0 & 1 & -l_2 \cos(q1_l) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.39}$$

The transformation matrices found in this section give the transformation from the origin in the Inertial frame to the different points on the left side of the robot in the lateral 2D frame. The transformation to a point is found by multiplying the transformation matrices between the point and the origin of the Inertial frame. As in the section for the longitudinal frame, the position of the point is found in the third column of the transformation matrices. The two first elements of the third column represent the position (or change in position) along the y-axis and z-axis, respectively. The positions of the knee actuator mass and foot mass are found by this method, multiplying the transformation matrices between origin and point and then extracting the two first elements of the third column. For simplicity, the positions will be referred to as $\boldsymbol{p_m}$, $\boldsymbol{p_{m2_l}}$ and $\boldsymbol{p_{m3_l}}$.

To compute the kinetic energy of the system, the velocity vectors of the masses are needed. To find the velocity vectors, it suffices to take the time derivative of the position vectors $\boldsymbol{p_m}$, $\boldsymbol{p_{m2_l}}$ and $\boldsymbol{p_{m3_l}}$. This can be done with Symbolic Matlab, as before, or computed by hand. For simplicity and to avoid common calculation errors, it is computed with Matlab, and the differentiation function `diff(f,x)`. The first velocity vector, $\boldsymbol{v_m}$, is simply the derivative of the vector $[y, z]^T$ and will be as written below:

$$\boldsymbol{v_m} = \begin{bmatrix} \dot{y} \\ \dot{z} \end{bmatrix} \tag{3.40}$$

The remaining two velocity vectors, $\boldsymbol{v_{m2_l}}$ and $\boldsymbol{v_{m3_l}}$ are found by taking the time derivative of $\boldsymbol{p_{m2_l}}$ and $\boldsymbol{p_{m3_l}}$ in Matlab.

3.2.2 Right legs

The right legs will be the legs on the right side in Figure 3.2. These are the legs furthest away from the z-axis. The angle of the hip actuator will for these legs be the angle $q1_r$, instead of $q1_l$. The only difference between these legs and the left legs is the second transformation matrix, from the center of mass in the BODY frame to the right-side hip actuator instead of the left hip actuator. This transformation, T_0^1, will have zero rotation because both points are in the same BODY frame, while the displacement vector will be a vector that goes from the center of mass to the bottom right corner of the robot main body. The resulting transformation matrix will then be as written below:

$$T_0^1 = \begin{bmatrix} 1 & 0 & \frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.41}$$

The transformation matrix T_I^B will be equal to the ones used in the calculations for the left legs, and the matrices T_1^2 and T_2^3 will be equal as well, except with the change in angle from $q1_l$ to $q1_r$. The transformation matrices are written below:

$$\begin{gathered} T_I^B = \begin{bmatrix} \cos\phi & -\sin\phi & y \\ \sin\phi & \cos\phi & z \\ 0 & 0 & 1 \end{bmatrix}, T_1^2 = \begin{bmatrix} 1 & 0 & l_1\sin(q1_r) \\ 0 & 1 & -l_1\cos(q1_r) \\ 0 & 0 & 1 \end{bmatrix} \\ T_2^3 = \begin{bmatrix} 1 & 0 & l_2\sin(q1_r) \\ 0 & 1 & -l_2\cos(q1_r) \\ 0 & 0 & 1 \end{bmatrix} \end{gathered} \tag{3.42}$$

The position vector of the center of mass $\boldsymbol{p_m}$ will be equal to the one computed for the left side. The same applies to the velocity vector $\boldsymbol{v_m}$. The position vectors of the right side masses m_2 and m_3 are found using the same method as the left side masses by multiplying the transformation matrices and extracting the first two elements of the third column in the third column matrices. The resulting position vector are denoted $\boldsymbol{p}_{\boldsymbol{m2}_r}$ and $\boldsymbol{p}_{\boldsymbol{m3}_r}$. The velocity vectors $\boldsymbol{v}_{\boldsymbol{m2}_r}$ and $\boldsymbol{v}_{\boldsymbol{m3}_r}$ are found by taking the time derivative of the position vectors, $\boldsymbol{p}_{\boldsymbol{m2}_r}$ and $\boldsymbol{p}_{\boldsymbol{m3}_r}$, in Matlab with the function `diff(f, x)`.

3.2.3 LaGrange calculation

The method of Lagrange will be used for the lateral dynamics in the same way as for the longitudinal dynamics. The Lagrangian $\boldsymbol{L}$ is found by subtracting the potential energy U from the kinetic energy T.

The total kinetic energy T is the sum of all the kinetic energies involved in this system in 2D. When the velocity vector for each mass of the robot is known, each kinetic energy part will take the form below:

$$T_i = \frac{1}{2}\boldsymbol{v}^T m_i \boldsymbol{v} \tag{3.43}$$

where $\boldsymbol{v}$ is the velocity vector of the mass m_i. These vectors are already computed in the section above by differentiation of the position vectors. Summarizing the kinetic energies for the masses m, $m2_l$, $m3_l$, $m2_r$ and $m3_r$ will give the total kinetic energy of the system, T.

The total potential energy U is the sum of the potential energies of each particle in the system, or in this case, the sum of the potential energies for each point mass. The only force contributing to potential energy in this system is the gravity force caused by the gravity constant g. The gravity constant points directly downwards, parallel to the z-axis in the inertial frame. Each potential energy part will then take the form as written below:

$$U_i = m_i g p_{mi_z} \tag{3.44}$$

where p_{mi_z} is the position along the z-axis for the mass m_i, and g is the gravity constant. Summarizing the potential energy parts will give the total potential energy of the system, U.

The Lagrangian $\boldsymbol{L}$ is computed as before with the equation:

$$L = T - U \tag{3.45}$$

The state vector, $\boldsymbol{q}$, for this system was set to be the positions in $[y, z]$-coordinates, and the angles ϕ, $q1_l$ and $q1_r$, and the Lagrangian will be a function of these five states. The next step is to solve the Lagrange equation:

$$\frac{d}{dt}\left(\frac{\delta \boldsymbol{L}}{\delta \dot{\boldsymbol{q}}}\right) - \frac{\delta \boldsymbol{L}}{\delta \boldsymbol{q}} = \boldsymbol{\tau} \tag{3.46}$$

The vector $\boldsymbol{q}$ is the state vector consisting of the five states, while $\dot{\boldsymbol{q}}$ is the vector that consists of the derivatives of the states. The vector $\boldsymbol{\tau}$ consists of the actuator torques and will have zero for the elements associated with the position states and ϕ, and a torque variable τ_i for the actuator angle states, $q1_l$ and $q1_r$. The partial derivatives and time derivatives are calculated with the symbolic toolbox in Matlab in the same manner as for the longitudinal dynamics. All time-dependent variables are defined with `syms`, and then the different derivatives for each state q_i and each time derivative state $\dot{q_i}$ are computed with the function `diff`. The variables τ_i are control variables and do not need any computation in this step. The final computed Lagrange equations will look like the system below:

$$\begin{bmatrix} \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{y}}\right) - \frac{\delta L}{\delta y} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{z}}\right) - \frac{\delta L}{\delta z} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\phi}}\right) - \frac{\delta L}{\delta \phi} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta q\dot{1}_l}\right) - \frac{\delta L}{\delta q1_l} \\ \frac{d}{dt}\left(\frac{\delta L}{\delta q\dot{1}_r}\right) - \frac{\delta L}{\delta q1_r} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \tau_{1l} \\ \tau_{1r} \end{bmatrix} \tag{3.47}$$

Equation 3.47 above gives the symbolic result of the Lagrange equations. Each row in the final system is associated with one of the states from $\boldsymbol{q}$. The three first states will have zero torque input, while the two last angular states each have a torque input τ_i. This system of equations has the same characteristics as other systems of equations of motion described in Section 2.2, in the sense that it can be rewritten in the form below:

$$M(q)\ddot{\boldsymbol{q}} + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{\tau}_g(\boldsymbol{q}) = B\boldsymbol{u} \tag{3.48}$$

This is done by reordering the system in Equation 3.47 and extracting the different terms into the matrices M, C and the vector $\boldsymbol{\tau}_g$. The vector $\boldsymbol{u}$ consists of the input torques τ_i, and the matrix B maps these control inputs correctly. This is done in the same way as for the longitudinal dynamics, and the resulting matrices M and C will be 5x5 matrices, while the vector $\boldsymbol{\tau}_g$ will be a 5x1 vector.

When the system of equations from Equation 3.47 is rewritten into the matrices and vectors M, C, $\boldsymbol{\tau}_g$, B and $\boldsymbol{u}$, the vector for the double derivative of the states, $\ddot{\boldsymbol{q}}$, can be found by rearranging the terms in Equation 3.48. The final equation for, $\ddot{\boldsymbol{q}}$ will be as written below:

$$\ddot{\boldsymbol{q}} = M(\boldsymbol{q})^{-1}\Big(B\boldsymbol{u} - C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{\tau}_g(\boldsymbol{q})\Big) \tag{3.49}$$

Chapter 4

Validation of Mathematical Models

Mathematical Model Validation is defined as the process of determining the degree of how accurate a mathematical model is to the real-life system it tries to describe [17]. This process is accomplished by comparing predictions from a model to experimental results. A hypothesis or prediction of how the real-life system would behave when subjected to some action or event is compared to the result of the mathematical model when the model is subjected to the same actions or events. This comparison will evaluate how trustworthy the model is and how it might differ from the real-life system.

Mathematical model validation is related to how accurate the models need to be. With a higher need for accuracy, the more critical the validation will be, and the more time should be spent on this part to ensure the model can be used for the intended purpose. The 2D models that were found in the previous chapter will be the basis on which the control laws will depend. The different control laws, such as a PID algorithm or an MPC control law, will have different needs for accuracy. PID does not necessarily depend on having a perfectly accurate model, while the MPC algorithm will perform predictions based on the mathematical models. For this reason, accurate modeling is much more critical when using MPC than a common PID control. Since the goal is to control the attitude angles of a complex robotic system, it is highly likely that the PID control proves insufficient and that an MPC will perform better. Consequently, it is reasonable to assume that a model of decent accuracy is needed later when determining what control law to use.

In order to determine the accuracy of the model, a series of experiments or tests is often conducted, tailored to the specific system and model. This is also how the longitudinal and lateral mathematical models will be evaluated. Some of the tests that will be performed are to observe how the moments of inertia around the robot affect the attitude angles and robot motions. Other tests are performed to determine the robot's motions when all initial conditions are zero.

4.1 Validation Experiments

The first set of Validation Experiments aims to determine how the different masses around the robot body and in the legs affect the attitude angles and motion. In other words, the goal is to observe how the moments of inertia originating from the masses in the leg actuators affect the overall motion of the robot's body.

If one of the actuators on the leg rotates counter-clockwise, the resulting torque will point upwards. Following the same notion, a torque that points directly upwards caused by the rotation of an actuator will lead to a counter-clockwise rotation about the center of mass. To sum this up, a counter-clockwise, or positive, rotation of one of the actuator angles will contribute to a counterclockwise, or positive, rotation about the center of mass, and vice versa. The force is dependent on the mass of a rotating object, in this simplified case, a point mass for each actuator. The more mass there is, the bigger the force will be. If there is no mass, there will be no force. To test how well the mathematical model correlates to the theory, each mass of the legs should be tested 'in isolation'. All the masses are set to a value very close to zero, except for one. Torque is then applied to the different legs to observe if it is correct that the leg parts without mass do not contribute to a rotation in the attitude angle when rotated. At the same time, the leg part with mass does contribute to a rotation in the attitude angle when rotated. It is difficult to determine a threshold for when the results are satisfactory but to set a reasonable limit; the massless leg parts should not contribute to an attitude angular rotation of more than $0,5°$ for the mathematical models to be deemed accurate.

When the actuators are viewed as pendulums, there will be a tension force, $\boldsymbol{T}$, working from the end of the pendulum where the mass is placed, along with the link towards the top part of the pendulum. However, this will no longer be the case when the robot is in free fall and there is no angular velocity of the pendulum. The tension force is present when there is tension on the link, usually caused by the pendulum mass being subjected to the gravity force $\boldsymbol{G}$, while the top of the pendulum is fastened. In a free fall, both the robot body and the legs will fall towards the ground with the same acceleration g. Hence there will be no tension in the link caused by gravity. When the mass moves with an angular velocity ω, there will again be a tension force, $\boldsymbol{T}$, present. This is simply because the end of the pendulum has a different motion than the top of the pendulum. The resulting net force working from the mass of the pendulum and upwards will be as given below:

$$F = \frac{mv^2}{l} \tag{4.1}$$

This force will have an impact at the top of the pendulum and, for this reason, impact the attitude angle and motion of the robot. Since the force works in the direction of the link of the pendulum, the impact of the force on the robot body is dependent on where the mass is placed. For example, if the actuator angle of the pendulum is positive and is moving in some direction, the force will point backward. However, if the angle is negative, the force will point forwards (positive x or y-direction). Since the center of mass of the robot's body is placed in the middle of the robot body, the left leg will have a force working from behind, while the right leg will have a force pointing from the front. A positive angle of $q2_l$ when the actuator is moving in some direction with angular velocity ω will lead to a force contributing towards a negative θ or ϕ angle as long as $q2_l$ remains positive. On the other hand, a positive angle of $q2_r$ will lead to a force contributing towards a positive θ or ϕ angle when an angular motion is present. The contribution on the θ and ϕ angle will change sign somewhere on the negative angle side of the left leg and the positive angle side of the right leg. This will happen when the force vector following the link passes under the center of the mass. Precisely at what angle this will happen can be calculated, but it will have to happen before $q2_l$ reaches $-90°$ from the positive side and before $q2_r$ reaches $90°$ from the negative side.

The force equation above describes how the force is dependent on mass m, the velocity $\boldsymbol{v}$, and the length of the pendulum arm l. The linear velocity $\boldsymbol{v}$ is related to the angular velocity ω in a manner described by the equation below

$$v = r\omega \tag{4.2}$$

where r denotes the radius of the pendulum circle. This radius will have the same length as the length l from the force equation, which leads to a new way of writing the force equation for $\boldsymbol{F}$:

$$\boldsymbol{F} = \frac{m\boldsymbol{v}^2}{r} = \frac{m(r\omega)^2}{r} = mr\omega \tag{4.3}$$

The equation above is proportional to the radius r, such that a small radius length should result in a minimal force $\boldsymbol{F}$. If the length is zero, there would be no force, regardless of how large the angular velocity ω is. To test this theory on the mathematical models, the leg lengths l_1 and l_2 are tested in 'isolation' in the same way as for the masses, such that only one of the four leg parts have a value at the same time. In contrast, the remaining other three leg parts are set to a value very close to zero. The same tests are conducted for the masses by rotating the different actuator angles to observe how this affects the robot's attitude angles and overall motion. The threshold is set to the same limit as for the test with the masses, where the actuators with zero-length leg parts should not contribute to an attitude angular rotation of more than $0,5°$.

The last validation experiment aims to determine if the mathematical models correlate correctly to Newton's first law, which states that a body at rest or with constant speed will remain at rest or with constant speed unless a force acts upon it. The quadruped robot is in free flight and is, for this reason, not at rest nor a constant speed. However, as the gravitational force only acts in a straight line parallel to the z-axis, there should be no change in the motions along the x- and y-axis when there is no other force acting than the gravitational force. The object of the validation experiment is to verify that when the actuator angles have no velocity, there will be no change in any of the angles of the robot, neither the actuator angles nor attitude angles. The threshold for when the mathematical models are deemed accurately enough is set to $0,1°$. If either actuator angle or attitude angle has a change of more than $0,1°$, the mathematical models need to be revisited and changed.

The threshold limits from the different validation experiments can be summed up in the table below:

Threshold Limits during 10 seconds	
Actuator with no mass	+/- 0.5°
Actuator with no scalar leg length	+/- 0.5°
Zero initial rotation	+/- 0.1°

Table 4.1: Table of threshold limits for the attitude angles and zero-initial condition actuator angles

Table 4.1 above gives the threshold limits from the three different validation experiments. The threshold limits for the two first experiments apply to the attitude angles. In contrast, the limit for the last validation experiment applies to all actuator angles and the attitude angles.

4.2 Set Up

The mathematical models need to be simulated over time to perform the validation experiments given in the previous section. This can be done using a computational program, preferably one where it is feasible and easy to solve the differential equations found in chapter 3 over time. This method is chosen to simulate the models because of previous experience and knowledge with Matlab's ODE-solvers in Matlab. The ODE solver `ode45` is a good starting point to test the simulation of models as it is viewed by many as the general-purpose nonstiff differential equation solver. A stiff differential equation is a differential equation for which certain numerical methods are numerically unstable for solving it. Whether or not these differential equations are stiff is not known, but it is at the same time not that important. Nonstiff differential equation solvers can still solve them, but it will take more time, and they might need to make the step size very small. If the ode45 solver takes an unreasonably long time to solve the equations or comes to a halt, it is time to consider changing the ODE solver to another more suitable for solving stiff equations.

The program in Matlab is set up around the differential equation given below:

$$\ddot{\boldsymbol{q}} = M^{-1}(B\boldsymbol{u} - C\dot{\boldsymbol{q}} - \tau_g) \tag{4.4}$$

where the matrices and vectors M, C, B, and τ_g were found in the previous chapter. The ODE solver ode45 solves a differential equation over a determined timespan, tspan, with a set of initial conditions, $x0$. The differential equation solver also needs an ode function as input, which is a function that takes a column vector $\boldsymbol{x}$ as input, and returns the column vector $\dot{\boldsymbol{x}}$. This is where the differential equations found earlier are added, and the input $\boldsymbol{x}$ is usually only the state vector that in this system is denoted $\boldsymbol{q}$. Since the vector of variables $\ddot{\boldsymbol{q}}$ is a double derivative, the ode45 solver will integrate the equations in order to find the vector of variables, $\dot{\boldsymbol{q}}$. It only retrieves the first derivative. However, the actual states $\boldsymbol{q}$ need to be found as well. This can be done by rearranging the system of equations and using the fact that the derivative of the state q_i is the derivative state $\dot{q_i}$. The state vector used in ode45 will then be double in length compared to the original state vector and will have the general form following the equation below:

$$\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= \ddot{x_1} = \ddot{q_1} \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= \ddot{x_3} = \ddot{q_2} \\
&. \\
&. \\
&.
\end{aligned} \tag{4.5}$$

The pattern of Equation 4.5 is continued for all of the states in the state vector $\boldsymbol{q}$, and the system of equations is then the combination of the double derivatives and the second derivatives. The differential equation ode45 can then be used with this as the state vector, and it will return the integrated states $\boldsymbol{q}$ and the derivative states $\dot{\boldsymbol{q}}$.

Ode45 is set up with an ode function as described above and a time span of 10 seconds. The initial conditions $x0$ is a vector of the initial conditions for each of the states returned by the ode function, that in this case, will be twice the size of the original state $\boldsymbol{q}$. The vector $\boldsymbol{x0}$ will include both the initial conditions for the state derivatives $\dot{\boldsymbol{q}}$ and the states and can be changed in order to, for example, give the actuator angles initial velocities. Additionally, the torques τ_i can be set to different values in the vector $\boldsymbol{u}$. The masses and lengths in the equations are defined as constant at the top of the program, such that they can easily be changed throughout the validation experiments.

4.3 Longitudinal

The first validation experiment is verifying that the moments of inertia behave correctly by setting the different masses to zero and rotating the robot legs in different directions.

In the first test, all the actuator masses $m2$ and $m3$ on the left and right legs are set to 0.0001, except the left side mass of the knee actuator, $m2_l$, which is set to have the original value, $2 \cdot 0.4kg$. All the leg lengths l_1 and l_2 have their original value. There is no input u on any of the actuator angles. Lastly, the initial conditions for x and z are set to $3m$ and $10m$, respectively. The initial angular value of θ is set to 20°. All other initial conditions are set to zero, except the angle $q2_l$, which is set to 10°, and the angular velocity of the left side actuator angle $q2_l$, $dq2_l$, which is set to 5 degrees per second.

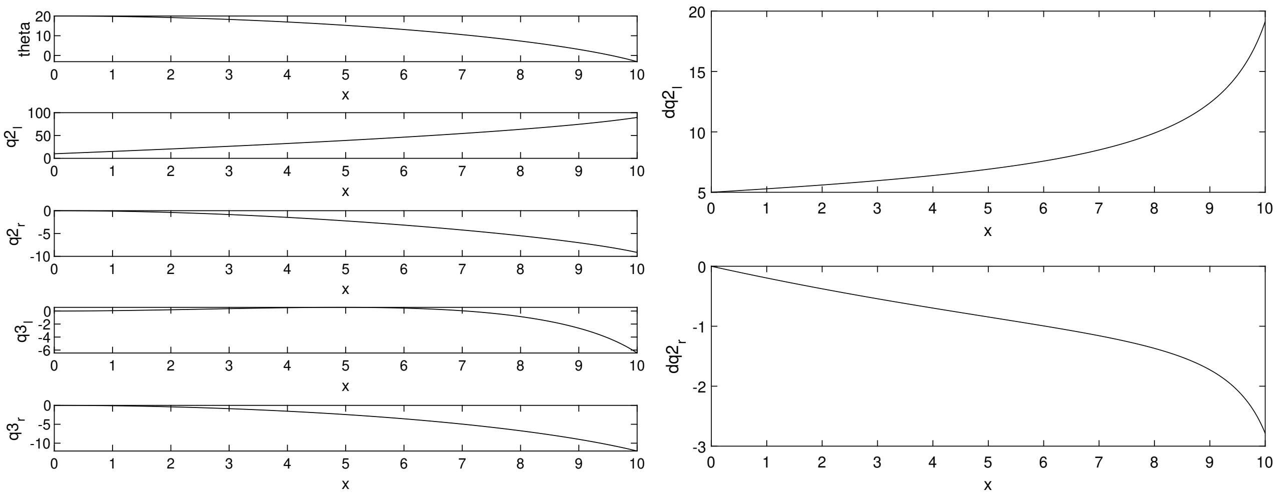


Figure 4.1: Simulation of the situation where only the left side mass $m2_l$ has a non-zero value. The attitude angle θ is given a start value, and the angular velocity $dq2_l$ has a start value. The left side knee actuator, $q2_l$ has a start value of 10°

Figure 4.1 above shows the first described situation, where only the left side knee actuator has a non-zero mass. When the rotation angle of $q2_l$ start at with a positive angle, 10°, and is given an initial angular velocity for $dq2_l$, the force caused by the link tension will contribute to a negative rotation of the attitude angle θ, as described in the previous section. This is also the case in the simulation, as the angle θ starts at 20° and slowly decreases to a negative valued angle. Observing the right-side plots of the angular velocities, they follow the states on the left side well, with an increase in the left side $q2$ angle and a decrease of the right side $q2$ angle.

To verify how the force contributing to the change of θ is dependant on the position of the rotating mass m_2, the following simulation scenario is equal to this previous scenario in Figure 4.1, except the start value of the angle $q2_l$ which now begins at $-70°$ instead. The results are shown below.

