A NEW FOOTSTEP PLANNING FOR SLIP AND TD-SLIP MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Serkan İSLAMOĞLU
December 2020

A New Footstep Planning for SLIP and TD-SLIP Models
By Serkan İSLAMOĞLU
December 2020

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Ömer Morgül(Advisor)

Hitay Özbay

İsmail Uyanık

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

A NEW FOOTSTEP PLANNING FOR SLIP AND TD-SLIP MODELS

Serkan İSLAMOĞLU
M.S. in Electrical and Electronics Engineering
Advisor: Ömer Morgül
December 2020

Spring Loaded Inverted Pendulum (SLIP) is a well-known model and an accurate descriptive tool, which can scientifically represent the dynamics of the legged locomotion. Torque actuated Dissipative SLIP (TD-SLIP), on the other hand, is fundamentally an enhanced version of the SLIP model. Inclusion of more realistic damping model and the hip torque actuation has led the researchers to develop a sufficiently better analytic approximation. This thesis proposes a new methodology to achieve footstep planning on the SLIP and TD-SLIP models, distinctly. It contributes a novel planning algorithm by utilising the constructed touchdown-to-touchdown map, and a novel recursive function to plan and execute the planning. The thesis provides a background information about the modelling and simulation of both of the used models, and an auxiliary function, which administers a derivative-free method to calculate the minimum of an input function. After defining the problems and the corresponding proposed solutions, the foundations of the preparation phase is established. This phase is fundamentally constructed to accumulate required information for the algorithm implementation and simulation phase. The main phase consists of subsections, which can be composed of the combination of following properties; planning type, as online and offline, policy type; as forward and backwards and output type; as based on distance or based on minimum step count. According to the stated problem, the planning is successfully realised not only for a single desired distance, but also an array of waypoints. In addition to this, the presented illustrations of different initial states show that the planning can also be constructed via any different initial touchdown state. Therefore, the obtained results are quite promising, since all of the cases and their combinations successfully reach the destinations with a negligible error value, which is less than 1%. Although, the offline planning type provides the results in a rapid way, the obtained data to use the plan requires much more space,

which also increases dramatically when the step count (level) is incremented. In addition to this, the forward planning is faster than the backwards one, but they both generate very similar results.

Keywords: SLIP dynamics, Online - Offline Planning, Forward - Backwards Planning, Footstep Planning, Legged Locomotion.

ÖZET

SLIP VE TD-SLIP MODELLERİ ÜZERİNE ADIM PLANLAMASI

Serkan İSLAMOĞLU
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Ömer Morgül
Aralık 2020

Yay Yüklü Ters Sarkaç (SLIP) bacaklı hareket yeteneğini bilimsel olarak temsil eden geniş akademik alanlarda kabul görmüş hassas tanımlayıcı bir modeldir. Bir diğer yandan, Tork ile işletilmiş dağıtıcı sarkaç ise (TD-SLIP) yay yüklü ters sarkacın temelde geliştirilmiş bir sürümüdür. Daha gerçekçi sönümlendirilmiş model ve kalçadan uygulanan torkun bu sisteme dahil olmasıyla modelin daha gerçekçi ve daha iyi analitik yaklaşımları geliştirilmiştir. Bu tez adım planlamasının hem SLIP hem de TD-SLIP ortamlarında nasıl sağlandığını göstermektedir. Hazırlanan yere-dokunuştan yere-dokunuşa fonksiyonlar ile yeni bir planlama algoritması ortaya koyan bu tez, aynı zamanda tekrarlanan bir fonksiyon ile planlamaya da katkı sağlar. Tezi daha detaylı anlayabilmek için geçmişteki çalışmaları içeren gerekli bilgilerle başlayıp, simülasyon ortamındaki hesaplamalar için türevsiz bir yöntemle verilen bir fonksiyonun çıktısını en küçük seviyede sunan bir yardımcı bir fonksiyonuna değinilir. Ardından hangi problemlerin çözüleceği ile ilgili detaylı bir açıklamayla devam edilip aynı problemlere uygulanan çözümleri de açıklığa kavuşturulur. Devamında, simülasyon öncesinde yapılan işlemlerin anlatımına geçilip, ana amacı simülasyon sırasında kullanılmak üzere verilerin nasıl oluşturulduğuyla ilgili detaylı bilgiler verilmiştir. Simülasyon aşaması alt kısımlardan oluşmaktadır. Bunlar çevrimiçi veya çevrimdışı plan oluşturulması, ileriye doğru veya tersten oluşturulmuş prensip seçimi ve en kısa mesafeye veya en düşük adım sayısına göre çıktı tipidir. Sağlanan figürler sadece tek bir hedef pozisyonun ulaşılmasına yönelik olmayıp aynı zamanda birden fazla ara noktaların da erişimine yöneliktir. Buna ek olarak, aynı hedef noktaya ulaşmak amacıyla farklı başlangıç durumlarının çizimlemeleri de gösterilmiştir. Bütün durumlarda ve onların birbirleriyle oluşturdukları kombinasyonlar hedeflerine %1'den az yanlışlık payıyla ulaştığından alınan sonuçlar oldukça umut verici sayılabilir. Çevrimdışı planlama çok daha hızlı sonuçlar üretmesine rağmen

elde edilen verinin kullanılması için bir o kadar fazla alana ihtiyaç vardır. Bu alan aynı zamanda adım sayısının artmasıyla beraber artmaktadır. Buna ek olarak, ileriye dönük planlama geriye dönük olana göre biraz daha hızlı sonuçlar üretmiş olup her iki yöntem de hedef pozisyona çok yakın noktalara gitmeyi başarmıştır.

Anahtar sözcükler: SLIP dinamikleri, Çevrimiçi - Çevrimdışı Planlama, İleriye doğru - Geriye doğru Planlama, Adim Planlamalari, Bacakli Hareket Yeteneği.

Acknowledgement

I would like thank to my supervisor Ömer Morgül for his guidance and providing me this great opportunity to work on. I would extend my gratitude with my friends; Ezgi Abıcılar for being my everything on everything. My study partner Ahmet Safa Öztürk, and my other always-there friends Giray İlhan, Ahmet Can Varan, Çetin Taştekin, Barış Coşkun, Kuter Kırımer, Kaan Yilmam and Arda Ata Erdogdu.

Also, special thanks for Ismail Uyanik and Hasan Hamzacebi for helping me out to construct the fundamentals of my work. Their supervision and advice led me to figure out beneficial solutions.

Finally, I always thank to my family, who are always there to help me when I am in need. My parents; Tunç İslamoğlu and Yeşim Büyükmeriç, my aunt Filiz İslamoğlu, and my little and beloved sister Ece İslamoğlu.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spring Loaded Inverted Pendulum (SLIP) model's foundations were established in the early years of 1990s [1]. It has been widely accepted in the literature, and it provides a beneficial model on animal locomotion[2, 3, 4], and ensures the capabilities of robust dynamic locomotion[5, 6, 7]. Animal locomotion on widely different sized morphologies were recognized as the center-of-mass movement pattern, and the discovery of these movement patterns led the researches to develop simple yet accurate SLIP models. The descriptive and dynamical model utilises the locomotion behavior by adopting running, walking and leaping modes, in the light of providing speed, agility, and efficiency[8, 9, 10]. Therefore, it was expected that the well-studied model encourage the development of other different structured, but similar logical models[11].

The Torque actuated Dissipative SLIP (TD-SLIP), on the other hand, was established to overcome the structural deficits and problematic model assimilation of the SLIP model[12]. TD-SLIP incorporates damping and hip torque actuation with the constant energy SLIP model. The comparison between two models clarified in the scientific work, which states the significance of the damping factor on human and animal modeling, and the use of torque to decrease the energy dissipation[12]. Although the analysis and control involves difficult challenges, from an implementation point of view, the enhanced model is actually more

realistic.

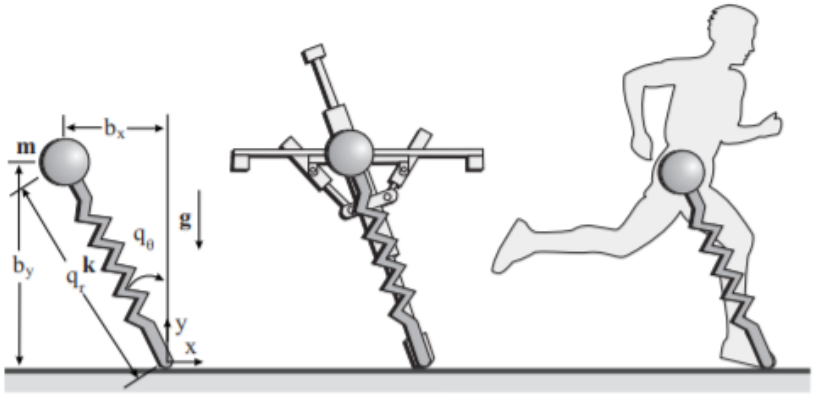

Figure 1.1: SLIP Model, Raibert's Hopper, A human runner

Using the principles of Lagrangian dynamics, the derivation of physics based mathematical models can be achieved. These methods are interestingly effective on describing center of mass trajectories of the legged locomotion[13, 14, 15]. The expression is not only limited as formulations, but also influenced the real physical robotic systems such as [16, 17, 18]. Additionally, the state-space models [19, 20] and data-driven analogies to utilise transfer functions [21, 22, 23, 24] are also studies to use input and output data for legged locomotion. Differently, central pattern generator approach of the models are also known in the literature so that the investigation of the legged locomotion extends in a broader pattern [25, 26, 27]. Many researchers have been anchored SLIP model template to achieve more complex models for running with legged robots so that its practicality can be increased [28, 29, 30, 31]. In addition to this, as the inclusion of the damping in the leg causes the system to energy loss, in order to compensate it a single linear actuator is combined with the model [32, 33], and its validated in [34] by modelling the muscle activation. The activation includes injection of energy during stance phase with a force-free leg length activation. Since the addition of a linear actuator increases the mass on the robot leg, studies enlighted the possible problem, and stated that it has a minor effect on system trajectories, although it affects the dynamics [35]. In the light of these the fundamentals of the footstep planning has successfully came into the picture [36, 37].

This thesis focuses on the footstep planning on the SLIP and TD-SLIP models.

The study enlightens the details behind how planning phase is constructed and execution stage is processed on the simulation environment. The planning phase is consisted of two fundamental chapters. The very first one is before the simulation (i.e preparation) phase, and the other one is the path construction phase. The preparation stage can be considered as the data accumulation part, in which the model's single steps' possible reach sets and their corresponding goal domains (possible goal sets) are combined as an array. The constructed goal domains are selected from a safe guard region, where the minimal input changes do not affect the implemented model to attain greater controllable input changes, and they are assigned as the target vectors. Each target vector becomes a goal vector for a calculated reach vector set, and all of the possible reach sets and their goal domains constitute a single cloud domain. Each cloud domain, which is constructed upon another actually covers a larger area. Therefore, as constructing a cloud takes time, the number of clouds and the region incremental values can actually be up to a predefined set of parameters. As preparation chapter is processed distinctly, the cloud construction part does not go along with the simulation. However, the gathered information is used within the simulation. On the other hand, the planning stage consists of main subsections, which are the online and offline planning construction types. The keyword online means the planning is constructed after the simulation is started, with respect to the initial state, whereas the keyword offline states another process in between the preparation and simulation phases to construct a plan for all possible starting states. Both of the implementation types includes the forward and backwards path generation, and both of the generation types are consisted of the planning based on the distance and step count. The main purpose behind the planning with respect to distance is to minimize the error to reach the target position, and the aim behind the step count is to minimize the step count as much as possible to reach the closest desired position. Additionally, the algorithm is executed with different initial states, and the waypoint logic also implemented for both of the models. All of the mentioned different planning executions are successfully achieved with at most 0.1% positional error, and their corresponding results are provided in the related chapter.

The main contributions of this thesis are as follows. First, it provides a novel

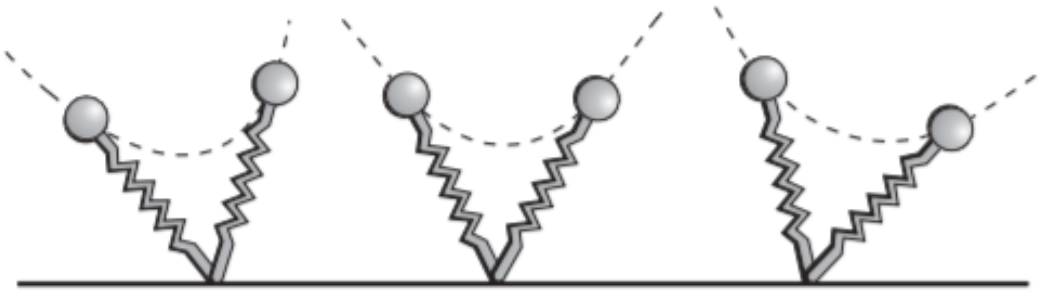

Figure 1.2: Gravity effect on angular momentum at the end of the stance phase compared to touchdown instant; decreasing effect on magnitude, symmetric stride, increasing effect on magnitude

way to plan the footstep of the well studied models in the literature, regardless of the initial starting state. The planning is not only successfully achieved by providing a single goal distance, but also an array of desired distances. Secondly, the cloud domain term is proposed, which can actually holds the required information to connect the successive model steps. Additionally, for the algorithm implementation part, comparisons between the online, which is the process after the simulation started, and offline, which is the procedure in between the simulation and preparation phase, plannings stated. The statement includes the selection of which one should choose with respect to their requirements. Also, the comparison between the forward and backwards planning is provided so that to understand and observe which one reaches better results. Finally, outputting the results based on minimum distance error or based on minimum step count is investigated. Therefore, a choice can be made according to the relevant situation, which can be reaching as close as it can, or reaching with smaller energy consumption.

