

Simulation Based Validation of MPC for Two wheeled Jumping Robot



Yusuf Abduraman Edo

A Thesis Submitted to

The Department of Electrical Power and Control Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Master of Science
Degree in Electrical Power and Control Engineering (Control Engineering)

Office of Graduate Studies

Adama Science and Technology University

Adama, Ethiopia

December, 2022

Simulation Based Validation of MPC for Two wheeled Jumping Robot

Yusuf Abduraman Edo

Advisor: prof. S.Kim

A Thesis Submitted to
The Department of Electrical Power and Control Engineering
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Master of Science
Degree in Electrical Power and Control Engineering (Control Engineering)

Office of Graduate Studies
Adama Science and Technology University

Adama, Ethiopia

December, 2022

DECLARATION

I hereby declare that this thesis entitled “Simulation Based Validation of MPC for Two wheeled Jumping Robot“ is my original work. That is, has not been submitted for the award academic degree, diploma or certificate in any university. All sources of material used for this thesis have been duly acknowledged through appropriate citation.

Yusuf Abduraman Edo

Name of student

Signature

Date

RECOMMENDATION

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled “Simulation Based Validation of MPC for Two wheeled Jumping Robot“ prepared under my guidance by Yusuf Abduraman Edo submitted in partial fulfillment of the requirements for the degree of Mater’s of Science in Electrical Power and Control Engineering (Control Engineering). Therefore, I recommend the submission of revised version of the thesis to the department following the applicable procedures.

_____	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-advisor	Signature	Date

APPROVAL SHEET

We, the advisors of the thesis entitled “Simulation Based Validation of MPC for Two wheeled Jumping Robot” and developed by Yusuf Abduraman Edo hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

<u>Prof. S. Kim</u> _____	 _____	 _____
Advisor	Signature	Date
_____	_____	_____

Co-advisor	Signature	Date
------------	-----------	------

We, the undersigned, members of the Board of Examiners of the thesis by Yusuf Abduraman Edo have read and evaluated the thesis entitled “Simulation Based Validation of MPC for Two wheeled Jumping Robot” and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Electrical Power and Control Engineering (Control Engineering).

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____

_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____

External Examiner Signature Date

Final approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____

_____	_____	_____
School Dean	Signature	Date
_____	_____	_____

Office of Postgraduate Studies, Dean Signature Date

ACKNOWLEDGEMENT

First of all, I want to thank my almighty Allah (sw) for keeping me until I finished my MSc Thesis. I have been very blessed to start my thesis work under the supervision and guidance of *Prof. S. kim* He introduced me to the field of Soft tuning techniques, educated me with the methods and principles of research, and guided me through the details of MPC controllers. Working with him, a person of values has been a rewarding experience. I am highly indebted and express my deep sense of gratitude for his valuable guidance, constant inspiration and motivation with enormous moral support during difficult phase to complete the work. I acknowledge his contributions and appreciate the efforts put by him for helping me complete this project. At this moment I would also like to express my gratitude for my friends for helping me out in my difficulty during my project. They have always helped me in every-way they can during my experimental phase of the work.

TABLE OF CONTENTS

DECLARATION	i
RECOMMENDATION	ii
APPROVAL SHEET	iii
ACKNOWLEDGEMENT	iv
LIST OF TABLE	viii
LIST OF FIGURES	ix
LIST OF ABBREVIATION	x
ABSTRACT.....	xi
CHAPTER ONE	1
INTRODUCTION	1
1.1 Background of Thesis	1
1.1.1 Leg-and-Foot-Based Robot.....	1
1.1.2 Wheeled Based Robot.....	2
1.1.3 Prismatic Joint.....	4
1.1.4 Model Predictive Control (MPC).....	5
1.2 Statement of the Problem.....	6
1.3 Objective of Thesis	6
1.3.1 General Objective	6
1.3.2 Specific Objectives	6
1.4 Significance of the Thesis.....	7
1.5 Scope of the Thesis	7
1.6 Limitation of the Thesis	7
CHAPTER TWO	8
LITRATURE REVIEW.....	8
CHAPTER THREE	14
METHODOLOGY	14
3.1 Introduction.....	14
3.2 Dynamic Modeling of System	14
3.2.1 Modeling for Swing-up and Balance Phase 1	15

3.2.2	Modeling of Variable Length Wheel Inverted Pendulum phase.....	20
CHAPTER FOUR.....		31
MODEL PREDICTIVE CONTROL		31
4.1	Introduction.....	31
4.1.1	Principle of Model Predictive Control	32
4.1.2	Mathematical Formulation of MPC	33
4.1.3	Solving MPC problem	34
4.1.4	Nonlinear Programming.....	34
4.1.5	Transcription Method.....	35
4.1.6	Multiple Shooting	35
4.2	System Control and Problem Formulation.....	38
4.2.1	Solver Decision Variables.....	38
4.2.2	Specify Initial Guesses.....	39
4.2.3	Configuration of fmincon Options	39
4.2.4	Problem Formulation Planning	39
4.2.5	The Matlab Code.....	43
4.3	Proportional Derivative Controller	46
4.3.1	Proportional and Derivative Controllers	47
CHAPTER FIVE		49
RESULT AND ANALYSIS		49
5.1	Introduction.....	49
5.2	Closed loop Response of the System	49
5.2.1	Swing up and Balance Phase	49
5.2.2	Closed loop response for the second (VI-WIP) phase	50
5.3	Proportional Derivatives versus Model Predictive Control	53
5.3.1	Simplified Model	53
5.3.2	Comparison of PD and MPC	56
CHAPTER SIX.....		59
RECOMMENDATION AND CONCLUSION		59
6.1	Recommendation	59

6.2	Conclusion	59
	RESEARCH FUND ACKNOWLEDGMENT	61
	REFERENCES	62
	<i>Appendix: MATLAB Code</i>	65
	<i>Appendix I: main for phase1</i>	65
	Appendix II: main for phase2	68
	<i>Dynamic model function</i>	71

LIST OF TABLE

Table 3.1: System parameters and its value.....	30
Table 5.1 Control performance PD vs MPC.....	58

LIST OF FIGURES

Figure 1.1 Human leg based robot	2
Figure 1.2 Different types of wheeled	3
Figure 1.3: Proposed Variable-Length Prismatic joint	4
Figure 2.1 The wheeled jumping robot jumps across a gap	8
Figure 2.2: Current prototype of the Ascento robot.....	9
Figure 2.3: AYNmal quadrupedal robot	10
Figure 2.4: Skaterbot Robotic with arbitrary arrangements of legs and wheels to locomote.....	11
Figure 2.5: Handle and sand Flea Robot.....	12
Figure 1.1: The braking car model.....	15
Figure 3.2: VL-WIP proposed template.....	21
Figure 4.1: Principle of model predictive control.....	32
Figure 4.2: Multiple shooting method	36
Figure 4.3 Fourth order Runge-Kutta method	37
Figure 4.4: Jumping over a gap.....	40
Figure 4.5: model predictive control pipeline.....	42
Figure 4.6: PD controller - block diagram	46
Figure 5.1: States response during the jumping over a gap phase	51
Figure 5.2: Controls inputs response during the jumping over a gap phase	51
Figure 5.3: Two-stage controller switch from the swing-up and balance phase model.	53
Figure 5.4: VI-WIP on the ground.....	54
Figure 5.5: State variables response – MPC	57
Figure 5.6: Closed loop response for PD controller	57

LIST OF ABBREVIATION

DOF	Degree of Freedom
EoM	Equation of Motion
FMINCON	Find minimum of constrained nonlinear multivariable function
LIP	Spring Loaded Inverted Pendulum MRP Mobile Robot Platform
LQR	Linear Quadratic Regulator
MATLAB	Matrix Laboratory
MPC	Model predictive control
NLP	Nonlinear Program
NMPC	Nonlinear Model predictive control
OCP	Optimal Control Problem
PID	Proportional Integral Derivative
PID	Proportional Integral Derivative
SQP	Sequential Quadratic Program
TWJR	Two Wheel Jumped Robot TWR Two Wheel Robot
VI-WIP	Variable Length wheeled invert pendulum

ABSTRACT

This thesis is presents the study a two wheel jumped robot with a nonlinear model predictive controller. In this research the propose template is a variable length wheeled inverted pendulum template model for the class of such two wheeled jumping robots. The robot havevariable length due to prismatic joint extension. The robot can create velocity through this extension joint movement to jump over obstacles and swing up to the equilibrium position after losing equilibrium. This model can be considered as the simplest two wheeled legged inverted pendulum system. Basically this robot have contained two phase which are swing up and balance phase and the second is Variable length wheeled Inverted pendulum phase. Initially the robot in car mode in the first phase the robot perform swing-up and balancethe robot mode changed from car mode to two wheeled invert pendulum mode second phase. In the second phase the robot perform three task pre-takeoff, flight and landing tasks. The analytical modeling of the system dynamics are presented thatapplied to thenonlinear model predictive controller (NMPC) to help robotto complete its two phase mission. Additionally, I observed the model's behavior and performed highly dynamic motions like swinging up, balancing, and jumping. These movements are also found through optimization using fundamental principles. And I can able to show the task of the robot such as swing up, balancing and jumping motion in result section. Additionally the performanceNMPCcontroller and proportional derivative (PD) controller are compare and evaluatedthrough control characteristics performance based and also can get better result.

Key words: - Prismatic joint, NMPC, MPC control, PD control, VL-WIP

CHAPTER ONE

INTRODUCTION

1.1 Background of Thesis

Since the Bronze Age, humans have used wheels. Wheeled vehicles' simplicity and efficiency have made it possible to move products over long distances quickly and reliably. However, they need prepared ground, like roads or railways. The growth of inspection robotics in research and industry has been accelerated by the appearance of capable mobile robots over the past ten years (M. Hutter R. D., 2017), (R. Siegwart and I. R. Nourbakhsh, 2004). While drones as well as other flying robot systems currently exhibit excellent maneuverability (M. Kamel, 2018.), these robots are constrained in their payload and flight time. Ground robots that can travel fast and overcome indoor barriers are still a research area and often lack speed or maneuverability. The movement of ground robots can be loosely divided into two categories: either a leg-orleg-and-foot-based approach.

1.1.1 Leg-and-Foot-Based Robot

A robot leg, also known as a robotic leg, is a mechanical limb that can carry out the same tasks as a human or animal leg. Usually, the robotic leg is programmed to carry out tasks that a human leg would carry out. A prosthetic leg and a robotic leg are comparable. A robotic leg, however, can be controlled mechanically or electronically. The nerves that previously controlled parts of the person's lower-leg muscles must be redirected in order to make the thigh muscles contract in order for the robotic leg to mimic human leg activities. The electrical pulses produced by the existing thigh muscle and by re-innervated muscle contraction are both measured by sensors built into the robotic leg.(Hayden, 30 September 2013)



Figure 1.1 Human leg based robot (Hayden, 30 September 2013)

While some indoor walking robots operate admirably, leaping over barriers and navigating gaps like stairs and slick surfaces, they usually still take a significant amount of time to execute these complex movements and with highly complex body structure. Because of this complexity, it is much expensive to controlling the robot in terms of processing and energy use. (M. Hutter C. G., 2017.)

1.1.2 Wheeled Based Robot

Robots called "wheeled robots" move across the ground by employing motorized wheels as wheels. Using wheels instead of treads or legs makes this design simpler and makes it simpler to design, manufacture, and program for mobility on flat, moderately difficult terrain. Additionally, they can be controlled better than other kinds of robots. Wheeled robots have the drawback of having difficulty navigating over obstacles like rocky terrain, sharp declines, or areas with low friction. The consumer market is where wheeled robots are most popular; their differential steering offers low cost and simplicity. Three wheels are all that are needed for static and dynamic balancing on robots of any size. When the terrain is not level, additional mechanisms will be needed to maintain all of the wheels on the ground. Additional wheels can help with balancing (wikibooks, 2022). There are different types of wheeled-based robot depending on their number of wheels. Such as two wheels, three wheels, four wheels, and so on.

On the other hand, leg-based robots with rotating components, such as wheels, are more suited for flat terrain since they can move swiftly, smoothly, and easily. However, they frequently struggle over rocky terrain, particularly when there are obstructions bigger than the radius of their wheel. Systems with continuous rails are an exception. These can easily navigate uneven terrain and small obstacles, but their sliding makes them clumsy and ineffective for turning maneuvers. (Online, n.d.).

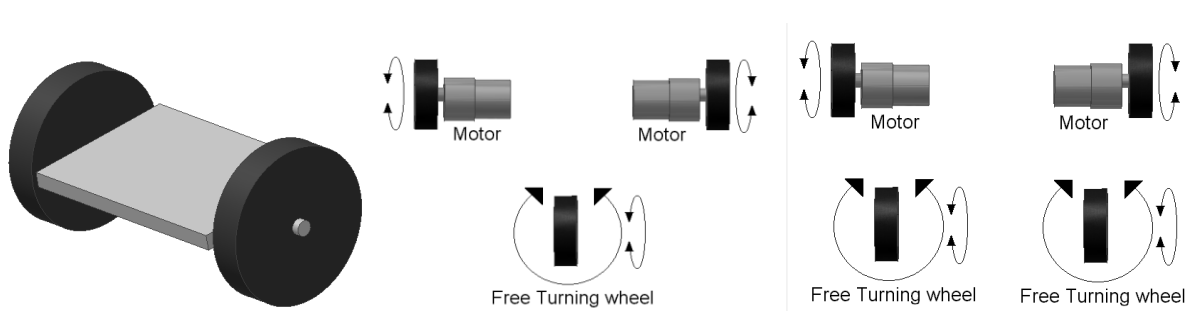


Figure 1.2 Different types of wheeled(Online, n.d.)

Robots with wheels and legs can combine the greatest features of both types, allowing them to move quickly across rough terrain on wheels and over legs. A model of the system that can manage both the step contact changes typical of legged robot step up and hopping motions and the rotating contact movement of the wheels is required by the integrated design. But instead of adding the rotating contact to a legged robot model to make it more complicated, due to this I interested to concentrate on creating a simple and easy model with just one extension joint to simulate the application of leg. This gives us a good tool for mimicking the robot behavior than some of the much intricate models. The whole body trajectory optimization challenging since the majority of wheeled leg mobile robots have a large degree of freedom in nonlinear systems. In a different study, the planning of kinematic motion in which the system is managed as a moving car and its legs serve as a suspension was examined. These methods by definition do not take the system dynamics into account.

The Spring-loaded Inverted Pendulum (SLIP) is a well-known framework model for legged robots in the wheeled leg-based robot. This template has been served to design for high speed moving and hopping (jumping)(Wensing, May 2014). For wheel balancing application frequently have been used, wheeled inverted pendulum structure as sample model (WIP) (R. P. M. Chan, April 2013.). Wheel-legged systems can use template models based on the inverted

pendulum with spring loaded and the inverted pendulum with wheels. However, they have each been classified as either the only leg or the only wheel. The wheel-legged robot template model that was suggested for this study. Two wheels front and back are joined by a supposed extension joint (prismatic) in this system.

1.1.3 Prismatic Joint

A prismatic joint is a junction that connects two things and permits motion along a single axis. It is impossible to move in a direction perpendicular to this axis or rotate around any other axis. The joint has one degree of freedom as a result (1-DOF). Only linear motion in one axis is possible with a prismatic joint. This joint is often called a slider for example a “crank-slider linkage.” An idealized technique to represent how motions are restrained within machines, this is a particular kind of kinematic pair. A prismatic joint can be physically represented in various ways, such as slide ways. To enable linear motion while avoiding rotation about the axis of movement, rectangular cross-sections can be utilized in combination with plain or rolling bearings. Prismatic joints are often used to model hydraulic and pneumatic cylinders, though they may also allow rotation about their axis. Higher pairs and lower pairs are two categories for kinematic pairs. Higher pairs involve point or line contact, such as a ball rolling over a surface or a roller. Lower pairs can be visualized as surface contact. (A fastenengineering, 2022) (See Figure 3).

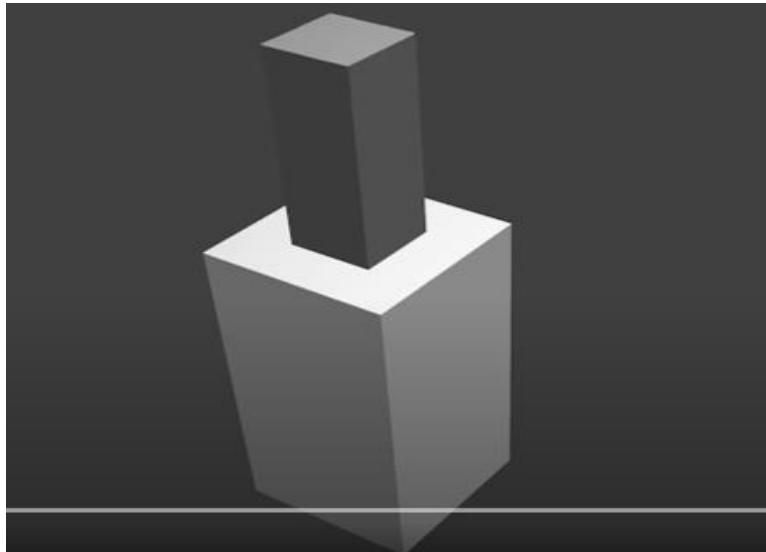


Figure 1.3: Proposed Variable-Length Prismatic joint (A fastenengineering, 2022)

As a result, (Traiko Dinev, 2020)it is important to create the Variable-Length Wheeled Inverted Pendulum (VL-WIP) template dynamic model for two wheeled-legged systems and put it into practice by using a motion planner and the direct transcription method to create a Model Predictive Controller (MPC).