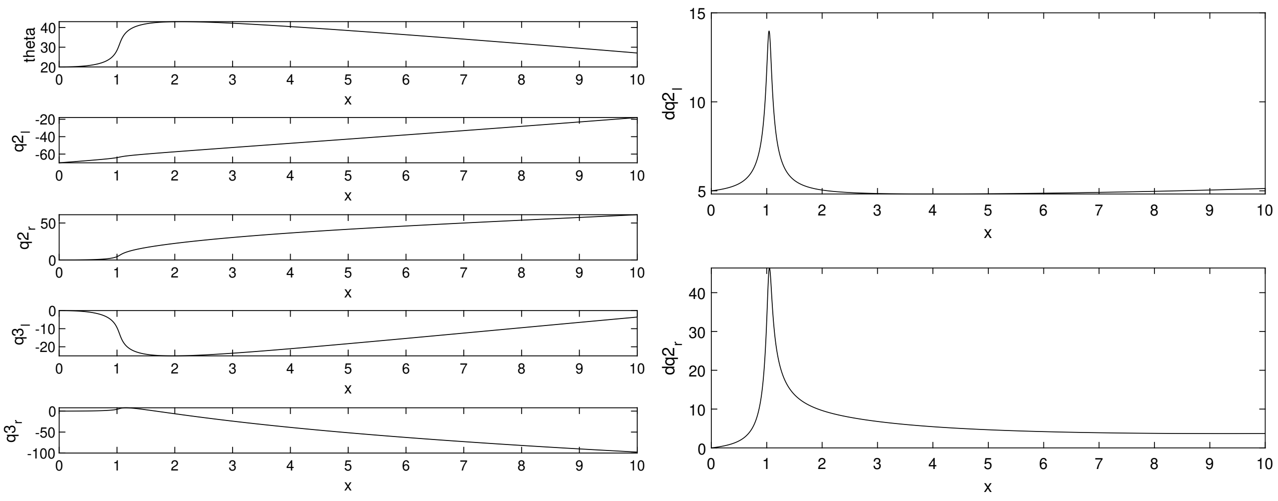


Figure 4.2: Simulation of the situation where only the left side mass $m2_l$ has a non-zero value. The attitude angle θ is given a start value, and the angular velocity $dq2_l$ has a start value. The left side knee actuator, $q2_l$, has a start value of $-70°$

The plots in Figure 4.2 show that the attitude angle θ increases in value with the angular velocity $dq2_l$ instead of decreasing as in the previous scenario. The mass m_2 on the left side is placed much further to the left side, and the force vector following the link from mass m_2 to the hip actuator will travel below the center of mass of the robot body, as explained in the previous section. The plots fit well with the theory.

The scenario simulated in the following plots shows instead how the system behaves when it is the right hip angle $q2_r$ that has an initial angular velocity of 5 degrees per second. In contrast, the left side angular velocity $dq2_l$ has zero initial value. The masses have the same value as before, with only the left side mass m_2 having a non-zero value. The results over ten seconds are shown in the plots below.

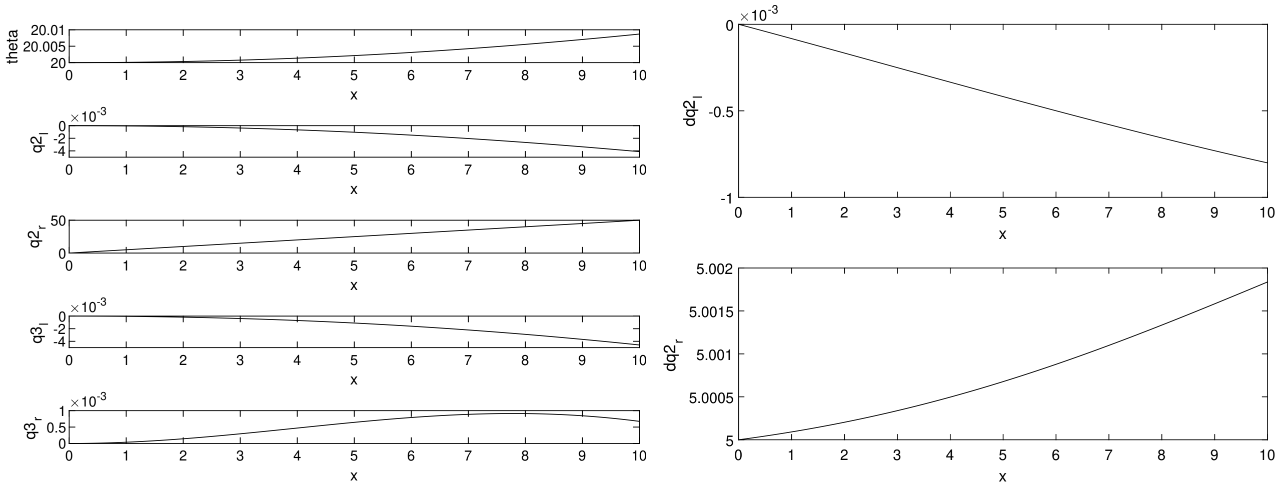


Figure 4.3: Simulation of the situation where only the left side mass $m2_l$ has a non-zero value. The attitude angle θ is given a start value, and the angular velocity $dq2_r$ has a start value of $5\frac{^\circ}{s}$.

The plots in Figure 4.3 show how the angular velocity of the right side mass $m2$ has no effect on the attitude angle θ, nor any real effect on any of the other angles either. There are minor changes in the states shown in the left plot, but apart from the angle $q2_r$, they are so small that they most likely stem from numerical errors of the Matlab computation and not any real change. The results from these plots fit very well with the theory and hypothesis of how the system would behave. Only the angular rotation of a mass would be able to change the rotation of θ, while the angular rotation of massless parts has no effect.

The same validation experiment is performed for the same scenario with initial angular velocity for the masses $m3_l$ and $m3_r$, meaning the variables $dq3_l$ and $dq3_r$ were tested with initial values. The results showed that neither of these angular velocities affected the attitude angle θ. The result for θ is well within the threshold limit of $+/-0.5°$.

This validation experiment was tested for the scenario where only the left side mass $m3$ had a non-zero value. The initial value for θ was still set to 20°, and the initial value for the left side angular velocity $q3_l$ was set to 5 degrees per second, while the rest of the initial conditions were set to zero. The results are shown in the plots below.

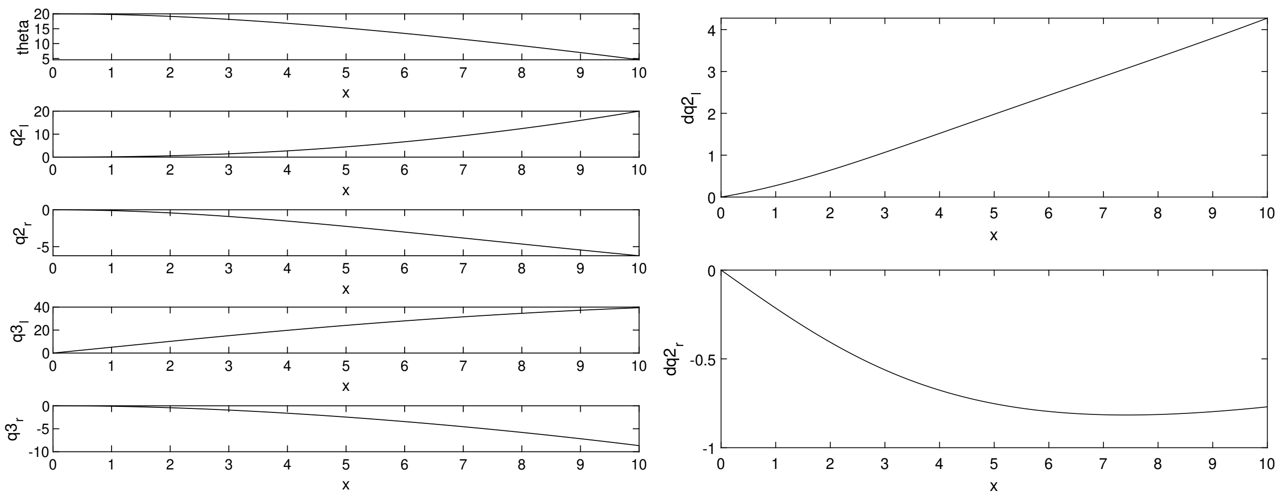


Figure 4.4: Simulation of the situation where only the left side mass $m3_l$ has a non-zero value. The attitude angle θ is given a start value, and the angular velocity $dq3_l$ has a start value.

Figure 4.4 shows that the left side mass $m3$ in combination with the angular velocity of the mass, $dq3_l$, affected the attitude angle. This fits well with the theory. Setting the angular velocity $dq2_l$ to some initial condition instead had about the same effect as $dq3_l$. This makes sense as an angular velocity $dq2_l$ would also mean an angular velocity of the mass $m3$. Setting the other angular velocities, $dq2_r$ and $dq3_r$, to some initial velocity instead did not affect the change of θ. Again, this follows the theory well.

The next step was to repeat this process for the right side masses $m2$ and $m3$ and vary the initial conditions for all the angular velocities. The results from this process showed the same as this one that only the actuators' angular velocity that affected the mass's position could lead to a change in the attitude angle θ. Additionally, the results confirmed that the right side masses with an angular velocity generally lead to a positive rotation of the attitude angle θ, except when larger than about 70°. It would instead lead to a negative rotation of θ.

The second validation experiment involves the lengths of the links in the robot legs. According to Equation 4.3, the tension force F from the mass following the link is proportional to the link length. Setting this length very close to zero should have minimal effect on the attitude angle θ, even when the angular velocity is very large.

In the first test, all the leg links, l_1 and l_2 are set to $0.0001m$, except the left side upper leg link l_1, which has the original value, $0.15m$. The masses in this validation experiment have their original values as well. The left side angular velocity $dq2_l$ has an initial value of 5 degrees per second. The simulation results are shown in the plot below.

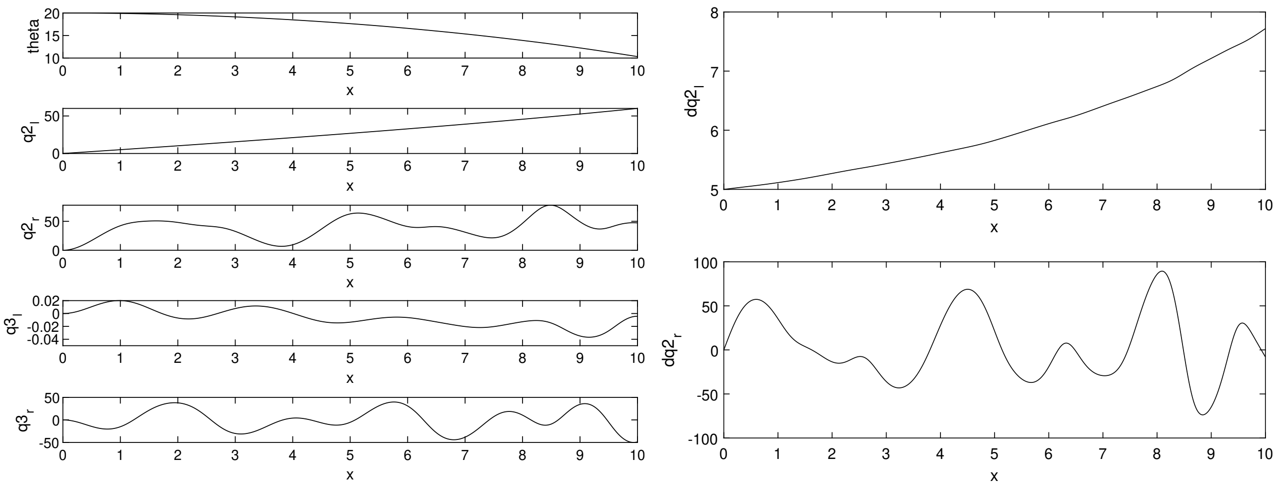


Figure 4.5: Simulation of the situation where only the left side upper link l_1 has a non-zero value. The attitude angle θ is given a start value of 20°, and the angular velocity $dq2_l$ has a start value of $5\frac{^\circ}{s}$

Figure 4.5 above shows how the value of θ decreases when the left side angle $q2_l$ has an angular velocity. If the angular velocity $dq2_l$ is set to zero, and $dq2_r$ instead set to the same initial condition, $5\frac{^\circ}{s}$, the simulation results will be as below:

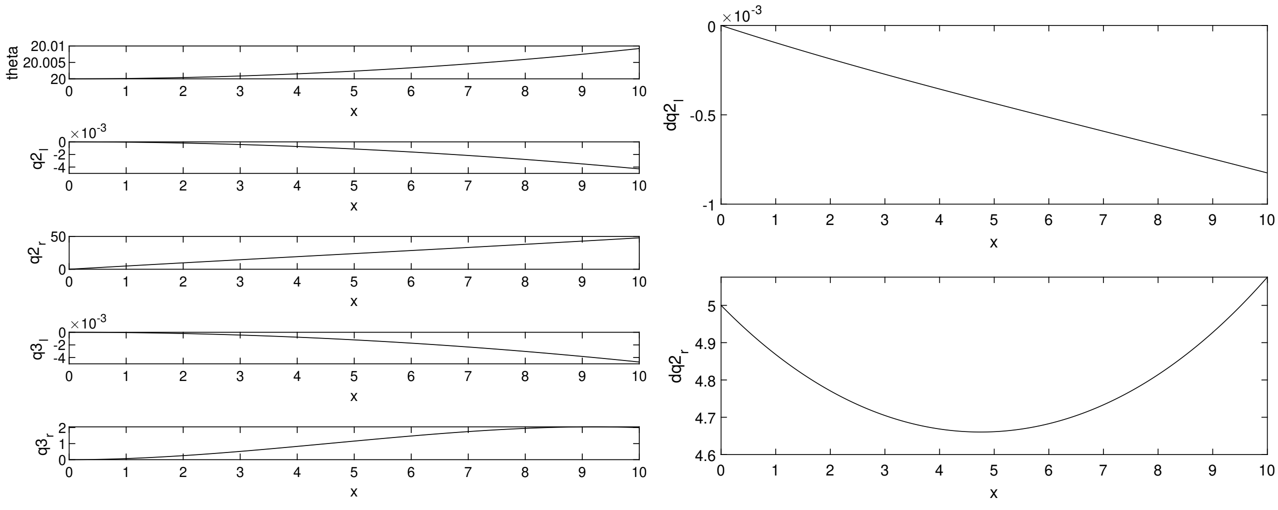


Figure 4.6: Simulation of the situation where only the left side upper link l_1 has a non-zero value. The attitude angle θ is given a start value of 20°, and the angular velocity $dq2_r$ has a start value of $5\frac{^\circ}{s}$

The plots in Figure 4.6 above shows how angular velocity of the right side angle $q2_r$ has no effect on the attitude angle θ. This fits well with the theory because the right side leg links l_1 and l_2 are close to zero. There will then be no tension force affecting the robot's body.

Testing with initial values for the angular velocities for $q3_l$ and $q3_r$ gives results as expected. There will be no effect on the attitude angle θ unless the leg link rotating has a non-zero length.

Setting the length of the left side bottom leg link l_2 to the original value and setting the rest close to zero had the same effect as for the masses. The angular velocities on the left side, both by $dq2_l$ and $dq3_l$, affected the attitude angle θ, while the right side angular velocities had no effect. This is because the left side angular velocities in this scenario are combined with a length l_2, even though the leg link l_1 might be zero.

The same validation experiment is performed for the right side leg links, and the results are as expected. The link l_1 in combination with an angular velocity for the angle $q2_r$ had an effect on θ, and the link l_2 in combination with an angular velocity for either $q2_r$ or $q3_r$ had an effect on θ. The left side angular velocities had no effect. All results from this validation experiment are well within the threshold limit of 0.5° during ten seconds, and this part of the model validation is verified.

The last validation experiment aims to see how the model behaves when there is no other initial motion than the inevitable falling motion caused by gravity. If there are no initial rotations in the system, it should stay like this. All masses $m2$ and $m3$, and links l_1 and l_2 have their original values again. The attitude angle θ has an initial rotation of 20°, and the rotations $q2_l$, $q2_r$, $q3_l$ and $q3_r$ have the initial values $[-20°, 20°, -10°, 10°]$.

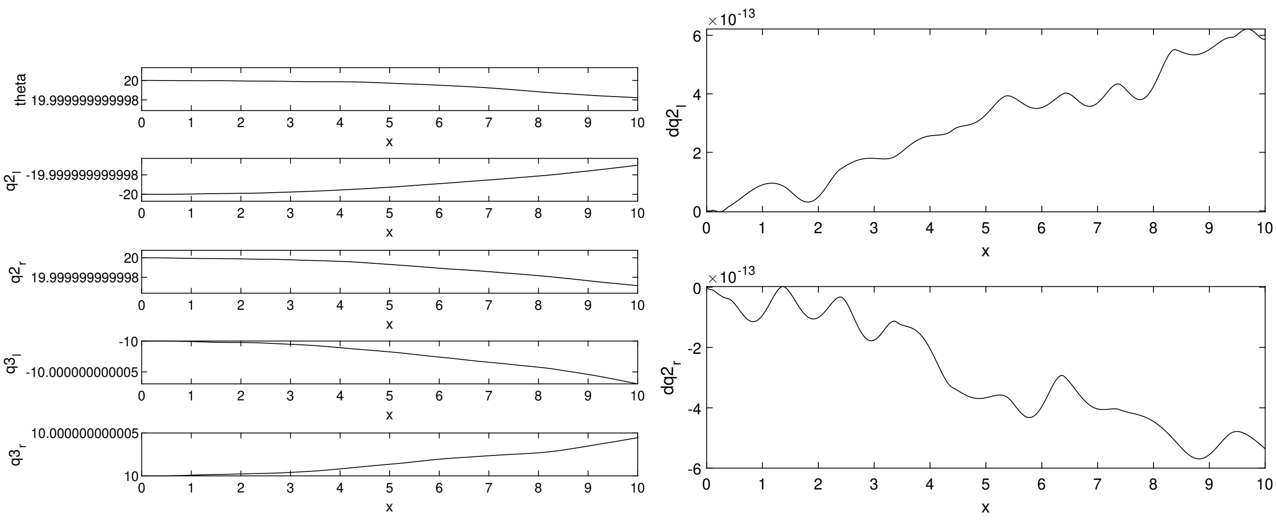


Figure 4.7: Simulation of the situation where all initial angular velocities are set to zero. The actuator angles have initial values, and θ has an initial value of 20°

The plots in Figure 4.7 illustrates how all the actuator angles and attitude angle remains constant at their initial values. This confirms the theory that the system remains constant unless when introduced to a motion.

The validation experiments performed on the longitudinal models showed that the models under simulation correlate well with the theory and expectations. All resulting values were kept well below the threshold limits, and the accuracy of the longitudinal models have a satisfactory accuracy.

4.4 Lateral

The validation experiments performed for the longitudinal dynamics need to be performed again for the lateral dynamics in the $y-z$ 2D coordinate frame.

The first experiment tests how well the rotation of leg masses $m2$ and $m3$ correlate to the theory. This is tested in the same way as for the lateral dynamics, by setting each mass close to zero, except for one, and varying the different angular velocities to see how it affects the change of attitude angle ϕ. The results for when only the left side mass $m2$ has a value and the left side angular velocity $dq1_l$ has an initial value are shown in the plots below.

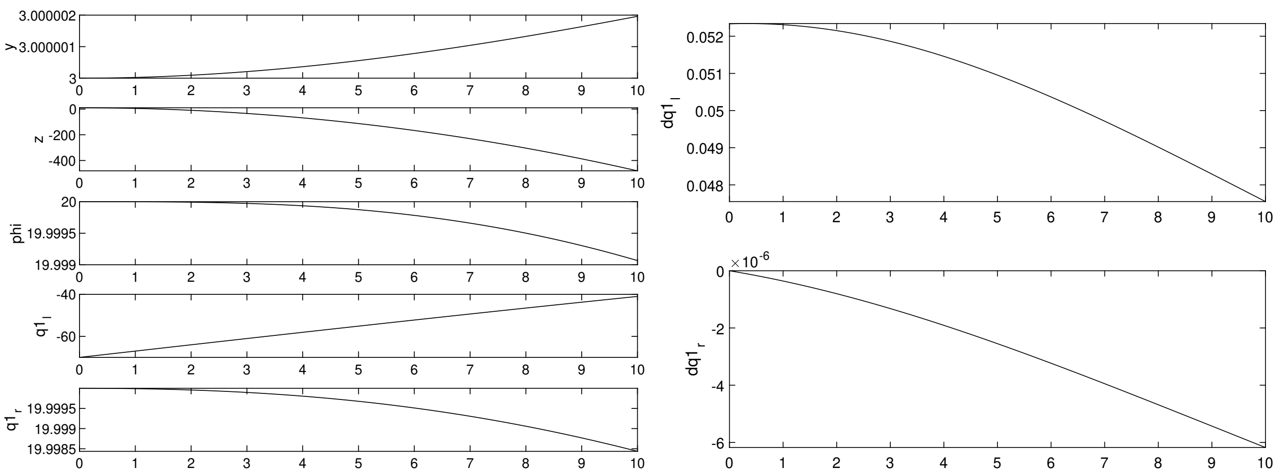


Figure 4.8: Simulation of the situation where only the left side mass $m2_l$ has a non-zero value. The attitude angle ϕ is given a start value of 20°, and the angular velocity $dq1_l$ has a start value of 3. The right side hip actuator $q1_r$ has initial value of 70°.

The plots in Figure 4.8 show that the angular velocity of the left side mass $m2$ very much affects the angle of ϕ. This answers well to the theory behind Equation 4.3. Instead, setting the initial condition of the right side angular velocity $dq1_l$ to a value did not affect the motion of the robot's main body. This follows that the right side masses $m2$ and $m3$ have values close to zero. This validation experiment is repeated for all the masses. The results showed that the system behaves similarly to the longitudinal dynamical system. Only the angular velocity of masses with values not equal to zero affected the rotation of the attitude angle ϕ. The results were well within the threshold limits of $0,5$ degrees per second, and the model following these experiments is deemed accurate enough.

The second validation experiment targets the link lengths l_1 and l_2. The procedure is to let all the link lengths have a value close to zero except for one and then vary the initial angular velocities. The plots below show the scenario simulation where only the right side link length l_1 has a length, while the others are close to zero. The left side angular velocity $dq1_l$ is then given an initial value of 4 degrees per second.

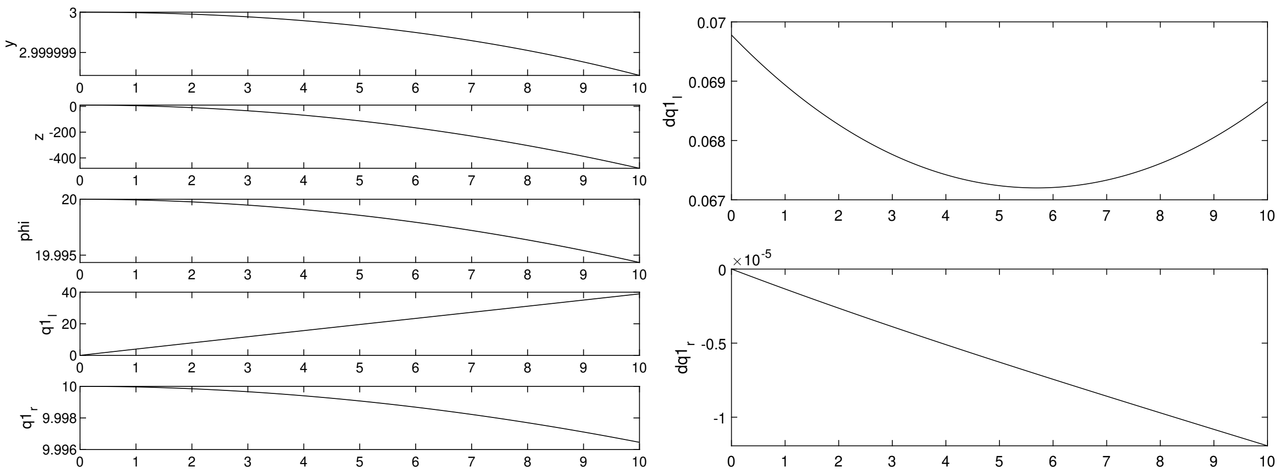


Figure 4.9: Simulation of the situation where only the right side link l_1 has a non-zero length value. The attitude angle ϕ is given a start value of 20°, the angular velocity $dq1_l$ has a start value of 3, while the right side actuator $q1_r$ has an initial value of 10.