This thesis organized as follows. Chapter 2 provides beneficial background information about the SLIP and TD-SLIP models. It includes the general information, their locomotion phases and the phase dynamics for both of the models. Chapter 3 defines the problem and its possible proposed solution. Chapter 4 emphasizes the work done in the preparation part or also described as the data accumulation phase, which will be used in the next chapter. Additionally, Chapter 5 realises the implemented algorithm for SLIP and TD-SLIP models, for online and offline ways, respectively. It uses the mentioned work in Chapter 4,

and provides different ways so that the algorithm can be executed. Chapter 6 demonstrates the obtained results, and finally the very last chapter concludes the work, and provides possible future extensions.

Chapter 2

Background Information

2.1 SLIP - Constant Energy - Model

2.1.1 General Information

Foundations of the Spring Loaded Inverted Pendulum were laid in the very late years of 20^{th} century [9, 38].

Figure 2.1 illustrates the general structure of the SLIP Model. The model is consisted of a point mass (m), which is attached to a compliant, mass-less leg, and a leg length of r. The spring constant is given with k, and the parameter b is used to represent the viscous damping. Finally, the leg angle is indicated as θ.

On the other hand, Figure 2.2 exemplifies single step locomotion of the SLIP model. There are basically two fundamental phases, which can be observed as the Flight and Stance phases. During the flight phase, leg does not touch the ground and during the stance phase leg touches the ground with toe position

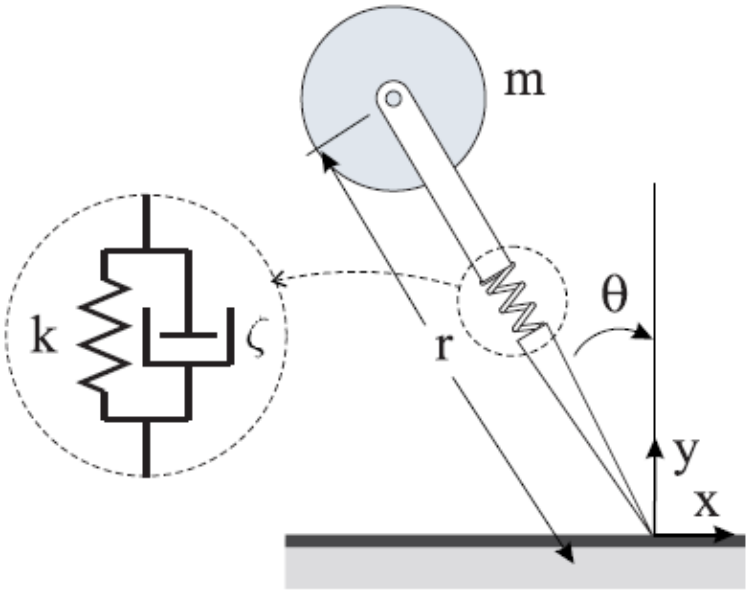

Figure 2.1: SLIP Model Coordinates & Parameters

stays fixed. Flight phase can be divided further into two sub-phases (Descent-Ascent) and Stance phase can also be divided into two-sub-phases (Compression-Decompression). Different transition events, which are demonstrated on the Figure, identifies the current phase and the current behavior of the model.

1. **Flight Phase**

   The Flight phase is the period where the leg remains completely untouched to the ground and the body follows a ballistic trajectory. Therefore, according to the model's vertical velocity the phase is divided into two main sub-phases

   (a) **Ascent Flight Phase**: This is exactly the half sub-period of the flight phase where the vertical velocity is always positive. Although, the velocity is decreasing in magnitude, the model vertical position keeps increasing.

   (b) **Descent Flight Phase**: This is the remaining half sub-period of

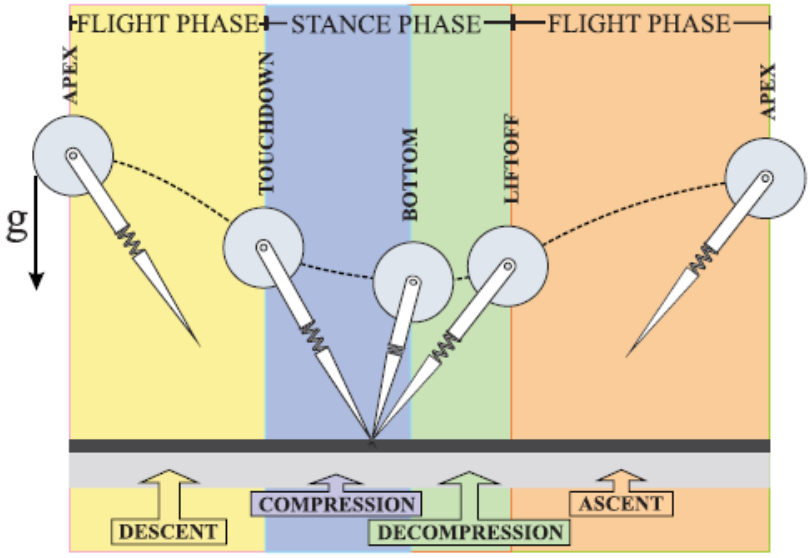

Figure 2.2: SLIP Model Phases & Transitions

the flight phase where the vertical velocity is always negative. The velocity is increasing in magnitude, but the model vertical position is decreasing. It can be considered as the opposite behavior of the ascent phase.

2. **Stance Phase**

When the model touches the ground, the stance phase begins. The dynamics of this phase are non-integrable, because of the gravitational attraction. Just like the Flight phase, this phase is also consisted of two main sub-phases, but these sub-phases are not identically occurred in half of the period, because of the touchdown leg angle.

(a) **Compression Stance Phase**: In this sub-period of time, the leg length becomes smaller, or in other words the rate of change of the leg length is negative. The energy is stored on the compressed string, and this phase occurs until the vertical leg velocity is zero.

(b) **Decompression Stance Phase**: This is the remaining sub-period of the stance phase, the rate of change of the leg length is positive. After the vertical velocity reaches to zero, the string converts its potential

SLIP States	
r, θ	Leg length and Leg angle
$\dot{r}, \dot{\theta}$	Leg compression and swing rates
q	Body state vector in polar coordinates $q = [\theta, \dot{\theta}, r, \dot{r}]$
p_r, p_θ	Radial and angular momenta
$r_{td}, \theta_{td}, t_{td}$	Touchdown leg length, angle and time
r_b, θ_b, t_b	Bottom leg length, angle and time
$r_{l0}, \theta_{lo}, t_{lo}$	Liftoff leg length, angle and time
x, y	Horizontal and vertical body positions
tx	Horizontal toe position
$\dot{x}, \dot{y}$	Horizontal and vertical body velocities
b	Body state vector in cartesian coordinates $b = [x, \dot{x}, y, \dot{y}, tx]$
$y_a, \dot{x_a}$	Apex height and velocity

Table 2.1: Notation for SLIP states throughout the thesis

SLIP Parameters	
m, g	body's mass, gravitational acceleration
k_c, k_d, ζ	Leg stiffness during compression, decompression and damping coefficient
E	Total mechanical energy
$F_s(r, \dot{r})$	Spring force function. For a given leg length it returns spring force based on the stance phase of SLIP
$U_s(r, \dot{r})$	Spring potential energy function. For a given leg length it returns stored energy on compliant leg based on the stance phase of SLIP

Table 2.2: Notation for SLIP model parameters throughout the thesis

energy to kinetic energy, and the leg begins to move on the opposite direction.

- **Transition Events**

Since the SLIP model is a hybrid one, it includes both continuous and discrete dynamics with respect to the flight and stance dynamics. These events can be considered as the boundaries in between the phases. They fundamentally occur at the point where a phase is finished, and the next

Mapping Functions	
$H_{td \to td}(b_{td})$	Touchdown to touchdown map
$H_{a \to a}(b_a)$	Apex to apex map
$H_{a \to td}(b_a)$	Apex to touchdown map
$H_{lo \to a}(b_{lo})$	Liftoff to apex Map
$H_{lo \to td}(b_{lo})$	Liftoff to touchdown Map
$t_{c \to p}(b)$	Cartesian to Polar Transformation
$t_{p \to c}(q)$	Polar to Cartesian Transformation

Table 2.3: Notation for mapping functions throughout the thesis

SLIP Events	
t_{td}, t_{lo}	touchdown and liftoff times
r_{td}, θ_{td}	touchdown leg length and angle
r_{lo}, θ_{lo}	liftoff leg length and angle
$\dot{r_{td}}$, $\dot{\theta_{td}}$	touchdown leg compression and swing rates

Table 2.4: Notation for SLIP events

one is started. SLIP events table demonstrates the basic definitions of the model events.

1. **Apex Event**: This event occurs in the Flight phase, and in between the Ascent and Descent sub-phases. When this event occurs, the SLIP body is on its maximum height, the maximum potential energy, and zero vertical velocity. In other words, it occurs where the following equation is equal to zero.

$$\dot{y} = 0$$
$$SLIP \to flight$$

2. **Touchdown Event**: This is a phase transition event. It occurs, when the slip body touches at the ground, or transitions into compression sub-phase from descent. Differently from the apex event, this event occurs at the crossing point of the following equation.

$$y - r_{td}\cos\theta_{td} = 0$$
$$\dot{y} < 0$$

3. **Bottom Event**: This event occurs in between the compression subphase to decompression subphase. It occurs when the vertical velocity is zero, it actually indicates the minimum height, and minimum potential energy. It occurs at the zero crossing point of the following equation.

$$\dot{r} = 0$$
$$SLIP \rightarrow stance$$

4. **Liftoff Event**: This is the other phase transition event. It occurs, when the slip body removes connection with the ground, and begins to fly. It appears in between the decompression and ascent sub-phases. The following equation identifies the zero crossing function.

$$y - r_{lo}\cos\theta_{lo} = 0$$
$$\dot{y} > 0$$

- **SLIP Dynamics**

Based on the hybrid nature of the SLIP model, the stance and flight phases exhibits different dynamics. Therefore, the following information is going to clarify the flight phase's and stance phase's dynamics

1. **Flight Phase Dynamics**: During the Flight phase, the SLIP model follows a ballistic trajectory. In Cartesian coordinates, the state vector can be considered as;

$$b := \begin{bmatrix} x & \dot{x} & y & \dot{y} & tx \end{bmatrix} \tag{2.1}$$

Therefore, the flight dynamics becomes;

$$\dot{b} := \begin{bmatrix} \dot{x} & 0 & y & -g & \dot{x} \end{bmatrix} \tag{2.2}$$

In Equation 2.2, the last variable of the vector is used for the multi-step purposes. Therefore, for a single movement from apex to apex, the locomotion is not necessary and can be considered as zero. The variable remains constant during the stance phase, and has identical dynamics with the body position state over the flight phase. Since the controller action is assumed to be executed on each apex, the toe position is instantaneously updated with the new touchdown angle, independently from its dynamics.