1.1.4 Model Predictive Control (MPC)

Model Predictive controller is a control approach that was first used in the process sector in the 1970s. It was a well-known computationally expensive control technique that was mostly applied to slow operations. In recent years, MPC has gained popularity as a solution for applications requiring faster sample times because of improvements in computing power and effective new solvers. The robotics sector is one of the novel domains where MPC is now being tested.

In this research the two wheeled jumped robot should have to perform swing up and balancing, drive forward and jumping tasks in different phases. A control system must be set up before a driver assistance system may be offered. In this thesis, the maneuvers problem of this system is researched using MPC. Recently this controller techniques has significantly increased use for a wide range of new applications, including the robotics sector. The ever-increasing processing capability of contemporary computers can be blamed for the recent surge in interest in using model predictive control for new function. This makes it possible to solve complicated optimization issues that were previously thought to be intractable.

Model Predictive Controller (MPC) is an advance controller that enables present timeslot optimization while taking into account future timeslot. This is achieved by implementing only the present time slot while optimizing along a finite time horizon. And also show that the robot can perform actions like swing up, balancing jumping over the gap that are only possible when the wheels and base work together by using prismatic joint model. From the dynamics model the ability to find dynamic motion and to exploit the predictive power of the controller stems also engaged.

1.2 Statement of the Problem

Most researchers have completed their work and proven successful in stabilizing the system in relation to two wheel robots. Additionally, these two wheeled systems able to navigate on smooth surface; they can able to overcome obstacle larger than the wheel radius. However, these two wheeled developed with complex mechanical structure and not future predictive controller. To improve those problem, author (Traiko Dinev, 2020)develop the variable length wheeled inverted Pendulum simplified model using prismatic joint and MPC controller with additional knitro solver packages to Matlab Tool which not found free. In order to validate the simulation result of (Traiko Dinev, 2020) I use nonlinear model predictive control (nlmpc) solver which Matlab nonlinear system solver tool. This system controlled stabilizing by nonlinear model predictive controller (NMPC).

1.3 Objective of Thesis

1.3.1 General Objective

General objective of this thesis is to Simulation based validation Model Predictive controller (MPC) for Two Wheel Jumped Robot (TWJR)

1.3.2 Specific Objectives

The specific objectives of the thesis are:

- To study the mathematical model of the two wheel jumped robot system for control design
- To design nonlinear model predictive controller (NMPC) using nmpc solver for the system.
- studying the transcription method and incorporating it into a controller (MPC)
- To demonstrate that the robot can carry out actions like jumping using the template model
- Compare the characteristics of the response performance of the nonlinear model predictive controller (NMPC)with proportional derivatives (PD) controller.

- To simulate and analyze the controller in MATLAB

1.4 Significance of the Thesis

Now days, this kind of robotic systems have wide range of research and application fields. In the study of control systems, the two-wheeled robot (TWR) is one of the fascinating plants. The robot is quite inexpensive, but it presents some challenging control system study difficulties. In Robotics, industries most maneuver robots walking slowly to move one place to another but the Two Wheel Jumping Robot (TWJR) helps maneuver Robots to high maneuver on the ground and overcome the obstacle through jumping. Due to this reason developing this control mechanism that improve the stability of the system using model predictive controller have play vita role in robotics industries. Due to those improvement this thesis can be also input in mobile robotics Industries.

1.5 Scope of the Thesis

This thesis is organize from various section as follows; The First section describes introduction of the thesis that mention background of the thesis, statement of problem, objective of the thesis which is contain the general and specific objective of the thesis, second section is explained literature survey regarding with thesis, The third section is mentioned the methodology and mathematical modeling of two wheel jumped system with proposed variable length wheeled inverted pendulum, the fourth section explained designing the optimal control of Two Wheel Jumped Robot (TWJR) system, model predictive controller (MPC) to handle error. The Fifth Section contain simulation and result analysis work using MATLAB/Simulink and The last section of this thesis mention the conclusion and recommendation.

1.6 Limitation of the Thesis

The thesis will be limited to simulation of the system using MATLAB Simulink by showing how the different components of the system interact with each other. This is because implementation of the real system is difficult because of the component are not available in our country and along with it the additional time and cost incurred in making the actual implementation.

CHAPTER TWO

LITRATURE REVIEW

In the robotic research industries most researchers and engineers have performed several studies and application fields two wheeled legged mobile robots. Using kinematic motion planning the majority of two wheel robots that can stand on their own and move on a smooth surface are built, whereby the robot is steered like a moving vehicle and suspended by its wheels. The challenge of this system is to overcome obstacles. The application range of wheeled robots must go beyond flat areas, hence they must have all-terrain capabilities. In the robotic industries four six-wheeled legs which have been well known for years which are cable to overcome obstacle. These days, two-legged (bipedal) robots are more and more common due to improved gait algorithms.

Authors of (Traiko Dinev, 2020) design two wheels jumped robot using Variable length wheeled inverted pendulum templates model and they use model predictive controller (MPC) and knitro solver tool packages. Artely knitro is commercial software packages for solving large scale mathematical optimization problems. Instead of using knitro solver I use Model predictive control (MPC) with nonlinear model predictive controller (nmpc) which is nonlinear system matlab tool solver. The nmpc is not commercial packages software solver like knitro, it is found in matlab MPC nonlinear solver tools.



Figure 2.1 Two wheeled jumping robot (Traiko Dinev, 2020)

The authors of (V. Klemm, 2019) design the two-legged (bipedal) robots and use a LQR controller and use Kalman filter to estimate state for balancing the Ascento robot. The nonlinear dynamics model of system is linearized at fixed equilibrium point. Although it offers a good approximation, this constrains boundary of motion is around the fixed point.



Figure 2.2: Current prototype of the Ascento robot (V. Klemm, 2019).

(M. Bjelonic, Jan. 2020) Developed ANYmal a quadrupedal robot for autonomous movement in rough terrain. Unique compliant and precisely torque controlled actuators move the system, which is capable of dynamic running and high-mobile climbing. A framework for online trajectory optimization for a quadrupedal robot that can use hybrid walking-driving locomotion techniques was described by the study's authors. By segmenting the optimize issue into a wheeled and base trajectory motion planning. A centroidal dynamics model has been used. One stage of a two-stage model predictive controller is for the wheels and the other is for the center of mass (CoM). Because of this, this method does not fully utilize the coupled synergy between wheel and body dynamics.



Figure 2.3: AYNmal quadrupedal robot (M. Bjelonic, Jan. 2020)

Skaterbots (M. Geilinger, July 2018.), this system employs a two stage controller and a centroidal dynamic model similar to ANYmal robot. A controller for the wheels and one for the center of mass (CoM). Because of this, this method does not fully use the coupled synergy between wheel and body motion. The complete optimization is solved. But since the resultant nonlinear program is so complicated, the limitations are stated via Newton method. In contrast to Skaterbots, I can able to tackle the nonlinear challenge thanks to the proposed model of a wheeled inverted pendulum with a changeable length. In contrast to Skaterbots' proportional derivatives (PD) control, I also employ a Model Predictive Controller (MPC) method.



Figure 2.4: Skaterbot Robotic with legs and wheels arrangements to locomote (M. Geilinger, July 2018.)

Flea sand and Handle exhibit highly dynamic motions using Boston Dynamics. Flea sand robot is capable of jumping to a 10 meter(around 30 feet) height into air using combustive propulsion [(A. Saunders, 2012.)]. Handle is a mobile robot that combines the effectiveness of wheels with the ability to navigate uneven terrain on its legs. It has a manipulator arm that can pick up bulky boxes and is intended for material-handling operations. Additionally, it has a swinging "tail" that helps in its dynamic movement and balancing in small areas. Handle robot moves forward in a jumping motion while overcoming rocky terrain (Dynamics, 2019)The techniques utilized to manage these systems, however, have not been made public.

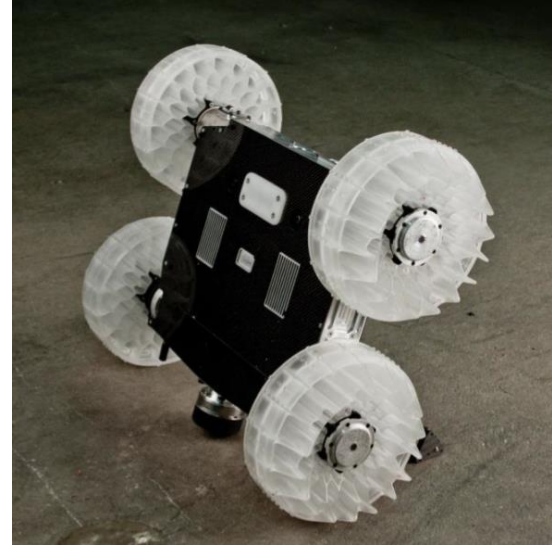


Figure 2.5: Handle and sand Flea Robot (A. Saunders, 2012.).

By linearizing the nonlinear dynamic equation and estimating the states, (Nur Uddin, November 2017.)Presents linear Quadratic linear (LQR) controller for stabilizing balancing two wheeled robots (TWR).A control system is necessary to stabilize a two wheel robot since it is an unstable system. Full-state feedback control called LQR determines the control gain by minimizing a performance index. When using LQR to stabilize a TWR, pitching angle and pitching rate angle measurement must be used as feedback. The Luenberger observer state estimator is used to predict the pitching angle based on the measurement of the pitching rate angle. The linearizing dynamic model of any system has an effect on the sensitivity to sounds, and this two-wheeled robot can only travel on smooth surfaces.

Another robotic vehicle idea, known as the robotic mobile platform (RMP),also created by Segway. The Segway RMP outperforms equivalent current platforms in terms of speed, cost, and agility. It can carry a heavier cargo while being tough, having a short footprint, and having a zero turning radius.

One of the powerful methods in the linear control theory is linear quadratic regulator (LQR). The LQR is used to minimize a performance index, which is represented by a quadratic performance index, to determine the control gain of the states feedback control law. Studies of using LQR to stabilize a TWR have been published. Control design for a TWR has been presented in employing a linear quadratic regulator (M. O. Asali, 2017). The study findings reveal that both

PID and LQR are capable of stabilizing the TWR, although the LQR controller performs better than the PID controller. A comparison of employing PID control and LQR control to stabilize TWR has been published in (A. Nasir, 2010).

CHAPTER THREE

METHODOLOGY

3.1 Introduction

It is necessary to provide a mathematical description of the problem before design a controller for the system. A mathematical description must first be constructed to simplify the procedure before a controller for the system can be developed. The dynamic model of two wheel-jumped robot has contained two phase, which are swing-up and balancing phase and variable length wheeled Invert pendulum phase. Initially the robot in car mode stand horizontally then the robot swing-up which change its mode from car mode to two wheeled mode and balance its self. These tasks are performed in the swing-up and balancing phase. The next phase is Variable-Length wheeled Invert pendulum phase the robot show when jumping over the gap. The dynamic model of this robot system is nonlinear equation. An adequate model of the proposed nonlinear system is developed using the lagrangian method and presented as an explicit Equation of Motion ((EOM).) in order to develop and design appropriate controllers. The process control industry frequently employs the nonlinear model predictive control (NMPC) approach, particularly for systems with nonlinear dynamics models. The mathematical model of the system affects the performance of the controller in MPC approach. Model Predictive Controller (MPC) maintains optimality while maintaining incoming future timeslots by taking present time into account. This is an iterative and finite time horizon optimization method. NMPC successfully predicts the nonlinear plant's nonlinear model's future states. The MPC prediction is practically sensitive to prediction mistakes. For meager nonlinear systems, this is acceptable.

3.2 Dynamic Modeling of System

Two wheel jumped robot is essentially a car with four wheels, which is stand horizontally on the ground with four wheels. The robot's prismatic joint, which allows adjusting the body length, and the fact that the front wheels are motorized make it unique. The plant model developed in this chapter describes the physical behavior of the two-wheeled jumped robot over a specific operating range. This system has two modeling and simulation phases, Swing-up and Balance Phase and variable-length wheeled inverted pendulum phase

3.2.1 Modeling for Swing-up and Balance Phase 1

In the starting position, the robot is placed horizontally with respect to the floor, with the entire wheel touching the ground. The robot able to swing-up starting position, by braking and accumulating kinetic energy. To accumulating kinetic energy the starting or initial velocity must be different from zero. During this phase of the assumption simulation or the goal of the robot is to slow-down and swing-up by applying torque to the front wheels. The model developed based on the following model figure 3.1.

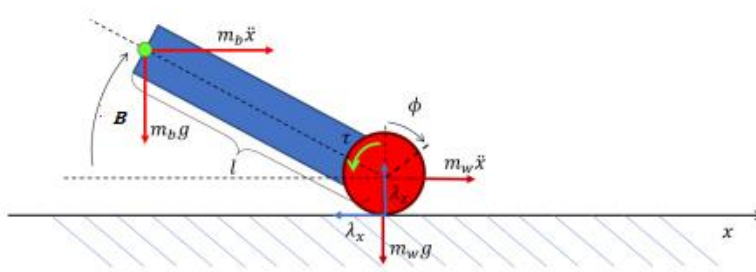


Figure 3.1: The braking car model

Where, m_w is mass of wheel, m_p is represent point mass (body mas), ϕ is represent the wheel angles along its z -axis and β , represent the wheel angles along its x -axis.

The prismatic joint is the linkage of front wheel and back wheel which is capable to increase and reduce its length. However, in this swing up phase the length of pendulum l constraint as constant. So that, the force acting on prismatic joint is ignored that is used to extend and retract the prismatic joint. Therefore, the model simplifies. For this reason, we have to consider as input only the torque acting on the wheel.

In this phase state variable of the system, $x = [q^T \dot{q}^T]$, where q is represent general position coordinate of the robot and \dot{q} is general velocity of the system.

The robot generalize position $q = [\phi \ \beta]^T$ and generalize velocity $\dot{q} = \frac{dq}{dt} = [\dot{\phi} \ \dot{\beta}]^T$, as you see in the above Figure: 3.1.

A controls input for this system for this phase is $u = [\tau]$ where τ is actuation torque that applied by wheels. By analyzing the torques applied to the car then we can possibly derive its dynamic model using Lagrangian Method of first phase:

The total number of states and inputs for this phase is then given by the following:

$$x = \begin{bmatrix} \phi \\ \beta \\ \dot{\phi} \\ \dot{\beta} \end{bmatrix}, u = \tau \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Where, $\dot{\phi}$, is the angular speed of the robot from the inertia *vertical* coordinates, $\dot{\beta}$, is the angular speed of robot body from the inertia *horizontal* coordinates and τ , is the torque applied to the system or robot.

To drive the robot equation of motion we have famous Lagrange equations. Lagrange equation states that:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = f_{ext} \dots \dots \dots (3.1)$$

Where L is Lagrangian of any system and f_{ext} is force or energy that applied to the system. Lagrangian of a system is the difference of total kinetic energy T of the system and total potential energy U of the system.

$$L = T - U \dots \dots \dots (3.2)$$

Based on the previous figure 3.2, it is now feasible to separately drive the system's potential and kinetic energy.

Kinetic Energy of System:

Every connection in the system will typically have kinetic energy in both a rotational and a translational component. In the back wheels kinetic energy is present in both component which are rotational and translational. The kinetic energy in the rotational is expressed in term of $I_w \dot{\phi}^2$, where $I_w = m_w R_w^2$ is the moment of the wheel, where m_w mass of wheel. Since that a body mass has no moment inertia, due to this it possesses only kinetic energy $m_p v_p^T v_p$, which equals the product of the mass of the body's velocity v_p and its mass m_p .