The plots in Figure 4.9 above show how there is minimal effect on the attitude angle ϕ when only the angular velocity $dq1_l$ has a value. The small changes in the left side angles are probably due to numerical calculation errors following the computations. Either way, the change is well within the threshold limit of $+/-0.5°$ during ten seconds. The validation experiment is repeated with an initial value for the right side angular velocity $dq1_r$ instead, which changed the attitude angle ϕ, as expected. This procedure was then repeated for all the left- and right-side link lengths l_1 and l_2, and the results showed that only the angular velocity of masses related to a link with a value well over zero had any effect on the motion of the robot and the attitude angle ϕ.

The last validation experiment aims to confirm that when there are no initial velocities in the system, it will continue to stay at rest, except for in the z-direction as it is in free fall. The plots of this simulation are shown below:

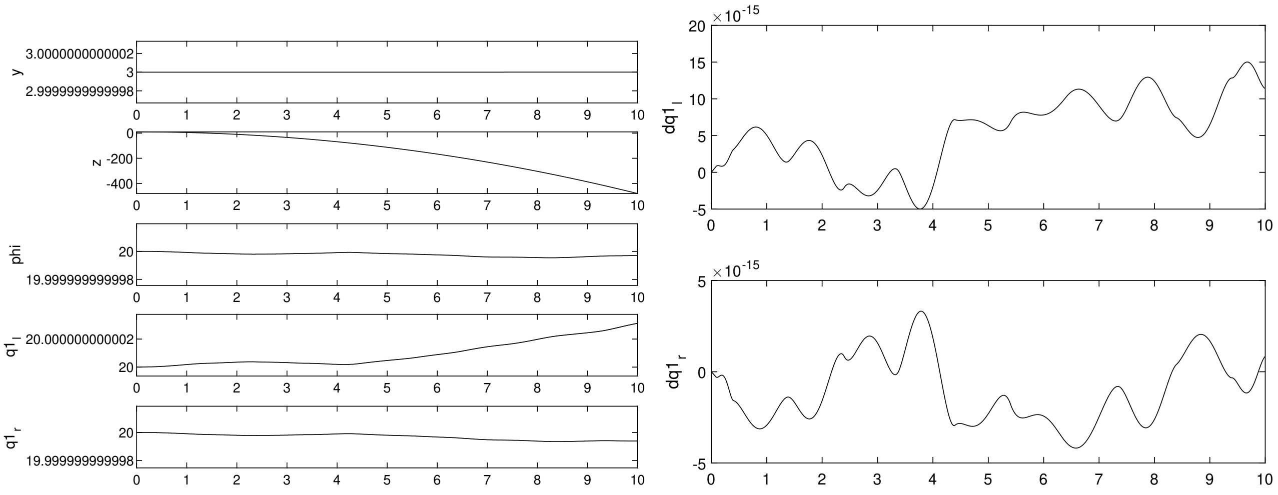


Figure 4.10: Simulation of the situation where all initial angular velocities are set to zero. The actuator angles have initial values of 20°, and ϕ has an initial value of 20°.

The plots in Figure 4.10 confirm that when there are no initial angular velocities, the system remains at rest, and none of the angles of the system change. This correlates with the theory.

The validation experiments performed on the lateral dynamical model confirmed that the model is well within all the threshold limits, and the model is deemed accurate enough to continue.

Chapter 5

PID Control

The mathematical models describing the system's longitudinal and lateral coordinates were evaluated and validated through experiments. The results from the experiments implied that the models followed the physical system very well and that they were accurate enough to continue using for the following processes. Since the mathematical models are in place, the next phase is to evaluate what controller should be used to control the attitude angles θ and ϕ. There are many options for controllers, but some are more suitable than others, depending on the system. It is becoming more and more popular in modern applications to develop controllers based on machine learning. However, the machine learning methods rely on large amounts of data. Additionally, they are based on not needing to know or understand why the controller works or not. It takes away the theoretical guarantees so well developed for classical control theory. In classical control theory, there are methods to determine the stability of closed-loop systems or regions of stability. This is why the methods of machine learning will not be considered for solving the control problem of this problem. Instead, both the PID and MPC controllers will be evaluated and tested.

The PID controller is widely used for process control, mainly because it is simple to implement and understand. If the system works with a PID control, there is often no good reason to use another control scheme. However, it will likely not produce good results for all systems because it is simple and trivial. The more complex the system becomes, the harder it gets to implement and tune a PID controller that will give good results. The tuning part of the PID is another topic entirely, as there are multiple schemes and techniques to follow in order to tune it correctly. Still, many ends up having to resort to the trial and error scheme, for the most part, either way. Sometimes, the PID controller is difficult to use on some systems because there is no golden rule as to how it should be tuned. The description given for the different controller gains in the theory section is a good starting point, but there is no guarantee that this will provide decent results.

The longitudinal and lateral dynamics have already been simulated in Matlab using the ODE solver ode45. The PID controller algorithm can be added to this already developed Matlab program. The ode45 solver takes an ode function as input. This ode function is where the equations to be integrated are described, and the user makes it. The ode function takes the previous state values, q, as input. It performs the necessary computations based on the equations of motion found in Section 3 using these previous state values. The input vector u that contains the actuator torques τ_i are also set in this ode function. The PID controller implementation is also done in this ode function, either by directly performing the necessary calculations in this function file or calling another function that performs the PID calculations. In order to keep the Matlab program files tidy, it seemed the most reasonable to use a new function file for the PID calculations.

5.1 Longitudinal

The state vector for the longitudinal dynamics is given below:

$$\begin{bmatrix} x \\ z \\ \theta \\ q2_l \\ q2_r \\ q3_l \\ q3_r \end{bmatrix} \tag{5.1}$$

The main object of the control law is to control the attitude angle θ. There are several other objects of using a PID controller, such as controlling the speed of the actuators or the position of the actuator's angles. However, as the control of θ is the most important object, it seems more reasonable to not focus on other control objectives unless it is clear that it can solve the central object without other constraints or interests. This is why other objective goals are disregarded for the moment, and the physical constraints are ignored. When using a PID controller to perform this, a reference θ_r is needed for θ to describe what angle θ should approach. The general algorithm of the PID controller is given below:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{det}\right) \tag{5.2}$$

where $e(t)$ gives the error or the difference between the present value and the reference value. The gain K, T_i, and T_d are the design parameters that need to be tuned, and the output of the function, $u(t)$, gives the control inputs for the system.

In the longitudinal dynamical system, the vector $\boldsymbol{u}$ will consist of the four different actuator inputs $u2_l$, $u2_r$, $u3_l$ and $u3_r$, for each of the actuators present in this system. Since the errors in the positions are disregarded in this first step, it makes no sense to set the error $e(t)$ to the difference between the actuator angle states and their reference. The only error e of interest is the attitude angle θ. In order to determine if it seems possible to control the attitude with a PID control, the $e(t)$ is chosen to be the attitude θ angle and is hence equal for all the four actuators at all times. However, the different controller gains can change and be different and will act as the method for tuning the controller. To sum this up, the PID controller split into the four actuator PID controllers can be written as below:

$$\begin{aligned}
u2_l(t) &= K_{2l}\left(e(t) + \frac{1}{T_{i_{2l}}}\int_0^T e(\tau)d\tau + T_{d_{2l}}\frac{de(t)}{dt}\right)\\
u2_r(t) &= K_{2r}\left(e(t) + \frac{1}{T_{i_{2r}}}\int_0^T e(\tau)d\tau + T_{d_{2r}}\frac{de(t)}{dt}\right)\\
u3_l(t) &= K_{3l}\left(e(t) + \frac{1}{T_{i_{3l}}}\int_0^T e(\tau)d\tau + T_{d_{3l}}\frac{de(t)}{dt}\right)\\
u3_r(t) &= K_{3r}\left(e(t) + \frac{1}{T_{i_{3r}}}\int_0^T e(\tau)d\tau + T_{d_{3r}}\frac{de(t)}{dt}\right)
\end{aligned} \tag{5.3}$$

These PID controllers for the actuators in longitudinal dynamics are implemented in a separate Matlab function file, which takes in the reference signal θ_r and the current value of θ. The function file returns these control input values, which are used in the simulation file that uses the ode45 solver to simulate the states of the system over time.

The tuning of the gains is the most challenging part, and it seemed wise to follow the principle of setting the derivative gain T_{d_i} to zero and varying the two other gains until the system response meets the reference signal. It is then most subject to oscillations, and the derivative gain is then varied and increased to lower the oscillations. Apart from this method, the trial and error method needs to be performed to find the best response possible.

5.2 Lateral

The state vector for the lateral dynamics is given below:

$$\begin{bmatrix} y \\ z \\ \phi \\ q1_l \\ q1_r \end{bmatrix} \tag{5.4}$$

The attitude angle for these dynamics is the angle ϕ. The principle of the PID controller is as before, but with a new main objective. The goal is to control the attitude rotation ϕ. The control input vector $\boldsymbol{u}$ contains the two different control inputs $u1_l$ and $u1_r$, which are are torque inputs for the actuator $q1_l$ and $q1_r$. The method for controlling this angle is the same as for the longitudinal dynamics, and the PID control laws for the two actuators can be written as below:

$$\begin{aligned} u1_l(t) &= K_{1l}\left(e(t) + \frac{1}{T_{i_{1l}}}\int_0^T e(\tau)d\tau + T_{d_{1l}}\frac{de(t)}{dt}\right) \\ u1_r(t) &= K_{1r}\left(e(t) + \frac{1}{T_{i_{1r}}}\int_0^T e(\tau)d\tau + T_{d_{1r}}\frac{de(t)}{dt}\right) \end{aligned} \tag{5.5}$$

These control laws are implemented in a separate Matlab function file, which takes the reference angle ϕ_r and the current value for ϕ. The function file returns the control inputs used in the simulation file, where the ode45 solver is used to simulate the states of the system over time.

The tuning of the gains follows the same principle of varying the integral and constant gains until the system response hits the reference signal and then tuning the derivative gain to minimize the oscillations of the response. The results are shown in the result section.

Chapter 6

Model Predictive Control - MPC

The final objective of making a control law, any control law, is to control a set of states. In the case of the quadruped robot in free flight, the states to be controlled are mainly the attitude angles θ, ϕ, and ψ. Additionally, the derivatives should go towards zero such that the robot is stabilized. In this project, ψ is omitted to focus entirely on the two others. The final control algorithm should be able to control both of these attitudes. If this algorithm is only one single algorithm that handles both attitudes or if it is split up into separate algorithms is something that needs to be decided, and there are pros and cons to both choices.

When considering the process of building a reliable controller, the simpler is often better. The PID controllers from the previous section already had 3 tuning variables for each actuator. The MPC - Model Predictive Controller, will have more, such as sample time, prediction and control horizons, and weights for the input and output variables. Splitting the two attitude angle control laws into two separate controllers minimizes complexity. In addition, the mathematical models are already made for the two separate coordinate frames. The controllers can be tuned separately, with a much smaller state vector for each, compared to the state vector for the combined controller. This is a considerable advantage when tuning and finding the correct configurations. The downside is that the split control algorithm does not include the whole system for each control algorithm. A solution that works for one isolated part of the system might experience difficulties when employed on the whole system. This is simply because the isolated controller does not account for all the states and characteristics of the system.

Still, the benefits of splitting the controllers into two separate MPC controllers appear to be a better choice for this system. The quadruped robot has the characteristic that the motions of the legs can be split into two separate 2D frames, longitudinal and lateral, where the dynamics of the two frames are very minimally dependent on the states of the other frame, when the actuator angles are relatively small. This is why the solution for this project will be based on having two separate MPC controllers for the attitude control of θ and ϕ.

The next problem is deciding whether the MPC controllers will be nonlinear MPC controllers, NMPC, or if the mathematical models should be linearized around some equilibrium to use a regular MPC controller. There are benefits, and disadvantages to both methods [18]. The regular MPC optimization problem is generally a linear or quadratic program. These are convex problems, for which there will exist a global minimum that most computers will be able to solve unless the models are unreasonably large and complex. This is not the case for nonlinear MPC optimization problems, for which it is rarely possible to guarantee a global solution. The optimization problem may have several local minimum points, and the computational cost will increase with these points. The development of the nonlinear models and the nonlinear state estimators can also be challenging. However, for many systems, successful implementation of a nonlinear MPC control scheme has performed better and more efficiently than a linear MPC control scheme [18]. M. Kamel, M. Burri, and R. Siegwart write in [19] that for their trajectory tracking of Micro Air Vehicles, the NMPC showed better disturbance rejection, step response tracking, tracking performance, and computational effort than the MPC. The NMPC involves a much more accurate system representation than the MPC. This is why it has the potential to outperform the MPC. However, the disadvantages of potentially not finding good local minimum points in the optimization problem or sufficiently good nonlinear models and state estimators are why the choice fell on a regular MPC controller scheme.

The MPC uses a set of linear models, and since the Equations of Motion are nonlinear, they need to be linearized. The linear models of the MPC controllers will be a state-space system of the form $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$. As of now, the equations of motion are on the form

$$\ddot{\boldsymbol{q}} = M(\boldsymbol{q})^{-1}\Big(B\boldsymbol{u} - C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \tau_g(\boldsymbol{q})\Big) \tag{6.1}$$

The overall method of linearizing the equations of motion and translating the equations to a state-space system of the form above has two general steps.

The first step is to linearize the equations of motion using a linearization method. This can be done in several ways, among which a popular method is a jacobian linearization. This linearization is based on the Taylor expansion series but neglects all higher than 1st order terms. This leaves only the terms written below:

$$\dot{\delta}_x = f(x,u)|_{\bar{x},\bar{u}}\delta_x + \frac{df}{dx}|_{\bar{x},\bar{u}}\delta_x + \frac{df}{du}|_{\bar{x},\bar{u}}\delta_u \tag{6.2}$$

However, since it is about an equilibrium point, the term $f(x,u)|_{\bar{x},\bar{u}}\delta_x$ will be equal to zero. The expression is then left with one term for the partial derivative with respect to the states, and one with respect to the input $\boldsymbol{u}$. The term for the partial derivative with respect to $\boldsymbol{x}$,

$$\frac{df}{dx} \tag{6.3}$$

is the jacobian of the system, with respect to the states, and when replacing the states with the equilibrium states and the inputs u with the equilibrium inputs, the result will be the state space matrix A:

$$A = \frac{df}{dx}|_{\bar{x},\bar{u}}\delta_x \tag{6.4}$$

In the same way, the state space matrix B will be the jacobian of the system with respect to the input $\boldsymbol{u}$, when the states and inputs are replaced with their equilibrium value,

$$B = \frac{df}{du}|_{\bar{x},\bar{u}}\delta_u \tag{6.5}$$

When the equilibrium states are found, the method is to find the two jacobians of the system and then replace the states and inputs with their equilibrium values.

The second step is to expand the state-space system. The equations of motion are of the second derivatives of the states, $\ddot{\boldsymbol{q}}$, so that when performing the first step described above, the state space system will be in the form: $\ddot{\boldsymbol{q}} = A\dot{\boldsymbol{q}} + B\boldsymbol{u}$. However, the system will need to include both the derivative states $\dot{\boldsymbol{q}}$ as well as the states $\boldsymbol{q}$. The method of finding the state $\boldsymbol{q}$ from the equations of motion is as written below:

$$\begin{aligned} \dot{x_1} &= \dot{q_1} = x_2 \\ \dot{x_2} &= \ddot{q_1} \\ \dot{x_3} &= \dot{q_2} = x_4 \\ \dot{x_4} &= \ddot{q_2} \end{aligned} \tag{6.6}$$

The structure of the method described above results in a state vector that has twice the size of the original state vector, and it will return both the time derivative of the states, as well as the time derivative of the derivative states, as written below:

$$\begin{matrix} q_1 \\ \dot{q_1} \\ q_2 \\ \dot{q_2} \\ . \\ . \\ . \end{matrix} \tag{6.7}$$

To perform this, the A and B matrices found when linearizing will be expanded such that every other line of the matrices will be devoted to the states $\boldsymbol{q}$. The A matrix will have the size $2n \times 2n$, where n denotes the number of states, and the B-matrix will have the size $2n \times m$, where n is the number of states, while m gives the number of inputs.

The linearized models of the system are used in the prediction block of the MPC controllers. The actual mathematical models, or equations of motion, are used as the plant of the system. However, to verify how the linearization works, the MPC controller is first tested with the linearized models as both the prediction and plant models. If this does not produce good results, there is no point in continuing with these models, and the linearization should be revisited. The first MPC algorithm scheme will then have the form as shown below.

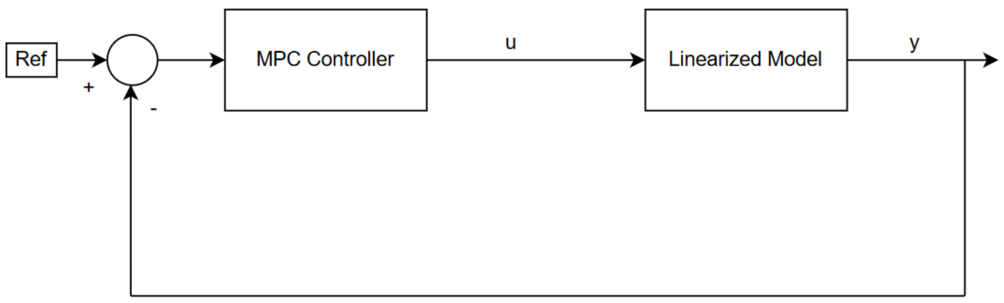


Figure 6.1: MPC Controller scheme with the linearized model as both prediction model and plant

If the results from the MPC controller scheme in Figure 6.1 are good, the next step would be to add the actual plant of the system to the controller scheme. For this project, as there are no real-life signals or sensors in use, the plant of the system will be the complete mathematical model, meaning the equations of motion derived in Chapter 3. The predictions in the MPC controller scheme will still use the linearized model for the predictions, but the predicted control inputs $\boldsymbol{u}$ will be the input of the plant, which outputs the system states. These plant states will act as the feedback of the controller scheme in order to correct the errors. The resulting MPC algorithm scheme will have the form as below:

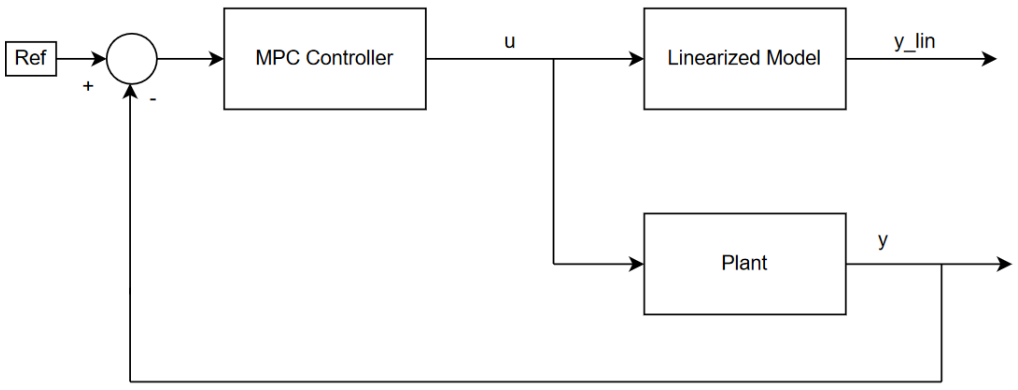


Figure 6.2: MPC Controller scheme, where the linearized model act as prediction model inside the MPC controller block. The plant equations are the equations of motion derived in Section 3

The actual MPC controller block in Figure 6.2 can be made with the MPC Toolbox extension in Matlab Simulink. This solution largely simplifies developing the MPC controller because the Toolbox performs all the necessary computations itself. What remains for the user is to define the linear models to be used and the tuning parameters. The models will be the linearized state-space system, while the tuning parameters must be appropriately determined.

The tuning parameters of the MPC controller include the sample time T_s, the prediction horizon n, the control horizon m, and the penalty matrices used in the cost function for the optimization problem. These penalty matrices consist of weights for all manipulated variables, control inputs u, and output signals. The output signals will be the states fed back to the MPC controller.

The common way of determining the parameters is to tune the MPC step-by-step and only change one parameter at a time [20]. There is also a consensus about the order in which the parameters are tuned. It is ordinary to set the sample time T_s first, followed by the prediction horizon n. This is because these parameters affect the step response time, and depending on the system, there might be a desired step response time for the closed-loop system. A rough guideline is to set the sample time T_s between 10% and 25% of the desired closed-loop response time [21]. However, as the sample time decreases, the computational complexity increases drastically, such that an optimal choice is a balance between the performance and computational effort. The prediction horizon n will be the number of future control intervals for the MPC to compute at the prediction step at each time interval. Naturally, a smaller sample time results in more rapid control interval computations, and a larger prediction horizon results in more computation per time step. A general tip if a response time T is desired is to choose the prediction horizon n such that $T = n \cdot T_s$.

The next step is usually to determine the control horizon m. Matlab's MPC Toolbox default is to set the control horizon to two. A general rule is to set m such that $m << n$, but most importantly, the control horizon should be smaller than the prediction horizon. This is to ensure that all predicted control inputs affect the output variables before the end of the prediction horizon. Another reason for keeping the control horizon short is to seek internal stability. The longer the control horizon, the more time it takes to solve the optimization problem, and the more information is discarded, the longer the control horizon is.

Following the determination of sample time, prediction, and control horizon, comes the decision-making process of determining the penalty matrices for the cost function for the optimization problem. The MPC solves an optimization problem at each control interval, where the solution determines the control input variables. The optimization problem in the Matlab MPC controller is a quadratic problem of the form:

$$J(z_k) = J_y(z_k) + J_u(z_k) + J_{\Delta u}(z_k) + J_\epsilon(z_k) \tag{6.8}$$

The cost function in Equation 6.8 above aims to minimize the cost J, where z_k is the control decision that will be decided in the Quadratic Program.

The term J_y includes the output signal weights or the weights of the system states. The controller aims to keep selected output signals at their reference tracks, and the weights determine how important a state's reference tracking is. A higher weight implies it is more important that the specific state follows the reference track, while a lower weight implies lower importance. This can be used to prioritize the different states according to how important it is that they follow their reference tracks. Zero ensures no control action is taken to follow the track, while one implies average priority. Five or above implies above average priority.