Therefore, over a flat surface, the apex to touchdown map ($H_{a\to td}(b_a)$), becomes;

$$H_{a\to td}(b_a) := \begin{bmatrix} x_a + \dot{x_a}\sqrt{2(y_a - y)/g} \\ \dot{x_a} \\ r_{td}\cos\theta_{td} \\ -2\sqrt{2g(y_a - y)} \\ x + r_{td}\sin\theta_{td} \end{bmatrix} \tag{2.3}$$

In addition to this, another simple way to derive the flight dynamics is using polar coordinates. If we assume the state vector as;

$$q := \begin{bmatrix} \theta & \dot{\theta} & r & \dot{r} \end{bmatrix}^\intercal \tag{2.4}$$

Therefore, the touchdown states can now be mapped with a transformation from Cartesian coordinates to polar coordinates;

$$t_{c\to p}(b_{td}) := \begin{bmatrix} \theta_{td} \\ (-y\dot{x} + (x - tx)\dot{y})/r^2 \\ r_{td} \\ ((x - tx)\dot{x} + y\dot{y})/r \end{bmatrix} \tag{2.5}$$

For a given liftoff state, a transformation from polar coordinates to Cartesian coordinates are required, so;

$$t_{p\to c}(q_{lo}) := \begin{bmatrix} -r_{lo}\sin\theta_{lo} \\ -\dot{r_{lo}}\sin\theta - r_{lo}\cos\theta_{lo}\dot{\theta_{lo}} \\ r_{lo}\cos\theta_{lo} \\ \dot{r_{lo}}\cos\theta_{lo} - r_{lo}\sin r_{lo}\sin\theta_{lo}\dot{\theta_{lo}} \\ 0 \end{bmatrix} \tag{2.6}$$

According to the assumption of a flat surface with zero height, and toe is located at the origin of the Cartesian coordinate frame. The liftoff to apex map ($f_{lo\to a}(b_{lo})$), is derived as follows;

$$H_{lo\to a}(b_{lo}) := \begin{bmatrix} x_{lo} + \dot{x_{lo}}\dot{y_{lo}}/g \\ \dot{x_{lo}} \\ 0.5\dot{y_{lo}}^2/g \\ 0 \\ \dot{x_{lo}}\dot{y_{lo}}/g \end{bmatrix} \tag{2.7}$$

2. **Stance Phase Dynamics**: In the light of the assumption of the friction-less revolute joint, the stance phase occurs in between the first contact point of the ground and first lose-contact point with the ground. As the best fit for the derivation of the stance phase dynamics is the polar coordinates, the Lagrangian equation of the Figure 2.1 is given by;

$$L = \frac{1}{2}m(\dot{r}^2 + r^2\dot{\theta}^2) - \frac{1}{2}k(l_0 - r - mgr\cos\theta) \tag{2.8}$$

Therefore, equations of the motion of the SLIP model in stance phase can be derived as follows;

$$m\ddot{r} = mr\dot{\theta}^2 + k(l_0 - r) - mg\cos\theta - \zeta\dot{r} \tag{2.9}$$

$$0 = \frac{\partial}{\partial t}(mr^2\dot{\theta}) + mgr\sin\theta \tag{2.10}$$

Using 2.9 and 2.10, the dynamics in polar coordinates, $\mathbf{q}$, are given by;

$$\dot{q} = \begin{bmatrix} \dot{\theta} \\ -g\sin\theta/r - 2\dot{r}\dot{\theta}/r \\ \dot{r} \\ F_s(r,\dot{r})/m + r\dot{\theta}^2 - g\cos\theta \end{bmatrix} \tag{2.11}$$

where $F_s(r,\dot{r})$ is the spring force function;

$$F_s(r,\dot{r}) = \begin{cases} k_c(l_0 - r) & \text{if } \dot{r} \leq 0 \\ k_d(l_0 - r) & \text{if } \dot{r} > 0 \end{cases} \tag{2.12}$$

Note that Equations 2.9 and 2.10 are nonlinear differential equations. In fact, due to the gravitational form, these equations are non-integrable, hence a closed-form solutions cannot be found [39, 40]. Although, there are no exact solutions for the stance map, there are also several studies on approximate solutions of SLIP's stance dynamics, in the literature. The following simulation sections consider [39] and [40], as the approximations.

2.1.2 Model Simulation

The model has been implemented on the MATLAB environment, so the simulation utilises a few fundamental functions. These functions simulate the model to jump from one point to another with respect to the parameters provided as an input to the MATLAB's ODE45 function. The ODE45 function solves the differential equations by integrating the input function with respect to the provided $t_{initial}$ and t_{final} values, and an initial function value.

The first MATLAB function we utilize is called $slip_{tr}$, which is the main transition events function and symbolically, which is given as in Eq. 2.13.

$$[y_{new}, c_{new}, stop, params] = slip_{tr}(t, y, c, ie, params) \tag{2.13}$$

Simulation Parameters	
$slip_{tr}$	slip transition events
$slip_{vf}$	slip vector field
$slip_{ev}$	slip event functions in between transitions
t	current time
y	current state vector
c	current phase
y_{new}	next state vector
c_{new}	next phase
ie	stop conditional
$stop$	boolean value
$params$	required update parameters for the corresponding function

Table 2.5: Notation for SLIP simulation parameters

It contributes the model to execute required calculations in between two subsequent phases. It does not only provide the new position, the new state, and the is-fall boolean value, but also updates the model specific parameters, respectively. The aim of the next function, $slip_{vf}$ is to evaluate the vector;

$$y_p = slip_{vf}(t, y, c, params) \tag{2.14}$$

The output of the Equation 2.14 represents the vector field. It calculates the current state values in a sub-phase. For example, after changing the model state (from liftoff event to apex event), this function calculates current states all values, based on the provided time interval, until the next state is reached. The last function $slip_{ev}$ is symbolically given below;

$$[value, isterminal, direction] = slip_{ev}(t, y, c, params) \tag{2.15}$$

Function provided in Equation 2.15 handles to a function that computes event functions to detect hybrid transitions. Throughout the simulations we use the following parameters;

SLIP Simulation Parameters	
spring constant (k)	$(350\,\mathrm{N/m})$
mass (m)	$(1\,\mathrm{kg})$
gravitational force(g)	$(9.81\,\mathrm{N/kg})$
leg length(l_0)	$(1\,\mathrm{m})$

Table 2.6: SLIP simulation parameters

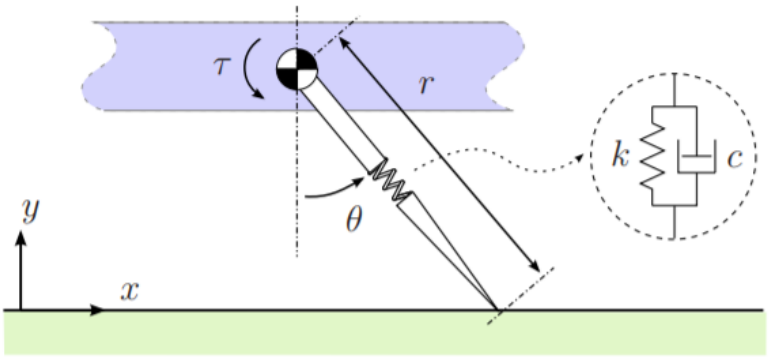

Figure 2.3: TD-SLIP(MONOPOD) Model Coordinates & Parameters

2.2 MONOPOD - Torque actuated with Damping - Model

2.2.1 General Information

Torque actuated Dissipative Spring Hopper is a develop version of the SLIP model [12]. Differently from the constant energy model, it utilises a torque parameter and the damping constant. Therefore, the system dissipates energy with the damping, but regains it with the applied torque.

The phases, sub-phases and transition events are completely the same as the standard SLIP model as given in the Section 2.1. Therefore, will not be repeated here.

- **Flight Phase Dynamics**: The flight phase dynamics are completely the

same with the previous section. The model fundamentally follows a ballistic trajectory.

- **Stance Phase Dynamics**: The stance phase dynamics, on the other hand, are a little bit complicated than the previous section. In the light of the inclusion of the damping and torque actuation parameter, the phase dynamics become more complicated.

$$m\ddot{r} = mr\dot{\theta}^2 - mg\cos\theta - k(r - l_0) - b\dot{r} \tag{2.16}$$

$$\frac{\partial}{\partial t}(mr^2\dot{\theta}) = mgr\sin\theta + \tau \tag{2.17}$$

The equations are derived from the Euler-Lagrange formulation. The τ parameter represents the applied hip torque, which can be observed from the Figure 2.3.

2.2.2 Model Simulation

The used functions in the MATLAB simulation is logically with the same with the previous ones. The difference occurs from the incorporated effects of the hip torque and damping. Therefore, the $monopod_{tr}$ function, which provides the required calculations in between phase changes, takes the effect of torque into account from the touchdown event to bottom, and bottom event to liftoff. Additionally, the vector field among these compression and decompression subphases (**stance phase**) are also calculated by the $monopod_{vf}$ function.

MONOPOD Simulation Parameters	
spring constant (k)	$\left(350\,\mathrm{N/m}\right)$
mass (m)	$\left(1\,\mathrm{kg}\right)$
gravitational force (g)	$\left(9.81\,\mathrm{N/kg}\right)$
leg length (l_0)	$\left(1\,\mathrm{m}\right)$
damping constant (b)	$\left(24\,\mathrm{Ns/m}\right)$

Table 2.7: MONOPOD simulation parameters

2.3 Used Libraries

2.3.1 *fminsearch* function

fminsearch is a well-known MATLAB function. As the name suggests the method takes an unconstrained multi variable function as an input and calculates the minimum by following a derivative-free method. It's been explicitly used on variety of MATLAB functions and also individual implementations.

The description of the function can be demonstrated as follows

$$\min_x f(x) \tag{2.18}$$

where:

x = *is a vector or a matrix*
$f(x)$ = *is a function that returns a scalar*

Additionally, the algorithmic implementation of the function can be demonstrated as follows

$$x = fminsearch(fun, x_0, options) \tag{2.19}$$

where:

fun = *denotes a function to be minimized*
x_0 = *initial vector*
$options$ = *optimization options*

The options field is not required, but in order to improve the process,

- maximum number of function evaluations
- maximum number of iteration count
- termination tolerance

can be arranged. These optional fields not only provide a better solution, but if given appropriately it can also support to avoid being converged to a local minima.

2.3.2 *fminsearchbnd* function

fminsearchbnd is an improved version of the *fminsearch*. Likely *fminsearch* function, *fminsearchbnd* method uses the same input parameters with an inclusion of setting upper and lower boundary parameter.

$$x = fminsearch(fun, x_0, LB, UB, options) \tag{2.20}$$

where:

fun = *denotes a function to be minimized*
x_0 = *initial vector*
$options$ = *optimization options*
LB = *lower bound*
UB = *upper bound*

As *fminsearch* does not admit bound constraints, *fminsearchbnd* provides a solution to this need, by using a transformation method (quadric for single bounds, sin(x) for dual bounds) to convert bounded constrained problem into an unconstrained one. Then, applies the *fminsearch* procedure to calculate the function minimum. [41]

Chapter 3

Problem Definition & Proposed Solution

3.1 Problem Definition

Since we consider 2D-SLIP model, the SLIP touchdown points follow a one dimensional line. Let us assume that the initial touchdown position $x_0 \in \mathbf{R}$, and the fixed (derived) touchdown position $x_f \in \mathbf{R}$ be given, where $x_f > x_0$, which is demonstrated on Figure 3.1.

Our aim is to design a planning algorithm and a control law accordingly so that after an (yet) undetermined number of steps, the SLIP touches down in a sufficiently close neighborhood of x_f. Note that touchdown state vector $w_{td} \in \mathbf{R}^5$ is given as in the following equation, where we assumed that the leg length is given as 1 ($l_o = 1$).

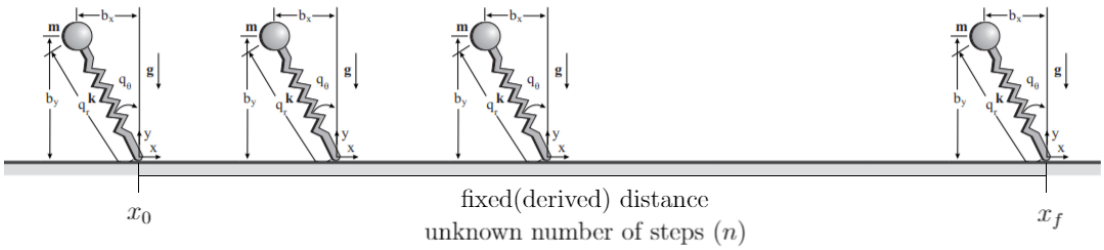

Figure 3.1: Problem Visualization

$$w_{td} = \begin{bmatrix} x_0 \\ \dot{x_0} \\ y_0 \\ \dot{y_0} \\ x_0 + \sin\theta_{td} \end{bmatrix} \tag{3.1}$$

The very last parameter in Equation 3.1 actually demonstrates the touchdown position in x direction. According to Figure 3.2, it's clear that we have;

$$x_{td} = x_0 + l_0 \sin\theta_{td} \tag{3.2}$$

For convenience, let us define the following projection vector $\Pi_5 : \mathbf{R}^5 \to \mathbf{R}$ as;

$$\Pi_5(w) = w_5 \tag{3.3}$$

for any $w = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} \in \mathbf{R}^5$. Clearly we have

$$\Pi_5(w_{td}) = x_0 + l_0 \sin\theta_{td} \tag{3.4}$$

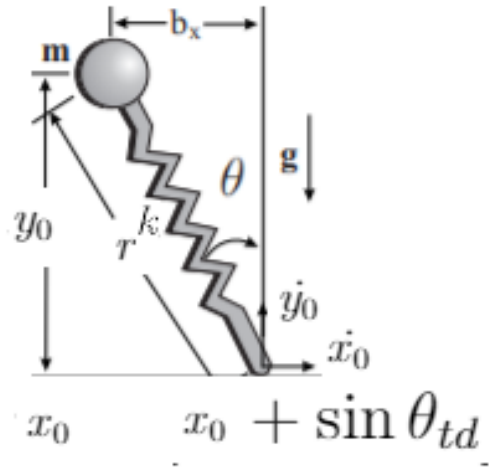

Figure 3.2: SLIP touchdown state vector visualization

Remark. *Note that θ_{td} is our control parameter and we assume that we can adjust it during the flight phase. Hence, if $x_0 \in \mathbf{R}$ is known, the next $x_{td} \in \mathbf{R}$ is determined by θ_{td} only. Moreover during the stance phase, x_{td} does not change until the liftoff. After the liftoff, the mass follows a ballistic trajectory, which is integrable. Hence if we determine next touchdown angle, then x_0 and y_0 of next touchdown states were also determined. This implies that while searching possible touchdown positions, we have only 3 free parameters; namely $\dot{x}$, $\dot{y}$ and θ.*

Likewise, from Figure 3.2, we have

$$y_0 = l_0 \cos\theta_{td} \tag{3.5}$$

Hence, if θ_{td} is known, then y_0 is also known.