The movement of the point mass or the pendulum is compute by considering the movement the end of pendulum or front wheel. Under this phase the movement across z-axis is consider as zero because there is no up and down movement across this axis.

$$\text{So, } T_r = \frac{1}{2} I_w \dot{\phi}^2 \text{ is Rotational kinetic energy and}$$

$$T_t = \frac{1}{2} (m_w \dot{x}^2 + m_p \dot{x}_p^2), \text{ is Translational kinetic energy.}$$

Therefore, the kinetic energy (T) of system is the sum of translational and rotational kinetic energy. So, kinetic energy of the system become,

$$T = T_r + T_t$$

$$T = \frac{1}{2} (I_w \dot{\phi}^2 + m_w \dot{x}^2 + m_p \dot{x}_p^2) \dots \dots \dots (3.3)$$

Where,

$$x_p = x + l \sin \beta$$

$$z_p = z + l \cos \beta$$

$$v_p = \dot{x}_p = \dot{x} + \dot{l} \sin \beta + l \dot{\beta} \cos \beta \dots \dots \dots (3.4)$$

Potential Energy of the system

Due to the gravitational forces acting just on the point mass the potential energy will be formed. Consequently, the robot potential energy equation in this phase becomes,

$$U = m_p g z_p \dots \dots \dots (3.5)$$

Therefore, after substitute equation (3.3) and (3.5) in the (3.2) then Lagrangian Equation will become,

$$L = \frac{1}{2} (I_w \dot{\phi}^2 + m_w \dot{x}^2 + m_p \dot{x}_p^2 - m_p g z_p) \dots \dots \dots (3.6)$$

By inputting Equations (3.6) into Equation (3.1) we can drive the equations of motion of the system. In this phase, aim is to swing up the robot by braking and accumulating kinetic energy. In this phase, it has a starting velocity different from zero or apply torque to the back wheel that will be used to accumulating kinetic energy and braking by applying counter torque to the wheel.

Accumulating kinetic energy:

From lagrange method, substitute equation (3.6) and (3.3) into equation (3.1) in terms of ϕ is the wheel angle along its coordinates and ϕ is the rotational of body from z - coordinates. It is easy to understand that, if the car does not slide, we can also determine the coordinate x as $x = \phi R_w$.

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} = \tau, \text{ where } x = \phi R_w \text{ this leads to } \dot{x} = \dot{\phi} R_w$$

$$\frac{\partial L}{\partial \dot{\phi}} = I_w \dot{\phi} + m_t \dot{\phi} R_w,$$

$$\frac{d}{dx} \frac{\partial L}{\partial \dot{\phi}} = I_w \ddot{\phi} + m_t \ddot{\phi} R_w, \text{ and } \frac{\partial L}{\partial \phi} = 0$$

$$\text{Therefore, } I_w \dot{\phi} + m_t \dot{\phi} R_w = \tau$$

Consequently, the body's angular acceleration relative to the z-coordinates of inertia will become,

$$\ddot{\phi} = \frac{\tau}{I_w + m_t R_w} \dots\dots\dots(3.7)$$

Braking:

We can be braking the car by applying counter torque to the wheel. Initially the robot stand horizontally on the ground and Interaction force between torque due to accumulating energy and counter torque will be used to swing up the robot. Substitute equation (3.6) and (3.3) into equation (3.1) in terms of β is wheel angles along its coordinate and $\dot{\beta}$ is the rotational of the body from x- coordinates.

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\beta}} - \frac{\partial L}{\partial \beta} = -\tau, \text{ where } x = \phi R_w \text{ this means } \dot{x} = \dot{\phi} R_w$$

$$\frac{\partial L}{\partial \beta} = m_p \dot{\beta} l^2 \cos^2 \beta + m_p \dot{\phi} R_w \dot{\beta} \sin \beta,$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\beta}} = m_p \dot{\beta} l^2 + m_p l \sin \beta \ddot{\phi} R_w, \frac{\partial L}{\partial \phi} = m_p g l \cos \beta$$

$$\text{Therefore, } m_p \dot{\beta} l^2 + m_p l \sin \beta \ddot{\phi} R_w - m_p g l \cos \beta = -\tau$$

Therefore, the angular acceleration of the body from the inertia x-axis $\dot{\beta}$ become,

$$\ddot{\beta} = \frac{1}{m_p l^2} (-m_p l \sin \beta \ddot{\phi} R_w + m_p g l \cos \beta - \tau) \dots\dots\dots(3.8)$$

Therefore, the overall dynamic model in ODE form of the robot for the first phase are equation (3.7) and equation (3.8),

$$\ddot{\phi} = \frac{\tau}{I_w + m_t R_w}$$

$$\ddot{\beta} = \frac{1}{m_p l^2} (-m_p l \sin \beta \ddot{\phi} R_w + m_p g l \cos \beta - \tau)$$

Where,

- ϕ is the wheel's rotational angle.
- β is the robot's angle with respect to the x-axis.
- l is the length between point mass and the center of mass of the body
- m_w is the mass of the wheel
- m_p is the mass of point mass,
- $m_{m_p} = m_w + m_p$ is the total car or robot mass,
- R_w is the radius of the wheel,
- I_w is the inertia of the wheel,
- τ is torque applied to the wheel.

3.2.2 Modeling of Variable Length Wheel Inverted Pendulum phase

A sampled model known as a variable Length wheel inverted pendulum is used to simulate the behavior of the system. The Wheeled Inverted Pendulum model, basically served to steer balanced robot, is the model on which this one is based(Orin, may, 2014). This one is based on a model called the Wheeled Inverted Pendulum (WIP), which is used to drive balanced robots. As illustrated in the accompanying proposed template. This proposed model is extended to incorporate a prismatic extension joint and a jumping direction defined on the vertical z-coordinate(Traiko Dinev, 2020)Figure 3.2.

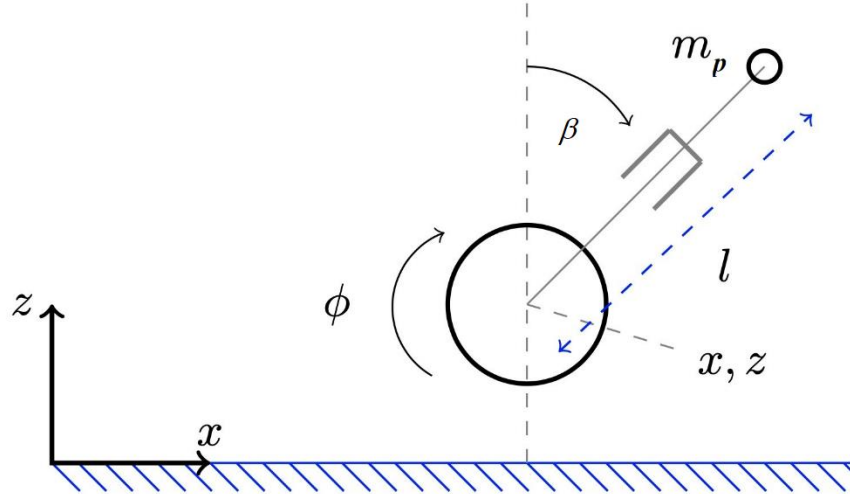


Figure 3.2: VL-WIP proposed template

As mentioned in first phase every nonlinear system model which describes the system behavior is expressed as $\dot{x} = f(x, u)$

Where,

- x is stands for state of the system and,
- u is stands for input controls.

One of the variables used to define the mathematical "state" of a dynamical system is a state variable. In this phase State variable of the system, $x = [q^T \dot{q}^T]$, where q general position coordinate and \dot{q} is general velocity of the system. For this variable length wheel pendulum

(VL-WIP)phase system $x = [q^T \dot{q}^T]$,

Where,

Generalize position of the system, $q = [x \ z \ \phi \ l \ \beta]^T$ and

Generalize velocity of the system, $\dot{q} = \frac{dq}{dt} = [\dot{x} \ \dot{z} \ \dot{\phi} \ \dot{l} \ \dot{\beta}]^T$, as you see in the

above figure.

Where,

- x is the wheel's center's coordinates to the x-axis.
- z is The wheel's center's coordinates to the x-axit.
- ϕ is the angle of the wheel's axis..
- l distance between the body point mass and the wheel center
- β is the body's rotation with respect to the z-axis of inertia..
-
- \dot{x} is the speed of the wheel's center relative to the x-axis.
- is the speed of the wheel's center relative to the z-axis.
-
- $\dot{\phi}$ is the angle of rotation of the wheel..
-
- \dot{l} is the velocity movement of prismatic joint up and down
-
- $\dot{\beta}$ Rotational angle with respect to the inertia z-coordinates.

A controls input for this system are $u = [\tau \quad F]$ and $\lambda = [\lambda_x \quad \lambda_z]$

Where,

- τ is torque applied to robot wheels
- F Force applied to extension joint.
- λ_x Contact forces acting on the x directions.
- λ_z Contact forces acting on the z directions.

The dynamic modeling approach used for this system (robot) is based on Lagrange equation (Morin, 2007). The overall number of states and inputs for second phase for the above floating figure and the ground contact force is zero in the floating under floating phase is then given by:

$$x = \begin{bmatrix} x \\ z \\ \phi \\ l \\ \beta \\ \square \\ x \\ \square \\ z \\ \square \\ \phi \\ \square \\ l \\ \square \\ \beta \end{bmatrix}, \text{ States of the system } u = \begin{bmatrix} \tau \\ F \end{bmatrix}, \text{ Where } \lambda = [\lambda_x \quad \lambda_z] = 0$$

Before derive the Lagrange equation of the system first let define the position of x_p, z_p , the

body (point) mass m_p and its velocity \dot{x}_p, \dot{z}_p based on the proposed prismatic joint template.

$$x_p = x + l \sin \beta$$

$$z_p = z + l \cos \beta$$

$$\dot{x}_p = \dot{x} + \dot{l} \sin \beta + l \beta \cos \beta \dots \dots \dots (3.9)$$

$$\dot{z}_p = \dot{z} + \dot{l} \cos \beta - l \beta \sin \beta \dots \dots \dots (3.10)$$

Now we can compute the kinetic energy and potential energy of the system using Lagrangian methods that are mention on the above equation (3.1) and equation (3.2),

Kinetic Energy of the system Variable Length wheel Inverted Pendulum Phase:

Every system connection will typically have both a rotational and a translational component of kinetic energy. kinetic energy on the wheel expressed as $I_w \dot{\phi}^2$, where $I_w = m_w R_w^2$ is inertia of the wheel, where m_w mass of wheel. Since the wheel of the point mass (body) has no rotation and

hence has no moment of inertia, the point of mass only kinetic energy that possesses transitional $m_p v_p^T v_p$, where v_p is velocity of the point mass and m_p is mass of the point

Rotational kinetic energy:

$$T_r = \frac{1}{2} I_w \dot{\phi}^2 \dots$$

Translational kinetic energy of the robot in the second phase:

$$T_t = \frac{1}{2} (m_w \dot{x}^2 + m_w \dot{z}^2 + m_p \dot{x}_p^2 + m_p \dot{z}_p^2),$$

Translational and rotational kinetic energy are added to provide the system's total kinetic energy (T). As a result, the system's kinetic energy becomes,

$$T = T_r + T_t$$

$$T = \frac{1}{2} (I_w \dot{\phi}^2 + m_w \dot{x}^2 + m_w \dot{z}^2 + m_p \dot{x}_p^2 + m_p \dot{z}_p^2) \dots \dots \dots (3.11)$$

Potential Energy of the system Variable Length wheel Inverted Pendulum Phase:

The potential energy due to gravity, which acts on both the wheel and the point mass. As a result, the robot's potential energy becomes,

$$U = m_w g z + m_p g z_p \dots \dots \dots (3.12)$$

Therefore, the Lagrangian Equation (3.2) will become,

$$L = \frac{1}{2} (I_w \dot{\phi}^2 + m_w \dot{x}^2 + m_w \dot{z}^2 + m_p \dot{x}_p^2 + m_p \dot{z}_p^2) - (m_w g z + m_p g z_p) \dots \dots \dots (3.13)$$

Inputting Equations (3.13) into Equation (3.1) leads to a system to have five equations due to five generalize position (5 DOF) of the robot, the solution is the equations of motion.

Case (1) – Compute Lagrange Equation (3.1) in terms of x generalize position (coordinate) and the control input is contact force λ_x using Equation (3.13) and the lagrangia will become,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = \lambda_x$$

$$\frac{\partial L}{\partial x} = m_w \ddot{x} + m_p (\ddot{x} + l \ddot{\beta} \sin \beta + l \dot{\beta}^2 \cos \beta)$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = m_w \ddot{x} + m_p \ddot{x} + m_p (l \ddot{\beta} \sin \beta + l \dot{\beta}^2 \cos \beta + l \ddot{\beta} \cos \beta - l \dot{\beta}^2 \sin \beta)$$

$$\frac{dL}{dx} = 0,$$

Therefore, $m_w \ddot{x} + m_p \ddot{x} + m_p (l \ddot{\beta} \sin \beta + l \dot{\beta}^2 \cos \beta + l \ddot{\beta} \cos \beta - l \dot{\beta}^2 \sin \beta) = \lambda_x \dots \dots \dots (3.14)$

Case (2) - Compute Equation (3.1) in terms of z generalize position and the contact force acting on the system and the lagrangia will become,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{z}} - \frac{\partial L}{\partial z} = \lambda_z$$

$$\frac{\partial L}{\partial z} = m_w \ddot{z} + m_p (\ddot{z} + l \ddot{\beta} \cos \beta - l \dot{\beta}^2 \sin \beta)$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{z}} = m_w \ddot{z} + m_p \ddot{z} + m_p (l \ddot{\beta} \cos \beta - l \dot{\beta}^2 \sin \beta - l \ddot{\beta} \sin \beta - l \dot{\beta}^2 \cos \beta)$$

$$\frac{dL}{dz} = 0 - (m_w g + m_p g) = -m_t g, \text{ where, } m_t = m_w + m_b$$

$m_w \ddot{z} + m_p \ddot{z} + m_p (l \ddot{\beta} \cos \beta - l \dot{\beta}^2 \sin \beta - l \ddot{\beta} \sin \beta - l \dot{\beta}^2 \cos \beta) + m_t g = \lambda_z \dots \dots \dots (3.15)$

Case (3) - Compute Equation (3.1) in terms of ϕ generalize position and the torque acting on the system the lagrangia will become,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} = \tau$$

$$\frac{\partial L}{\partial \phi} = I_w \ddot{\phi}, \text{ and then } \frac{d}{dx} \frac{\partial L}{\partial \dot{\phi}} = I_w \ddot{\phi}$$

$$\frac{\partial L}{\partial \phi} = 0,$$

$$I_w \ddot{\phi} = \tau \dots \dots \dots (3.16)$$

Case (4) - Compute Equation (3.1) in terms of l generalize position and the lagrangia will become,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{l}} - \frac{\partial L}{\partial l} = F,$$

$$\begin{aligned} \frac{\partial L}{\partial l} &= m_p ((x+l \sin \beta + l \beta \cos \beta) \sin \beta + m_p (z+l \cos \beta - l \beta \sin \beta) \cos \beta) \\ &= m_p (x \sin \beta + z \cos \beta + l) \end{aligned}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{l}} = m_p (x \sin \beta + z \cos \beta + l + x \beta \cos \beta - z \beta \sin \beta)$$

$$\frac{\partial L}{\partial l} = m_p g \cos \beta$$

$$m_p (x \sin \beta + z \cos \beta + l + x \beta \cos \beta - z \beta \sin \beta) - m_p g \cos \beta = F \dots \dots \dots (3.17)$$

Case (5). Compute Equation (3.1) in terms of θ generalize position and the counter torque is acting on system the lagrangia will become,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\beta}} - \frac{\partial L}{\partial \beta} = -\tau,$$

$$\frac{\partial L}{\partial \beta} = m_p (xl \cos \beta + zl + \beta l^2)$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\beta}} = m_p (x \dot{l} \cos \beta + z \dot{l} \sin \beta + x l \ddot{\beta} \cos \beta - z \dot{l} \sin \beta - x l \beta \sin \beta - z l \beta \cos \theta + l^2 \ddot{\beta} + 2 \dot{l} \dot{\beta})$$

$$\frac{\partial L}{\partial \beta} = m_p g l \sin \beta$$

$$m_p (x \dot{l} \cos \beta + z \dot{l} \sin \beta + x l \ddot{\beta} \cos \beta - z \dot{l} \sin \beta - x l \beta \sin \beta - z l \beta \cos \theta + l^2 \ddot{\beta} + 2 \dot{l} \dot{\beta}) - m_p g l \sin \beta = -\tau \dots \dots (3.18)$$

Equations of Motion (EoM)

The explicit Equations of Motion (EoM) or the dynamics equation of the system is derived using equation (3.13), equation (3.14), equation (3.15), equation (3.16) and equation (3.17) with the Lagrangian method for the system during ground contact can be written as:

$$m(q) \ddot{q} + c(q, \dot{q}) \dot{q} + g(q) = S^T u + J_c^T \lambda \dots \dots \dots (3.19)$$

$m(q)$ Mass matrix;

Mass matrix $m(q)$ is crucial for both dynamic modeling and controller design. The inertia matrix is a symmetric, positive, square matrix $n \times n$ whose members rely only on the generalized coordinates, and it also has a strong link to kinetic energy.