The term J_u involves the weights of the manipulated variables or control inputs $\boldsymbol{u}$. It may sometimes be necessary that the control inputs follow specific references, and these weights are used to prioritize this, as with the weights of J_y. However, in this situation, there is initially no need for the control inputs to follow specific references, such that all the weights in J_u are set to zero.

The term $J_{\Delta u}$ is a matrix created to penalize manipulated variable moves. These are in Matlab MPC referred to as the manipulated variable rate weights, and the higher the weight, the higher the penalty for large control input moves. The default is set at 0.1, but it might be beneficial for some systems to have smaller control input moves, such that the rate weights should be increased. Smaller control input moves usually provide more robust controllers.

The last term, $J_\epsilon(z_k)$ includes the constraint violation weights. If the MPC controller includes constraints for the states or control inputs, these weights determine how much slack and penalty should be given for each specific constraint. However, no constraints will be set in the lateral or longitudinal MPC. It would be more beneficial for the MPC controllers to work well without setting any constraints. One of the reasons for this is that the need for setting constraints often is a sign that the parameters are not properly tuned. For example, it might be too aggressive, and the weight choices should be revisited. The second reason is the quadratic optimization problem because it grows in complexity with the number of constraints. The constraints limit the space for possible control actions, and the optimal solution might violate a constraint. A solution could be to opt for soft constraints, which can be violated when necessary, although, the optimal method should not need constraints, which is why they are not set during this project.

The last tuning parameters available in Matlab's MPC control designer are two sliding bars. These sliding bars are directly connected to the performance of the MPC and simplify some of the tunings for the user.

The first slider is the Closed-Loop Performance slider, which, when moved further to the left, ensures a more robust performance, and when moved to the right, ensures a more aggressive performance. When moving the slider towards the left, the manipulated variable rate weights are increased, and the output signal weights are decreased.

The second slider is the State Estimation slider, which, when moved further to the left, leads to slower state estimations, and when moved to the right, leads to faster state estimations. When moving the slider to the right, the gains for the disturbance models are increased, while the gains for noise models are decreased. This is a way of tuning the responses to be faster because the time spent on state estimation is shortened. When working with a linearized model for a nonlinear system, the developed noise models of the MPC might be quite extensive because the MPC controller interprets there to be much noise because the output signals of the plant are different from the predicted outputs. When decreasing the weights of the noise models, this is less emphasized, and the time spent on improving the state estimations is cut down.

6.1 Lateral

The linearization of the lateral dynamics will be performed as described above. The linearization needs to be done about an equilibrium point, $\bar{\boldsymbol{x}}$, for which the differential equations are equal to zero. This equilibrium point will have $\bar{\phi}$, $\bar{\dot{\phi}}$, $\bar{\dot{q1}}_l$ and $\bar{\dot{q1}}_r$ equal to zero, since the point of the controller scheme is to stabilize the robot about zero attitude angle ϕ. Naturally, the input torques $u1_l$ and $u1_r$ should be zero. The position states y and z are not decided by the actuator angles or derivatives and, in turn, do not affect the actuator angels, attitude angle, or angle derivatives. This is why the differential equations for y and z are excluded from the state space system, leaving a state vector of six states (three angular states and three angular velocity states) instead of ten states. A computer calculation computes the remaining angles $q1_l$ and $q1_r$ in Matlab to determine for what values the differential equations will be zero. The result of solving the system of differential equations for $q1_l$ and $q1_r$ in Matlab fives that both angles at the equilibrium point are zero. The equilibrium point is zero for all states in the state vector and all control inputs.

The A matrix formed using the features above will be a somewhat empty matrix. All elements from the linearization will be zero, while the elements for the system states will form an identity matrix. This will not work very well because the double derivatives of the states with this state-space system will be independent of all other states. The linearization needs to be changed. The first step is to perform a small angle approximation of some of the angles. This is described in the Article from [22], and the main concept is that when some angle θ is small, the sine of the angle can be approximated to the angle, while the cosine of the angle can be approximated to one. This approximation is summed up in the equation below:

$$\begin{aligned} \sin\theta &\approx \theta \\ \cos\theta &\approx 1 \end{aligned} \tag{6.9}$$

The assumption for the system is that the attitude angle ϕ is small enough to use small angle approximation as described above. Additionally, the angular expression below are approximated as described below:

$$\begin{aligned} \sin(\phi - q1_r) &\approx \phi - q1_r \\ \sin(\phi + q1_l) &\approx \phi + q1_l \\ \cos(\phi - q1_r) &\approx 1 \\ \cos(\phi + q1_l) &\approx 1 \\ \cos(q1_l) &\approx 1 \\ \cos(q1_r) &\approx 1 \\ \sin(q1_l) &\approx q1_l \\ \sin(q1_r) &\approx q1_r \end{aligned} \tag{6.10}$$

The next step is to evaluate the equilibrium point. The differential equations and states are very much dependent on the actuator angular velocities, and setting them to zero will eliminate many terms. The actuator angular velocities $dq1_l$ and $dq1_r$ are set to have the equilibrium value -0.001 instead of zero to let the terms dependent on these angular velocities not be zero. The final linearization with these updates will have the matrices A and B as written below.

$$A = \begin{bmatrix} 0.0276 & 0 & 0.0138 & -0.0001 & -0.0138 & 0.0001 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ -0.0165 & 0 & 0.0082 & 0.0001 & 0.0082 & -0.0001 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0.0165 & 0 & 0.0082 & -0.0001 & -0.0082 & 0.0001 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{6.11}$$

$$B = \begin{bmatrix} -14.0910 & 14.0910 \\ 0 & 0 \\ 31.6544 & -10.4904 \\ 0 & 0 \\ -10.4904 & 31.6544 \\ 0 & 0 \end{bmatrix} \tag{6.12}$$

The matrices C and D are related to the system's output, as written below

$$y = Cx + Du \tag{6.13}$$

where $\boldsymbol{y}$ denotes the output vector, $\boldsymbol{x}$ is the state vector, and $\boldsymbol{u}$ is the control input vector. Since both the angle states $q1_l$, $q1_r$ and ϕ, and the angular velocities $dq1_l$, $dq1_r$ and $d\phi$ are wanted, the C matrix will be the 6x6 identity matrix. There are no control inputs that affect the output, meaning the matrix D will have all elements equal to zero. The matrix D will have the size 6x2 since there are six states and two control inputs. These matrices are summed up below:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.14}$$

The tuning of the MPC controller begins with deciding how long the response needs to be. Since the robotic system depends on controlling the actuator angles mid-air, the response time has to be much faster than for most process control systems. Given that the control will happen during the time span of a few seconds, the response time should be well under 0.5 seconds to give the controller time to both reach the reference value and stabilize. As a minimal response time, 0.1 seconds is chosen as the value for determining sample time T_s and prediction horizon n.

If the response time T is set at 0.1 seconds, the sample time T_s will be chosen from the $10-25\%$ interval of $[0.01, 0.025]$ seconds. The initial test for sample time should start at one of the ends and is therefore chosen to be 0.01 seconds. The prediction horizon n follows the general rule $T = n \cdot T_s$, such that the initial value of n will be 10.

The output state weights that determine the priority of the reference tracking of the output states should follow the controller's objective. It is essential that the attitude angle ϕ follows the reference value, while the actuator angles do not need to follow a reference. However, the actuator angle velocities, or derivatives of the actuator angles, should converge to zero when ϕ has reached its reference. As an initial tuning, the output state ϕ will have a weight of one, while the actuator angle velocities will have weights of 0.2. The remaining states will have zero for weight. These values will make up the penalize matrix Q, which penalized error between output signals and reference trajectory [23]. The Q matrix will, in this case, look like below, following the same state pattern as for the state space system, with the order of states as $[d\phi, \phi, dq1_l, q1_l, dq1_r, q1_r]^T$.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.15}$$

The Q matrix has to be positive semi-definite but should generally also be positive definite [4]. Because of the zeroes along the diagonal of the matrix, this Q-matrix is only positive and semi-definite. To change this, one can either omit the states that will not follow a reference, such as $d\phi$ and the actuator angles, or one could give them a minimal value in the Q-matrix. However, it will not make any difference in practice, even though leaving them in the Q-matrix generally is not a good practice.

The control input rate weights, or manipulated variable rate weights, form the diagonal of the penalize matrix R. This matrix penalizes the manipulated variable moves, depending on the sizes of the weights. With the weights of 0.1, the R-matrix will look like below:

$$R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{6.16}$$

This R-matrix needs to be positive definite, which means each rate weight has to have a value above zero.

The initial tuning parameters can be summarized in the table below:

T_s	n	m	w_ϕ	w_{dq1_l}	w_{dq1_r}	$w_{\Delta u1_l}$	$w_{\Delta u1_r}$
0.01	10	2	1	0.2	0.2	0.1	0.1

Table 6.1: First set of tuning parameters for lateral MPC.

It is eventually clear that the initial weights for state outputs are insufficient. The priority of ϕ needs to be higher in order for it to track the reference properly. Following the guide for prioritizing state output weighting, the weight of ϕ is set to five, which implies above average priority. At the same time, the weights of the angular actuator velocities are decreased, such that they are not a priority unless ϕ already tracks its reference. These weights are set to 0.02, which indicates low priority. The control inputs' rate weights seem reasonable, and there is no immediate need to alter these weights. A few tests with higher values and distinct values for the control input rate weights showed not much change for the ϕ-response with the changes. The new Q-matrix will look like below:

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.02 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.17}$$

and the seconds set of tuning parameters are given in Table 6.2 below.

T_s	n	m	w_ϕ	w_{dq1_l}	w_{dq1_r}	$w_{\Delta u1_l}$	$w_{\Delta u1_r}$
0.01	10	2	5	0.02	0.02	0.1	0.1

Table 6.2: Second set of tuning parameters for lateral MPC.

Finally, there might be a problem with the computational complexity of the MPC, mainly because the sampling time T_s is set relatively low. If the prediction step does not have enough time to perform all necessary computations before the next time step, the system behavior might differ from what would be expected. The State Estimation slider might come in handy because it can speed up the computational time and ensure the prediction step has sufficient time to perform all necessary operations. If this step is necessary or not will be visible through the tests and experiments.

6.2 Longitudinal

The longitudinal dynamics' linearization will follow the same procedure as for the lateral dynamics. The state-space system state vector will be the combination of all the angular states and the angular velocities, such that the state vector will include the ten states $[d\theta, \theta, dq2_l, q2_l, dq2_r, q2_r, dq3_l, q3_l, dq3_r, q3_r]$. The longitudinal linearization will also have the same issue regarding the equilibrium point when the point is zero for all the states, as the lateral dynamics had. This is why small-angle approximation is used for the attitude angle θ, as well as for the terms below:

$$\begin{aligned}
\sin(\theta - q2_r) &\approx \theta - q2_r \\
\sin(\theta + q2_l) &\approx \theta + q2_l \\
\cos(\theta - q2_r) &\approx 1 \\
\cos(\theta + q2_l) &\approx 1 \\
\cos(q2_l) &\approx 1 \\
\cos(q2_r) &\approx 1 \\
\sin(q2_l) &\approx q2_l \\
\sin(q2_r) &\approx q2_r \\
\cos(q3_l) &\approx 1 \\
\cos(q3_r) &\approx 1 \\
\sin(q3_l) &\approx q3_l \\
\sin(q3_r) &\approx q3_r
\end{aligned} \tag{6.18}$$

The equilibrium point is initially the point for which all the states are zero. However, this results in a very sparse A-matrix such that the equilibrium angular velocity states instead are changed to have the value -0.001. The final linearization for the longitudinal dynamics will have the matrices A and B as written below.

$$A = \begin{bmatrix}
0.0378 & 0 & 0.0189 & -0.0001 & -0.0189 & 0.0001 & 0.0089 & 0 & -0.0089 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.0126 & 0 & -0.0063 & 0.0001 & 0.0063 & 0 & -0.003 & 0.0004 & 0.003 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.0126 & 0 & 0.0063 & 0 & -0.0063 & 0.0001 & 0.003 & 0 & -0.003 & 0.0004 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
-0.0252 & 0 & -0.0126 & 0 & 0.0126 & -0.0001 & -0.0059 & -0.0008 & 0.0059 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0.0252 & 0 & 0.0126 & -0.0001 & -0.0126 & 0 & 0.0059 & 0 & -0.0059 & -0.0008 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix} \tag{6.19}$$

$$B = \begin{bmatrix} -8.2305 & 8.2305 & -16.4609 & 16.4609 \\ 0 & 0 & 0 & 0 \\ 97.1879 & -8.2990 & -177.8464 & 0.0686 \\ 0 & 0 & 0 & 0 \\ -8.2990 & -97.1879 & 0.0686 & -177.8464 \\ 0 & 0 & 0 & 0 \\ -177.8464 & 0.0686 & 483.1962 & -16.5295 \\ 0 & 0 & 0 & 0 \\ 0.0686 & -177.8464 & -16.5295 & 483.1962 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.20}$$

The C matrix will be the 10x10 identity matrix, while the D matrix will be the 10x4 zero-input matrix. These are given below.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.21}$$

The tuning process follows the theoretical background behind the MPC controller scheme and the tuning already performed for the lateral MPC. The sample time T_s should be small enough to follow the system dynamics, and with a value of 0.01 for the lateral case, it is natural to choose the longitudinal sample time equal to that. The prediction horizon n is initially set at 10, and the control horizon at the default value of 2.

The weights of the state outputs are set such that the attitude angle θ has priority in tracking the reference value. The actuator angles should eventually converge to some values and not oscillate, but not in a manner that hinders θ in tracking its reference. This is solved by setting the reference of the actuator angle velocities to zero and giving these output states a weight. As an initial tuning, the output state θ will have a weight of one, while the actuator angular velocities $dq2_l$, $dq2_r$, $dq3_l$ and $dq3_r$ are given a weight of 0.2. The remaining states will have a weight of zero, rendering the Q-matrix positive semi-definite but not positive definite. The Q-matrix will have the form as written below.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.22}$$

The initial rate weights for the control input moves will have the default values of 0.1. With these weights, the R-matrix will look like below.

$$R = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{6.23}$$

The initial tuning parameters can be summarized in the table below:

T_s	n	m	w_ϕ	w_{dq2_l}	w_{dq2_r}	w_{dq3_l}	w_{dq3_r}	$w_{\Delta u2_l}$	$w_{\Delta u2_r}$	$w_{\Delta u3_l}$	$w_{\Delta u3_r}$
0.01	10	2	1	0.2	0.2	0.2	0.2	0.1	0.1	0.1	0.1

Table 6.3: Initial set of tuning parameters for longitudinal MPC.

Later during the simulation and experiments, it becomes clear that the weights chosen are not sufficiently tuned. As for the lateral case, the weight of θ will be increased from 1 to 5, while the actuator angular velocity weights will be decreased to 0.02. This is done to ensure the controller properly tracks the θ-reference as the highest priority and that deviating from the angular velocity references is allowed to achieve that.

The rate weights of the control inputs were too low, and the closed-loop system was not robust enough. By increasing the weights of the control inputs, the penalty for large control moves is higher such that the responses will become smoother and less aggressive. The rate weights of the $u3$-control inputs were increased more than the $u2$-control inputs because it is a more wanted behavior that the hip actuators and hip motion will contribute more towards the attitude control than the knee-actuator movements. This is partly because the system is less complex if there are only two main contributors to attitude control. It is also because it is not wanted for the landing that the bottom leg parts have a large angle away from the top leg parts. The control inputs $u2_l$ and $u2_r$ will have a new rate weight of

0.127, while the control inputs $u3_l$ and $u3_r$ will have rate weights of 0.38. The updated Q- and R-matrices will then be as written below.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.02 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.02 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.24}$$

$$R = \begin{bmatrix} 0.127 & 0 & 0 & 0 \\ 0 & 0.127 & 0 & 0 \\ 0 & 0 & 0.38 & 0 \\ 0 & 0 & 0 & 0.38 \end{bmatrix} \tag{6.25}$$

Another change made for the tuning parameters regards the prediction horizon n. There appears to be no difference in simulation when decreasing the prediction horizon from 10 to 8. As a prediction horizon of 8 ensures less computational complexity and memory usage, the choice fell on lowering it to 8. Lowering n further had a negative effect on performance in terms of response time. Increasing it above 10 had the same negative performance effect.

The second set of tuning parameters can be summarized in the table below:

T_s	n	m	w_ϕ	w_{dq2_l}	w_{dq2_r}	w_{dq3_l}	w_{dq3_r}	$w_{\Delta u2_l}$	$w_{\Delta u2_r}$	$w_{\Delta u3_l}$	$w_{\Delta u3_r}$
0.01	8	2	5	0.02	0.02	0.02	0.02	0.127	0.127	0.38	0.38

Table 6.4: Second set of tuning parameters for longitudinal MPC.

Chapter 7

Results

7.1 PID Control

The results from the PID control scheme described in the PID chapter are shown in the plots below. The first set of plots shows the overall best results from the lateral PID control scheme over three seconds. The configuration of the PID gains and initial conditions of states are given in Table 7.1 below. All initial derivatives are equal to zero, and the states not shown in the table are also zero.

Lateral							
y	z	ϕ	$q1_l$	$q1_r$	K	T_i	T_d
3 m	10 m	20°	20°	20 °	0.570	1630	0.333

Table 7.1: Lateral initial state conditions and PID controller gains

Table 7.1 above give the PID gains K, T_i and T_d for the lateral PID controller. The gains are equal for both actuator controllers $q1_l$ and $q1_r$. The simulation was also tested with varying controller gains for the two actuators, but it did not improve the results. These controller gains are the gains that led to the so-far best results found. There are most likely gains that could improve the response further, but through extensive testing, no combination was found that led to satisfying results. It did not seem like there was any point in continuing with the PID controllers. The results from the gains in the table above are shown below in figure 7.1.

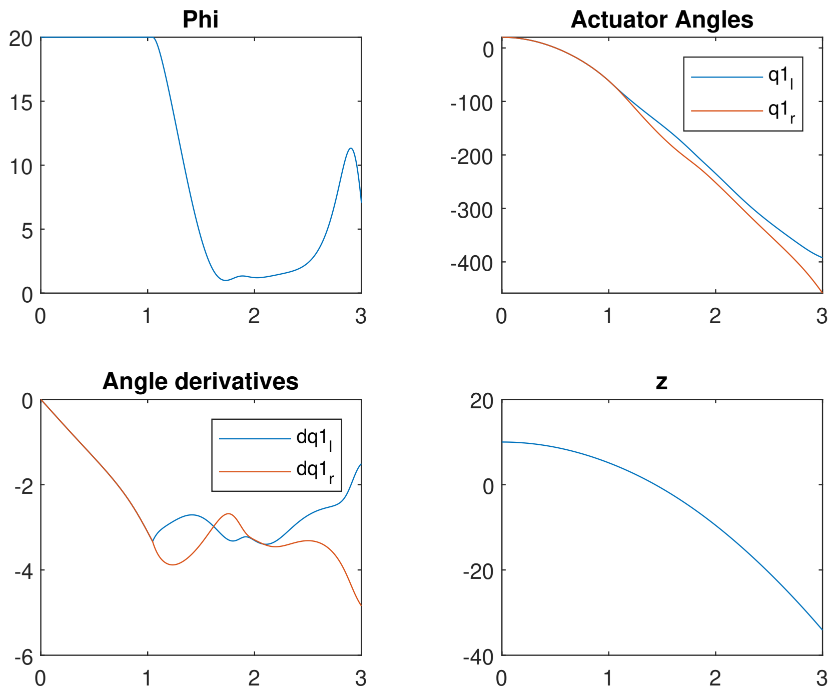


Figure 7.1: Simulation results from the Lateral PID controller scheme, with gains and initial conditions as described in Table 7.1

Figure 7.1 above show the system response from the PID controller with the described initial conditions and controller gains. The robot has an initial free flight height of 10 meters, as given by z, and an initial attitude angle ϕ of 20°. When observing the plots, the first thing to note is that the response is too slow. Almost a full second passes without any change in ϕ. In addition, when it does rotate and reach the reference angle 0, it does not stabilize on the value but switches direction again. The actuator angles $q1_l$ and $q1_r$ also have slow response in the beginning but quickly travel towards tremendous values. The time duration of three seconds is too short to observe what happens further with the actuator angles, but it seems evident that the system response is unstable. In order to achieve a quicker response time for the attitude angle ϕ, the controller gains need to be higher, which leads to even more unstable system response. To sum this up, either the system response will be much too slow or much too unstable.

The final controller gains and initial conditions for the longitudinal PID controller scheme are given in Table 7.2 below. All initial angular velocities are set to zero. Additionally, all controller gains are equal for the actuator PID controllers of $q2_l$ and $q2_r$, while the control inputs $u3_l$ and $u3_r$ are set to zero.

Longitudinal									
x	z	θ	$q2_l$	$q2_r$	$q3_l$	$q3_r$	K	T_i	T_d
3 m	10 m	20°	-20°	20°	0	0	0.006	0.15	10

Table 7.2: Longitudinal initial state conditions and PID controller gains

The results from the longitudinal PID controller scheme given the controller gains and initial conditions as given in the table above and the control scheme described above are given in the plots below.

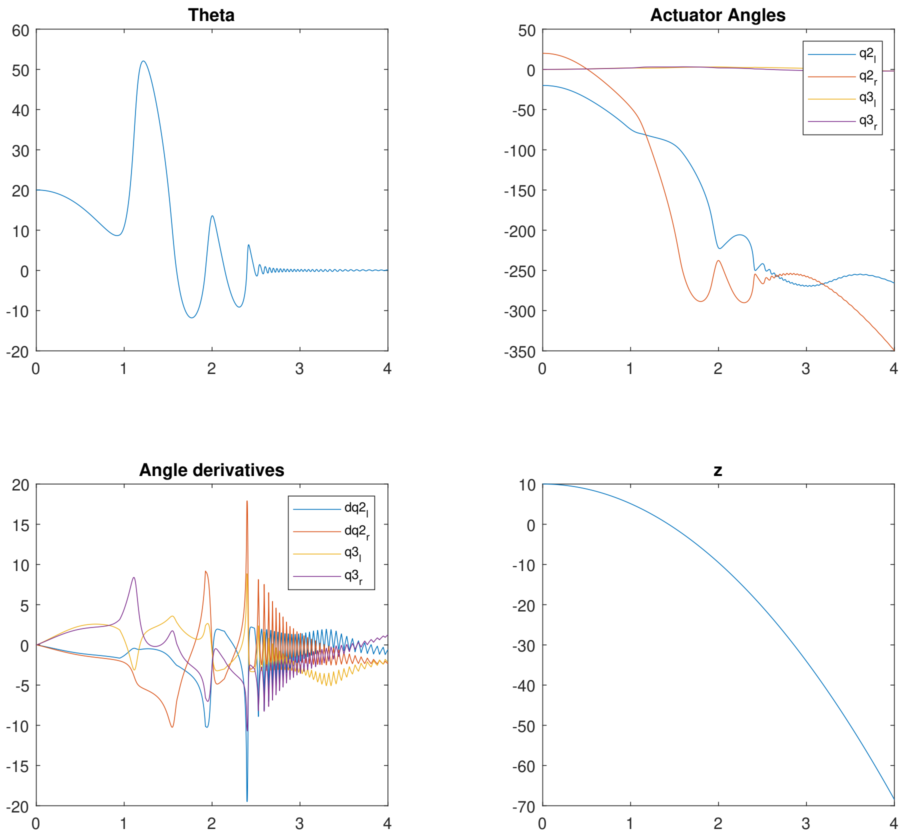
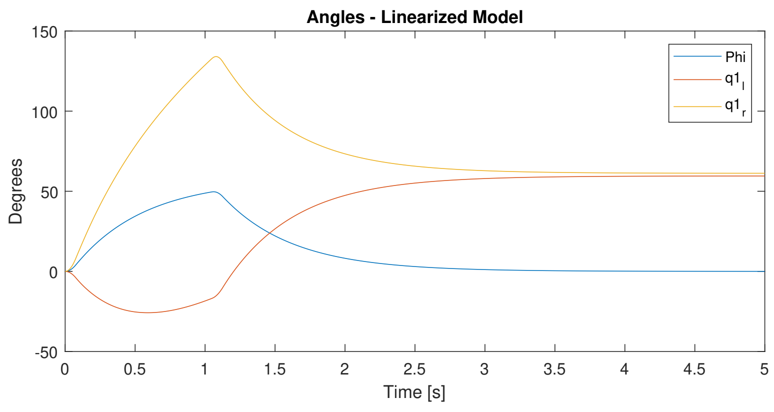


Figure 7.2: Simulation results from the Longitudinal PID controller scheme, with gains and initial conditions as described in Table 7.2

The plots in Figure 7.2 show the simulation of the attitude angle θ, actuator angles, the angle derivatives, and the height above ground, z, over a time period of 4 seconds. In the same way, as with the lateral PID controller, the response of this system is much too slow with this PID controller. The plot of z shows that the robot reaches the ground somewhere between one and two seconds, which means the response has to be stable at zero before this. The system does not achieve this but only reaches zero after about 2.2 seconds. In addition, it is unstable, which is seen especially in the plot of the angle derivatives. They oscillate continuously and fast, even after the attitude angle θ reaches zero. When the controller gains are configured for a faster response, the system becomes even more unstable, making it a poor choice. The actuator inputs $u3_l$ and $u3_r$ were set to zero, but adding a PID controller for these made the system even harder to control. This is why the choice to set them at zero was made to focus on the $u2_l$ and $u2_r$ controllers solely.