3.1.1 Touchdown-to-Touchdown Return Map

The touchdown event initiates the stance phase, where the body follows the dynamics demonstrated in Equations 2.9 and 2.10. Therefore, by following a similar representation to provided in [42], the touchdown to liftoff map can be formulated as

$$(r_{lo}, \dot{r}_{lo}, \theta_{lo}, \dot{\theta}_{lo}) = H_{td \to lo}(\dot{r}_{td}, \dot{\theta}_{td}) \tag{3.6}$$

where the variables with subscript of lo means the liftoff phase, and td corresponds the touchdown instant. Additionally, the $H_{td \to lo}$ is the mathematical function which describes for the touchdown to liftoff map, and it also depends on the control parameter θ_{td}.

In addition to this, the liftoff the apex map can be formulated as

$$(\dot{z}_{td_1}, \dot{y}_{td_1}) = H_{lo \to td_1}(\dot{z}_{td}, \dot{y}_{td}, r_{lo}, \theta_{lo}) \tag{3.7}$$

where $H_{lo \to td,1}$ corresponds to the liftoff to next touchdown map. Therefore, the map from one touchdown to another becomes as

$$H_{td_0 \to td_1} = V_{td1} \circ H_{lo \to td_1} \circ V_{lo} \circ H_{td_0 \to lo} \circ V_{td_0} \tag{3.8}$$

where $H_{td_0 \to td_1}$ is the touchdown to touchdown return map. The symbols subscripted with V label is actually the mathematical representation of the coordinate transformation matrices, to switch between the Cartesian and Polar Coordinates. Hence, combining all together the touchdown to touchdown return map can be formulated as follows

$$(\dot{x}_{td_1}, \dot{y}_{td_1}) = H_{td_0 \to td_1}(\dot{x}_{td_0}, \dot{y}_{td_0}) \tag{3.9}$$

Let $w_{td}^{(n)} \in \mathbf{R}^5$ and $w_{td}^{(n+1)} \in \mathbf{R}^5$ be n^{th} and $(n+1)^{th}$ consecutive touchdown states, respectively. Clearly we have,

$$w_{td}^{(n+1)} = H_{td,n\to td,(n+1)} w_{td}^{(n)} \tag{3.10}$$

where $H_{td,n\to td,(n+1)}$ is the touchdown-to-touchdown map between n^{th} and $(n+1)^{th}$ touchdowns. Due to the non-integrability of the stance dynamics, exact mathematical expression of this map is not available. It could only be approximated by some analytical methods [43, 42]. In this work, we will rely on simulations of stance dynamics to evaluate this mapping. Clearly, we utilize the iteration of this map. Let $w_{td}^{(0)} \in \mathbf{R}$ be an initial touchdown state and $w_{td}^{(n)} \in \mathbf{R}^n$ be the corresponding touchdown state off the n^{th} step. Clearly we have;

$$w_{td}^{(n)} = H_{td,(n-1)\to td,(n)} \circ H_{td,(n-1)\to td,(n)} \circ \dots H_{td,0\to td,1} w_{td}^{(0)} \tag{3.11}$$

$$= H_{td,0\to td,(n)} w_{td}^{(0)} \tag{3.12}$$

where, $H_{td,0\to td,(n)}$ could be considered as n-step touchdown-to-touchdown map.

With this information, we can define the footstep planning problem as follows;

- **Problem 1 (Single waypoint footstep planning)**:

  Given $x_0 \in \mathbf{R}$ and $x_f \in \mathbf{R}$ with $x_f > x_0 > 0$, find an initial touchdown state $w_{td}^{(0)} \in \mathbf{R}^5$ with $\Pi_5(w_{td}^{(0)}) = x_0$, such that for some $n \in \mathbf{N}$, the final touchdown state $w_{td}^{(n)}$ given by 3.12 satisfies

  $$|\Pi_5(w_{td}^{(n)}) - x_f| < \epsilon \tag{3.13}$$

  for some sufficiently small $\epsilon > 0$.

To define multiple waypoint footstep planning, let us assume that $x_0 \in \mathbf{R}$ and $x_{f_i} \in \mathbf{R}$ be given where $i = 1, 2, \ldots k$ with $x_{f_k} > x_{f_{k-1}} > \cdots > x_{f_1} > x_{f_0}$. For an initial touchdown state $w_{td}^{(0)}$, let us define the following for some integers $n_i, i = 1, 2, \ldots, k$;

$$w_{td}^{(n_1)} = H_{td,0\to td,n_1}(w_{td}^{(0)}) \tag{3.14}$$

$$w_{td}^{(n_i)} = H_{td,1}^{td,n_i}(w_{td}^{(i-1)}), i = 2, \ldots, k \tag{3.15}$$

- **Problem 2 (Multiple waypoint footstep planning)**:

  Given $x_0 \in \mathbf{R}$ and $x_{f_i} \in \mathbf{R}, i = 1, 2, \ldots, k$, find an initial touchdown state $w_{td}^{(0)} \in \mathbf{R}^5$ with $\Pi_5(w_{td}^{(0)}) = x_0$, such that for some integers $n_i \in \mathbf{N}$ the following inequatilities are satisfied

  $$|\Pi_5(w_{td}^{(n_i)}) - x_{f_i}| < \epsilon \tag{3.16}$$

  for some sufficiently small $\epsilon > 0$.

Remark. *Clearly the solvability of the problems given above depends on finding an appropriate initial touchdown state as well as appropriate touchdown angles at each touchdown. Note that the touchdown angle is our basic control parameter for the standard SLIP model.*

3.2 Proposed Solution

As stated in the previous **Remark**, solvability of the problems given in the previous section depends on finding appropriate initial touchdown state(s) and corresponding touchdown angle(s) so that Equation 3.13 and 3.16 are satisfied. This requires solving equations 3.12 and 3.15. But, since the map $H_{td,i\to td,i+1}(.)$ is not available analytically, we will resort to the numerical solutions, i.e. numerical integration of the SLIP equation given by Eq. 2.9 and Eq. 2.10. Since, initial

touchdown state w_{td} is not known, but is only required to satify $\Pi_5(w_{td}^{(0)}) = x_0$, we choose a region of possible initial states D_0 by choosing the components of initial touchdown state components in a reasonable range by considering the velocity values in a reasonable running behavior.

3.2.1 Safe Guard Region

The region has been constructed by observing a single SLIP stride. As each stride consists of a touchdown, bottom, liftoff, apex and next touchdown state, different next touchdown state vectors with respect to the different starting touchdown states can be examined by changing the starting touchdown angle (touchdown-to-touchdown map). Therefore, by using this method the state variances can be investigated.

Observing the horizontal and vertical velocities are the fundamental idea behind constructing the safe guard region. According to the SLIP configuration, in the simulation, selecting an inappropriate starting velocity might cause the SLIP to fall independent from the touchdown angle. Therefore, observing and avoiding the undesired or unstable(irrecoverable) states provides a better and controllable scope.

Hence we define D_0 as follows;

$$D_0 = \{w_{td}^{(0)} \in \mathbf{R}^5, \quad \Pi_5(w_{td}^{(0)})\} = x_0 \tag{3.17}$$

Then numerically we evaluate the next touchdown state from D_1 which are reachable from D_0, i.e.

$$D_1 = \{w_{td}^{(1)} \in \mathbf{R}^5, \quad \in w_{td}^{(0)} D_0 \ni w_{td}^{(1)} = H_{td,0\to td,1} w_{td}^{(0)}\} \tag{3.18}$$

$$= H_{td,0\to td,1}(D_0) \tag{3.19}$$

3.2.2 Goal Domain

The goal vector can also be considered as the desired vector. It is specifically used as the aimed next touchdown state. According to the algorithm, different starting touchdown states can address to reach the same goal domain. Different reachable vector sets are defined based on a provided desired vector. The function logic can be considered as an inverse one, as the next touchdown vectors are provided as an input, so that the current reachable vector sets can be obtained.

3.2.3 Reachable Touchdown State Set

A reachable touchdown state is a starting touchdown state set that is aiming to react the provided goal domain. It consists of different horizontal and vertical velocities with different touchdown angles.

Clearly, now we can define all successive touchdown states;

$$D_i = H_{td,0\rightarrow td,i} D_0, \quad \text{and} \quad i = 1, 2, \ldots, n \tag{3.20}$$

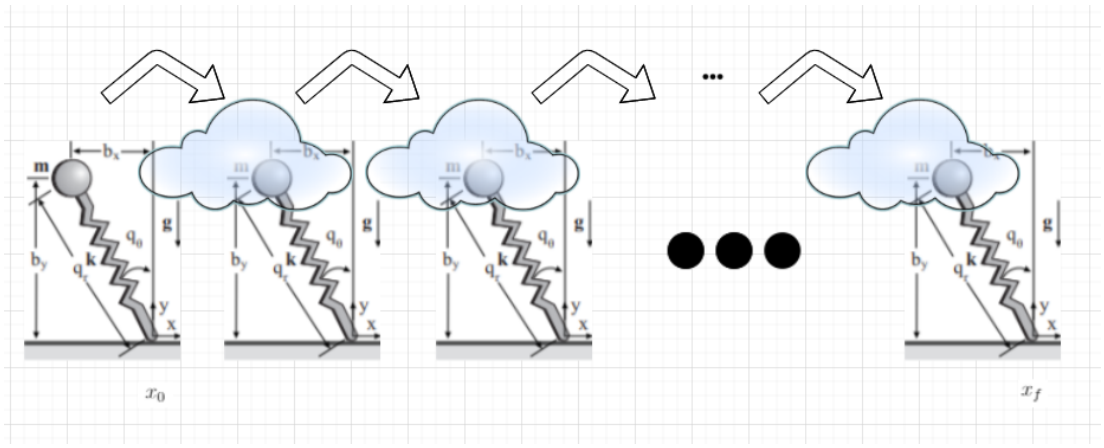

Figure 3.3: Proposed Solution Example (Forward Planning)

Now our Problem 1 will have a solution if for some $w_{td} \in D_n$, and Equation 3.13 is satisfied. The Figure 3.3 could be considered as the forward planning.

Alternatively, we can start from the final touchdown point and evaluate the steps mentioned above in a backwards fashion. More precisely, let $w_{td}^{(0)} \in \mathbf{R}^5$ be a touchdown state satisfy $\Pi_5(w_{td}^{(n)}) = x_f$. As before, similar to Equation 3.17 we define the initial possible states D_0 as follows;

$$D_0 = \{w_{td}^{(n)} \in \mathbf{R}^5, \quad \Pi_5(w_{td}^{(n)})\} = x_f, \tag{3.21}$$

This time, using the inverse of touchdown-to-touchdown map, we find D_{n-1} as follows;

$$D_{n-1} = \{w_{td}^{(n-1)} \in \mathbf{R}^5, \quad H_{td,0\to td,1} w_{td}^{(n-1)} \in D_n\} = H_{td,0\to td,1}^{-1}(D_n) \tag{3.22}$$

Similar to Equation 3.20, we can define all previous feasible touchdown states D_i as in Figure 3.4;

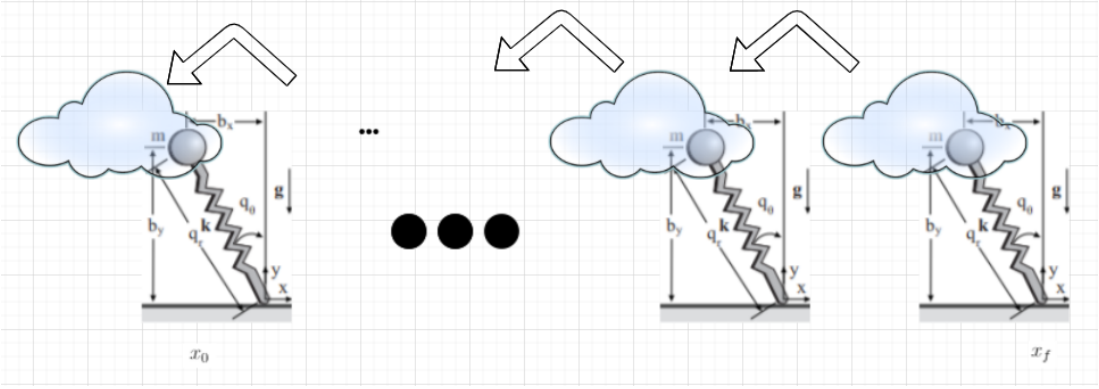

Figure 3.4: Proposed Solution Example (Backwards Planning)

Chapter 4

Preparation Phase

The preparation phase consists of a single sub-phase. In general, the work done in this chapter has been used in the main simulation phase. It is a separate part which has been processed before starting the simulation. It also can be considered as data acquisition and domain construction, or in other words, the gathered information will explicitly be used on the simulation chapter.