$$m(q) = \begin{bmatrix} m_t & 0 & 0 & m_p \sin \beta & m_p l \cos \beta \\ 0 & m_t & 0 & m_p \cos \beta & -m_p l \sin \beta \\ 0 & 0 & I_w & 0 & 0 \\ m_p \sin \beta & m_p \cos \beta & 0 & m_p & 0 \\ m_p l \cos \beta & -m_p l \sin \beta & 0 & 0 & m_p l^2 \end{bmatrix} \dots \dots \dots (3.20)$$

$c(q, \dot{q})$ Coriolis matrix:

The study of stability in mechanical systems and other systems, such as control systems, requires the use of centrifugal and Coriolis matrices $C(q, \dot{q})$. This matrix's members rely on the generalized coordinates and velocities and is square of the form $n \times n$.

$$c(q, \dot{q}) = \begin{bmatrix} 0 & 0 & 0 & 2m_p \sin \beta \dot{\beta} & -m_p l \sin \beta \dot{\beta} \\ 0 & 0 & 0 & m_p \cos \beta \dot{\beta} & -m_p l \cos \beta \dot{\beta} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -m_p l \dot{\beta} \\ 0 & 0 & 0 & 2m_p l \dot{\beta} & 0 \end{bmatrix} \dots\dots\dots(3.21)$$

g Gravity vector:

In mechanical systems without counterweights or springs, the gravity vector $G(q)$ is present, which is then present in systems with displacement from the horizontal plane. This $\times 1$ vector solely depends on the joint locations.

$$g(q) = \begin{bmatrix} 0 & gm_t & 0 & gm_p \cos \beta & -gm_p \sin \beta \end{bmatrix}^T \dots\dots\dots(3.22)$$

where, $m_t = m_w + m_p$

S Selection matrix:

The selection matrix S which used to translates controls inputs torque τ applied to wheel and force F that applied to linearly to prismatic joint into the generalized coordinates of the system. A counter-torque $-\tau$ will be imparted to the robot's body as a result of the torque τ supplied to the wheel. Write this down as:

$$u = \begin{bmatrix} \tau & F \end{bmatrix},$$

$$S = \begin{bmatrix} 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \dots\dots\dots(3.23)$$

J_C^T Contact jacobian:

The relationship between the contact point's velocity v_C and the robot's generalized velocities is represented by a contact jacobian matrix.

$$v_C = \begin{bmatrix} x_c \\ z_c \end{bmatrix}^T$$

$$v_c = J_C q = \begin{bmatrix} 1 & 0 & -R_w & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} q, \text{ where, } q = [x \ z \ \phi \ l \ \beta]^T \dots\dots\dots(3.24)$$

Where,

- x_c and z_c is contact point velocity on the x and z axis
- R_w is the radius of the wheel

λ Contact force:

$$\lambda = [\lambda_x \ \lambda_z] \dots\dots\dots(3.25)$$

When, where, and how are the contact forces operating on the directions. I have created dynamics model ODE form because of the simplicity of our system which is called analytic dynamics model. Analytic dynamics and their derivatives are quicker to analyze than the whole body dynamics of complicated systems, such as those computed using recursive techniques. By using analytical dynamics we can arrange coordinated motions for the point or body and wheels. The optimizer can plan for motions by using a unified planning method. Finally, the full system equation of motion can be derive using equation from (3.19) to (3.25) to the standard Ordinary

Differential Equation (ODE) form (analytic dynamic model form). $\ddot{x} = f(x, u, \lambda)$.

We know that $\dot{x} = [\dot{q} \ \ddot{q}]$, Where,

$$q = [x \ z \ \phi \ l \ \beta]^T \text{ Velocity, and } \dot{q} = \frac{dq}{dt} = \begin{bmatrix} \dot{x} & \dot{z} & \dot{\phi} & \dot{l} & \dot{\beta} \end{bmatrix}^T, \text{ acceleration}$$

So again, we can compute system acceleration \ddot{q} from Equation (19)

$$\ddot{q} = -m(q)^T [c(q, \dot{q}) \dot{q} + g(q) - S^T u - J_c^T \lambda] \dots\dots\dots(25)$$

System Parameters

In this thesis I investigated to validate the simulation of jumping robot using several tasks, namely swing-up and driving up right, balancing and jumping performance in order to validate approach in the Matlab simulator. To simulate the robot several tasks the following system parameters are consider.

No	Parameters	Symbol	Value
1	Mass of wheels	m_w	2 Kg
2	Mass of body	m_p	4 Kg
3	Length of Prismatic joint	l	2 m
4	Radius of wheels	R_w	0.17 m
5	Moment of Inertia of wheels	I_w	0.0578 Kg m^2
6	Gravity constant	g	9.81 m / s 2

Table 3-1: system parameters and there value

CHAPTER FOUR

MODEL PREDICTIVE CONTROL

4.1 Introduction

Model predictive control (MPC), a sophisticated approach to process control, is used to uphold process control while complying with a variety of constraints. Since the 1980s, Model predictive control has been used to process industries as well as chemical and oil refineries. It has been used in power electronics and models for balancing power systems in recent years (Michèle Arnold, August 22-26, 2011). The MPC controller's key feature is that it allows timeslot optimization while taking future timeslots into account. Instead of using a linear-quadratic regulator (LQR), this is accomplished by optimizing a limited time horizon, but only using the current timeslot before optimizing once again, repeatedly. Although they both represent optimal control, model predictive control and linear-quadratic regulators do so in distinct ways. LQR optimizes throughout the whole time frame (horizon), whereas MPC optimizes in a receding time window (Wang, 2009.). This is one of the main differences between MPC and LQR. And that LQR uses a single (optimal) solution for the entire time horizon. Additionally, MPC is capable of predicting future events and taking appropriate control measures. Contrarily, Proportional-Integral Derivative (PID) controllers lack the capacity for future prediction. Although there is study into using specially created analog circuitry to achieve faster response times, MPC is mostly always implemented as a digital control (Vichik & Borrelli, 2014).

In MPC controller, a the objective function is improved to ensure that a to tracked reference a prediction horizon of N . Current time step data and predicted outputs for future time steps are helps to follow the reference along the prediction horizon. A system model is used to anticipate the expected outputs. A series of ideal control inputs is calculated across a prediction horizon by minimizing a cost function. The present time step control input is then put into action at each time step. The following time step's procedure is then repeated. Since the horizon travels with the present time step, it is known as a receding horizon. Future time steps can be taken into consideration thanks to MPC's predictive nature.

4.1.1 Principle of Model Predictive Control

An open-loop optimum control issue with a limited horizon that is subject to system dynamics and constraints, such as states and controls, is often characterized as the model predictive control problem. Figure 4.1 shows the basic principle of model predictive control.

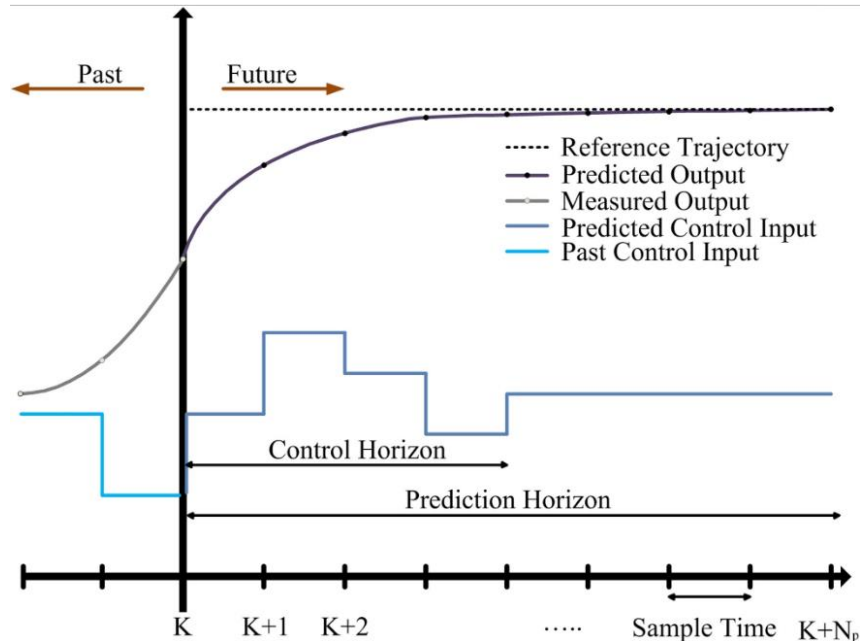


Figure 4.1: Principle of model predictive control.

Based on measurements taken at time, the controller chooses an input across a control horizon $k + N_p \geq k + N_c$ to achieve an ideal open-loop performance target. The controller then makes predictions about the system's future dynamic behavior over a prediction horizon. The input function discovered at time $k = 0$ could be applied to the system for all times if there were no disturbances, no model-plant mismatch, and the optimization issue could be solved for infinite horizons $k \geq 0$. But generally speaking, this is not achievable.

Because of disturbances and model-plant mismatch, the actual system behavior deviates from the anticipated behavior. In order to incorporate a feedback mechanism, the resulting open-loop modified input function will only be employed up until the availability of the next measurement. The interval between the recalculation and the measurement can vary, although it is frequently believed to be constant, meaning that the measurement will occur every sampling time-units δ . To discover a new input function with the control and prediction horizons moving

ahead, the entire prediction and optimization operation is repeated using the new measurement at time $k + \delta$. It will be shown how factoring in input and state limits as well as cost function optimization is possible when computing the applied input based on the anticipated system behavior. The intended system behavior will generally differ from the closed-loop behavior, thus effort must be taken to achieve closed-loop stability.

4.1.2 Mathematical Formulation of MPC

Model Predictive Control includes a family of control methods that allow to achieve optimal performance while satisfying constraints. It finds the optimal control action by solving an optimal control problem (OCP) over a finite prediction horizon. The cost function is minimized taking into account the process dynamics along the horizon.

To describe the expected behavior in order to specify an aim cost function have vital role. The cost function seeks to reduce over a horizon the difference between the reference states and the measured anticipated states. An MPC optimization problem's generic formulation is as follows:

$$\begin{aligned} & \min_{X(k), U(k)} J(X(k), U(k)) \\ & \text{Subject to } \begin{cases} X(k) \in X, U(k) \in U \\ X(k) = f(X(k), U(k)) \end{cases} \dots \dots \dots (4.1) \end{aligned}$$

Where the states and inputs can be contained within sets X and U :

$$\begin{aligned} X & := \{ X \in R^n \mid X_{\min} \leq X \leq X_{\max} \} \\ U & := \{ U \in R^m \mid U_{\min} \leq U \leq U_{\max} \} \dots \dots \dots (4.2) \end{aligned}$$

The nonlinear dynamics $f(X(k), U(k))$ also apply to the optimization issue. The following is the MPC's generic cost function:

$$J = \| X(N) - X_r(N) \|^2 P + \sum_{k=1}^N \| X(t+k) - X_r(t+k) \|^2 Q + \| U(t+k) - U_r(t+k) \|^2 R \dots \dots \dots (4.3)$$

Where,

X_r represent the reference state vector and U stands for control input. The present time step is represented by t and N represent the horizon. The iterations for which the output is predicted and the cost function is minimized are $k = 1, N$. The weights represented by Q which tracking the

reference states and R which punishing the input. These weight matrices are designated as symmetric, positive-definite weighting matrices.

$$Q = \begin{bmatrix} q_1 & 0 & \cdots & 0 \\ 0 & q_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & q_n \end{bmatrix}, R = \begin{bmatrix} r_1 & 0 & \cdots & 0 \\ 0 & r_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & r_m \end{bmatrix} \dots\dots\dots(4.4)$$

Each state's and input's unique weights are identified by the symbols $q_1 \dots q_n$ and $r_1 \dots r_m$, respectively. A larger weight means that monitoring some state or punishing an input is valued highly.

4.1.3 Solving MPC problem

The major methods to resolving trajectory optimization issues fall into one of two categories: direct or indirect. Direct techniques will be the main emphasis of this thesis. A direct method's distinguishing characteristic is that it discretizes the trajectory optimization issue at hand, often turning it into a nonlinear program. Due to the conversion process known as transcription, direct collocation techniques are also known as direct transcription techniques. Typically, all continuous functions in the problem statement are approximated as polynomial splines in order to discretize a continuous trajectory optimization problem utilizing direct transcription techniques.

A function called a spline is constructed from a series of polynomial segments. Since they may be represented by a limited (finite) number of coefficients and because it is simple to compute integrals and derivatives of polynomials in terms of these coefficients, polynomials are employed.

4.1.4 Nonlinear Programming

It is necessary to transform the optimization issue into a Nonlinear Program in order to solve a MPC controller problem. The majority of direct collocation methods transform a continuous-time trajectory optimization problem into a nonlinear program (NLP). If the objective function or constraint function of an optimization problem with restricted parameters contains nonlinear elements, the problem is referred to as a "nonlinear program".

4.1.5 Transcription Method

Different categories can be created to classify transcription techniques for best control. Direct and indirect approaches are the first to mention. The issue is discretized in direct approaches, followed by optimization. The prerequisites for optimality are computed first in indirect approaches. After being discretized and resolved, these circumstances. Direct transcription is a key component of the transcription techniques this thesis examines. Since they require extremely exact initial estimations to converge, indirect transcription methods are extremely accurate yet challenging to solve in reality (V. Leek, 2016.).

The two categories of direct procedures are simultaneous and sequential. An illustration of a sequential technique is a single firing technique. This approach discretizes only the control trajectory using a piecewise slight estimation. After that, the discretized control trajectory are introduced to the NLP. A cannon firing at a target is an analogy for a single shooting technique. First, a decent shot angle is estimated. The target is then examined to see if it was struck. If not, the angle is changed based on the prior outcome, allowing the issue to be resolved progressively across a number of rounds. The resulting NLP is often lower but extremely nonlinear when employing a single firing approach (Diehl, 2015). By taking into account a multiple shot approach, where the issue size is larger but contains less nonlinearities, convergence can be enhanced. A multiple strategy often yields better NLP convergence and more numerical stability than a single shot method (H. Bock and K. Plitt, 1984.). The resultant NLP also has a sparser format that can be used. Use a multiple shot strategy in this thesis since the state and control trajectories are both discretized.

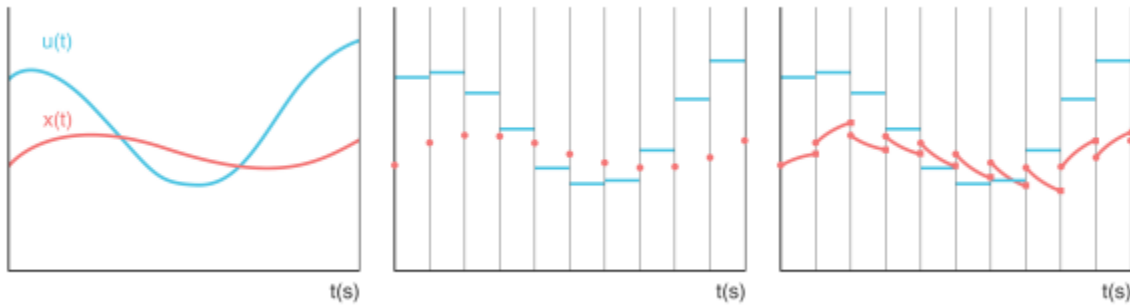
4.1.6 Multiple Shooting

A transcribing technique based on simulation is the multiple shooting method. To derive the discrete-time dynamics, a Runge Kutta, Euler method, and trapezoidal method scheme are often applied. At the start of each control interval the states and control inputs are only included in multiple shooting techniques. As observed in the direct collocation approach, there are no intermediary collocation spots.

In Figure 4.2, as you see in the figure below the multiple shooting approach, first, the continuous (4.2a) the trajectories are discretized (4.2b). N is represent the amount of control intervals. The

control inputs $u(t)$ are to the decision variables u_0, u_1, \dots, u_{N-1} obtained by discretizing the control input

Similarly, the state decision variables x_0, x_1, \dots, x_N are presented. Now, every interval k consists of u_k a control input signal and x_k a initial state, as shown in Figure 11b. Start from x_k and the state at the next interval x_{k+1} can be evaluated by integrating over each interval.



(a) Continuous (b) Discretized (c) Integrating state trajectory

Figure 4.2: Multiple shooting approach

Figure 4.2c illustrates the mismatch between the state gained through integration and the start state of the following time step. In order to ensure that these two states do not conflict, continuity restrictions are added.

A Runge-Kutta technique is employed in this thesis to determine the state at the following time step. A group of techniques for integrating ordinary differential equations are known as Runge-Kutta methods. A fourth order Runge-Kutta algorithm is used in this situation. It comes close to

providing the answer to a differential equation $\dot{x} = f(x, u)$ given beginning conditions $x(t_0) = x_0$.