7.2 Model Predictive Control

7.3 lateral



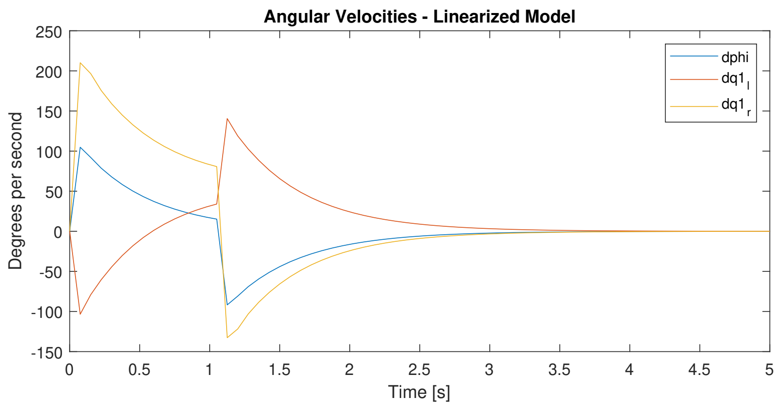


Figure 7.3: Simulation results from lateral MPC controller, when prediction model is the linearized lateral state space system. The reference signal is $[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]^T$ followed by - $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ after one second.

The plots in Figure 7.3 above show the simulation of the lateral linearized state-space system with the MPC controller scheme. Both the prediction plant and system plant is the linearized system. The MPC controller has a prediction horizon of 10 steps and a control horizon of 2 steps. The sample time is 0.075 seconds. The control input rate weights are 0.1. The output responses are weighted such that the attitude angle ϕ has a weight of one, while the angular velocities $dq1_l$ and $dq1_r$ have weights 0.2. There are no constraints on either control input or output signals. The plots show that ϕ reaches the reference value of zero with a rise time of 1 second. The angular velocities $dq1_l$ and $dq1_r$ and the angular velocity of ϕ all go towards zero. This attitude angle follows the reference well but is too slow to reach the reference value of one 57° before changing direction to reach the reference of zero. However, the MPC controller is too aggressive as the actuator angle $q1_r$ goes well above 90°, which is a hard limit for the actuator angles. It is also interesting to observe the actuator angular velocities, as they were meant to follow the references of one and then zero. However, they do not follow these references because they have lower weights than the attitude angle ϕ and behave such that the ϕ angle acts as wanted instead.

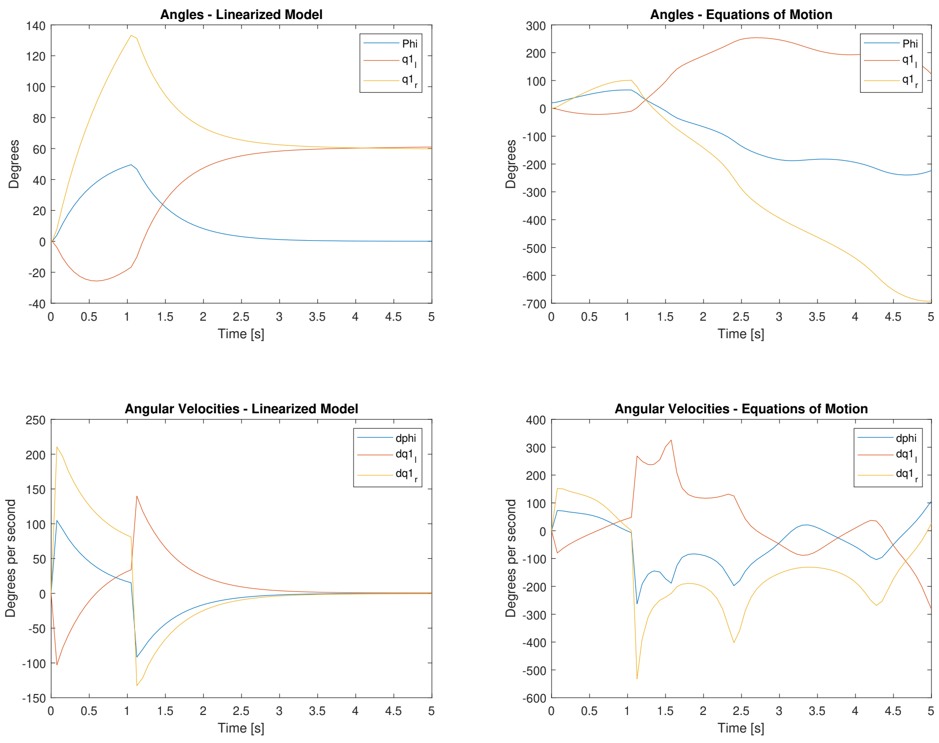


Figure 7.4: Simulation results from the lateral MPC controller, when the same control inputs u are fed into the Equations of Motion as well as the linearized model. The reference signal is $[1,1,1,1,1,1,1,1,1,1]^T$ followed by -$[0,0,0,0,0,0,0,0,0,0]^T$ after one second.

The plots above in Figure 7.4 show the same situation as the last plots, but when the control inputs $u1_l$ and $u1_r$ are fed in as inputs to the lateral Equations of Motion as well. The reference is still the same as before, one for all states until one second and zero afterwards. When comparing the two simulation results to the left and right, it is clear that the controller performs much better on the linearized model than on the actual Equations of Motion. The attitude angle ϕ at the right side plot somewhat follows the direction until about two seconds but continues way below zero. The actuator angles and angular velocities are drastically high and oscillatory. Overall, the MPC controller performs very poorly for the lateral system. The controller scheme still uses the linearized model as both prediction and model input, and the next step would be to change this.

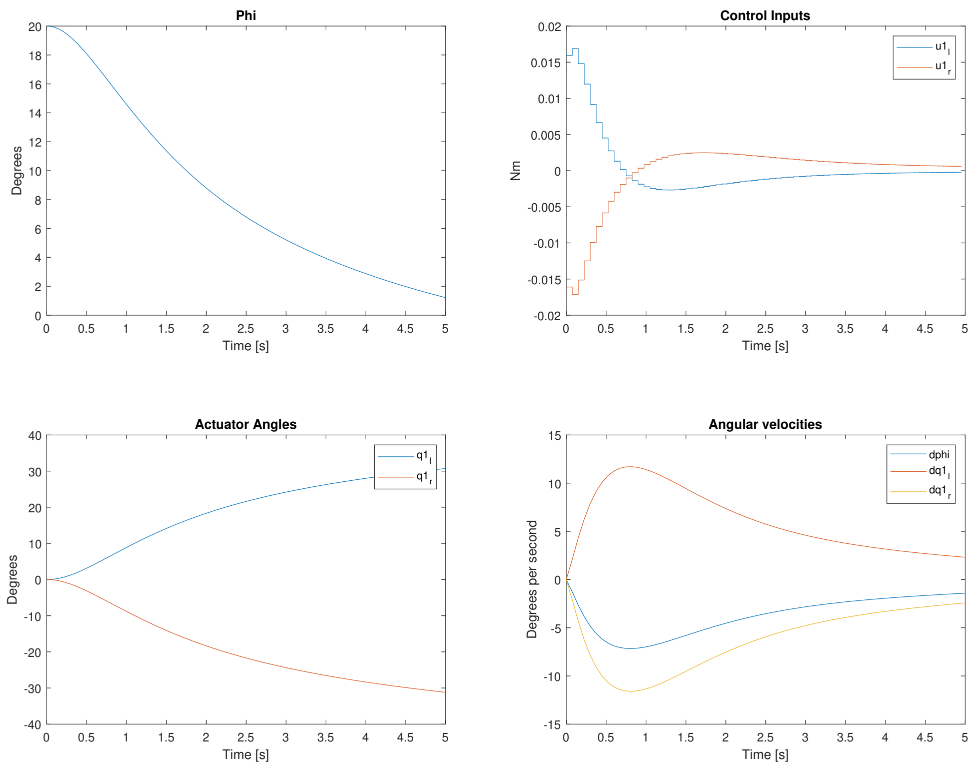


Figure 7.5: Simulation result of lateral MPC controller with plant outputs as the feedback signals to the MPC controller. Tuning parameters are equal to before, while the reference signals is $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ at all times.

In the simulation in Figure 7.5 above, the output feedback signal to the MPC controller has been switched from the linearized model outputs to the system plant outputs. The reference signal has been changed from a step function to a vector of zeros, such that each step has zero as reference value. Additionally, the attitude angle ϕ has been given an initial value of 20°. The MPC controller has equal tuning parameters as before. This change drastically improved the system responses. ϕ follows the reference, although very slow, and the angular velocities appear to converge towards zero. Additionally, the control inputs are low and appear to converge towards zero as well.

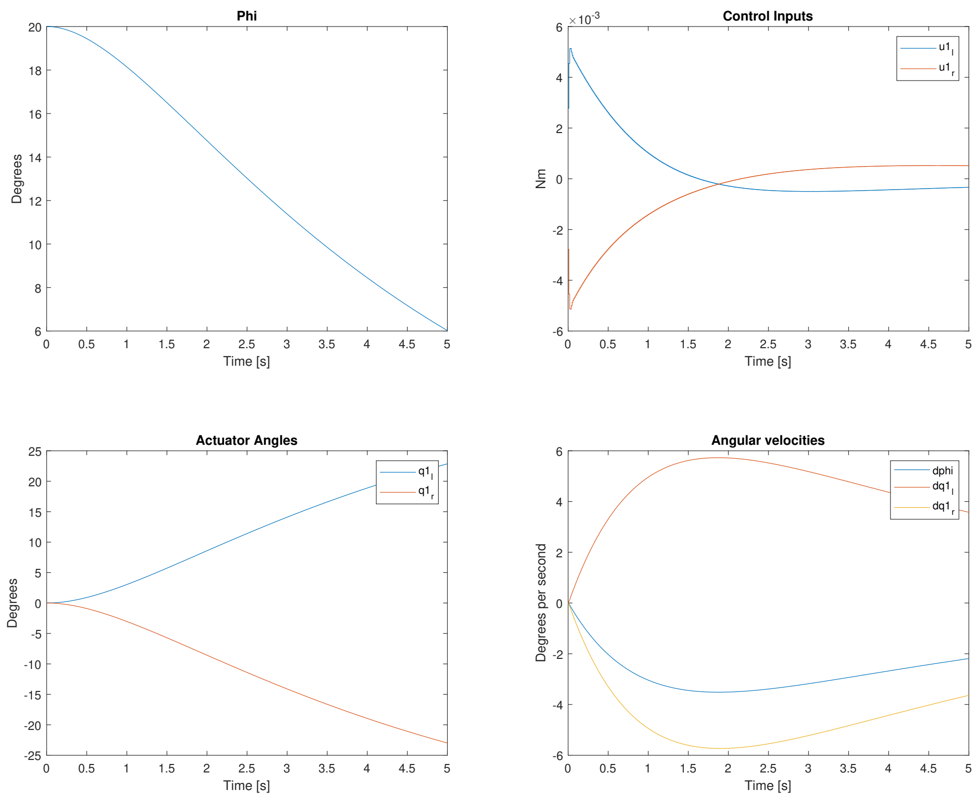


Figure 7.6: Simulation results of lateral MPC controller, when sample time T_s has been changed from 0.075 to 0.01 from the previous plots.

The plots in Figure 7.6 show the results of changing the sample time from 0.075 to 0.01. The remaining MPC controller parameters are left unchanged, and the plots clearly show that the response is slower than before. Both angular velocities, actuator angles, and ϕ has slower responses. The control inputs have significantly lower values than before, explaining the sluggish responses. Nonetheless, the closed-loop system seems robust, and the responses follow their references, although much too slow.

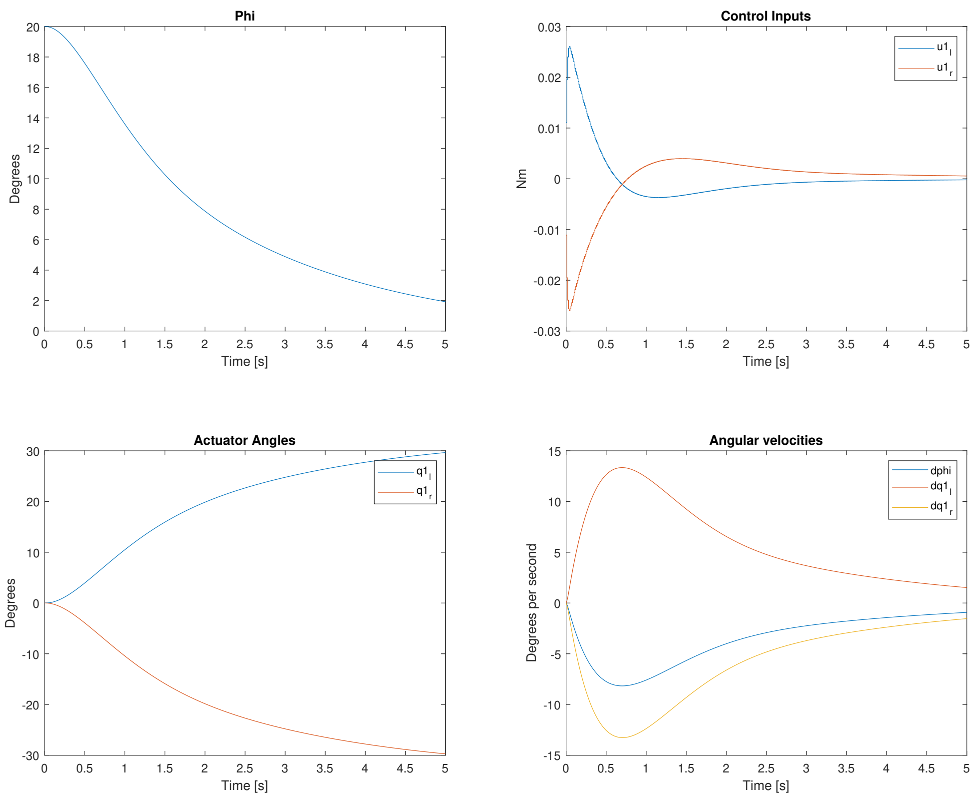


Figure 7.7: Simulation results of lateral MPC controller with the effect of the State Estimation slider.

Figure 7.7 above shows the effect of the state estimation slider in the MPC designer. When the slider is moved further to the right, meaning faster state estimation, the responses are significantly faster. Additionally, the control input values are larger, resulting in more control action in general. Still, the responses persist in being robust and smooth but too slow.

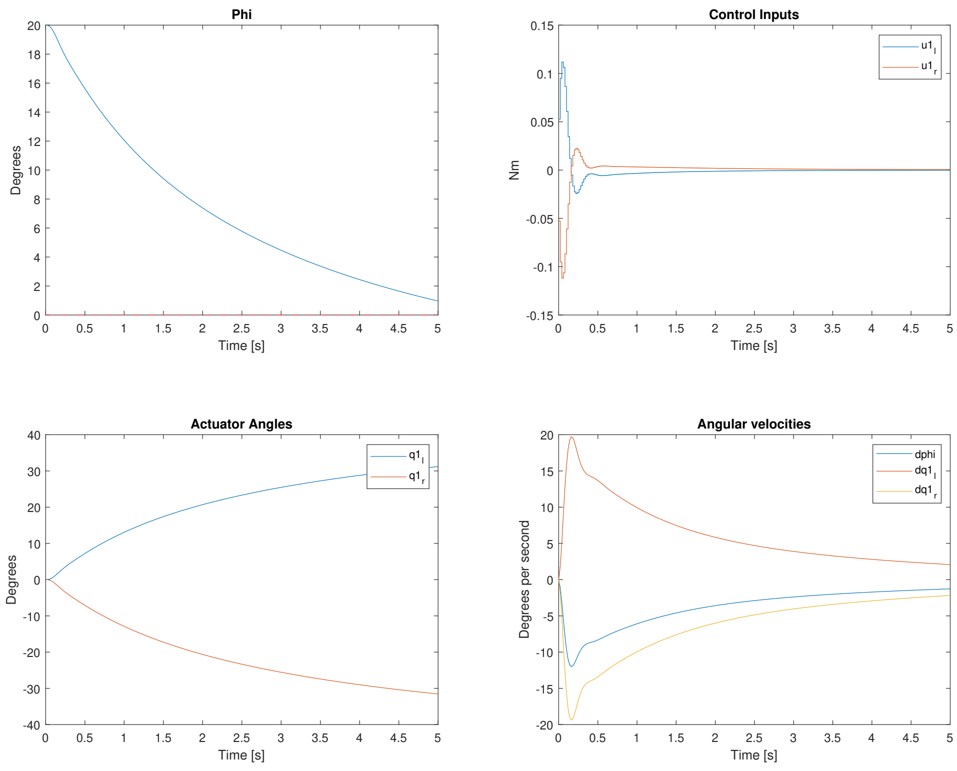


Figure 7.8: Simulation results of lateral MPC controller without the effect of the State Estimation slider, but with a decrease in prediction horizon from 10 to 8.

The slider is changed back so that there is no effect of the sped-up state estimation. However, the prediction horizon is decreased from 10 to 8. The effect of lowering the prediction horizon to 8 did not have any visible effect. However, it was kept at 8 because a lower horizon has lower memory usage and complexity than a higher horizon. The weights of the output variables have been changed as well. The weight of ϕ is set at five, while the weight of the angular velocities is set to 0.02. Compared to Figure 7.6, the response is much quicker, but the initial response time is still slower than with the sped-up state estimation effect shown in Figure 7.7. The actuator angles do not stabilize during the period of five seconds, even though the control inputs go towards zero.

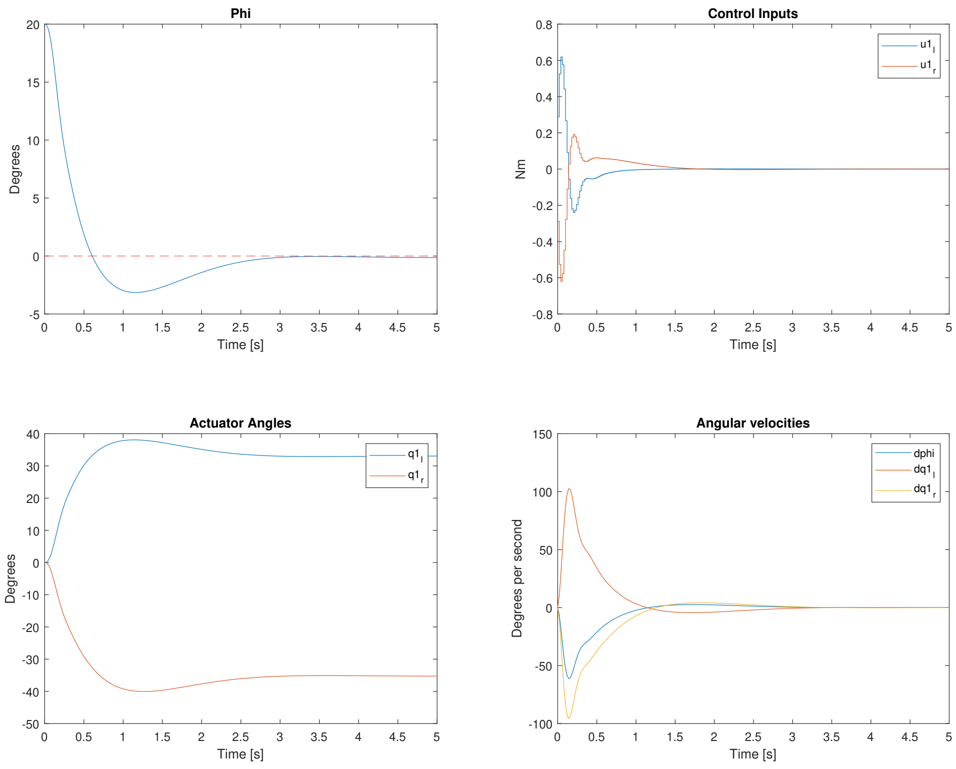


Figure 7.9: Simulation results of lateral MPC controller with the effect of the State Estimation slider combined with the decrease in prediction horizon from 10 to 8.

The plots in Figure 7.9 above show the results of combining the MPC controller from the previous plots with the effect of the state estimation slider from the MPC Control Designer options. The slider is moved from the initial, middle point towards the right, such that it reaches just over 3/4 of the sliding bar. The effect is seen in the plots, where ϕ has a significantly faster response while the system proceeds to stay robust and stable. The control inputs stay low, under $|1|Nm$, and the actuator angles stabilize around $40°$ and $-40°$, which is well beneath their threshold limit of $+/-90°$. The angular velocities and the control inputs go towards zero after about 2.5 and 1.5 seconds, respectively. The response of the attitude angle ϕ has an overshoot of $3°$ and a rise time of 0.55 seconds. The steady-state error is close to zero but has a minor deviation of $x°$. The results of these parameter settings are overall very good, but the settling time of 2.5 seconds is slightly longer than wanted.