Section 3.2 clarifies the notations and terms, which have been used after across the chapter, and also the structure. It provides the details behind the touchdown state, stride, safe guard region, goal domain, and the reachable touchdown state set. In addition to this, Section 4.1 enlightens the details behind why and in what behavior the domain term is used, and also how it's constructed. Also, the section provides detailed information about the notion of inner and outer domains.

4.1 Cloud Construction

The fundamentals of the cloud construction algorithm is based on connecting the touchdown state vectors starting from the safe guard region to the outer regions. The idea behind this logic is to keep the all of the touchdown states inside the safe guard region, or in order words, provide a way to push them into

the controllable region. If a touchdown state is outside the safe guard region, then, by using the constructed clouds, figure out the corresponding touchdown angles (also applied torque for MONOPOD model) in order to advance the state inside the stable(more controllable) region.

Scanning through all of the possible states and their corresponding goal domains reveals a distance array. As a pair, each initial and next touchdown state has a distance value. At the beginning, the initial position of the first touchdown position is 0. Hence, when the algorithm passes through 5 model states (touchdown, bottom, liftoff, apex, next touchdown) the next touchdown state has a non-zero position value. This value represents the distance between a specific starting touchdown state, a specific touchdown angle and a specific next touchdown state.

4.1.1 Construction of the Inner Cloud

The inner cloud can be realised as a set, in which the randomly or linearly generated goal domains and their corresponding starting touchdown states has been found together. In a touchdown state vector, the vertical position and the leg position are associated with the touchdown angle. Therefore, their illustrations on a figure is unnecessary and redundant. Excluding this information from the state vector practically forces the algorithm to benefit from the horizontal velocity, vertical velocity and the calculated distance value, together in between the starting and next touchdown states. Therefore, as the pair set becomes diagrammatically shapeless (not a line, nor a polygon), it can be also mentioned as a cloud.

The very first process of this part starts with figuring out the safe guard region. The definition of a safe guard region has provided in the Section 3.2.1. After evaluating or approximating the safe guard region, two different vector sampling methods can be achieved.

Two types of sampling has been used from the safe guard region. The very

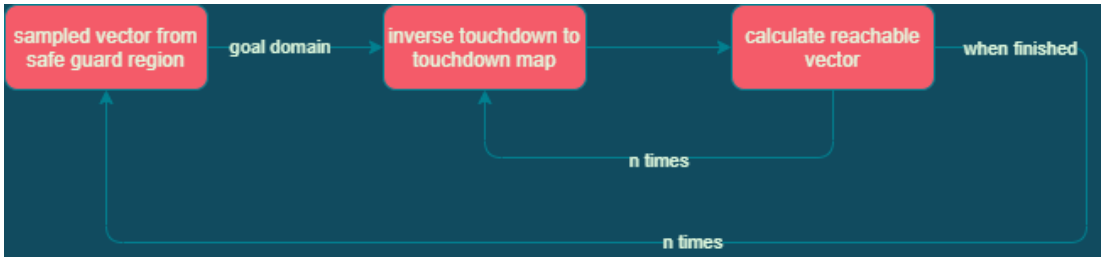

Figure 4.1: Cloud Construction Logic

first one uses to sample the goal domain vectors (horizontal and vertical velocity) in a random manner, and the second one utilises from linearly sampling ways. Linearly sampling can cover the safe guard region properly. However, randomly sampling can achieve a better precision. What is best to do is to cover the region with a better precision, which can be provided by sampling in a random manner with a greater sample count. However, covering a wider area takes more time. Therefore, the implementation path should be selected according to the computer's specifications.

Assume that the algorithm has sampled n goal vectors from the safe guard region. For each goal domain vector, another n number of vectors have been randomly sampled, and this new sampled vector set is called the reachable vector set. This set's sampling region can be considered as an extended or wider version of the safe guard region. The size of this covering region can totally up to the user, but extending it with a small number can contribute more robust results, because of the relatively narrower composed region. In this way, the size of the first cloud becomes $n * n$. Each element in the reachable vector has provided as an input to *fminsearchbnd* function.

The fun parameter in the *fminsearchbnd* function can be considered as a function to figure out a starting touchdown state with respect to the provided goal domain. The function takes two inputs; randomly sampled reachable vector as the initial condition, and provided goal domain as the next touchdown state. After providing the reachable set's region limits as the lower and upper boundaries, the *fminsearchbnd* easily calculates the best specific starting vector with a specific touchdown angle to reach the desired goal domain.

$$[state_{initial}, error] = fminsearchbnd(\\ @(state_{initial}) fun(state_{initial}, state_{goal}), LB, UB, options \\ ) \quad (4.1)$$

where:

$state_{initial}$ = *reachable vector, starting touchdown state*
$state_{goal}$ = *goal domain vector*
$error$ = *fminsearchbnd error*
fun = *touchdown to touchdown single step function*
LB = *reachable vector set region - LOWER*
LB = *reachable vector set region - UPPER*
$options$ = *includes MaxFunEvals, TolX, MaxIter*

The parameter provided in @() means that the values inside can be changed with respect to the error function (which is the output of the fun), and other parameters contributed in $fun()$ remains the same. For example, in the Equation 4.1, $state_{initial}$ has changed at each iteration, but $state_{goal}$ remains the same. The provided error values is actually the output of the euclidean distance between the given goal domain vector(goal horizontal and vertical velocity) and the output of the fun, which is the next touchdown state(horizontal and vertical velocity).

After figuring out the best (minimum error) starting touchdown state to reach the goal touchdown state vector, a datum has been created, and stored in an array to be used in the simulation part. Datum's fields are given in the equation 4.2.

$$Datum = \begin{bmatrix} distance & x'_0 & y'_0 & \theta_{td_0} & \theta_{td_1} & error \end{bmatrix} \quad (4.2)$$

where:

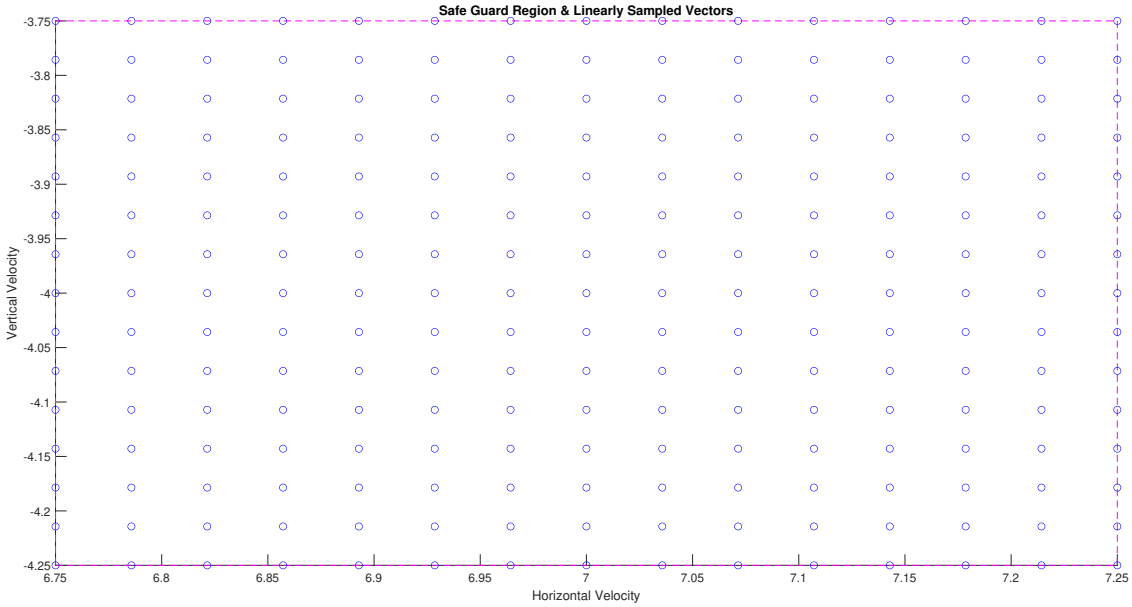

Figure 4.2: Linearly sampled horizontal and vertical velocity values from the Safe Guard Region

$distance$ = *the distance between the starting touchdown and the goal touchdown state*
x'_0 = *starting touchdown horizontal velocity*
y'_0 = *starting touchdown vertical velocity*
θ_{td_0} = *starting touchdown angle*
θ_{td_1} = *goal touchdown angle*
$error$ = *fminsearchbnd error*

The result of equation 4.2 is a single reachable vector set output. Inner cloud consists of n number of them for single goal domain, and $n * n$ number of them, in total.

Figure 4.2 illustrates the selected horizontal and vertical goal domain vectors for the linearly sampling type. The magenta border color demonstrates the assigned safe guard region. The sample size has been selected as $n = 225$, so that the row and the column size are equal and 15. The boundaries of the safe guard region is selected based on the system behavior. Which means that the model simulations were observed under specific conditions with different initial parameters.

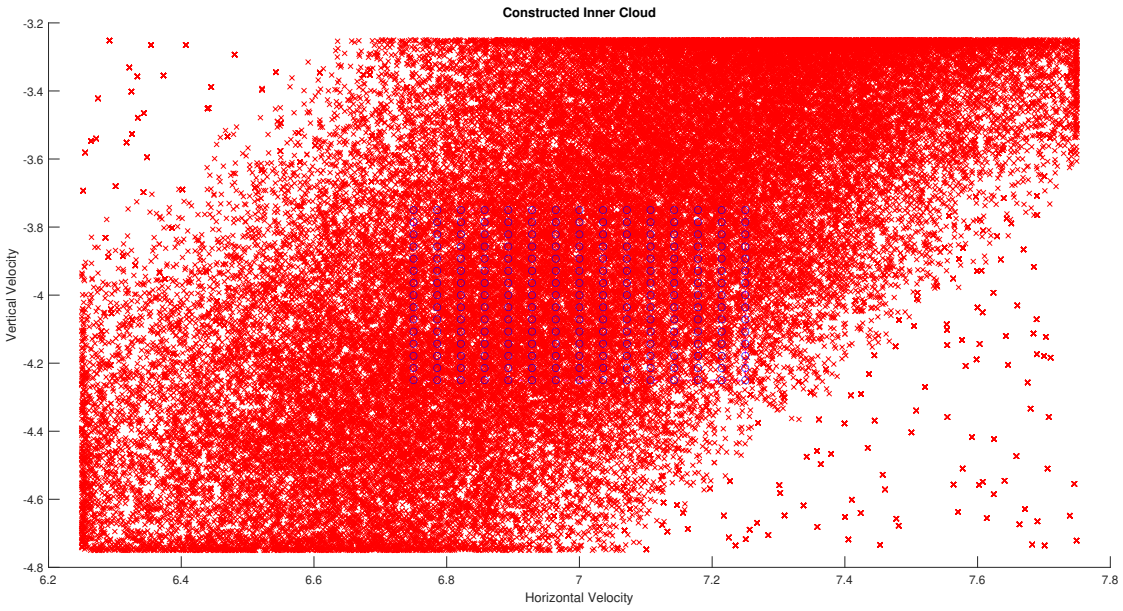

Figure 4.3: Constructed Inner Cloud — Red Ones Reachable Set ($n * n$, $n = 225$), Blue Ones Safe Guard Region ($m * m$, $m = 15$)

Then, according to the obtained results, the most robust points were observed. Therefore, the boundaries of the region are selected as $\begin{pmatrix} 6.75\,\mathrm{m/s} & 7.25\,\mathrm{m/s} \end{pmatrix}$ for the horizontal velocity, and $\begin{pmatrix} -4.25\,\mathrm{m/s} & -3.75\,\mathrm{m/s} \end{pmatrix}$ for the vertical velocity.