The number of slope estimates made during a single time step is referred to as the Runge-Kutta method's order. Figure 4.3 shows a representation of the Runge-Kutta technique for the fourth order. $R1$ and $R2$ specify the slope at the start and finish points. $R2$ and $R3$ are the slopes at the two midpoints.

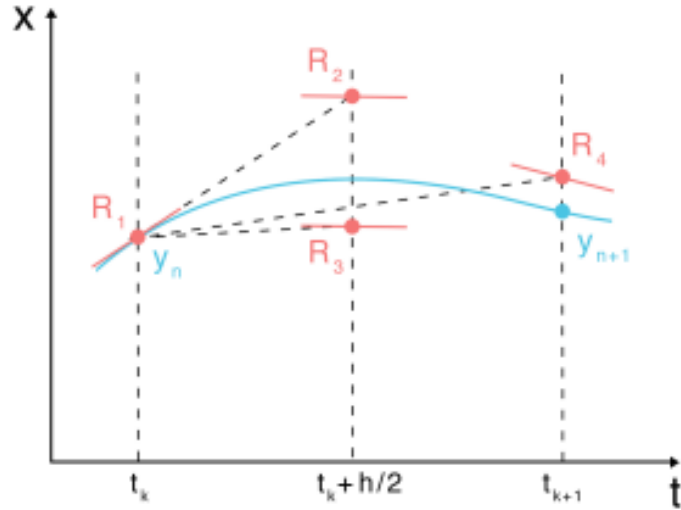


Figure 4.3 Runge kutta approach

To find an estimation of x_{k+1} , an averaged weighted of the slopes at R_1, R_2, R_3 and R_4 is used:

$$\tilde{x} = x_k + \frac{1}{6}(R_1 + 2R_2 + 2R_3 + R_4)h$$

$$t_{k+1} = t_k + h \dots \dots \dots (4.5)$$

The midpoint slopes are weighted twice as much as the initial and finish point. Two times as much weight is given to the halfway slopes as to the initial and finishing points. The specific slopes at each site are described as follows:

$$R_1 = f(t_k, x_k)$$

$$R_2 = f(t_k + h/2, x_k + R_1/2)$$

$$R_3 = f(t_k + h/2, x_k + R_2/2)$$

$$R_4 = f(t_k + h, x_k + R_3) \dots \dots \dots (4.6)$$

4.2 System Control and Problem Formulation

The usage of the resulting system dynamics in a pipeline for planning and controlling systems is examined in this section. An optimum control formulation, called direct transcription, is used to derive motion planning. The controller control mismatch and perturbations that occur during execution using predictive controller. And also the controller continually requests the planner to re-plan the motion by requesting the planner from the present robot state. In the same way as conventional linear MPC, nonlinear MPC determines the control actions at each control interval by combining model-based prediction with constrained optimization. Here are the main difference:

- The prediction model may have time-varying parameters and be nonlinear.
- Constraints on equality and inequality may not be linear.
- The cost function that has to be reduced might be a non-quadratic (linear or nonlinear) function of the choice variables. Constraints on equality and inequality may not be linear.

You can simulate closed-loop control of nonlinear plants under nonlinear costs and restrictions using nonlinear MPC. Nonlinear MPC controllers by default use the *nlmpcmove* function of the Optimization Toolbox software to solve a nonlinear programming issue, but if you don't have this software, you can define your own unique nonlinear solver. By configuring the *fmincon* option, the solver decision variable, and the initial guesses.

4.2.1 Solver Decision Variables

For nonlinear MPC controllers at time t_k , the nonlinear optimization problem uses the following decision variables:

- From time t_k to t_{k+p-1} predictions of state values. The values shown above correspond to rows 2 through $p+1$ in the input argument for your cost and constraint functions, where P is the prediction horizon.
- Predicted time t_k to t_{k+p-1} manipulated variables. These values correspond to the rows 1 through p of the u input parameter of your cost and constraint functions' modified variable columns.

4.2.2 Specify Initial Guesses

A common linear MPC optimization issue has a singular solution when configured appropriately. It might be challenging for the solver to find a solution to nonlinear MPC optimization problems since they frequently enable many solutions (local minima). In such cases, it is essential to provide a strong starting point that is not far from the global optimum.

Before doing closed-loop simulations, your nonlinear solver should be warmed up. Your initial predictions for the current control interval should be based on the predicted state and adjusted variable trajectories from the previous control interval.

When invoking *nlmpcmove*, return the *opt* output parameter. Any run-time options you supplied in the preceding *nlmpcmove* call are included in this *nlmpcmoveopt*. Object. Additionally, it contains the starting hypotheses for the state (*opt.x0*), manipulated variable (*opt.MV0*), and global slack variable trajectories (*opt.Slack0*). Pass this object in as the options input argument to *nlmpcmove* for the next control interval.

4.2.3 Configuration of *fmincon* Options

By default, nonlinear MPC controllers optimize their control motion using the *fmincon* function from the Optimization Toolbox. The *Optimization.SolverOptions* field of the *nlmpc* object initially contains the default *fmincon* parameters along with the following non-default settings:

- Use the SQP algorithm (*SolverOptions.Algorithm = 'sqp'*)
- Use objective function gradients (*SolverOptions.SpecifyObjectiveGradient = 'true'*)
- Use constraint gradients (*SolverOptions.SpecifyConstraintGradient = 'true'*)
- Do not display optimization messages to the command window (*SolverOptions.Display = 'none'*)

4.2.4 Problem Formulation Planning

The problem formulation planning is illustrated below. The formulation of the planning problem is a nonlinear optimization problem (NLP) whose goal is to discover routes that minimize the overall cost function while satisfying the problem constraints.

$$x = \{x_0, x_1, x_3 \dots, x_N\},$$

$$U = \{u_0, u_1, u_3 \dots, u_N\}, \text{ and } \lambda = \{\lambda_0, \lambda_1, \lambda_3 \dots, \lambda_N\}$$

Here, x refer to the robot state, u refer to the control inputs and λ refer to the contact forces. The first objective is to discretize the motion's duration prediction horizon (T) into N knot points and express it in seconds. The optimizer locates states x , controls u , and contact forces at each knot point that satisfy the system dynamics $x_{k+1} = f(x_k, u_k, \lambda_k)$ derived in system modeling section.

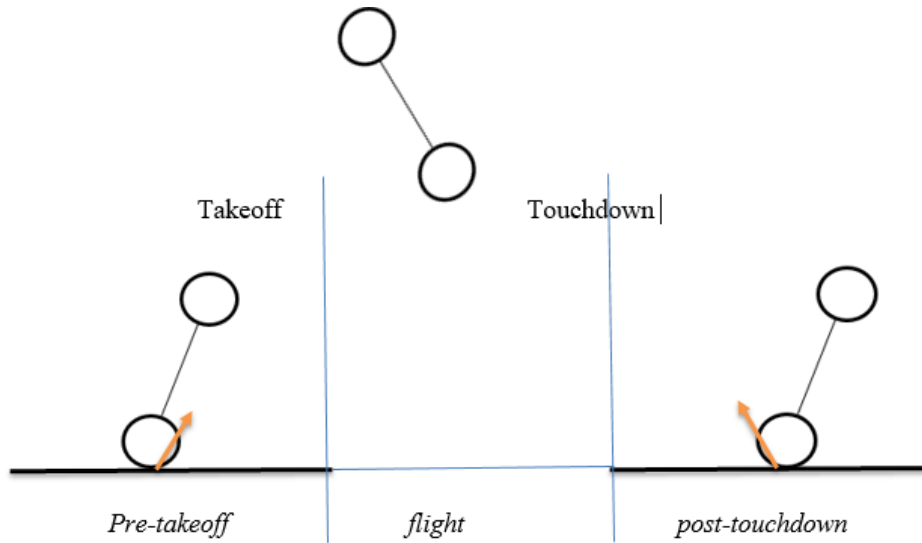


Figure 4.4: Jumping over a gap

The general problem formulation or the mathematical formulation for the problem are mentions as follows:

$$\text{Cost function} \quad \min_{X,U,\lambda} \sum_{k=1}^N ((x_N^* - x_k)^T Q (x_N^* - x_k) + u_k^T R u_k) + (x_N^* - x_N)^T Q_e (x_N^* - x_N)$$

Subject _to

Intial state

$$x_0 = x_0$$

Reference state

$$x_N = x_N$$

Dynamic (system equation)

$$x_{t+1} = f(x_t, u_t, \lambda_t), t \in [0, N]$$

State phase

$$x^l \leq x_t \leq x^u, t \in [0, N]$$

Control bounds

$$u^l \leq u_t \leq u^u, t \in [0, N-1]$$

Flight phase

$$t \in [T_{tf}, T_{td}]$$

(no contact force)

$$\lambda_t = 0$$

Ground Phase

$$t \notin [T_f, T_d]$$

(friction cone)

$$|\lambda_t^x| \leq \lambda_t^z$$

(unilateral force)

$$\lambda_t^z \geq 0$$

No slip on ground

$$\square x_t = R w \phi_t \square$$

Specific limits apply after the duration T setup job has been selected. The initial state x_0^* and the reference state x_N^* are specified for knots x_0 and x_N . At each knot point, state bounds $[x^l, x^u]$ as well as control bounds $[u^l, u^u]$ are enforced. These boundaries ensure the physical feasibility of the motion generated. The bounds of state and control will be different for the different tasks. To perform all tasks of robot will have three phases of state bounds. For instance, will show where the "flying" phase starts and concludes when jumping over a gap. A leaping action typically consists of the "pre-takeoff" phase, during which the robot is on the ground; the "flight" phase, during which the robot become floating; and the "post-touchdown" phase, during which

the robot landing and balance again balance to equilibrium point. I specified the takeoff time T_f and the touchdown time T_d to determine the length of each phase.

Finally, I define the limits for ground contact. During the robot become in the floating phase, the contact forces $\lambda_y = 0$ must be zero. The ground phases should have a positive vertical component $\lambda_y^z \geq 0$ and remain inside the friction cone $|\lambda_y^x| \leq \mu \lambda_y^z$, where μ is the friction coefficient.

While the robot is on the ground $x_t = R_w \phi_t$, I apply a no-slip restriction for kinematics: where R_w is radius of wheel. By omitting the limitations connected to the flight phase for motions that don't include leaping, I employ a comparable approach. The planner determines a legal course for the control job. The issue solution provides a set of optimum control actions.

$u_{t:t+N-1|t} = [u_{t|t}, \dots, u_{t+N-1|t}]$. wherever, I just apply the first one $u(t) = u_{t|t}(x_t)$

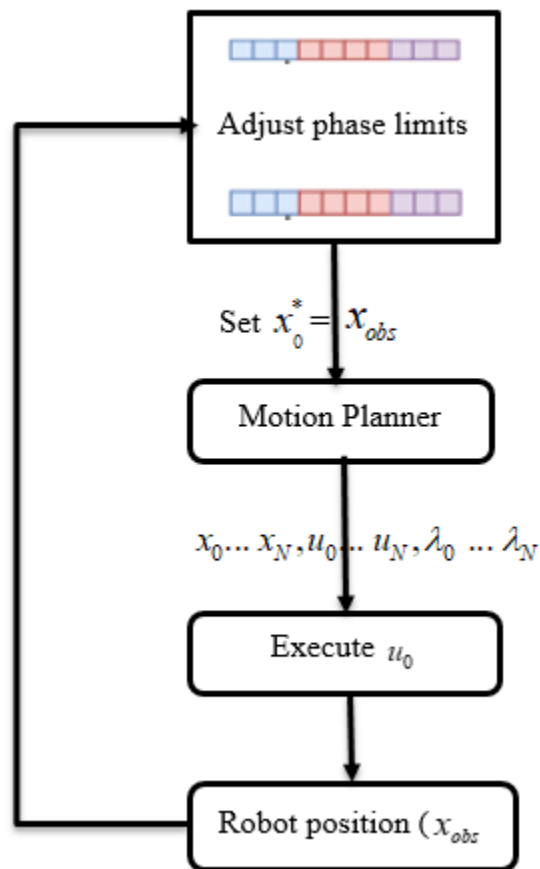


Figure 4.5: model predictive control pipeline.

The start state x_0 is set to the current state x_{obs} in the simulation by the controller, which then re-plans a motion at the beginning of each time step. The planner's phases and limitations are updated as the first control u_0 is then put into action. The planner updates the trajectory beginning from the observed robot x_{obs} state retrieved from the simulator at each time step. The plan's initial action u_0 is to be executed. The temporal horizon T is then retracted by the amount of time that has passed.

A natural issue results from T 's recession. The number of knot points N should ideally remain consistent as this allows the optimizer's state to be saved and prevents expensive re-initialization. To ensure that every phase lasts the same amount of time, the problem's limits must be changed. By adjusting the problem's bounds while keeping N constant, it is possible to retain the ratio of the pre-flight phase, flight phase, and post phases.

4.2.5 The Matlab Code

As mentioned in chapter three, there are two phases: first phase, which includes swinging up and balancing, and the second phase, which uses a variable length wheel inverted vehicle to execute "*pre-takeoff*," "*flying*," and "*post-takeoff*." For these two different phases, the code is prepared into different files:

- **Utils:** it contains a set of functions that are used to create videos and plots.
- **Robot models:** in these files we define the model in an appropriate way, by specifying the system's dynamic model, and, eventually, the constraints.
- **Mains:** in these files we define and solve the problem, so we implement the MPC controller or the PD controller depending on the simulation.

Moreover, we recall that the integrations have been performed using Runge-Kutta of 4th order and the sampling time of the controller is 0.25 s. First of all, in the code define the model parameters, and the state and control variables. These are:

- I. Swing-up and balance phase:

$$X = [\phi \ \beta \ \dot{\phi} \ \dot{\beta}]$$

$$u = \tau$$

Afterward, specify the dynamics explicitly as we saw in chapter three. To create the model's discrete version using Runge-Kutta of fourth order. The variable length wheeled inverted pendulum model receives the same mechanism. Then setup the following state and controls in the file.

$$X = [x \ z \ \phi \ l \ \beta \ \dot{x} \ \dot{z} \ \dot{\phi} \ \dot{l} \ \dot{\beta}]$$

$$u = [\tau \ f \ \lambda_x \ \lambda_z]$$

Additionally, to this specify each matrix required to define the dynamic model of the control system. The latter is then expressed in the form of explicitly. In addition, employ Runge-Kutta of fourth order in this situation.

In the source code file, after employ the above mention situation I define the problems to be solved with the MPC controller. In fact, I have implemented two different optimization problems to be solved with the MPC. The first drives the four-wheel robot from a velocity of $\dot{x} = 3m/s$ up to $\dot{x} = 0m/s$ using the *phase I* model that modeled in section 3.1. This stage lasts for a fraction of a second. I specified the following values for the state boundaries for the MPC constraints:

$$X_{lim} = \begin{bmatrix} -\infty & 0 & -\infty & -\infty \\ \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(4.7)$$

In actuality, $z = 0$ since otherwise the robot would sunk into the ground. I configured these control inputs boundaries:

$$u_{lim} = \begin{bmatrix} -10 \\ 10 \end{bmatrix}$$

After achieving the desired velocity, I confront another optimization challenge in section 3.2's *phase* (VL-WIP) model. In this phase as I mentioned before perform several tasks, which is namely “*pre-takeoff*”, “*flight*” and “*post-takeoff*” phase. Different constraints differentiate each phase. Prior to undertaking is in the ground phase, so for the ground phase we have the following boundaries:

$$X_{\lim}^{pre-tf} = \begin{bmatrix} -\infty & R_w & -\infty & l^l & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ x^u & R_w & \infty & l^u & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(4.8)$$

The next task after ground phase is flying phase we have the following boundaries:

$$X_{\lim}^{flight} = \begin{bmatrix} x^l & R_w & -\infty & l^l & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ x^u & z^u & \infty & l^u & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(4.9)$$

Last step: after the robot has returned to the ground contact after flying, I set the following boundary constraints:

$$X_{\lim}^{post-ft(td)} = \begin{bmatrix} x^u & R_w & -\infty & l^l & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ \infty & R_w & \infty & l^u & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(4.10)$$

Where $[x^l, x^u]$ are the boundary of the obstacle in the x direction. The value of z^u can be more than radius of the wheels during the robot must fly, but it cannot be smaller than radius of wheel (R_w) due to the existence of the ground. Length of body (l) is bounded to be always between two values that I choose, while β has to be between $-\pi/2$ and $\pi/2$.

Similarly, for the control inputs boundaries for ground phase before flight and after flight the constraint boundary should:

$$X_{\lim}^{ground} = \begin{bmatrix} -10 & -200 & -\infty & 0 \\ 10 & 200 & \infty & \infty \end{bmatrix} \dots\dots\dots(4.11)$$

Similar to this, the constraint boundary setup for the control inputs for the flight phase in the following manner:

$$X_{\text{lim}}^{\text{flight}} = \begin{bmatrix} -10 & -200 & 0 & 0 \\ 10 & 200 & 0 & 0 \end{bmatrix} \dots\dots\dots(4.12)$$

4.3 Proportional Derivative Controller

The proportional derivative controller is a form of controller in a control system whose output varies proportionally to the error signal as well as with the derivative of the error signal. Additionally called PD controllers or proportional plus derivative controllers. One of the most crucial feedback controllers is a proportional derivative (PD) controller. Combining both proportional and derivative control action, this sort of controller offers combined action. We are aware that any control system that includes controllers performs better as a whole. Therefore, the inclusion of two separate control actions results in a system that is more precise.