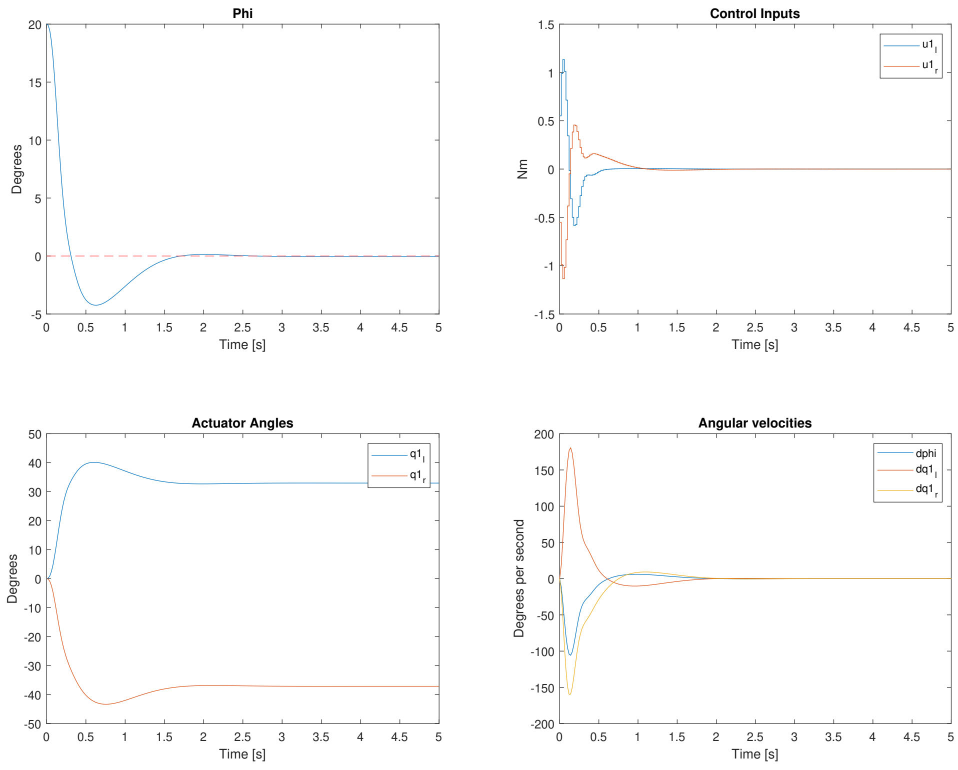


Figure 7.10: Simulation results of lateral MPC controller with even more effect of the State Estimation slider.

In the simulation in the plots in Figure 7.10 above, the state estimation slider is slid even further to the right, almost at the right end of the slider bar. The response difference compared to the last set of plots is distinct. The rise time and settling time are shorter, with 0.3 and 1.5 seconds, respectively. However, the overshoot is now almost 2° bigger at 5°. The response of the control inputs has slightly larger values, but they still converge to zero, along with the angular velocities. The actuator angles converge to about the same values as the previous plots. For the most part, it is only the response of ϕ that can determine which of the state estimation slider values give the best result. The response of ϕ in the plots in Figure 7.9 have a smaller overshoot but a longer rise and settling time. All three parameters say something about the controller's performance, but how to prioritize their importance depends on the specific control situation.

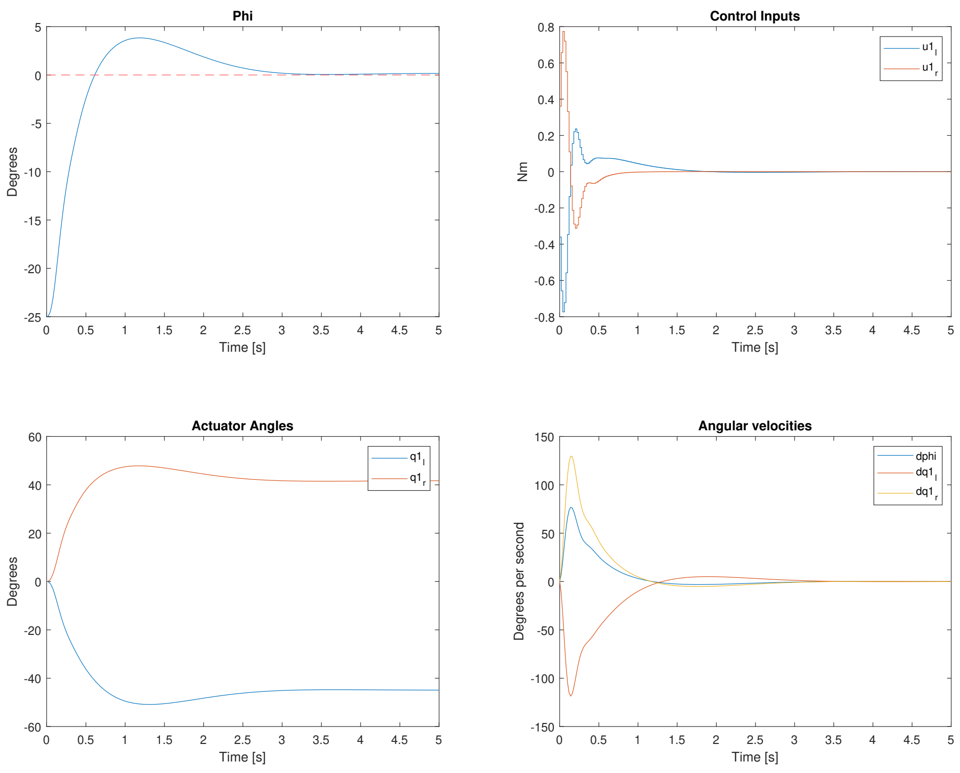


Figure 7.11: Simulation result of lateral MPC controller, with initial condition for ϕ $-25°$, instead of $20°$.

The initial ϕ value in the plots in Figure 7.11 above is set to $-25°$ instead of $20°$. The response proves that the controller performs virtually equal to the sign of the initial ϕ value. The MPC controller is equal to the controller from the simulation in Figure 7.9.

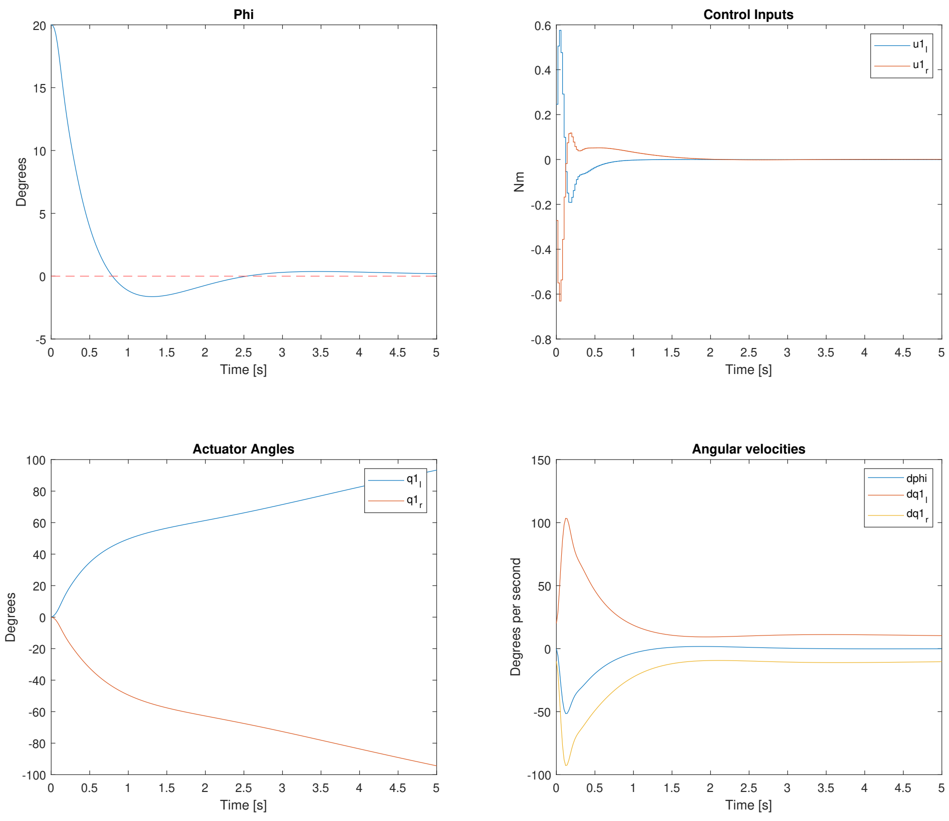


Figure 7.12: Simulation result of lateral MPC controller, with initial angular velocities for $q1_l$ and $q1_r$ of 20° and −10°, respectively.

The plots in Figure 7.12 above show the case where the actuator angular velocities $dq1_l$ and $dq1_r$ are given initial values of 20° and −10° respectively. The plots show that the response of ϕ tracks the reference value of zero very well, while the angular velocities are not able to converge to zero but instead to 10 and −10 degrees per second. This results in actuator angles that continue to grow, well past the point where ϕ has converged. The plots show in addition that the control inputs converge to zero.

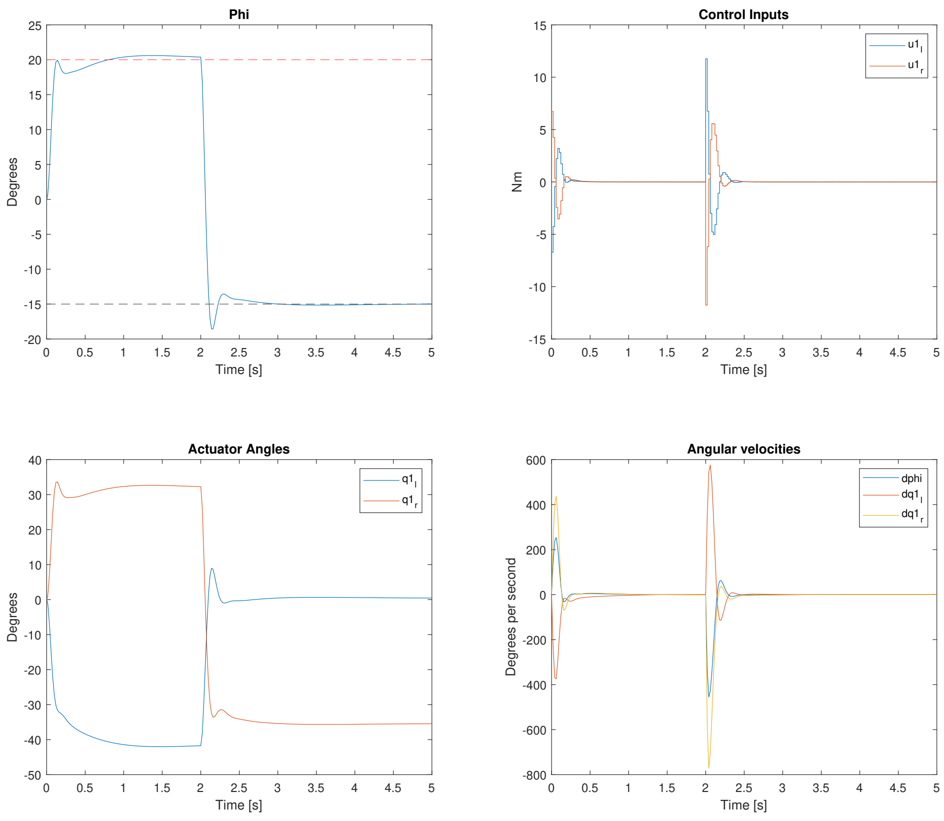
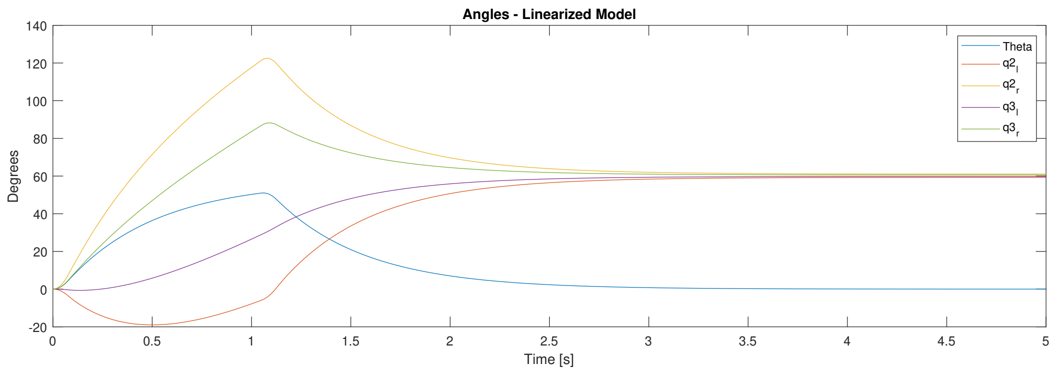


Figure 7.13: Simulation result of lateral MPC controller with a step of $[20°, -15°]$ as reference signal for ϕ.

In the simulation shown in the plots in Figure 7.13 above, the reference signal for ϕ is given a value of 20° for two seconds, and a value of $-15°$ for the remaining time. The initial value of ϕ is set to zero, along with the rest of the state's initial values. The resulting ϕ response tracks the reference fast and accurately. There is a small steady-state error for the 20° reference value and next-to-nothing for the $-15°$ reference value. Additionally, the overshoots are less than 4°, and the settling times are short. The actuator angels can almost converge for the 20° reference value, but not fast enough before the reference value is changed. They converge to zero and 35° for the second's reference value of $-15°$. These left side plots individually are satisfactory, while the plots to the right show signs of weakness regarding the MPC controller. The plot of the control inputs shows that the actuators reach substantial values for a short amount of time, both at the beginning and after two seconds, which coincide with the times when the reference value changes. This results in very large values for the angular velocities at the same time periods. Depending on the type, the actuators might be able to reach values of $10Nm$. However, the actuator velocities of well over 200 degrees per second seem unlikely and, not to mention harmful for the mechanical parts involved.

7.4 Longitudinal



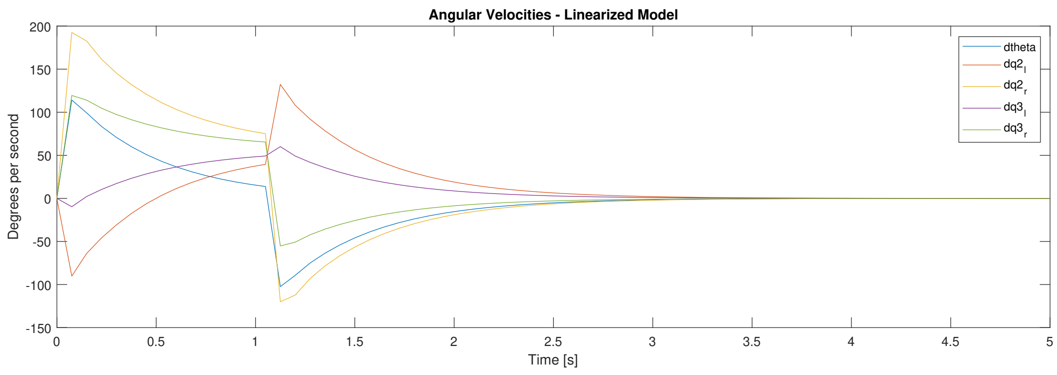


Figure 7.14: Simulation results of the longitudinal MPC controller, when both prediction model and plant is the linearized longitudinal state space model. The reference signal is $[1,1,1,1,1,1,1,1,1,1]^T$ followed by - $[0,0,0,0,0,0,0,0,0,0]^T$ after one second.

The plots in Figure 7.14 show the simulation of the longitudinal linearized state-space system found in the previous section. A simple MPC controller scheme is implemented as described with the Matlab MPC toolbox. Most of the tuning parameters for the MPC controller were kept at their default values, except the sampling time T_s, which was set to 0.075 seconds. The default control input weights were set to zero, and the default control input rate weights were set to 0.1 for all the control inputs $u2_l$, $u2_r$, $u3_l$ and $u3_r$. The weight for the attitude angle θ was set to 1, as the control of this angle is the main goal of the controller. The actuator angle velocities were given a weight of 0.2 each. The control horizon was kept at the default value of 2, while the prediction horizon was set to 10. The reference signal was as for the lateral case equal to 1 for all output states. After one second, the reference signal was set to zero for all output states. The plots show that the attitude angle θ follows the references to a certain degree but that the response is too slow for the angle to reach the reference value of 57° before one second has passed and a new reference signal takes over. Their weights for the angular velocities are also set to 1 up until one second. However, the plots show how the responses of these angular velocities do not follow this reference signal but instead behave such that θ can reach its reference value. The angular velocities follow the reference signal well for the zero reference signal after one second, and they all go towards zero. The attitude angle θ converges to zero, while the actuator angles converge to 60°. There is no overshoot of the θ response, and both the rise and settling time are relatively fast, at about one second. The two parameters are equal because there is no overshoot.

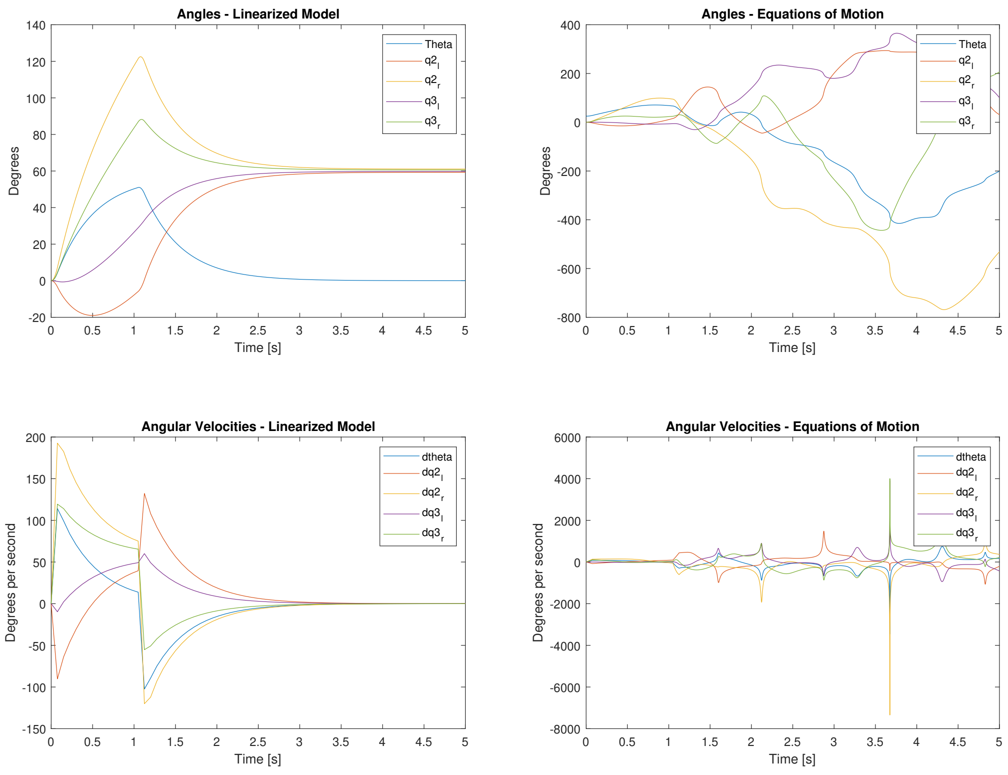


Figure 7.15: Simulation results of the longitudinal MPC controller, when the control inputs u are fed into the longitudinal Equations of Motion, as well as the linearized plant. The reference signal is $[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]^T$ followed by -$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ after one second.

The plots above in Figure 7.15 show the same situation as the previous plots, but the control inputs $u2_l$, $u2_r$, $u3_l$ and $u3_r$ are fed in as inputs to the Equations of Motion as well. The reference and control parameters have the same values as before. It is clear from the plots that the controller scheme performs much better for the linearized plant than for the Equations of Motion. The linearized plant is used for the prediction models, and these output signals are still the signals fed into the MPC controller. It is clear from the plots that the linearized plant and the Equations of Motion differ because the two signal responses from the same control inputs are very different. It is hard to tell how much they differ since a small deviation in accuracy for the MPC controller can cause quite big changes. One thing that is clear from the plots is that the actuator angles $q3_l$ and $q3_r$ behave more independently from the actuator angles $q2_l$ and $q2_r$ than what is translated by the left side plots. The linearized plant response to the left makes it seem like the left and right side actuator angles always have the same response. The right-side plots show that this is not the case. Additionally, the right side angular velocities are dominated by large oscillations, while the angular rotations seem to reach tremendous values and be unstable. The MPC controller, as it is, performs very poorly for the longitudinal Equations of Motion, and some upgrades are needed for it to perform better.

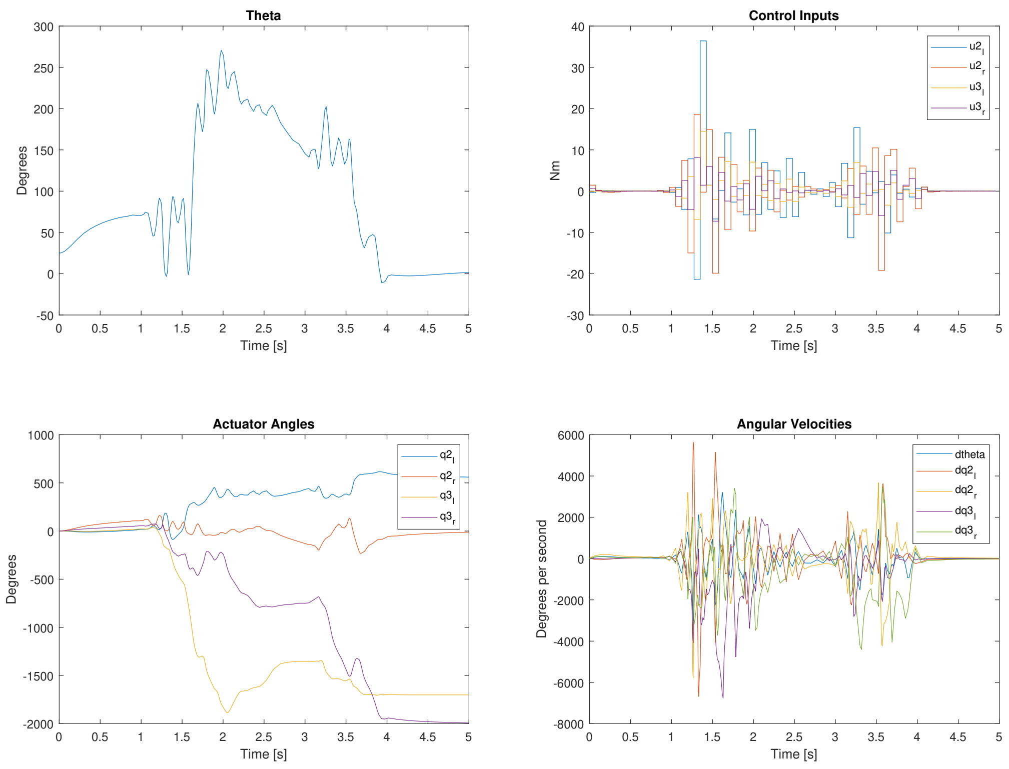


Figure 7.16: Simulation results of the longitudinal MPC controller, when the states from the Equations of Motion are fed into the MPC controller. The reference signal is $[1,1,1,1,1,1,1,1,1,1]^T$ followed by - $[0,0,0,0,0,0,0,0,0,0]^T$ after one second.