Figure 4.3 shows the final form of the constructed inner cloud domain. The blue vectors are the goal domains, and the surrounding red ones are the corresponding reachable vector set which can be connected to the vectors inside the safe guard region. The boundaries for the reachable vector set is selected as $\begin{pmatrix} 6.25\,\mathrm{m/s} & 7.75\,\mathrm{m/s} \end{pmatrix}$ for the horizontal velocity vector, and $\begin{pmatrix} -4.75\,\mathrm{m/s} & -3.25\,\mathrm{m/s} \end{pmatrix}$ for the vertical velocity vector. The upper left and the bottom right corner areas imply that the SLIP could not make successful strides. In addition to this, Eq. 4.3 demonstrates an example of a single element in the constructed array, where each index consists the values regarding with a single movement.

$$Datum_{monopod} = \begin{bmatrix} distance & x'_0 & y'_0 & \theta_{td_0} & \alpha_{td_0} & \theta_{td_1} & error \end{bmatrix} \tag{4.3}$$

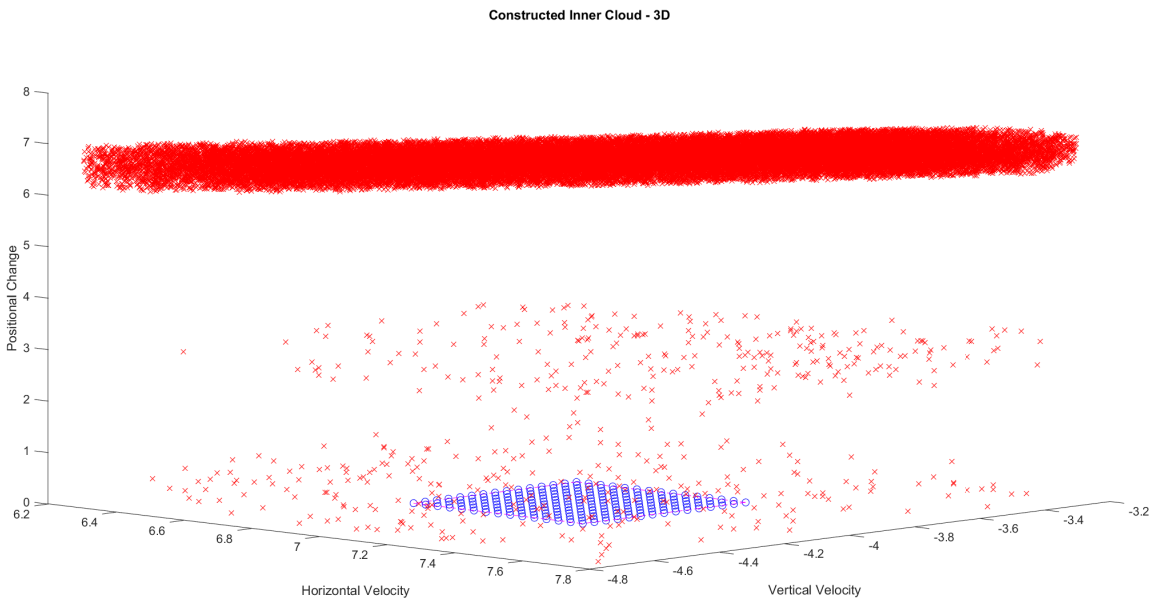

Figure 4.4: Constructed Inner Cloud(3D)

where:

α_{td_0} = *applied torque at the touchdown state*

Figure 4.4 demonstrates the 3D version of the Figure 4.3. The positional change axis illustrates the slip movement from one red cross to blue circle. According to the simulation, an approximate slip stride is around 6 to 7 meters, but for some specific cases it can have the lower values.

On the other hand, for the MONOPOD (torque actuated with dampling model) model, reachable set Datum is a little bit different. Differently from the Equation 4.2, Equation 4.3 has an extra field, which has been used to store the information of amount of the applied torque. The applied torque is a ramp function which has been implemented in between the starting of the touchdown state until the end of the bottom state [12].

The boundaries for the safe guard region is selected as $\begin{pmatrix} 3\,\mathrm{m/s} & 3.5\,\mathrm{m/s} \end{pmatrix}$ for the horizontal velocity vector, and $\begin{pmatrix} -3.5\,\mathrm{m/s} & -3\,\mathrm{m/s} \end{pmatrix}$ for the vertical velocity

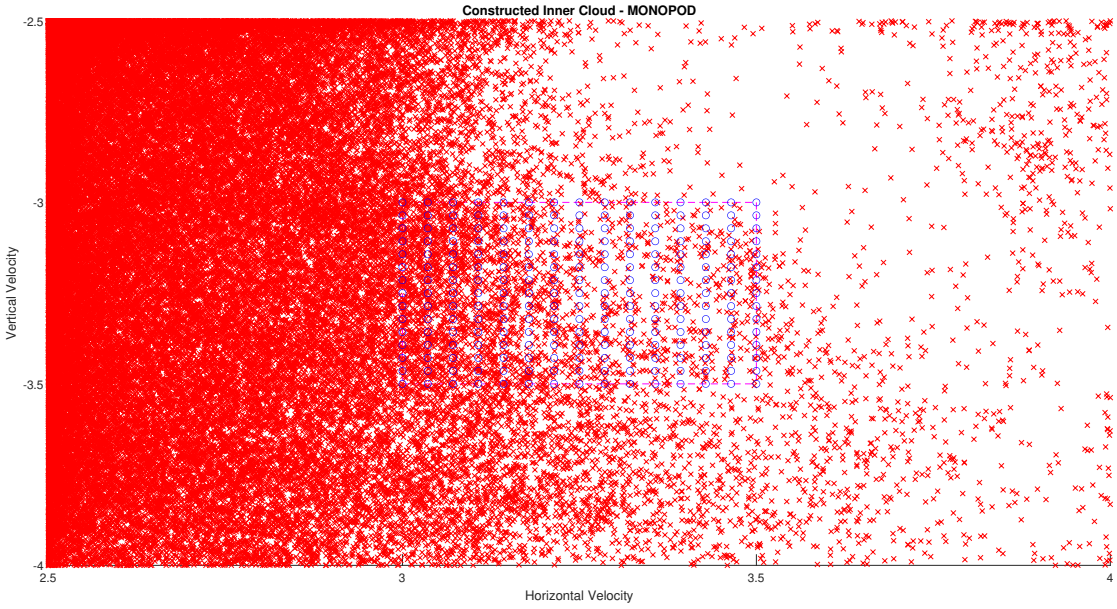

Figure 4.5: Constructed Inner Cloud - MONOPOD ($n * n$, $n = 225$)

vector. Also, for the reachable vector set, they are selected as $\begin{pmatrix} 2.5\,\mathrm{m/s} & 4\,\mathrm{m/s} \end{pmatrix}$ for the horizontal velocity vector, and $\begin{pmatrix} -4\,\mathrm{m/s} & -2.5\,\mathrm{m/s} \end{pmatrix}$ for the vertical velocity vector.

The MONOPOD's constructed inner cloud has shown in the Figure 4.5. Differently from the Figure 4.3, the density unfolds at the left side of the plot, and diminishes when it advances from left to right.

4.1.2 Construction of the Outer Clouds

The construction of the outer clouds almost follows the same procedure as in the Section 4.1.1. The main difference between them is the construction of the goal domains. In the previous section, goal domain vectors have been selected based on the randomly or linearly sampling processes. However, outer domain's goal vectors are selected based on inner domain's reachable vector set.

In order not to break the order, there are n number of vectors which are

randomly selected from the inner cloud's reachable vector set. In this execution, the most important thing to implement is that the selection process should be achieved with respect to their distinct goal domains. In other words, if the first goal domain has been selected from the reachable set index x, and row y, then the algorithm should not select the index x, again. Because selecting more than one value from the same index drives the algorithm to not cover every vector which is connected to the inner cloud.

$$\begin{array}{l}
\begin{bmatrix} distance_1 & x'_{01} & y'_{01} & \theta_{td_{01}} & \theta_{td_{11}} & error_1 \end{bmatrix} -- > \begin{bmatrix} x'_{goal} & y'_{goal} \end{bmatrix} \\
\begin{bmatrix} distance_2 & x'_{02} & y'_{02} & \theta_{td_{02}} & \theta_{td_{12}} & error_2 \end{bmatrix} -- > \begin{bmatrix} x'_{goal} & y'_{goal} \end{bmatrix} \\
\begin{bmatrix} distance_3 & x'_{03} & y'_{03} & \theta_{td_{03}} & \theta_{td_{13}} & error_3 \end{bmatrix} -- > \begin{bmatrix} x'_{goal} & y'_{goal} \end{bmatrix} \\
\vdots \\
\begin{bmatrix} distance_n & x'_{0n} & y'_{0n} & \theta_{td_{0n}} & \theta_{td_{1n}} & error_n \end{bmatrix} -- > \begin{bmatrix} x'_{goal} & y'_{goal} \end{bmatrix}
\end{array} \tag{4.4}$$

Equation 4.4 demonstrates n number of samples in the inner cloud, for a single goal domain. Selecting more than one value from this array pushes the algorithm to connect the same goal domain more than once. Therefore, for the outer cloud, each selected goal domain should push the algorithm into a distinct inner goal domain.

On the other hand, one way to prevent the restriction in the goal domain selection, is to use more than n samples. However, this process increases the construction time, and also the size of the constructed cloud array.

The provided algorithm uses n samples for the outer rings. Therefore, for a single outer cloud, the size of the constructed array becomes $2 * n * n$ and for more than m number of outer clouds, the size of the constructed array becomes $(m + 1) * n * n$.

Figure 4.6 shows the selected goal domains for the outer region. The green circles have been selected from the inner clouds reachable vector set, and the selection process has implemented for each distinct set. This proves that each

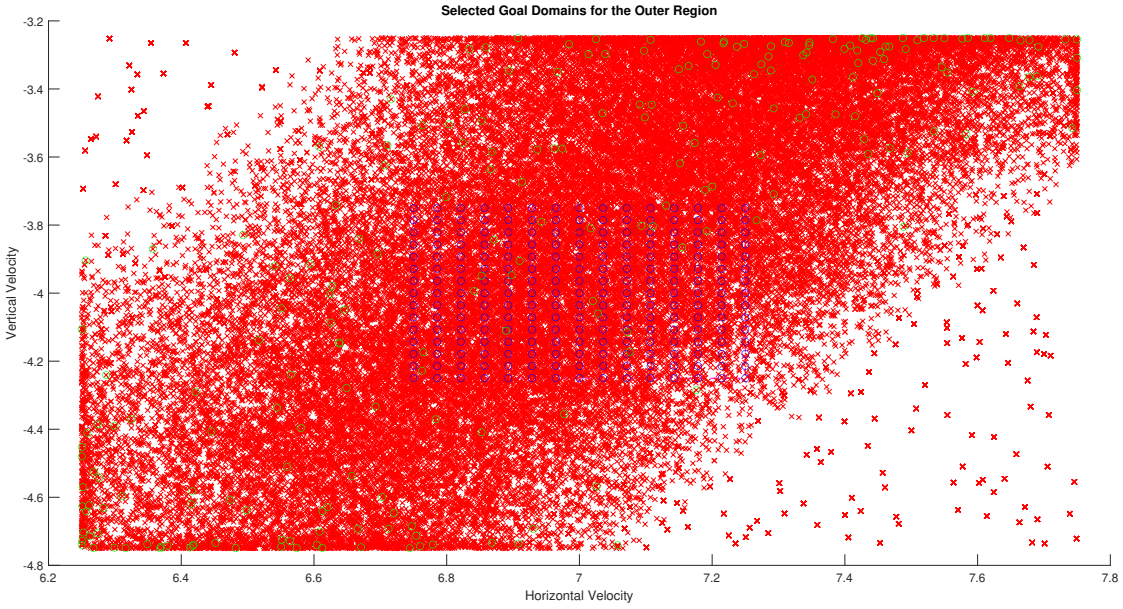

Figure 4.6: Selected Goal Domains for the Outer Cloud

green circle has a one-to-one relationship with the previous cloud's goal domain. Hence, any horizontal and vertical velocity combination can be pushed inside the safe guard region.

Differently from the previous figures, Figure 4.7 illustrates the first level of the outer constructed cloud. The reachable set of the outer cloud has been scattered with a deep orange color, and it has constructed by using the green circles, which are the outer cloud's goal domain vectors. Similarly in Figure 4.3, the upper left and lower right corner areas could not be sampled, but the remaining parts have covered the outer domain. The boundaries for the reachable vector set is selected as $\begin{pmatrix} 5.75\,\mathrm{m/s} & 8.25\,\mathrm{m/s} \end{pmatrix}$ for the horizontal velocity vector, and $\begin{pmatrix} -5.25\,\mathrm{m/s} & -2.75\,\mathrm{m/s} \end{pmatrix}$ for the vertical velocity vector.

Similarly in Figure 4.4, the Figure 4.8 demonstrates the 3D version of the Figure 4.7. As, the orange crosses are the reachable vector set for the green circles, the positional change between them shows the possible single stride movements. Not differently from the previous Figures, the obtained sets are similar, it has a wider range.