The block diagram of a control system comprising of PD controller is given below:

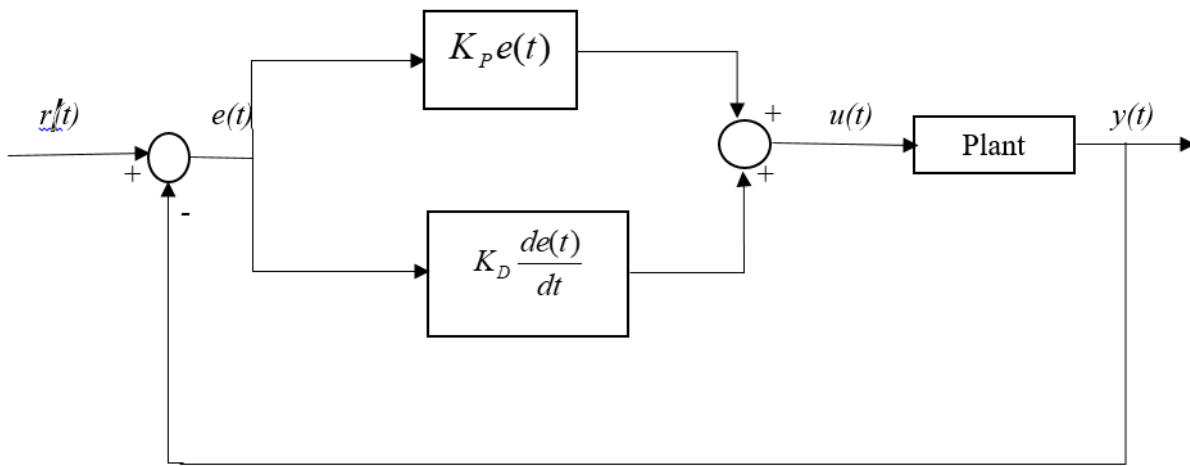


Figure 4.6: PD controller - block diagram

- **Proportional controller:** - It is a particular kind of controller where the output varies proportionally to the input. Mathematically it is expressed as:

$$u(t) \propto e(t)$$

$$u(t) = k_p e(t) \dots\dots\dots(4.13)$$

- **Derivative Controller:** The derivative controller's control action is set up such that its output is proportionate to how quickly the error signal is changing over time..

$$u(t) \propto \frac{de(t)}{dt}$$

$$u(t) = k_D \frac{de(t)}{dt} \dots\dots\dots(4.14)$$

A system is more efficient when a derivative controller and a proportional controller are used together. In situations like these, a proportional controller gets rid of the drawbacks of a derivative controller. We are aware that the fundamental design goal of derivative controllers is to have their output alter in response to changing error signals.

In the event of a constant error signal, it does not exhibit change. This is due to the fact that the error signal's rate of change over time will be zero if its value remains constant. Derivative controllers are therefore employed in combination with proportional controllers to take even constant error signal into account.

Sensitivity is increased by a proportional controller's use of a derivative control action. As a result, the system is more stable and may produce early corrective responses for even minor error signal values. However, we are also aware that the derivative controller causes steady-state error to grow. While reducing the steady-state error are the proportional controllers.(electronicscoach, 2022)

4.3.1 Proportional and Derivative Controllers

The following is the mathematical formula for a proportional derivative controller, which combines the actions of a proportional and a derivative controller:

$$u(t) \propto e(t) + \frac{de(t)}{dt} \dots\dots\dots(4.15)$$

Therefore, after removing the proportionality sign, the error signal and its derivative are added together with the proportionality constant. Thus

$$u(t) = k_p e(t) + k_D \frac{de(t)}{dt} \dots\dots\dots(4.16)$$

Where, k_p stands for constant of proportionality of error signal,

k_D Stands for constant of proportionality of derivative of the error signal.

As a result, in my thesis I just want to use a proportional derivative controller to operate the robot during the balancing phase, and the equation is:

$$\begin{aligned} \tau &= \tau_d - [k_D^\theta e(\beta) + k_D^\theta e(\dot{\beta}) + k_p^\theta e(\phi) + k_p^\theta e(\dot{\phi})] \\ f &= f_d + k_p^l e(l) + k_p^l e(\dot{l}) \dots\dots\dots(4.17) \end{aligned}$$

Where, $e(.)$ is error signal that is the difference between the desired and measured state while, k_d and k_p are the derivatives gain and proportional gains respectively. For the control input torque τ , I take into account the error on the generalized coordinates β and ϕ , and its derivatives, while for the force I consider the error on l and on it derivative. The control input force f also has a feed forward term equal to the desired value for it; and I set as following equation:

$$f = m_p g \cos \beta \dots\dots\dots(4.18)$$

Let's see how these two k_D and k_p action terms are defined and there role in order to maintain the stability of the robot:

- **Proportional action:** - it immediately reacts to the variations of the error, so it represents the present action. The error decreases as the proportional constant k_p increases, in fact, the rise time is the main characteristic that we can improve with this action, but, of course, there is a limit for the increase of the proportional constant.
- **Derivative action:** it is proportional to the derivative of the error, so it represents an anticipation of what will happen to the error. It is used to improve the transient of the response since it can decrease the overshooting. Here in this thesis the value both gain k_D and k_p are tuned manually.

CHAPTER FIVE

RESULT AND ANALYSIS

5.1 Introduction

The objective of this thesis to validate simulation of two wheel jumped robot. In previous section the mathematical modeling and control design are develop in section three and section four respectively. As mention in the previous modeling section this system have perform two-phase section. The simulations have been implemented in Matlab using the **nlmpcmove** solver which is used to computes optimal control action. **nlmpcmove** is a Matlab nonlinear optimization toolbox for finding the solution of optimal control problems. In order to minimize a cost function given the dynamics, restrictions, and initial conditions of the system, it solves a nonlinear Optimal Control Problem (OCP).

5.2 Closed loop Response of the System

5.2.1 Swing up and Balance Phase

In swing-up and balance Phase, the system is essentially a car with four wheels. The front wheels are actuated and the peculiarity of this robot is its prismatic joint that allows increasing and reducing the body length. In the starting position, the robot is placed horizontally with respect to the floor, with the entire wheel touching the ground. The robot has to swing up, by braking and accumulating kinetic energy. In this phase, it has a starting velocity different from zero $x = 7m/s$ that will be used to accumulate energy.

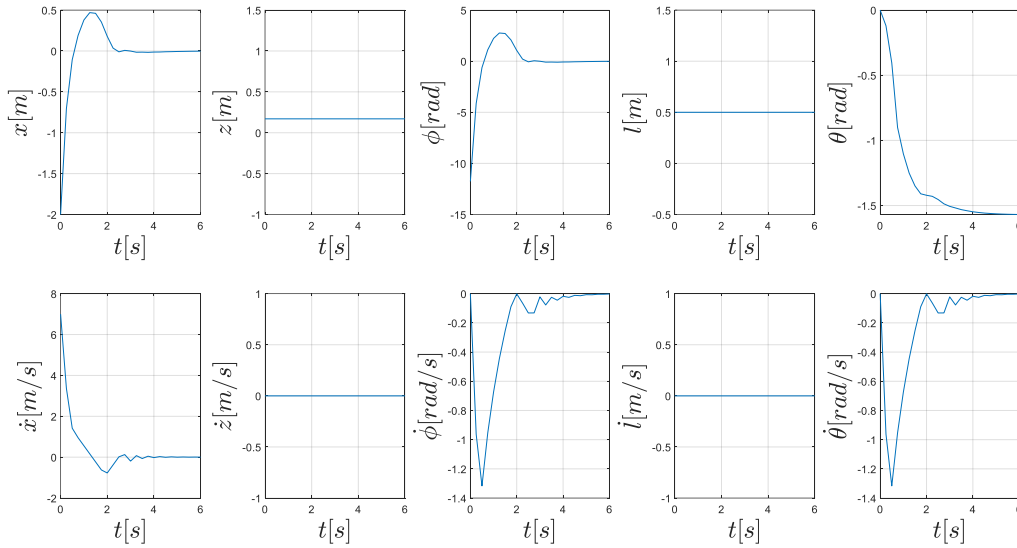


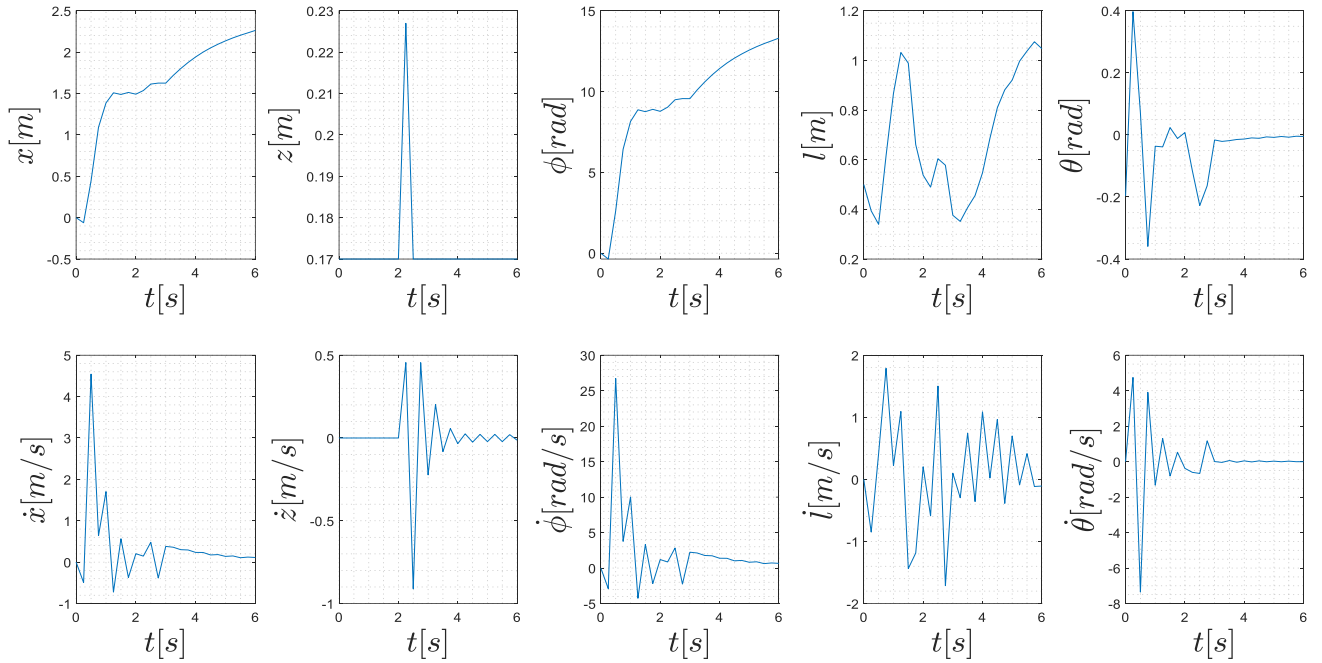
Figure 5.1: Closed loop response for swing-up and balance phase

In this phase as you see in the simulation show that the robot has first to drive horizontally on four wheels, then it should swing up (get up) and balance on two wheels. The ability to transition from a car mode on four wheels and on two wheels state. This illustrates how the robot can change state on its own.

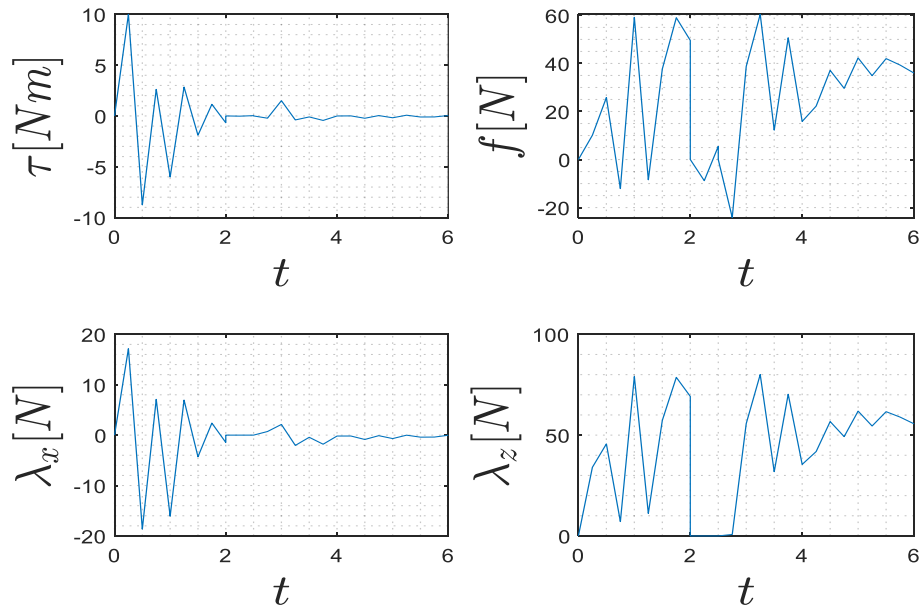
5.2.2 Closed loop response for the second (VI-WIP) phase

In this Phase, in this phase the length of prismatic joint allows increasing and reducing the body length of the robot. Change the controller to the suggested model to be controlled with a model predictive controller as soon as the robot's velocity reaches zero. In this phase system, perform three tasks, namely *pre-takeoff*, *flight* and *post-flight* tasks. Some simulation times, prediction horizon and control horizon are define as the follows:

- In 2 second the robot has to approach the initial edge of the gap (pre-takeoff) with 20 prediction horizon and 5 control horizon.
- In 0.5 second the robot has to jump over the gap (flight) with 20 prediction horizon and 5 control horizon.
- In 3.0.second the robot has to settle down and regain its balance after landing and drive upright again (post-flight) with 20 prediction horizon and 5 control horizon.



(a) States response during the jumping over a gap phase



(b) Controls inputs response during the jumping over a gap phase

Figure 5.1: States response and Controls inputs response during the jumping over a gap phase

Through the use of state and control constraints, Figures 5.2 and 5.3 depict the leaping action in its entirety. In addition to torque constraint $\pm 10Nm$ and force constraint $\pm 200N$, and also set the state constraints X_{lim} for each of the phases (from equation (4.7), (4.8) and (4.9)). Letting $[x^-, x^+]$ define the constraints of the gap in the x direction, as you see z specified the constraint during flight mode in the z direction, the length of prismatic joint and angle β also bounds in the following equation.

$$X_{lim}^{pre-tf} = \begin{bmatrix} -\infty & R_w & -\infty & 2R_w & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ 1.5 & R_w & \infty & 1+2R_w & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(5.1)$$

$$X_{lim}^{flight} = \begin{bmatrix} -\infty & R_w & -\infty & 2R_w & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ \infty & 2R_w & \infty & 1+2R_w & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(5.2)$$

$$X_{lim}^{post-tf} = \begin{bmatrix} 2 & R_w & -\infty & 2R_w & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ \infty & R_w & \infty & 1+2R_w & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \dots\dots\dots(5.3)$$

Figures 5.4 and 5.5 show the simulation plot that results from plotting the state and control histories. Prior to takeoff and landing, the prismatic joint in the robot's body is initially "contracted" by the optimizer by retracting it. In the motion that has been optimized for x , ϕ , and β , we can observe the preliminary movement.

The Model Predictive Control also designed in Fourth chapter for both phases. Therefore, the following simulation response show as how two-stage controller is used to switch from the swing-up and balance phase model to the variable length wheel inverted pendulum Phase.