The plots in Figure 7.16 above show the simulation of the situation where the output signals from the plant are fed back into the MPC controller instead of the outputs from the linearized model. The MPC controller is equal to before. The initial attitude angular rotation of θ was set to 25°. The MPC controller struggles very clearly with tracking the references and rendering a robust and stable system. All references are highly oscillatory and reach unreasonably large values. Especially the plot of the angular velocities shows how the signals oscillate between 6000 and -6000 degrees per second.

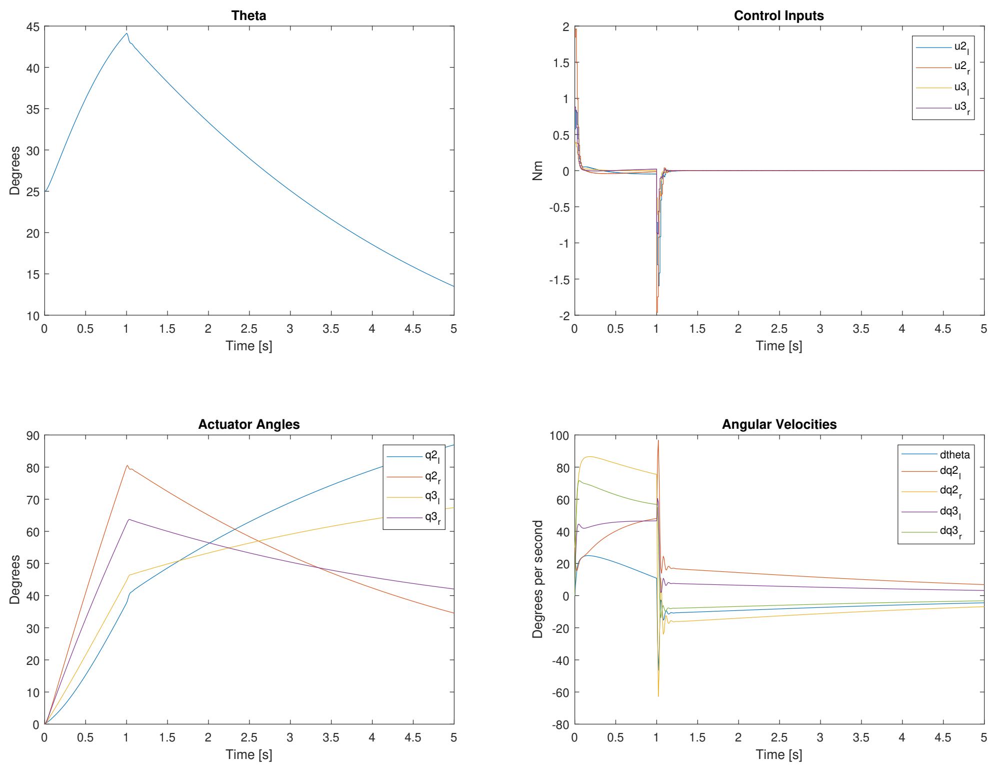


Figure 7.17: Simulation results of the longitudinal MPC controller, when the states from the Equations of Motion are fed into the MPC controller. The sample time T_s is changed from 0.075 to 0.01. The reference signal is still $[1,1,1,1,1,1,1,1,1,1]^T$ followed by - $[0,0,0,0,0,0,0,0,0,0]^T$ after one second.

The simulation in the plots in Figure 7.17 above has a new sample time T_s of 0.01 seconds as opposed to the previous value of 0.075 seconds. This smoothed the responses, as can be seen when comparing the responses to the previous plots. The prediction horizon was set to 8 instead of 10, but this change did not make a visible difference in the signal responses. The right side plots show that the angular velocities and control inputs have drastically improved responses compared to before. There are no longer large oscillations or unreasonably large output values. However, the plots still show some issues around the time of the reference signal change. The angular velocities oscillate very rapidly for a short time, after about one second. The actuator angles and θ also show that the trajectory change is far from smooth.

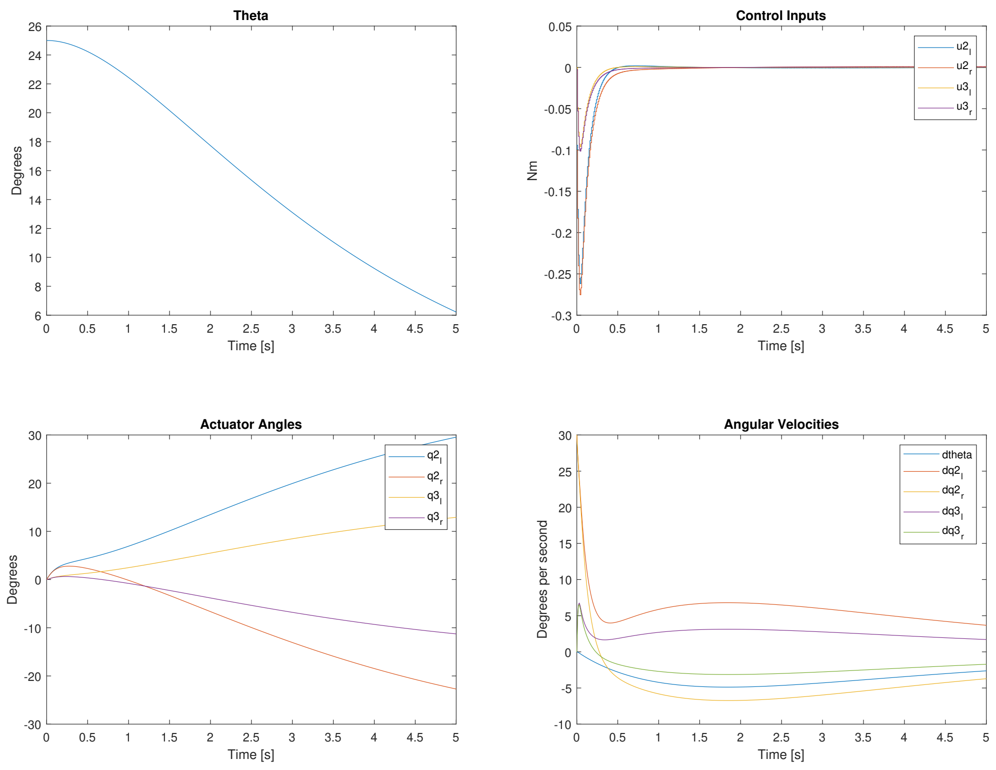


Figure 7.18: Simulation results of the longitudinal MPC controller. The sample time T_s is changed from 0.075 to 0.01. The reference signal is $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$.

The plots in Figure 7.18 above show the simulation when the reference signal is changed to simply being zero for all states at all times. The MPC controller and initial θ value are equal to before. The plot of θ shows that the response is far too slow. The response reaches 6° after five seconds, while it should have reached 0° after at most two seconds. The control inputs converge to zero, while the angular velocities converge to values between $-6°$ and $8°$. Since θ does not reach the reference value during the simulation time, there is no view of what happens after this reference value is reached.

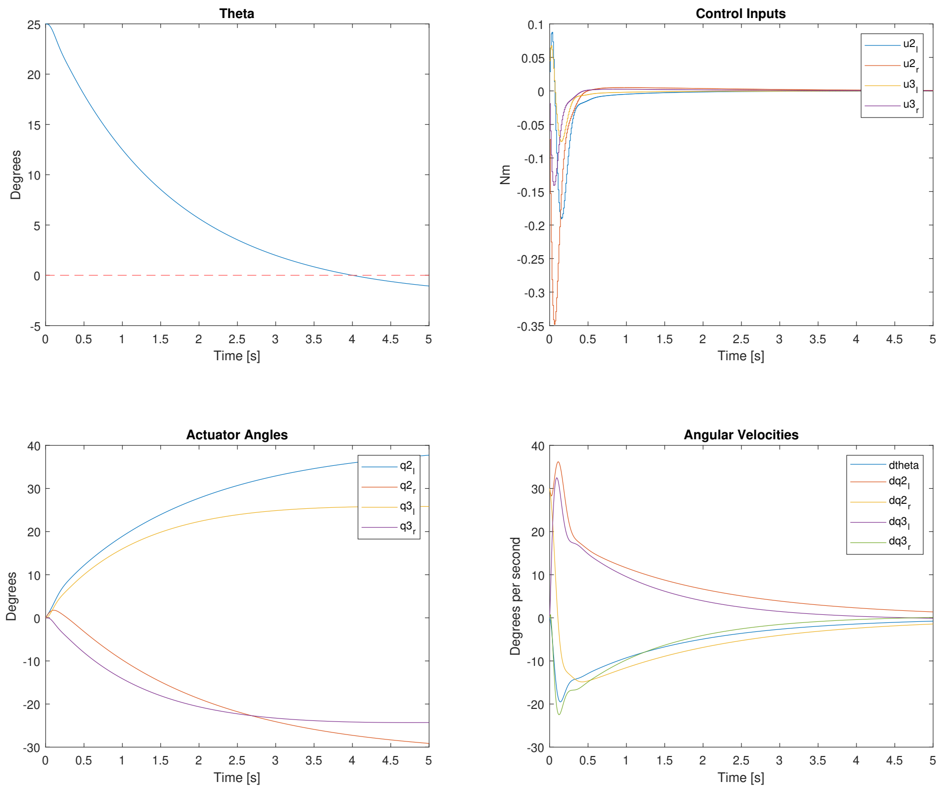


Figure 7.19: Simulation results of the longitudinal MPC controller. The weights of the output signals are set to $[\theta = 5, dq2_l{=}dq2_r = dq3_l = dq3_r = 0.02]$. The reference signal is $[0,0,0,0,0,0,0,0,0,0]^T$.

The plots in Figure 7.19 show the result of increasing the weight of the state θ to 5 and decreasing the weight of the actuator angular velocities to 0.02. Compared to the previous plots, the response of θ is faster, while the responses of the angular velocities have much larger values instead of residing around zero. Additionally, the angular velocity responses can converge to zero, along with the control inputs. The actuator angles have smooth responses and stay well within their limit of $+/90°$. Although the response of θ is faster than before, the rise time is still four seconds, which is far too long.

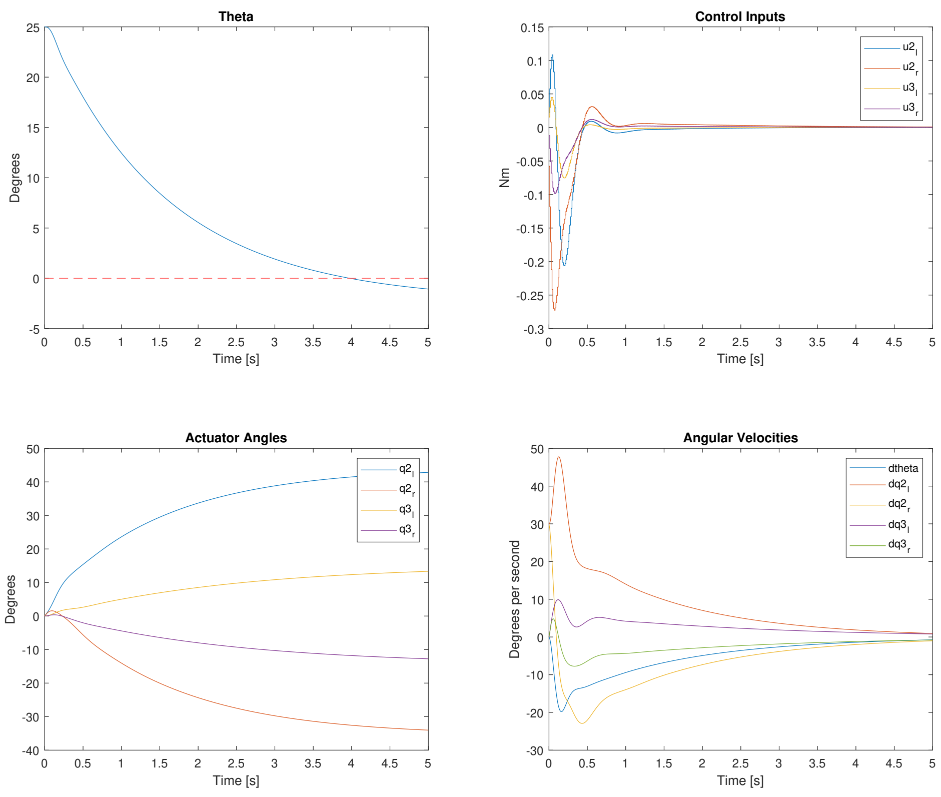


Figure 7.20: Simulation results of the longitudinal MPC controller. The rate weights of the manipulated variables are changed to $[u2_l = u2_r = 127, u3_l = u3_r = 0.38]$. The reference signal is $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$.

The plots in Figure 7.20 show that there is no visible change in θ of increasing the rate weights of the manipulated variables from 0.1 to $u2_l = u2_r = 0.127$ and $u3_l = u3_r = 0.38$. There are, however, changes in the other three plots, where the control inputs have slightly smaller values, and the actuator angles for the $q2$ actuators converge to larger angles than before. In contrast, the $q3$ actuator angles converge to smaller angles. Overall, the change did not aid the response of θ for this scenario, but the other responses might have become slightly more robust as the control inputs are smaller than before.

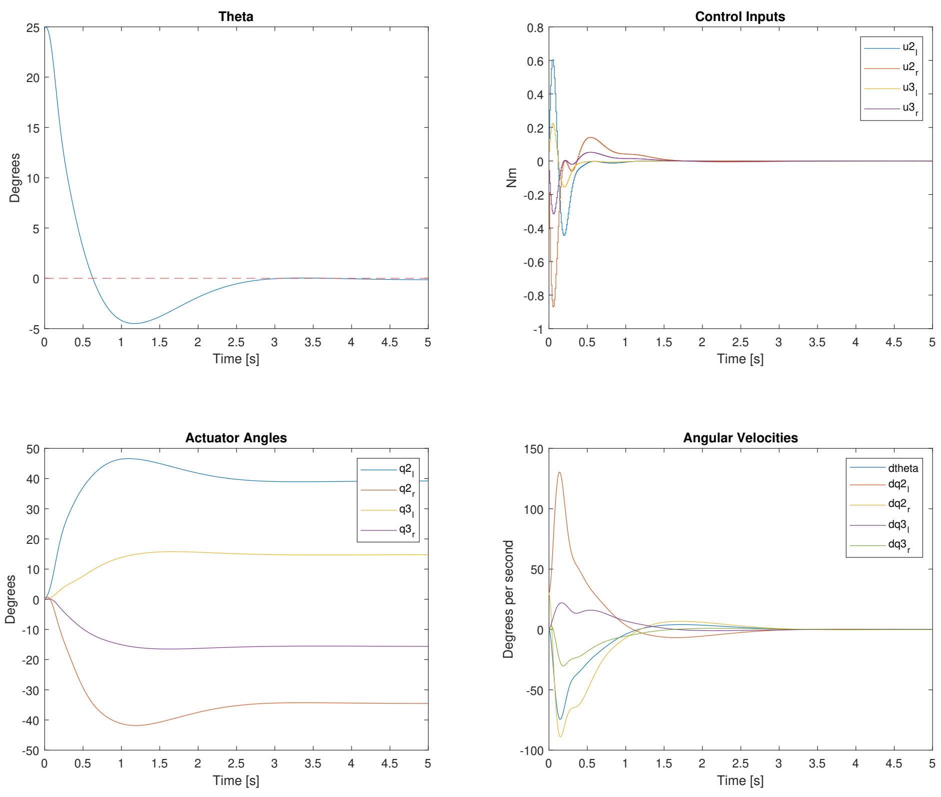


Figure 7.21: Simulation results of longitudinal MPC controller with the effect of the State Estimation slider. The reference signal is $[0,0,0,0,0,0,0,0,0,0]^T$.

The plots in Figure 7.21 above show the effect of the state estimation slider from the MPC designer toolbox in Matlab. The slider is moved to the right such that it reaches 3/4 of the bar. The system's response becomes much quicker, with a rise and settling time for θ of 0.65 and 2.5 seconds. The response of θ follows the reference closely after 2.5 seconds, with close to zero steady-state error. As for the lateral case, there is a slight overshoot in the θ-response plot. The overshoot reaches about 5° at most, slightly higher than wanted. The control inputs and the angular velocities converge to zero, while the actuator angles all converge to angles between −40° and 40°.

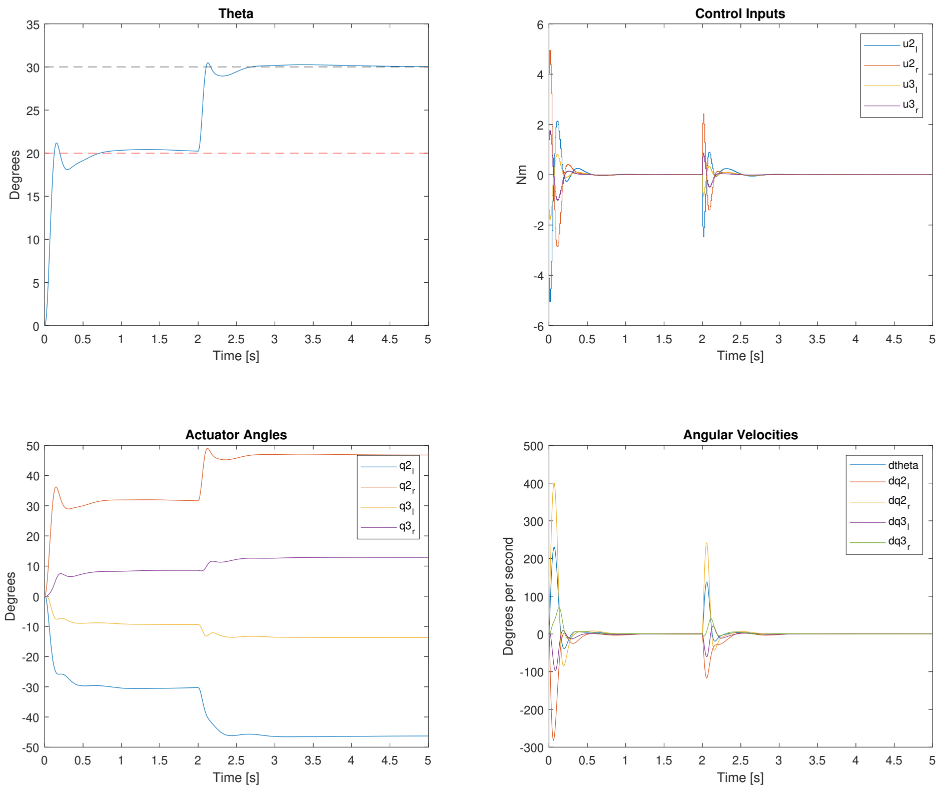


Figure 7.22: Simulation results of longitudinal MPC controller with a step reference signal of 20° initially, followed by 30° after two seconds.

The plots in Figure 7.22 above show that the θ response is able to track the a step reference signal of 20° up until two seconds, and 30° after two seconds. The response is fast, with rise times of well under 0.5 seconds, and the steady-state errors are negligible. Evaluating θ isolated will leave an impression of an MPC controller which performs very well. However, the plot of the control inputs and angular velocities shows that their response values are very large and slightly oscillatory. Especially the angular velocities have unreasonably high accelerations at the very beginning.

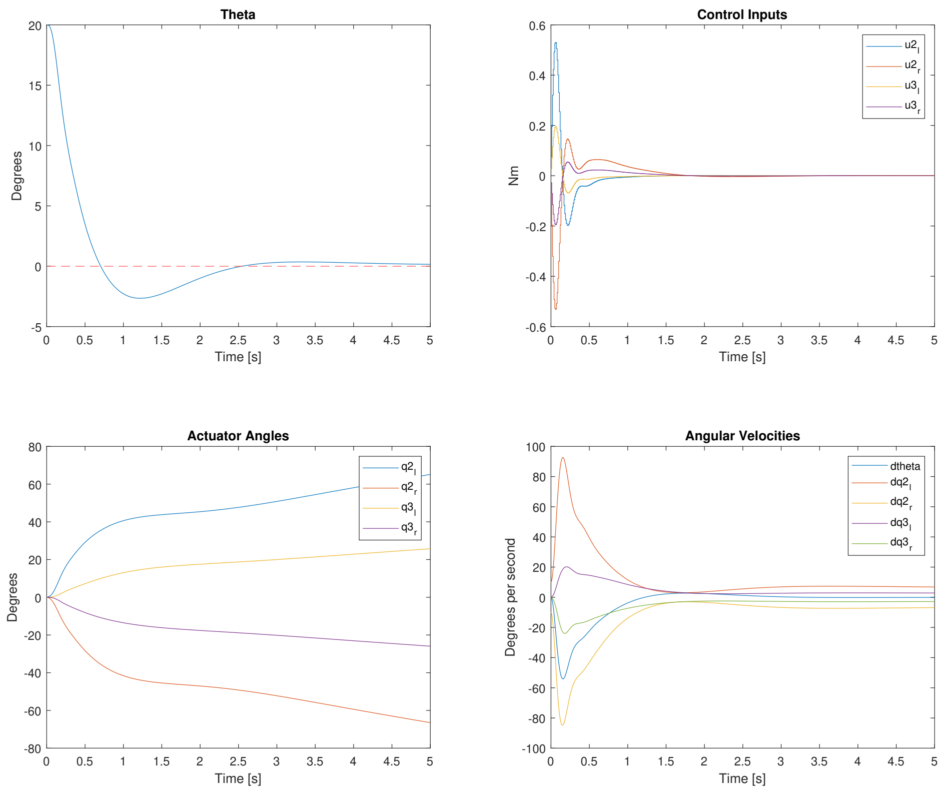


Figure 7.23: Simulation result of longitudinal MPC controller, with initial angular velocities for $q2_l$ and $q2_r$ of $10°$ and $-10°$, respectively.

The plots in Figure 7.23 above show the case where the actuator angular velocities $dq2_l$ and $dq2_r$ are given initial values of $10°$ and $-10°$ respectively. The plots show that the response of θ tracks the reference value of zero very well, while the angular velocities are not able to converge to zero but instead to values between 10 and -10 degrees per second. This results in actuator angles that continue to grow, well past the point where θ has converged. The plots show in addition that the control inputs converge to zero. The overshoot is smaller here than before, but the initial condition for ϕ is smaller, which is most likely the reason.

Mutual for the lateral and longitudinal MPC schemes is that only an interval of initial conditions for the attitude angles results in proper reference tracking. The plots of these limits for when the controller is unable to stabilize the attitudes are unfortunately not presented in this thesis. The lateral MPC struggled when the initial condition set on ϕ increased above $\sim 33°$ or decreased below $\sim -33°$. The interval for the initial condition on ϕ for lateral MPC was set to $[-32°, 32°]$ to ensure one degree of buffer at each end of the interval. The longitudinal MPC began to struggle already at $\sim 31^\circ$ and $\sim -31^\circ$, and thus the interval was set to $[-30^\circ, 30^\circ]$. Common features for both MPC schemes outside these intervals were oscillations, large control inputs, and actuator angular rotations.

Chapter 8

Discussion

The main object of the controllers developed during this project was to control the attitude angles towards some reference angle θ_r or ϕ_r. It was excepted that this would be possible as long as the initial attitude angle did not differ too much from the reference angle. The hypothesis for the angular difference for which it would be possible to control the angle was an interval of $+/-35°$. The robot system in question consists of the pitch attitude angle θ and the roll attitude angle ϕ since the yaw angle ψ is omitted. However, the two attitude angles θ and ϕ were split into two control schemes and regarded separately. The two different sets of models for the lateral and longitudinal dynamics were implemented in several control schemes. The first controller that was developed and tested was the PID controller. The second was a simple MPC controller where the predictions were made with the linearized models. The last controller scheme was an output feedback MPC controller, where an output feedback controller was added to the system output states. The results and tuning variables for these controllers are given in the previous chapter.