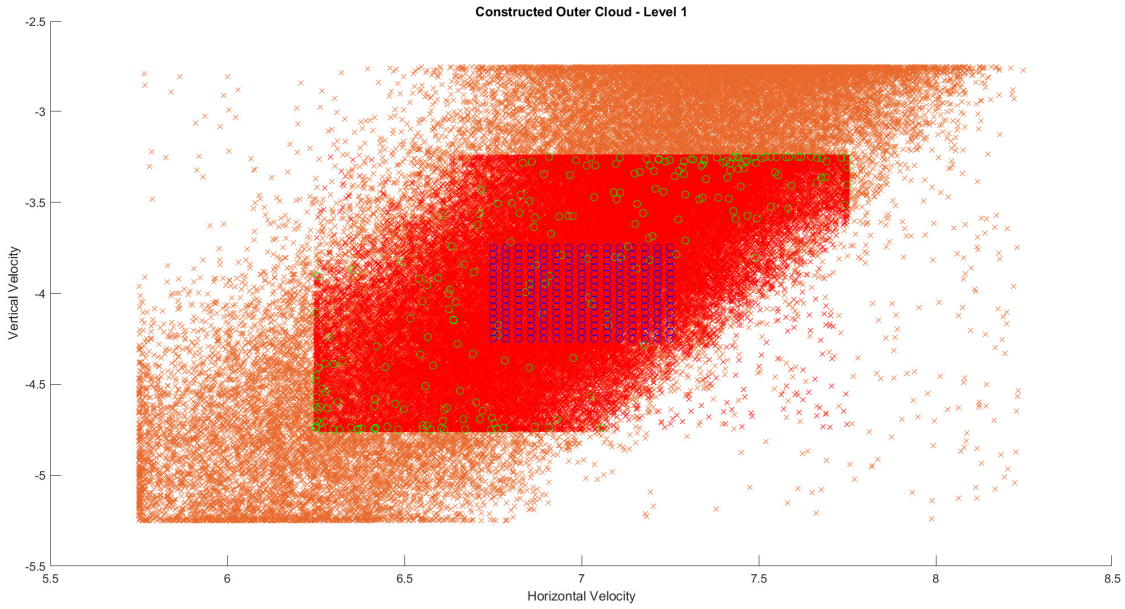

Figure 4.7: Constructed Outer Cloud - Level 1

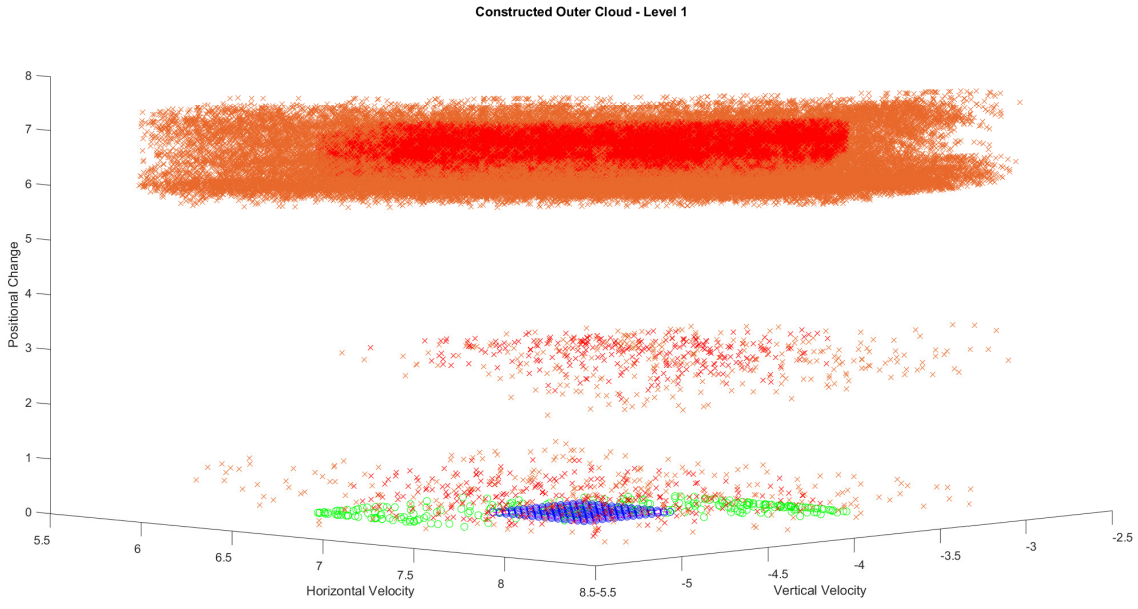

Figure 4.8: Constructed Outer Cloud (3D) - Level 1

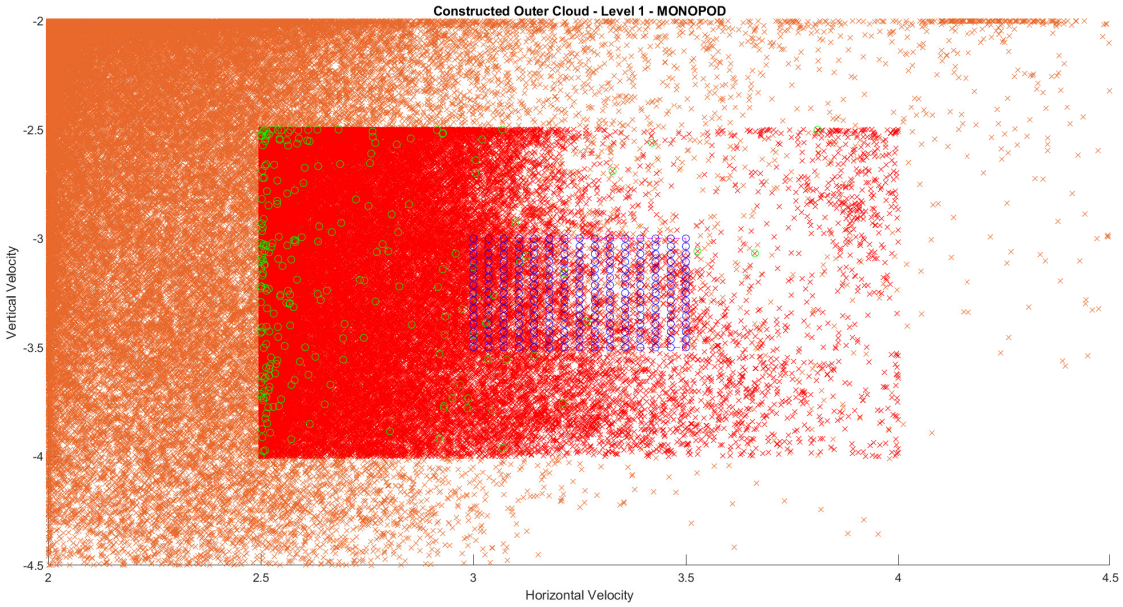

Figure 4.9: Constructed Outer Cloud - Level 1 - MONOPOD

In addition to this, the MONOPOD's constructed outer cloud has shown in the Figure 4.9. Differently from the Figure 4.7, the density unfolds at the left side of the plot, and diminishes when it advances from left to right. Just as demonstrated as in the Figure 4.5.

Chapter 5

Algorithm Implementation

The main idea behind the logic is that it is possible to connect each starting touchdown vector and goal domain vector with pairs. By connecting these pairs end to end, any distance becomes reachable, independent from the step count. The algorithm calculates the required number of steps to reach the destination by itself. The most important thing to not afford to overlook is that the amount of error when connecting the pairs.

The implementation part fundamentally consists of two parts. First one narrates the online implementation, whereas the second one emphasizes the offline implementation. Both of the parts have related subsections, and these subsections mainly highlights the forward and backwards planning implementation on SLIP and MONOPOD model, with respect to distance and minimum step count.

Figure 5.1 demonstrates the general overview of the Algorithm. After the initial touchdown state, the algorithm is executed based on the implementation, policy and planning type. As the Figure 5.2 illustrates the execution part fundamentally based on constructing the map in a recursive way and feeding the model based on the acquired results.

Regardless of the selected exploited model, the word online has been used in order to execute and construct the planning phase just after the beginning of the

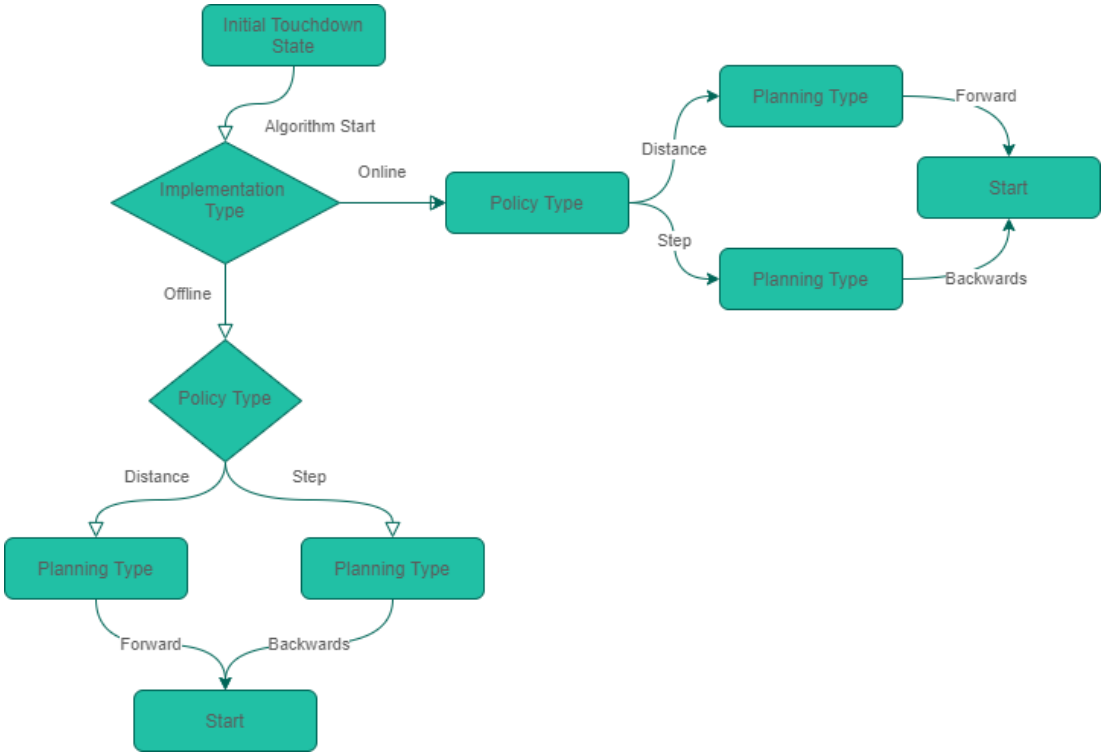

Figure 5.1: Algorithm General Overview

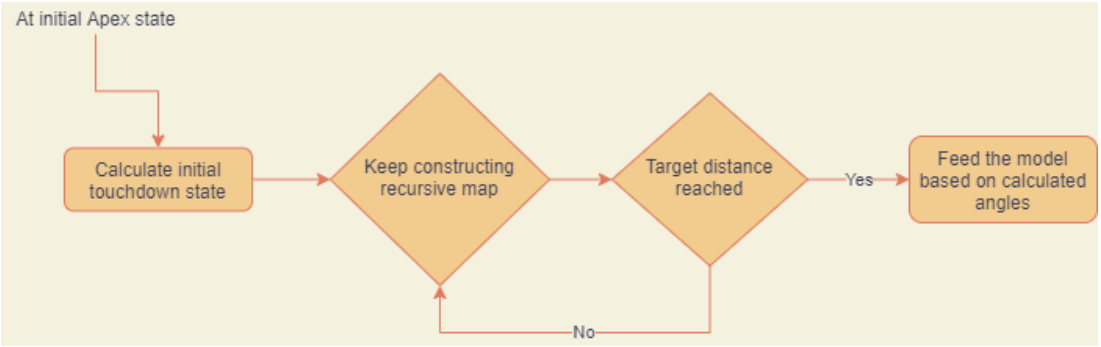

Figure 5.2: Logic General Overview

simulation. On the other hand, the offline keyword has been used to indicate that the planning of all possible ways to all possible distances has already been constructed before starting the algorithm.

At the beginning of the algorithm some parameters are assigned to select the implementation constraints. These constraints are as follows,

- ☐ implementation type: online — offline
- ☐ policy: distance — step
- ☐ planning type: forward — backwards
- ☐ waypoints: $\begin{pmatrix} waypoint_1 & waypoint_2 & \dots & waypoint_3 \end{pmatrix}$

The implementation type can be online or offline. If its assigned as online then the algorithm will construct the planning based on the starting touchdown horizontal and vertical velocity vectors, but if not then it will use the all possible ways mat file to choose the appropriate planning based on the same vectors. In addition to this the policy can be distance or step. If it's indicated as distance, then the algorithm will try to minimize error in between the given waypoints and model leg positions. If not, then the algorithm tries to construct the planning based on using the smallest step count. Forward and backwards planning are provided to choose the path construction type. Finally, the waypoints is an array where the algorithm or provider desires the model to place the leg positions.

In addition to this, Figure 5.3 shows how the recursive map is constructed. It basically follows similar procedures according to the planning type. Also, Figure 5.4 demonstrates the calculation of the possible list of θ_{td} and α (which is the amount of torque to be applied) values.