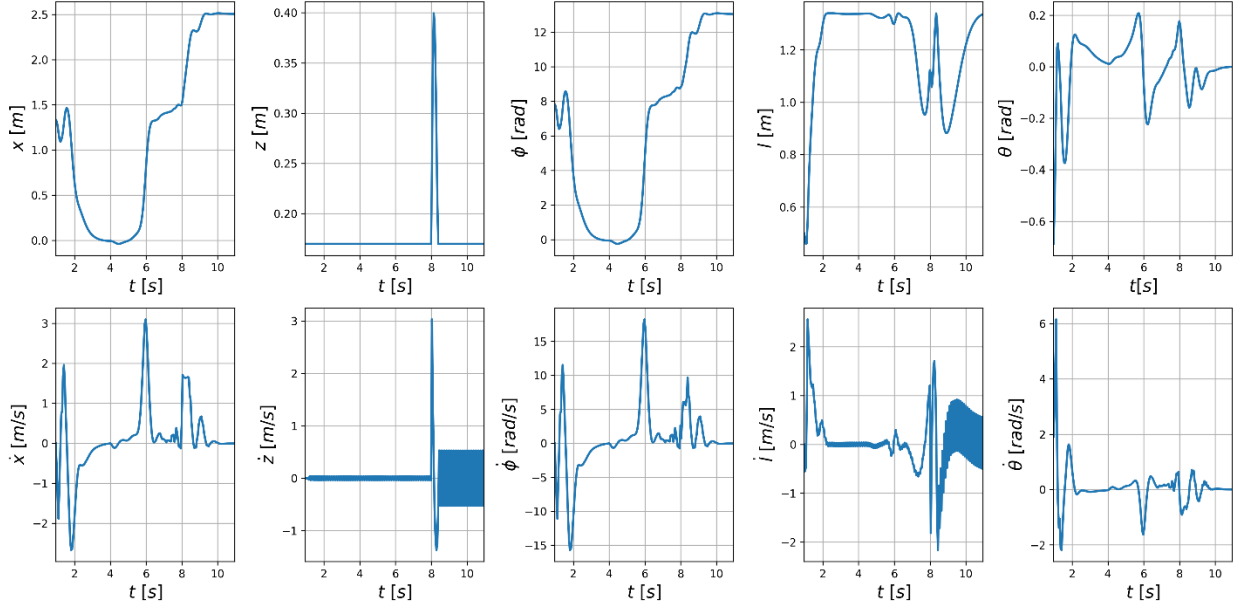


Figure 5.2: Two stage controller switch from the swing up and balance phase mode.

The variable x and ϕ have the same shape unless a scaling factor; in fact, $x = R_w \phi$. At first, the robot will accelerate to reach a target velocity, then it will immediately stop to pass from four wheels to two. It will then move with the aim of balancing, and will jump over the gap situated in $x \in [1.5, 2]m$. From the shape of z , we are able to understand the phase in which the robot has to jump over a gap; in fact, the z variable is almost always zero except for few seconds during the fly. Concerning l , it cannot extend more than $1 + 2R_w = 1.34m$, so we can state that it will keep its maximum extension for most of the time during the simulation. Finally, β oscillates around its target value that is 0 rad. At the bottom part, we have the shape of their derivatives $\dot{\beta}$.

5.3 Proportional Derivatives versus Model Predictive Control

5.3.1 Simplified Model

In this experiment, I am going compare Model Predictive Control (MPC) and Proportional Derivative (PD) controller by considering to control the robot during only the balancing phase, Thus, I am not concerned with the floating phase, and I am free to utilize a reduced mathematical

model of the robot. In this case, I derive a new simplified dynamic model based on Figure 3.2.in which considering the vector of generalized coordinates as:

Then the generalized velocity become:

$$\dot{q} = [\dot{\phi}, \dot{l}, \dot{\beta}]^T$$

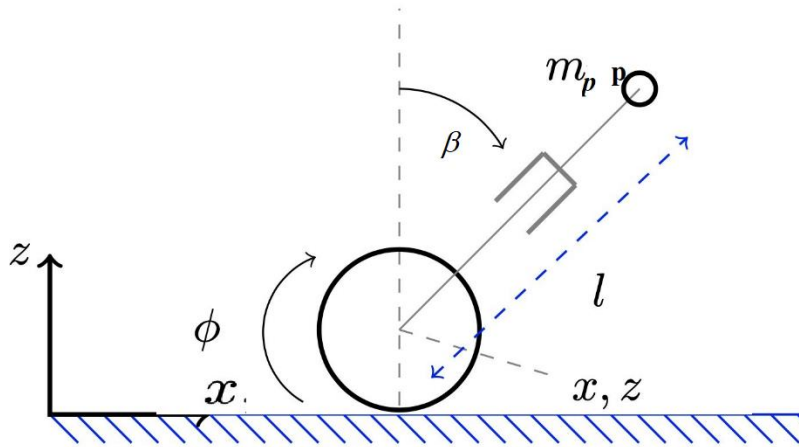


Figure 5.3: VI-WIP on the ground

In this simplified dynamic model I don't have included x and z coordinates as in the dynamic model of section 3.2, because now I am going to keep the z value constant and equal to the radius of the wheel. Actually, I prefer it when the wheel is in direct contact with the ground. Since $x = R_w\phi$, x is not dependent variable.

Here, the new simplified dynamic model is derived using the Lagrangian formulation as previous section 3.2 Variable Length Wheeled Inverted Pendulum (VLWIP) model. By define the position x_p and z_p of the point mass or body mass m_p and its velocity \dot{x}_p and \dot{z}_p .

$$x_p = R_w\phi + l\sin\beta$$

$$z_p = R_w + l\cos\beta$$

$$\dot{x}_p = R_w\dot{\phi} + \dot{l}\sin\beta + l\dot{\beta}\cos\beta$$

$$\dot{z}_p = l \cos \beta - l \beta \sin \beta \dots \dots \dots (5.1)$$

Then, define the kinetic as well as potential energy of the system. The wheel will only have a rotating term, the point mass will only have contribute to translational kinetic energy, and the joint will contain both rotational and translational components of kinetic energy:

$$T = \frac{1}{2} (I_w \dot{\phi}^2 + m_w \dot{x}^2 + m_w \dot{z}^2 + m_p \dot{x}_p^2 + m_p \dot{z}_p^2) \dots \dots \dots (5.2)$$

Due to the gravity acting on the point mass and wheel the potential energy is expressed as:

$$U = m_w g z + m_p g z_p \dots \dots \dots (5.3)$$

Using lagrangian equation (3.1) that are mention in the earlier section 3.2 we can derive the following Equation of Motion (EoM).

$$m(q) \ddot{q} + c(q, \dot{q}) + g = S^T u \dots \dots \dots (5.4)$$

Where $m(q)$ is the inertia matrix:

$$m(q) = \begin{bmatrix} I_w + R_w^2 m_p + R_w^2 m_w & R_w m_p \sin \beta & R_w l m_p \cos \beta \\ R_w m_p \sin \beta & m_p & 0 \\ R_w l m_p \cos \beta & 0 & l^2 m_p \end{bmatrix} \dots \dots \dots (5.5)$$

$c(q, \dot{q})$ is the vector of Coriolis and Centrifugal terms:

$$c(q, \dot{q}) = \begin{bmatrix} R_w m_p \dot{\beta} (2l \cos \beta - l \beta \sin \beta) \\ -l m_p \dot{\beta}^2 \\ 2l l m_p \dot{\beta} \end{bmatrix} \dots \dots \dots (5.6)$$

G is the gravity matrix:

$$c = \begin{bmatrix} 0 \\ gm_p \cos \beta \\ -glm_p \sin \beta \end{bmatrix} \dots\dots\dots(5.7)$$

S is the selection matrix:

$$S = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \dots\dots\dots(5.8)$$

The control input is the same to the earlier section:

$$u = \begin{bmatrix} \tau \\ f \end{bmatrix} \dots\dots\dots(5.9)$$

Consequently, the prismatic joint's linear force and actuation torque are considered in this phase, but this time I haven't expressly taken the contact forces into account.

5.3.2 Comparison of PD and MPC

After testing the MPC controller, we decided to use another kind of controller in order to make a comparison; in particular, I implemented a PD control law. For this experiments I simplify the robot's task and by consider only a balancing operation. The gains are tuned in order to obtain behaviors that are as similar as possible between the MPC and the PD controller. Then run both the simulations for 10 seconds; the initial configuration of the robot is $q_0 = [0, 0.5, \frac{\pi}{8}]^T$, while

the target one is $q_f = [-\frac{2}{R_w}, 0.9, 0]^T$ Even if the generalized coordinates of the model are only

ϕ , l and β then plot also the values of x and z ; the former is obtained through the relation $x = \phi R_w$, while the latter is always constant and equal to R_w .

The results obtained with the MPC controller:

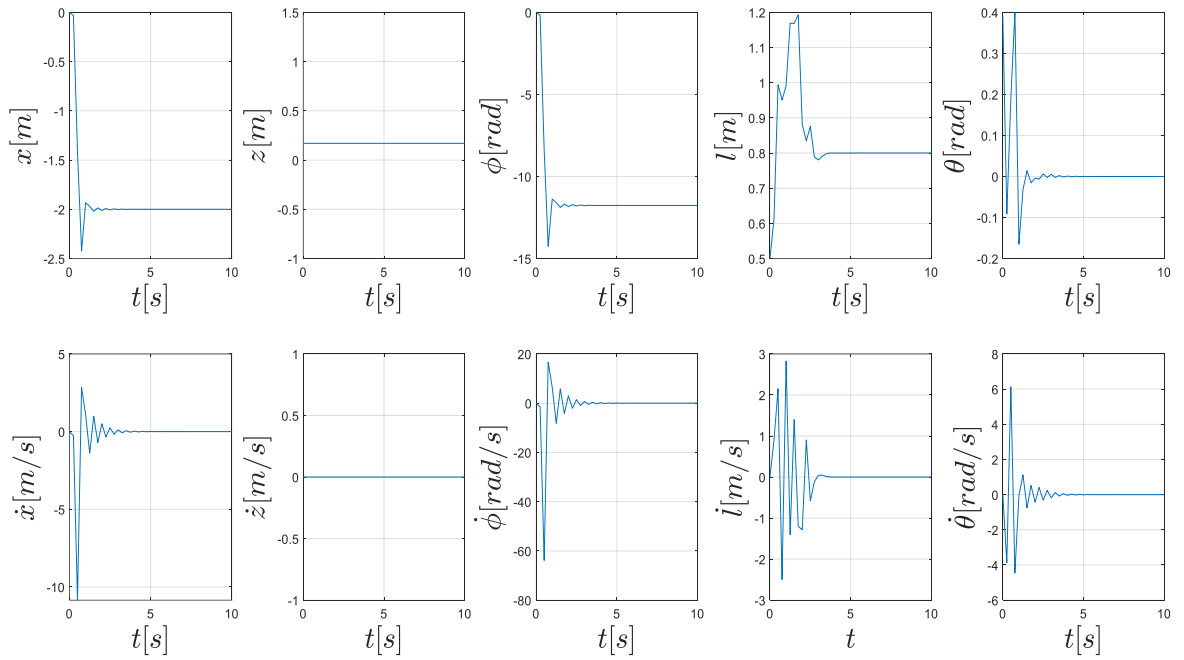


Figure 5.4: State variables response – MPC

And the output obtained with the PD controller

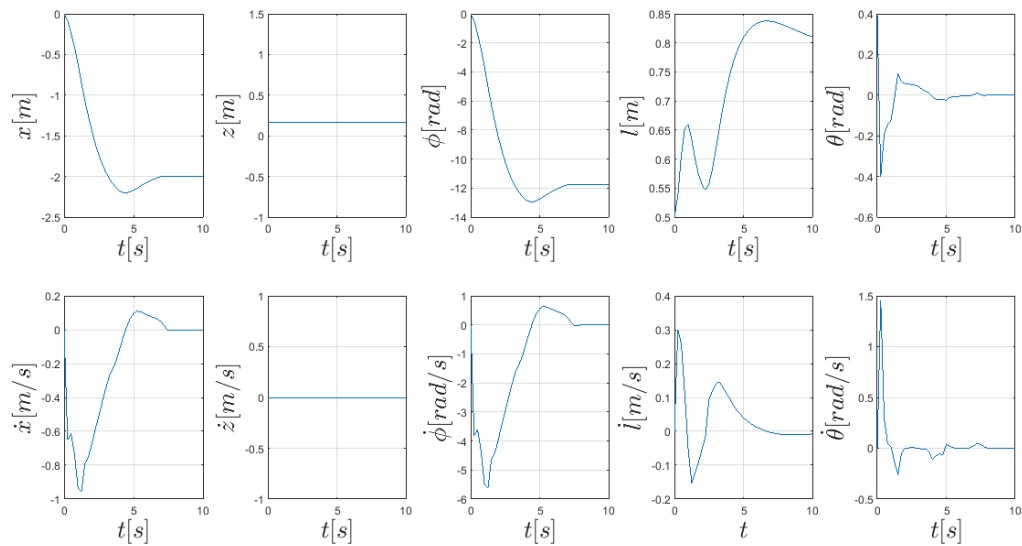


Figure 5.5: Closed loop response for PD controller

As we see from the above closed loop response of MPC and PD controller figure 5.7 and figure 5.8 have different control performance. From these plots we can see that the MPC controller is faster in reaching the desired; in fact, the state values stabilize after about 3.44 seconds of simulation, while for the PD controller would have had to run the simulation longer in order to reach a steady state. As expected, since the behavior with the PD controller is slower, the values of the control input τ are a little smaller compared to the ones for the MPC. However, in both cases l reaches the desired value in few seconds; this is reflected in the values of the control input f , which are similar in the two performances. Finally, I compare the performance to analyze data about the performance two cases in terms of β :

<i>States</i>	<i>Controller</i>	<i>Overshoot/undershoot</i>	<i>Settling time</i>	<i>Rise time</i>	<i>Peak time</i>
β	PD	-0.45	5.44	0.5	1
	MPC	0.4/-0.43	3.51	0.01	1

Table 5.1 Control performance PD vs MPC

CHAPTER SIX

RECOMMENDATION AND CONCLUSION

6.1 Recommendation

The findings of this thesis reveal that, despite its simplicity, the robot model is capable of producing complicated dynamic movements. This integrated pipeline of motion planning was created as a first step using the dynamics of two wheeled jumped robot, which are not often taken into account. I advise for future work to use the standard motion planning, control pipeline as well as the template model. This method is equivalent to using spring loaded inverted pendulum (SLIP) models for each leg on a biped.

In addition, the planner utilized the system dynamics for each motion without receiving explicit instructions to do so. In particular, it discovered that the preparatory movements for this type of robot were compressing and expanding the robot before takeoff, absorbing the shock upon landing, and extending the prismatic joint for simpler balance.

Basically in this thesis the switching times mechanism have predefined for the two phase case. In the future I recommended this will be to transfer from switching to automatically discovered, for example via phase-based parameterization. Finally, I recommended that transfer this approach to real world and to use better optimization toolbox packages and open source in order to increase the performance. And also recommend to show robustness of controller by introducing uncertainty noise.

6.2 Conclusion

The main objective of this thesis to control the two wheeled jumped robot to perform a task requiring hopping and balancing. I have shown that this system can produce complicated motions despite having extremely basic nonlinear dynamics model. With the nonlinear Model Predictive Control, I can able to manipulate it extremely precisely, achieving high accuracy motions throughout each phase. Additionally, in the balancing phase, I also compared the control characteristics performance of Model Predictive Control (MPC) results to those achieved with a Proportional Derivative controller (PD). I am also able to complete the intended task in this

instance. Anyhow, using the PD controller, I had to empirically adjust the gains by determining which configuration was most effective for our purpose. On the other hand, by simply incorporating limitations in the design of the issue, the MPC allowed us to more easily achieve even more challenging tasks (such as jumping). One of the drawbacks of MPC was that it needs more overshoot to perform the same task with respect to PD controller. In fact, the control action found by the MPC is the results of an optimization problem.

RESEARCH FUND ACKNOWLEDGMENT

This research thesis was funded by Adama Science and Technology University under the grant number ASTU/SM-R/540/22, Adama, Ethiopia.

REFERENCES

- A *fastenengineering*. (2022, 01 27). Retrieved from A fastenengineering website: <https://fastenerengineering.com/what-is-a-prismatic-joint>
- A. Nasir, M. A. (2010). “The control of a highly nonlinear two-wheels balancing robot: A comparative assessment between lqr and pid-pid control schemes. *World Academy of Science, Engineering and Technology*, vol. 70, , 227–232.
- A. Saunders, C. E. (2012.). Environmentally sealed combustion powered linear actuator. *US Patent US9 238 967B2*,.
- D. E. Orin, A. G.-H. (Oct. 2013.). “Centroidal dynamics of a humanoid robot,” . *Autonomous Robots*, vol. 35, no. 2, 161–176,.
- D., I. K. (2010). *Design of Two-Wheeled Twin Rotored Hybrid Robotic Platform*. Ankara: M.Sc Thesis, Atılım University.
- Diehl, J. A. (2015). *User Documentation for CasADi v2.2.0*.
- Dynamics, B. (2019). *ieee* . Retrieved from ieee web site: <https://www.ieee.org/>
- electronicscoach. (2022, 9 22). *electronicscoach*. Retrieved from electronicscoach.: <https://electronicscoach.com/proportional-derivative-controller.html>
- Geyer., T. (Nov. 2016.). *Model predictive control of high power converters and industrial drives*, . London: Wiley.
- H. Bock and K. Plitt. (1984.). A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems *. In pp., *IFAC Proceedings Volumes* ,”, vol. 17, no. 2 (pp. 1603–1608,).
- Hayden, E. C. (30 September 2013). Rewired nerves control robotic leg. *Nature*.
- J, B. Rawlings, D. Q. Mayne, M. M. Diehl. (October 2017). *Model Predictive Control: Theory, Computation, and Design 2nd Edition*. United States of America: Nob hill publishing, Madison, Wisconsin,.
- M. Bjelonic, P. K. (Jan. 2020). Rolling in the Deep – Hybrid Locomotion for Wheeled-Legged Robots using Online Trajectory Optimization. , *arXiv:1909.07193 [cs, eess]*.
- M. Geilinger, R. P. (July 2018.). “Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels,”. *ACM Transactions on Graphics*, vol. 37, no. 4, 1–12.