8.1 PID

The PID controller scheme was split into two separate controllers; one for the lateral dynamics and one for the longitudinal dynamics. The final gains K, T_i and T_d for the best results are given in Tables 7.1 and 7.2, along with the initial angular and position states for the two dynamical systems.

The best result for the lateral dynamical system is plotted in Figure 7.1. As explained in the Results Chapter, the response of the dynamic system is slow. Over one second passes before the attitude angle ϕ has any significant change in rotation. PID-controller gains generally work such that increasing the K results in a faster and more aggressive response, while increasing the integral gain (decreasing T_i) results in a response that minimizes the steady-state error or the gap between the reference value and steady-state response value. However, the response in these plots does not reach closer than about 2.5° to the reference value

0°, and the response is already unstable. It is only able to rest at this value for one second before the ϕ response again increases. In addition, the actuator angles $q1_l$ and $q1_r$ are rapidly decreasing and pass $-400°$, or over one whole rotation. The response is neither close enough to the reference value nor stable. An increase in the proportional or integral gain only resulted in a higher instability, oscillations on both angular velocities and ϕ, and a shorter time period where the angle ϕ could stay at the steady-state value. This is where an increase in the derivative gain T_d would be natural since this term dampens the oscillations because it penalizes rapid velocity changes. However, the derivative term could not sufficiently dampen the oscillations or minimize actuator angle changes. The effect of the decrease in derivative gain for this system was a delay in the system response. Intuitively, it makes sense that the PID controller scheme proves to be insufficient for the control problem.

The best result for the longitudinal dynamical system is plotted in Figure 7.2. The system response for the longitudinal system differs a bit from the response for the lateral dynamical system. The system response is faster but also much more unstable. It was challenging to control the longitudinal dynamical system to avoid large oscillations and, at the same time, reach the reference signal. This is also clear when comparing the results and the PID gains. The integral term for the longitudinal PID controller is a magnitude of five higher than the lateral integral term. This was necessary for the θ response to be able to reach the zero-valued reference signal. However, this caused large and rapid oscillations in the system, which is clear from the angle derivatives plot. The rotation of θ is able to oscillate around close to zero after 2.5 seconds, but this happens at the cost of large actuator angle velocity changes. This behavior is unacceptable for such a system with mechanical parts involved, as these would be damaged. This response is with a very large damper term T_d. The damper gain is much larger than the one used for the lateral PID controller, but it is still not possible to dampen the system's oscillations. The alternative is to decrease the integral term (increasing T_i), but this resulted in a system response that could not reach the reference rotation θ_r. Overall, these plots show the best possible result in terms of reaching the reference signal. The PID controller is insufficient for controlling the attitude angle θ for the longitudinal dynamics.

The dynamics are complex, non-linear and to a large degree, coupled. Each state of the system is dependent on the other states, including the double derivatives and derivatives of the other states. The controller scheme makes no prognosis or hypothesis as to how the system will behave, resulting in a PID controller that has to control 'blindly'. The more complex and coupled a dynamical system becomes, the more beneficial it will be to have some prediction step to the controller, and the PID controller has none. This is the main problem of the basic PID controller. There is no way of knowing whether a configuration is smart or not for the system's behavior without further investigating how the system will behave to different control inputs. A solution could be to spend more time evaluating how the system behaves with different motions. How does the system behave when the legs move slowly to the left or right, or faster to the left or right. How should the legs move in order to control the system correctly? This is an extensive task, but it would, as a result, be possible to develop different PID controllers for the different motion scenarios of the system. It would also be necessary to include the angular derivatives somehow because they need to go towards zero for the system to exclude oscillations. However, it seems more reasonable to switch to a prediction-based controller that takes both the states and the angular derivative states as input.

8.2 MPC - Model Predictive Control

As for the PID controller scheme, the MPC controller was split into two separate controllers; one to handle the lateral dynamics and one to handle the longitudinal dynamics. The MPC controller schemes were developed in Matlab Simulink with the MPC toolbox block. This MPC block in Simulink needs a model plant used to compute the predictions. The lateral and longitudinal dynamics predictions are based on the linearized lateral and longitudinal state space systems.

The first set of plots for lateral and longitudinal MPC, Figures 7.3 and 7.14, show how well the attitude angles ϕ and θ are able to track their references. The attitude responses are very similar; each of them reaches up to $\sim 50^\circ$ of the first reference value before the reference is changed to zero. Their settling times have about the same length, and there are no overshoots. The MPC schemes in these scenarios are able to perform so well because the system for which it makes predictions is equal to the plant of the systems. Additionally, there is no noise or disturbance present, such that the predicted output states from the MPC controllers will be equal to the actual plant outputs. This, however, does not say much of how the controllers will perform for the actual system plants - the Equations of Motion.

The following set of plots, from Figures 7.4 and 7.15, show the side by side comparisons of the linearized models and the Equations of Motion. As the same control inputs are fed into both the models and the plants, a perfectly modeled system would produce the same outputs as the plants. However, since this is not the case because of the linearization, the signal outputs are very different. This is as expected since the linearization of the Equations of Motion removed a large portion of the system dynamics. Even though the results are as expected, it proves how important the tuning process will be. The results from the first set of plots, with only the linearized models present, are from zero amount of parameter tuning.

The following sets of plots have MPC controllers where the output state feedback has replaced the linearized models as feedback to the MPC controllers. The Figures 7.5 and 7.16 show the immediate result of this without any further tuning or changes. The lateral MPC performs decently only with this immediate change, and the attitude angle ϕ is able to track the reference signal, although very slowly. The longitudinal MPC performs very poorly in comparison. This might be because the longitudinal MPC reference is still a step response, while the reference for the lateral MPC is set to be zero for all states. It makes more sense to compare the lateral MPC scheme from Figure 7.5 with the longitudinal MPC scheme from Figure 7.17, as both of these have zero as reference value. Both of these sets of plots have attitude responses that are able to track their references, although very slowly. Both magnitudes of the control inputs are relatively low, but the magnitude of the longitudinal control inputs lies about 10 times higher than for the lateral case. The reason for this could be the fact that the initial θ value is 5° higher than the initial ϕ value, such that the control inputs need to be larger to control the rotation. Another reason might be that two of the angular velocities of the longitudinal MPC scheme have initial values of 30 degrees per second. The control inputs are much larger during the time period where these angular velocities are large and shrink after these velocities approach closer to zero. It appears to be a deliberate control action move to guide these velocities towards 5 and -5 degrees per second, such that the control inputs at the beginning are much larger for the very purpose of bringing the velocities quickly towards 5 and -5 degrees per second, respectively. These velocity changes are reflected in the plot of actuator angles in Figure 7.17, where it is clear that the rotation of $q2_r$ changes direction the moment its actuator angular velocity move under zero degrees per second. It makes sense that these are the longitudinal control actions taken when evaluating the theory behind the movement of the attitude angles. When the control objective is to bring the attitude θ from 25° to 0°, the left side legs will want to rotate with a positive rotation such that the force from the movement of the actuator points back on the robot's body, and the attitude angle is shifted downwards. The right side legs move the opposite direction to somewhat dampen the movement, and smooth out the response. It is clear that the left side legs rotate slightly faster and for longer than the right side legs, meaning they are the main contributors for the attitude rotation change.

The plots in Figures 7.6 and 7.18 show the MPC controllers when the sample time is changed to 0.01 for both lateral and longitudinal MPC. When comparing the attitude responses to the previous plots, they both respond slower, even though the expectation was for the opposite to happen. A decrease in sample time T_s means that the optimization problem at runtime is solved more often, and generally, this leads to improved performance in terms of both response time (or bandwidth) and robustness. The response time intuitively makes sense because each new prediction happens faster than before, while improved robustness shows that the phase and gain margins are improved. As the sample time gets small, the controller can react faster to unexpected changes, which is the reason for improved robustness. It is clear from the longitudinal plots that the decreased sample time has led to smoother control inputs and thus smoother system responses in general. The robustness of the system seems to have improved with this change. This is, however, the only noticeable improvement. Both longitudinal and lateral MPC responses are slower than before, which opposes most of the reasoning behind lowering the sample time. It appears as if the sample time has decreased below the point where the optimization problem can complete within each sampling period. Since the optimization problems are not completed during the prediction steps, the outputs can not be expected to have the expected responses. It is difficult to determine what result has come from the optimization problems. A possible solution is to use a suboptimal optimization solution instead of waiting for the optimal solution. This is achieved by decreasing the complexity and size of the optimization problem.

Another critical remark about the plots regards the output signal weighting. Only the attitude angles and actuator angular velocities have weights, with the attitude angle weights a magnitude of five times the weights of the actuator angular velocities. These plots show how this affects the controllers to guide the attitude angles and the angular velocities towards their reference, although slower and less aggressive. The actuator angles have no weight and are free to move without any penalty. It is, however, still expected that they eventually converge to some angle because the reference of the actuator angular velocities is zero.

There are other possible solutions to shortening the response time before opting for a suboptimal solution. Figure 7.8 shows how the ϕ response of lateral MPC changes when the prediction horizon is decreased from 10 to 8. Compared to Figure 7.6, the response almost reaches the reference of zero after five seconds, compared to previously only reaching 6°. This substantiates the theory of the optimization problems not finishing on time. When the prediction horizon decreases, there are fewer control intervals to be solved during each time interval and a higher chance of solving the optimization problems on time. This appears to be the case for this MPC controller because the result of lower computational complexity is a drastically shorter attitude rise time. The prediction horizon for longitudinal MPC was also decreased, but this did not have the same result as for the lateral MPC.

Apart from the sample time being too low for the optimization problems to finish on time, the responses of the attitude angles might not be aggressive enough because the weighting of the output states does not emphasize the importance of reference tracking for the attitude angles. The plots in Figure 7.19 show just how much the weighting affects the responses. The plots here compared to the plots from Figure 7.18 show that the attitude response is considerably more aggressive and with a faster response time when the weight is changed from 1 to 5. Additionally, the weights of the actuator angular velocities are lowered to 0.02, allowing them to stray further and for a longer amount of time away from their reference of zero. This allows the actuator angles to grow larger as well. The key difference is the weight matrix Q, which previously had a dilemma regarding the control moves. Before, the R-matrix translated that the attitude angle should have priority and that the control moves should behave as such. At the same time, however, it translated that the angular velocities should approach zero and not stray far from the reference. Since the attitude state had priority, the result was a very half-way reference tracking of θ and actuator angular velocities that rapidly approached zero but were not able to converge there because the attitude never reached its reference during the simulation time. The control movements wanted are contradictory, which is partly why the response is slow. The new weighting hierarchy clarifies for the controller that the actuator angular velocities are allowed to stray for their reference to better track the θ reference.

The plots in Figure 7.20 compared to Figure 7.19 shows that there is hardly any difference for the θ response when the rate weights for the control inputs are increased. Increasing these rate weights results in an R-matrix with larger values along the diagonal and a higher penalty for control input moves. This is reflected in the control inputs plot, where the control inputs have lower values than before. This results in the actuator angles that converge to smaller values, especially for the $q2$-angles, which now converge to $10°$ and $-11°$ compared to previously converging to $25°$ and $-30°$. However, the change in θ is minimal, which means the increased control input rate weights have not limited the response. This again means that the controller can operate well under these increased rate weights, and there are more benefits in keeping them increased than decreasing them again in regards to robustness. If a situation comes up where the controller would want to move the control inputs rapidly, these rate weights ensure the system behaves more robustly than if there were no penalty given in these rapid movements.

The longitudinal set of plots from Figure 7.21 along with the lateral plots from Figures 7.7, 7.9 and 7.10 show the effect of the state estimation slider from the the Matlab MPC Designer toolbox. It is clear from the longitudinal results that the change towards a suboptimal solution drastically sped up the process. The rise time decreased from 4 seconds to just above 0.5 seconds. However, for the lateral case, a decrease in the prediction horizon was also needed to reach a response time of the same time period as for the longitudinal case. It would be more expected of the longitudinal case to have these results, given that the complexity and size are larger for the longitudinal system than for the lateral system. However, the results clearly show that it is the lateral MPC that struggles with this. For the longitudinal case, the improvement of lowering the optimization complexity is immediate. This supports the claim that the prediction step did not have sufficient time to solve the optimization problem. A suboptimal optimization problem was needed for that given time interval. The slider allows the MPC controller to regard the output states fed back as less affected by random noise and more affected by disturbances. This fits well with what is happening because there is no noise present in the system but rather a constant difference between prediction models and plant models. This difference can be viewed as a disturbance by the controller because it is constant. When it is regarded as noise instead, the MPC controller state estimations make alterations to better answer to noise being present. However, this does not relate well to the problem because it is not true that the difference comes from noise. It is possible to say that the inaccuracy between the linearized models and plants are some of the reason why the prediction step takes too long to compute. Suppose the difference between the lateral linearized model and plant is larger compared to the longitudinal linearized model and plant. In that case, it explains why the lateral MPC struggles more with an optimization problem that takes too long to solve. This theory should have been tested as well.

Another remark on the results concerns the response overshoots. When evaluating the best results from lateral and longitudinal MPC schemes, the attitude responses have overshoots of between $3°$ and $5°$. Given the strict requirements set on the rise time of the responses, overshoot will be challenging to avoid. Minimized overshoot and decreased rise time are often conflicted, meaning achieving both will be difficult, if not impossible. Some systems have specific demands set on overshoot, where the lowest possible overshoot is the goal. However, it makes little to no difference for the robotic system in question if the attitude angle lies below or above the reference value. It is, however, important that the attitudes can closely track their reference within a short amount of time. This is why a slightly higher overshoot is preferred over a higher rise time, and the overshoot acceptance limit can be set at as high as $5°$.

The plots in Figure 7.11 shows that the lateral MPC has equal performance when the initial ϕ value has a negative rotation instead of a positive rotation. As there are no dynamic differences between one or the other side, and the robot is symmetric, this follows the expectation for the system behavior. The same experiment was conducted for the longitudinal MPC, and the simulation showed that also longitudinal MPC had a comparable performance for both positive and negative initial θ values.

The plots in Figures 7.13 and 7.22 show that the MPC controllers are able to track the attitude references when the reference signal consists of a step. Both simulations show that the attitude angles can closely follow the reference signals and maintain a short rise and settling time. However, the control inputs are visibly more aggressive than in the previous plot. There are no signs of direct instability since all signals can converge, and there are no unnecessary oscillations. However, the rapid and large control movements are a sign that the system's robustness has a lower priority than a fast response. For a robotic system without unexpected changes in the environment, this should, in general, not be a problem. However, it would be wise to alter either the state estimation slider bar or the sample time T_s such that the behavior is slightly less aggressive. This is because oscillations are a very unwanted behavior for the mechanical parts, and any signs that might point towards the possibility of oscillations should be considered.

Figures 7.12 and 7.23 show that when the actuator angular velocities are given initial values, the angular velocities struggle with converging to zero. This theory was tested for a few cases. It appeared that the velocities struggled with converging to zero only when a left side angular velocity and a right side angular velocity had different initial values. The reason for this is unclear, and does not translate well through the plots. This problem should be tested further if the wish is to continue with these MPC controllers. However, for the real-life robotic system, there will rarely be the case that the actuator angular velocities have initial values when falling. They will most likely stay still, until a reference for the attitude angle is set and the control scheme begins.

The lateral and longitudinal MPC were able to track the attitude references within their given initial condition intervals, as long as there were no initial conditions set on the actuator angular velocities. The further away from the reference the attitude angles begin, the larger the control moves will be when the MPC parameters remain unchanged. An option is to revisit the parameter tunings and alter them to ensure a more robust system. However, this would, in turn, result in decreased performance for the smaller angular initial conditions in terms of rise and response time. The optimal performance of the MPC depends on what it is meant to be used for, as it is impossible to achieve a short response time and robustness, and at the same time, have an MPC that would work for all initial angular conditions. The choice fell on faster response time for the attitude initial conditions intervals described in the result chapter. Lateral with the interval: $[\ 32^\circ,\ -32^\circ]$ and longitudinal with the interval: $[\ 30^\circ,\ -30^\circ]$.

Chapter 9

Conclusion

The main objective of this master's thesis was to show that it is possible to stabilize the attitude of a quadruped robotic system during free-flight solely by the rotation of the legs. The legs are each equipped with three actuators and a mass at the foot, such that the rotations create centrifugal forces, which in turn affect the attitude of the robot's main body.

The dynamics of the robotic system were split into two sections; one describing the longitudinal motions and one describing the lateral motions. The longitudinal and lateral models describing the dynamics were found using the Lagrangian method before validating them through simulation experiments.

The attitude stabilization was achieved by testing PID and MPC control schemes, with parameters and tuning found through theoretical evaluation combined with testing. The results showed that the PID appeared insufficient for the task early on, while the MPC showed potential. Therefore, the PID controller scheme was rejected, and the focus shifted entirely towards tuning the lateral and longitudinal MPC parameters as best as possible.

The lateral and longitudinal MPC features needed to promote fast system response and settling time and, at the same time, ensure a robust closed-loop system response. Short rise and settling times were achieved by meticulously tuning the sample times and prediction horizons along with the weights of the output states. Increased robustness was promoted by an increase in control input rate weights, such that larger control moves were penalized stricter. The lateral and longitudinal MPC results showed that it is possible to stabilize the attitude by rotating the robot legs. Additionally, both MPC schemes were able to control their attitude angle to zero for initial attitude angle rotations between $-30°$ and $30°$, with rise times below 0.75 seconds and settling times below 3 seconds. Both MPC schemes showed clear signs of robustness with initial conditions of zero on angular velocities. Angular velocities and control inputs converged to zero, while the attitude angles closely tracked their reference. However, the MPC schemes showed signs of struggle with initial angular velocities set on the actuator angular velocities. Actuator angular velocities were not able to converge to zero, resulting in growing actuator angles.

With rapid progress in the field of legged robots and robots than can jump and land, there will be a growing need for attitude stabilization of falling objects. Mutually for most of these robots is their weight limitations and short fall time, such that the traditional attitude stabilizer actuators like thrusters and flaps are ineffectual. This is the main reason why the methods and results presented in this thesis are so promising. The individual results themselves are not yet sufficient for real-life employment, but it proves that it is possible to solve this control problem with this method.

During the thesis project, several choices could have been made to change the outcome of the results. Some of these choices would be natural to continue working on for further work. If the process were to be continued, an option would be to choose a nonlinear MPC instead of the explicit MPC. It became clear that the relationship between linearized and dynamical models was weak. It points towards a poor linearization process, but it also promotes the idea of switching to nonlinear MPC instead.

Additionally, it would have been beneficial to experiment more with the simulations to evaluate how robust the closed-loop systems were. This is an important aspect if the MPC controllers ever were planned to be used in real-life. However, there was not enough time to prioritize this along with the process of determining how far the attitude could be controlled. Additionally, the yaw angle ψ should be included, such that a third MPC scheme is implemented for the dynamics in the $x-y$-frame. Lastly, the situation evaluated throughout the project is an ideal situation without noise, disturbances, air, or wind. This is far from the case in real life, and a natural next step would be first to introduce disturbances and noise to the simulation and alter the tuning parameters after that. It would then be feasible to test the MPC schemes on a real-life robotic system and make further alterations.

Bibliography

[1] F. A. ADMINISTRATION, *Pilot's Handbook of Aeronautical Knowledge - Chapter 6 - Flight Control*. [Online]. Available: https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/phak/media/08_phak_ch6.pdf.

[2] T.-D. Chu and C.-K. Chen, 'Design and implementation of model predictive control for a gyroscopic inverted pendulum,' 2017.

[3] Ø. Hegrenæs, J. T. Gravdahl and P. Tøndel, 'Attitude control by means of explicit model predictive control, via multi-parametric quadratic programming,'

[4] S. J. Qin and T. A. Badgwell, 'A survey of industrial model predictive control technology,' 2001.

[5] B. University, *5 dynamics of rigid bodies*. [Online]. Available: https://www.brown.edu/Departments/Engineering/Courses/En4/notes_old/RigidKinematics/rigkin.htm.

[6] O. Egeland and J. T. Gravdahl, *Modeling and Simulation for Automatic Control*. Tapir Trykkeri, Trondheim, 2002.

[7] D. Morin and H. University, *Chapter 6 - the lagrangian method*. [Online]. Available: https://scholar.harvard.edu/files/david-morin/files/cmchap6.pdf.

[8] K. Academy, *What is kinetic energy?* [Online]. Available: https://www.khanacademy.org/science/physics/work-and-energy/work-and-energy-tutorial/a/what-is-kinetic-energy.

[9] R. Tedrake and MIT, *Underactuated Robotics - Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2022. [Online]. Available: http://underactuated.mit.edu/multibody.html#Craig89.

[10] G. S. University, *Rotational kinetic energy*. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/rke.html.

[11] K. J. Åstrøm, *Control System Designs Ch. 6 - PID Control*. [Online]. Available: https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf.

[12] S. D. E, E. T. F, M. D. A and D. F. J, *Process Dynamics and Control Ch. 20 - Model Predictive Control*. John Wiley and Sons Inc, 2011. [Online]. Available: https://folk.ntnu.no/skoge/vgprosessregulering/papers-pensum/seborg-c20ModelPredictiveControl.pdf.

[13] MatLab, *Mpc - model predictive controller*. [Online]. Available: https://se.mathworks.com/help/mpc/ref/mpc.html.

[14] M. Morari, C. E. Garcia and D. M. Prett, 'Model predictive control: Theory and practice,' 1998. [Online]. Available: https://reader.elsevier.com/reader/sd/pii/B9780080357355500061?token=2DC09ACE1626570DF7F45D3FECF9853481D78C8 originRegion=eu-west-1&originCreation=20220421075212.

[15] Gorinevsky and Stanford, *Lecture 14 - model predictive control part 1: The concept*. [Online]. Available: https://web.stanford.edu/class/archive/ee/ee392m/ee392m.1056/Lecture14_MPC.pdf.

[16] *Right hand rule*. [Online]. Available: https://www.real-world-physics-problems.com/right-hand-rule.html.

[17] T. L. Paez, 'Introduction to model validation,'

[18] T. A. Johansen, 'Introduction to nonlinear model predictive control and moving horizon estimation,' [Online]. Available: https://folk.ntnu.no/torarnj/nonlinear.pdf.

[19] M. Kamel, M. Burri and R. Siegwart, 'Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles,' 2017.

[20] G. Shah and S. Engell, 'Tuning mpc for desired closed-loop performance for mimo systems,'

[21] A. Bemporad, N. L. Ricker and M. Morar, *Model predictive control toolbox™ user's guide*. [Online]. Available: https://se.mathworks.com/help/pdf_doc/mpc/mpc_ug.pdf.

[22] Brilliant.org, *Small-angle approximation*. [Online]. Available: https://brilliant.org/wiki/small-angle-approximation/.

[23] D. Sotelo, A. Favela-Contreras, V. V. Kalashnikov and C. Sotelo1, 'Model predictive control with a relaxed cost function for constrained linear systems,' 2020.

Nora Graffer

Attitude Stabilization of a Quadruped Robotic System during Free-Flight