- M. Hutter, C. G. (2017.). Anymal-toward legged robots for harsh environments. *Advanced Robotics*, vol. 31, no. 17,, pp. 918–931,.
- M. Hutter, R. D. (2017). towards a generic solution for inspection of industrial sites. *11th Conference on FSR*.
- M. Kamel, S. V. (2018.). “Voliro: An omnidirectional hexacopter with tiltable rotors,.” *Computing Research Repository (CoRR)*, vol. abs/1801.04581.
- M. O. Asali, F. H. (2017). Modeling, simulation, and optimal control for two-wheeled self-balancing robot. *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 4, 2008–2017,.
- Michèle Arnold, G. A. (August 22-26, 2011). Model Predictive Control of energy storage including uncertain forecasts. *7th Power Systems Computation Conference (PSCC 2011)*,. Stockholm, Sweden: 17th Power Systems Computation Conference .
- Morin, D. (2007). The Lagrangian Method. In D. Morin.
- Nikolaou, M. (2001). Advances in Chemical Engineering,. In *Model predictive controllers, A critical synthesis of theory and industrial needs, Volume 26* (pp. 131-204). Academic Press.
- Nur Uddin, p. f. (November 2017.). “Stabilizing Two-Wheeled Robot Using Linear Quadratic Regulator and States Estimation”. *Research project on Autonomous Robot in 2017*.
- Online, S. A. (n.d.). *Bostondynamics corporation*. Retrieved 07 29, 2018, from Bostondynamics corporation Website: <https://www.bostondynamics.com/spot-mini>
- Orin, P. M. (may, 2014). “Development of high-span running long jumps for humanoids. *IEEE* (pp. 222-227). IEEE International Conference on Robotics and Automation (ICRA),.
- P. M. Wensing and D. E. Orin, i. 2. (May 2014). Development of high-span running long jumps for humanoids. *IEEE International Conference on Robotics and Automation (ICRA)*, 222–227.
- R. P. M. Chan, K. A. (Apr. 2013.). “Review of modeling and control of two-wheeled robots,” . *Annual Reviews in Control*, vol. 37, no. 1, , 89–103.
- R. Siegwart and I. R. Nourbakhsh, A. M. (2004). Autonomous Mobile Robots. *The MIT Press Cambridge, Massachusetts*,.
- Traiko Dinev, S. X. (2020). Modeling and Control of a Hybrid Wheeled Jumping Robot. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE.

- V. Klemm, A. M. (2019). "Ascento: A Two-Wheeled Jumping Robot,". *International Conference on Robotics and Automation (ICRA)* (pp. 7515-7521). Canada: IEEE.
- V. Leek, V. (2016). "An Optimal Control Toolbox for MATLAB Based on CasADi,".
- Vichik, S., & Borrelli, F. (2014). "Solving linear and quadratic programs with an analog circuit". In U. o.-1. Department of Mechanical Engineering, *Computers & Chemical Engineering*. 70 (pp. 160–171). Francesco: doi:10.1016/j.compchemeng.2014.01.011.
- Wang, L. (2009.). *Model Predictive Control System Design and Implementation Using MATLAB®*. Springer Science & Business Media.
- wikibooks. (2022, August 2). *wikibooks*. Retrieved from wikibooks web site: https://en.wikibooks.org/wiki/Robotics/Types_of_Robots/Wheeled

Appendix: MATLAB Code

Appendix I: main for phase1

```
% main for phase1
clearall
closeall
clc

mb = 4;
mw = 2;
Rw = 0.17;
Iw = (mw*Rw^2);

% dyn = dynamics_phase1model(mb,mw,Iw,Rw);
phase1utils_obj=phase1utils(Rw);

%% Swing-up and Balance

nx = 4;
ny = 4;
nu = 1;

nlobj = nlmpc(nx,ny,nu);

Duration=6;
%p = 5;
Ts = 0.25;
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 20;
nlobj.ControlHorizon = 5;

nlobj.ManipulatedVariables(1).Min = -10;
nlobj.ManipulatedVariables(1).Max = 10;

% nlobj.States(1).Max=pi/2;
nlobj.States(2).Max=0;
nlobj.States(2).Min=-pi/2;

nlobj.Optimization.CustomCostFcn = @(X,U,e,data)
phase1utils.cost_func(X,U,e,data);

nlobj.Optimization.CustomEqConFcn = @(X,U,data)
phase1utils_obj.eqConfunction(X,U,data);
```

```

% nlobj.Optimization.CustomIneqConFcn = @(X,U,e,data)
phaselutils_obj.ineqConfunction(X,U,e,data);
nlobj.Model.StateFcn= @(X,U)dynamics_phase1model1(X,U);
% nlobj.Model.StateFcn = @(X,U) dyn.next_state(X,U);
nlobj.Model.IsContinuousTime = true;
%% Setting Weights

% We are only interested in theta and dx
% (because we want the car to drive horizontally)
% W = [5 50];
W = [1000 1000 1 0];
% The last two inputs are the ground reaction forces
% We don't want to weight them
nlobj.Weights.OutputVariables = W;
nlobj.Weights.ManipulatedVariables = 1;

%%
% nlobj.Model.OutputFcn = @(x,u)
[x(1);x(2);x(3);x(4);x(5);x(6)];

x_d=[0 0 0 0];
% x_d =[1.5 Rw 1.5/Rw 0.5 0 0 0 0 0 0];
% x_d2 =[2 Rw 2/0.17 0.5 0 0 0 0 0 0];
% x_d3 =[2.5 Rw 2.5/0.17 0.5 0 0 0 0 0 0];

v0=7;
x0 = [2/Rw;-pi/2;v0/Rw;0];
u0 = 0;

validateFcns(nlobj, x0, u0, []);

nloptions = nlmpcmoveopt;

time = 0;

%% Closed loop
xk=x0;
uk=u0;
xHistory = x0';
uHistory = u0';

nlobj.Optimization.ReplaceStandardCost = false;

for ct = 1:(Duration/Ts)
tic
disp(['iteration.. ',num2str(ct),'/', num2str(Duration/Ts)])

```

```

        [uk,nloptions, info]=
nlpmove(nlobj,xk,uk,x_d,[],nloptions);
xk=info.Xopt(2,:);
%uk=info.MVopt(2,:);
uk
xHistory = [xHistory; xk];
uHistory = [uHistory; uk'];
time = time+Ts;
toc
end
% time=time
%% Plots

time_int=[0:Ts:Duration];

% addpath('C:\results')
% formatOut = 'yyyy.mm.dd-HH:MM:SS';
% name = datestr(datetime('now'),formatOut);
% mkdir('C:final\results',name)

phaselutils.plot_state(xHistory,time_int)
saveas(gcf, strcat('results/',name, '/plot_state.png'))
saveas(gcf, strcat('results/',name, '/plot_state.fig'))

phaselutils.plot_control(uHistory,time_int)
saveas(gcf, strcat('results/',name, '/plot_control.png'))
saveas(gcf, strcat('results/',name, '/plot_control.fig'))

closeall
utils.createVideo(0.17*xHistory(:,1), 0.17, xHistory(:,2),
xHistory(:,3), round(1/Ts), strcat('results/',name))

function dxdt = dynamics_phaselmodell(x,u)
%% Continuous-time nonlinear dynamic model of a pendulum on a
cart
%
% 4 states (x):
% 1 inputs: (u)
% force (F): when positive, force pushes cart to right
%
% Copyright 2018 The MathWorks, Inc.

%#codegen

%% parameters
mw = 2; % cart mass
mb = 4; % pendulum mass

```

```

g = 9.81;    % gravity of earth
l = 0.5;% pendulum length
Rw = 0.17;
Iw = mw*Rw*Rw;
mt = mw+mb;
% Kd = 10;    % cart damping
%% Obtain x, u and y
% x = [phi; theta; phi_dot; theta_dot];
%     phi = x(1);
theta = x(2);
    phi_dot = x(3);
    theta_dot = x(4);

tau = u;
%% Compute dxdt
phidd=tau/(Iw+mt*Rw);
dxdt = vertcat(phi_dot,...
               theta_dot,...
               phidd,...
               (1/mb*l*l)*(-mb*Rw*phidd*l*sin(theta) +
               mb*g*l*cos(theta) - tau)...
               );

```

Appendix II: main for phase2

```

clearall
closeall
clc

mb = 4;
mw = 2;
Rw = 0.17;
Iw = (mw*Rw^2);

dyn = dynamics_model(mb,mw,Iw,Rw);
utils_obj=utils(Rw);

nx = 10;
ny = 10;
nu = 4;

nlobj = nlmpc(nx,ny,nu);

Duration=6;
%p = 5;

```

```

Ts = 0.25;
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 5;

nlobj.ManipulatedVariables(1).Min = -10;
nlobj.ManipulatedVariables(2).Min = 0;
nlobj.ManipulatedVariables(1).Max = 10;
nlobj.ManipulatedVariables(2).Max = 0;
nlobj.ManipulatedVariables(3).Min = 0;
nlobj.ManipulatedVariables(3).Max = 0;
nlobj.ManipulatedVariables(4).Max = 0;
nlobj.ManipulatedVariables(4).Min=0;

nlobj.States(2).Max=Rw;
nlobj.States(2).Min=Rw;
nlobj.States(4).Max=0.5;
nlobj.States(4).Min=0.5;
nlobj.States(5).Max=pi/2;
nlobj.States(5).Min=-pi/2;

nlobj.Optimization.CustomCostFcn = @(X,U,e,data)
utils.cost_func(X,U,e,data);

nlobj.Optimization.CustomEqConFcn = @(X,U,data)
utils_obj.eqConfunction(X,U,data);
nlobj.Optimization.CustomIneqConFcn = @(X,U,e,data)
utils_obj.ineqConfunction(X,U,e,data);

nlobj.Model.StateFcn = @(X,U) dyn.next_state(X,U);
nlobj.Model.IsContinuousTime = true;
%% Setting Weights

% We are only interested in theta and dx
% (because we want the car to drive horizontally)
W = [1000 100 0 0 1000 0 0 0 0 0];
W2 = [1000 100 0 0 1000 0 0 0 0 0];
% The last two inputs are the ground reaction forces
% We don't want to weight them
nlobj.Weights.OutputVariables = W;
nlobj.Weights.ManipulatedVariables = [1 0 0 0];

%%
nlobj.Model.OutputFcn = @(x,u)
[x(1);x(2);x(3);x(4);x(5);x(6);x(7);x(8);x(9);x(10)];

yref=[2.5 Rw 2.5/0.17 0.5 0 0 0 0 0 0];

```

```

x_d =[1.5 Rw 1.5/Rw 0.5 0 0 0 0 0 0];
% x_d2 =[2 Rw 2/0.17 0.5 0 0 0 0 0 0];
% x_d3 =[2.5 Rw 2.5/0.17 0.5 0 0 0 0 0 0];

v0=3;
x0 = [-2/Rw;Rw;v0/7;0.5;-pi/2; 0;0;0;0;0];
u0 = [0;0;0;0];

validateFcns(nlobj, x0, u0, []);

nloptions = nlmPCMmoveopt;

time = 0;
tf1 = 2.5;
tf2 = 3;
current_phase = 1;

%% Closed loop
xk=x0;
uk=u0;
xHistory = x0';
uHistory = u0';

nlobj.Optimization.ReplaceStandardCost = false;

for ct = 1:(Duration/Ts)
tic
phase = utils.temporal_phase(time,tf1,tf2);
if current_phase ~= phase
    current_phase = phase;
if phase == 2
    nlobj.Weights.OutputVariables = W2;
nlobj.ManipulatedVariables(3).Min = 0;
nlobj.ManipulatedVariables(4).Min = 0;
nlobj.ManipulatedVariables(3).Max = 0;
nlobj.ManipulatedVariables(4).Max = 0;
nlobj.States(1).Min=1.5;
nlobj.States(2).Max=2*Rw;
nlobj.States(2).Min=Rw;
%         nlobj.Weights.OutputVariables = W2;
        x_d = [2 Rw 1.5/Rw 0.5 0 0 0 0 0 0];
disp('Phase 2')
elseif phase == 3
nlobj.ManipulatedVariables(3).Min = -inf;
nlobj.ManipulatedVariables(4).Min = 0;
nlobj.ManipulatedVariables(3).Max = inf;
nlobj.ManipulatedVariables(4).Max = inf;

```

```

nlobj.States(1).Min=2;
nlobj.States(2).Max=Rw;
nlobj.States(2).Min=Rw;
        nlobj.Weights.OutputVariables = W;
        x_d = [2.5 Rw 1.5/Rw 0.5 0 0 0 0 0 0];
disp('Phase 3')
end
end

disp(['iteration.. ', num2str(ct), '/', num2str(Duration/Ts)])
    [uk, nloptions, info]=
nlpmove(nlobj, xk, uk, x_d, [], nloptions);
xk=info.Xopt(2, :);
uk=info.MVopt(2, :);
uk
xHistory = [xHistory; xk];
uHistory = [uHistory; uk];
time = time+Ts;
toc
end
% u1 = uHistory(end,1:4)
%% Plots

time_int=[0:Ts:Duration];

% addpath('results')
% formatOut = 'yyyy.mm.dd-HH:MM:SS';
% yusuf= datestr(datetime('now'), formatOut);
% mkdir('results', name)

utils.plot_state(xHistory, time_int)
saveas(gcf, strcat('results/', '/plot_state.png'))
saveas(gcf, strcat('results/', '/plot_state.fig'))

utils.plot_control(uHistory, time_int)
saveas(gcf, strcat('results/', '/plot_control.png'))
saveas(gcf, strcat('results/', '/plot_control.fig'))

% close all
% utils.createVideo(xHistory(:,1), xHistory(:,2), xHistory(:,4),
xHistory(:,5), round(1/Ts), strcat('results/', name))

```

Dynamic model function

```

classdef dynamics_model
%DYNAMIC_MODEL Summary of this class goes here

```

```

% Detailed explanation goes here

properties
mb
mw
mt
Iw
g
    Rw
end

methods

function obj = dynamics_model(mb,mw,Iw,Rw)
    obj.mb = mb;
    obj.mw = mw;
    obj.mt = mb + mw;
    obj.Iw = Iw;
    obj.g = 9.81; % [m/s^2]
    obj.Rw = Rw;
end

function M = mass_matrix(obj,q)

    M = [obj.mt                0
0      obj.mb*sin(q(5))  obj.mb*q(4)*cos(q(5));
        0                obj.mt
0      obj.mb*cos(q(5)) -obj.mb*q(4)*sin(q(5));
        0                0
obj.Iw        0                0;
        obj.mb*sin(q(5))        obj.mb*cos(q(5))
0      obj.mb                0;
        obj.mb*q(4)*cos(q(5)) -obj.mb*q(4)*sin(q(5))
0      0                obj.mb*q(4)^2];
end

function C = coriolis_matrix(obj,q,dq)

    C = [0 0 0  2*obj.mb*cos(q(5))*dq(5) -
obj.mb*q(4)*sin(q(5))*dq(5);
        0 0 0 -2*obj.mb*sin(q(5))*dq(5) -
obj.mb*q(4)*cos(q(5))*dq(5);
        0 0 0                0
0;
        0 0 0                0
obj.mb*q(4)*dq(5);
        -

```

```

                                0 0 0      2*obj.mb*q(4)*dq(5)
0];

end

function G = gravity_vector(obj,q)

    G = [0 obj.g*obj.mt 0 obj.g*obj.mb*cos(q(5)) -
obj.g*obj.mb*q(4)*sin(q(5))]' ;

end

function S = get_S(obj)

    S = [0 0 1 0 -1;
         0 0 0 1 0];

end

function Jc = get_jacobian(obj)

    Jc = [1 0 -obj.Rw 0 0;
          0 1      0  0 0];

end

function ddq = get_ddq(obj,q,dq,u,lambda)
    M = obj.mass_matrix(q);
    C = obj.coriolis_matrix(q,dq);
    G = obj.gravity_vector(q);
    S = obj.get_S();
    Jc = obj.get_jacobian();

    ddq = M^-1*(S'*u+Jc'*lambda-C*dq-G);

end

function dx = next_state(obj,x,u)

    ddq = obj.get_ddq(x(1:5),x(6:10),u(1:2),u(3:4));

    dx = [x(6:10);ddq];
    %      dx = x + dx*0.1;

end

```

end
end