Figure 5.3: Recursive Map Construction

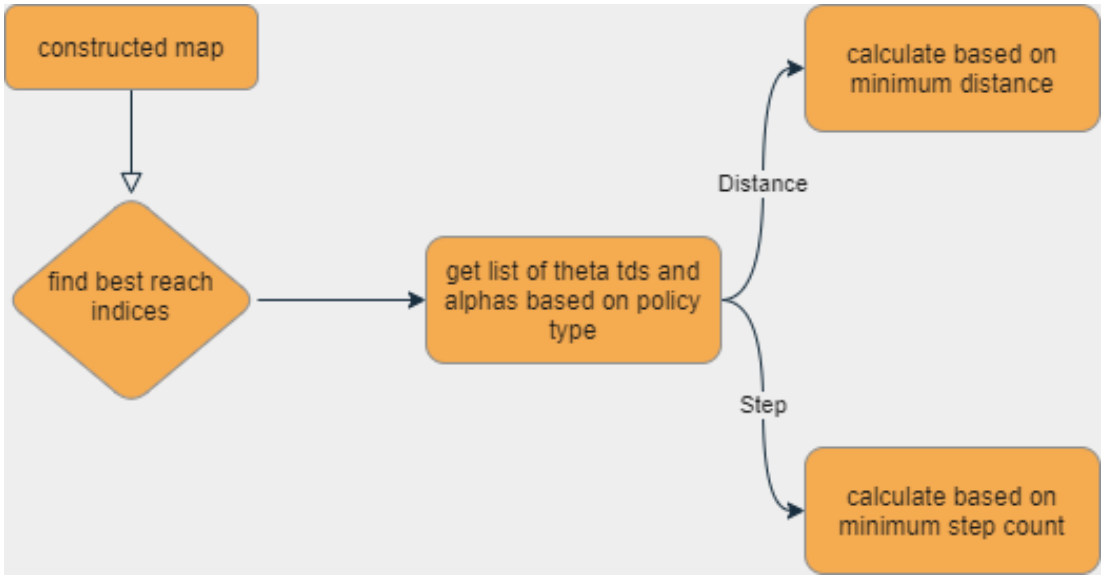

Figure 5.4: Best Theta TD and Alpha Calculation

5.1 Online Implementation

Online implementation takes place in two different models, i.e. constant energy and torque actuated with damping. Different construction methods and their combinations with each other take place, based on the parameter selection.

Every phase in this section begins with the very first touchdown state. The current horizontal and vertical velocity vectors are provided to an iterative algorithm as inputs, and the algorithm finds the possible goal domains. By feeding those goal domains as input to the same algorithm, it encounters new goal domains. In general this process iterates over all constructed clouds so that the desired waypoints can be reached.

In addition to this, the datum's constructed in Equation 4.2 have an error value. When the algorithm begins to plan the footsteps, this error value might lead it to an unsolved situation. Therefore, to eliminate the misleading plannings high errors ($> 1e{-}5$), a function always runs before everything else. The reason behind not eliminating these values in the Chapter 4 is to observe effects of distinct variations of errors on the simulation.

5.1.1 SLIP - Constant Energy - Model

5.1.1.1 Forward planning based on distance

After calculating the horizontal and vertical velocity vectors, based on the very first touchdown state, the forward planning starts with a function called $constructPathsCheckInterval$.

$$best\_vector\_set = constructPathsCheckInterval(\\ constructed\_vectors, velocity\_vector, target \\ ) \quad (5.1)$$

where:

$$\begin{aligned} constructed\_vectors &= constructed\ cloud\ domains\ (m*n*n) \\ velocity\_vector &= current\ touchdown\ velocity\ vector \\ target &= mainly\ includes\ the\ waypoints,\ error\ rates,\ max\ iteration\ count \end{aligned}$$

The Equation 5.1 takes 3 inputs, which are the constructed cloud domains, touchdown state velocity vectors, and a target value to control the conditions. This function creates a velocity region according to the provided target region error. Assume that the initial touchdown velocity vector is $\begin{pmatrix} 7.1000\,\mathrm{m/s} & -3.8763\,\mathrm{m/s} \end{pmatrix}$, and the region error as 0.2%. Then, the constructed velocity region becomes $\begin{pmatrix} 7.0858\,\mathrm{m/s} & -3.8685\,\mathrm{m/s} \\ 7.1142\,\mathrm{m/s} & -3.8840\,\mathrm{m/s} \end{pmatrix}$. This region can be considered as a square which has located on the Figures 4.7 or 4.9. This size of the region can be adjusted to different situations with respect to the assigned region error.

The reachable set vectors inside this region have been assigned as the proper vectors(minimum Euclidean distance), and their current touchdown state, goal domain vector, reach indices, and cloud indices are stored in the output object. In addition to this, in order to connect the goal domain vector and reachable vector set in a recursive way, the proper reachable vectors have given to function given below.

$$best\_vector\_set = constructPathsCheckIntervalRecursive(\\ constructed\_vectors, best\_vector\_set(i), target \\ ) \quad (5.2)$$

where:

$best\_vector\_set(i) =$ *the i-th reachable vector in the current set*

The recursive function, provided in Equation 5.2, identifies the next waypoint in the first place, and compares it with the current vector's position by using conditionals. The very first condition checks whether the first waypoint is the final one or not. If it's the last one, then the function continues with two additional conditions. Both of the conditions checks the distance between the simulation's current position and the last waypoint. If it's smaller than the desired last waypoint value (with an error interval), then it keeps iterating. If its not, but inside the acceptable interval, then it stores and continues with the next one, else it discards the path and moves to the next one. The acceptable range can be constructed by assigning a dynamic error value, because placing the same error value for any desired distance might be acceptable for small values, but causes greater fluctuation for greater distance values. Returning back to the conditionals, if the identified waypoint is not the last one, then we execute the same process for the smaller than case, if its not then we assign another adaptive error condition to check whether its passes the waypoint or not.

After constructing all of the possible paths, or in other words the *best_vector_set*, another important fundamental function which is called $findBestFitVectorBasedOnConstructedCloud$ is executed. The function is symbolically indicated below.

$$best\_reach\_indices = findBestReachIndicesBasedOnConstructedPaths(best\_vector\_set, target) \quad (5.3)$$

where:

$best\_reach\_indices$ = *closest paths to reach the provided waypoint array*

Equation 5.3 recursively calculates the best reach indices according to the closest distance values with respect to the provided waypoint inputs. It takes the constructed vector set in Equation 5.1 and the target object as inputs, and returns an array which stores the indices of the closest distances, according to an error value. To exemplify, assume that the last waypoint is $\left(80\,\mathrm{m}\right)$, and the error value is 0.1%, then the interval becomes $\left(79.92\,\mathrm{m} \quad 80.08\,\mathrm{m}\right)$. Therefore, the function, in Equation 5.3, gathers the possible indices whose positions falls inside this region, and returns them as an array.

After that, a minimization function takes the output of the Equation 5.3 as an input, and outputs a single index array which can be considered as the best cloud set combination to achieve the goal.

The below function calculates the necessary touchdown angle array based on the acquired vector set and their reach indices. It also outputs the best distance, which also the result of the provided policy -based on distance-.

$$[best\_theta\_tds, best\_reach\_index, best\_distance] = getClosestThetaTds\_basedOnDistance(best\_vector\_set, best\_reach\_indices, target) \quad (5.4)$$

where:

$$\begin{aligned} best\_theta\_tds &= best\ touchdown\ angles \\ best\_reach\_index &= best\ vector\ set's\ path\ indices \\ best\_distance &= calculated\ closest\ distance\ to\ achieve\ the\ goal \end{aligned}$$

In Equation 5.4, the function outputs three distinct objects, which are *best_theta_tds*, *best_reach_index* and *best_distance*. The variable *best_theta_tds* contains the best touchdown angles to reach the goal. The step size to reach the goal can also be noticed as the size of this array. Also, the variable *best_distance* is the distance that is closest to the desired waypoint array. The reason behind outputting the *best_reach_index* will be clarified in the Equation 5.5.

The functions contributed so far are only executed once, so that the best path can be revealed. Assigning the corresponding *best_theta_tds* value in each touchdown state pushes the algorithm to step on closest positions to the desired waypoint array. In this process, one important thing to remember is that, the closest positions can rarely become the desired waypoints (even most cases the *best_distance* in Equation 5.4), because the algorithm selects the closest velocity vector from the interval mentioned in part related with the explanation of Equation 5.1. Therefore, a positional correction (or can be considered as the angle correction) is required to take place at the each touchdown state. This function is symbolically given below.

$$\begin{aligned} &[recalculated\_theta\_td, error] = \\ &\qquad positionalCorrection(\\ &touchdown\_vector, best\_vector\_set, best\_reach\_index, best\_theta\_tds \\ &) \end{aligned} \tag{5.5}$$

where:

$$
\begin{aligned}
recalculated\_theta\_td &= corrected\ touchdown\ state\ angle \\
error &= correction\ error \\
touchdown\_vector &= current\ touchdown\ horizontal\ and\ velocity\ vector
\end{aligned}
$$

The function indicated in Figure 5.5 outputs the *recalculated_theta_td*, which can be considered as the corrected version of the corresponding value of *best_theta_tds* array. To exemplify this process, assume that current value of the *best_theta_tds* with respect to the touchdown count is $\left(0.4855\,\mathrm{rad}\right)$. When this value fed into the function, it becomes as $\left(0.4853\,\mathrm{rad}\right)$. The value of the corrected theta touchdown angle can vary according to the selected region error in target prop of the Equation 5.1. It is important to note that this minor improvements are very crucial, and must not be prevented to execute.

As mentioned, Equation 5.5 only corrects the touchdown angle if there exist another step to process. Therefore, for the final step, one closing minor correction takes place in order to reach the *best_distance* output of the Equation 5.4. The last step correction function fixes the final position of the simulated model, and eliminates the effect of the accumulated error so far.

The fundamental functions of this part can be considered as mentioned one at the start of this section. The remaining ones are essentially the updated, changed or adjusted versions of them.

5.1.1.2 Backwards planning based on distance

The planning phase of the constructing the possible paths in a backwards approach is the same until the beginning of the Equation 5.1. However, this time, rather than using that, the algorithm exploits a function called $constructPathsCheckIntervalBackwards$ to establish the paths.

$$best\_vector\_set = constructPathsCheckIntervalBackwards(\\ constructed\_vectors, target \\ ) \quad (5.6)$$

where:

$constructed\_vectors$ = *constructed cloud domains (m*n*n)*
$target$ = *mainly includes the waypoints, error rates, max iteration count*

Differently from the Equation 5.1, the backwards construction function takes 2 inputs. The initial touchdown horizontal and vertical velocity information is redundant for this step, because, the process can start with any final touchdown velocity vector. Hence, the construction process can be completed without the knowledge of the initial touchdown state velocity vectors.

Equation 5.6 begins with looping through all of the possible goal domains with respect to the provided constructed cloud set. For each goal domain, the possible reach set has been extracted and for each possible reach vector a region has been constructed. This region is very similar to the one mentioned in subsection 5.1.1.1. Then, among all possible reach vectors inside this region, a recursive function has been called to repeat this process until the the paths around the goal distance has been constructed. The functions symbolically given as follows;

$$best\_vector\_set = constructPathsCheckIntervalBackwardsRecursive(\\ constructed\_vectors, best\_vector\_set(i), target \\ ) \quad (5.7)$$

where:

$best\_vector\_set(i)$ = *the i-th reachable vector in the current set*

The recursive function in Equation 5.7, exploits the same logical process in Equation 5.2, but implements the construction method emphasized in this section. At the end of this process, a *best_vector_set* array has been constructed, and it carries the combination of all possible paths, which has positioned around the goal distance.

After acquiring the *best_vector_set*, same procedures are executed as after the Equation 5.2, which are fundamentally finding the best reach indices, acquiring the best touchdown angles from the provided indices, and executing the positional correction function on each stride.

5.1.1.3 Forward and Backwards planning based on minimum step count

Constructing a footstep plan based on the minimum step count is valid for both forward and backwards methods. It basically replaces the function indicated in Equation 5.4, and implements a different method to establish the essentials. The functions given symbolically as follows;

$$\begin{aligned}[best\_theta\_tds, best\_reach\_index, best\_distance] = \\ getClosestThetaTds\_basedOnMinimumStep(\\ best\_vector\_set, best\_reach\_indices, target \\ )\end{aligned} \quad (5.8)$$

where:

$best\_theta\_tds$ = *best touchdown angles*
$best\_reach\_index$ = *best vector set's path indices*
$best\_distance$ = *calculated closest distance to achieve the goal*

In Equation 5.8, the provided function's main purpose is to reach the goal

domain with minimum error and step count. When selected, among the provided best index array, the algorithm tries to find the least possible step count to reach the closest position to goal distance. The execution loops through all rows in the $best\_reach\_index$, and compares them with the local best distance and step count variable.

5.1.2 MONOPOD - Torque Actuated with Damping - Model

Differently from the constant energy model, the path planning part for the MONOPOD model, brings another controllable variable, which can be considered as the applied torque value. Therefore, new functions to be referred are the modified and updated versions of the functions indicated so far.

5.1.2.1 Forward planning based on distance

Planning phase begins at the very first touchdown state of the simulation. The acquired horizontal and vertical velocity have been given to a function sequence to execute a planning to achieve the goal. The planning phase starts with constructing the $best\_vector\_set$. The functions given symbolically as follow;

$$best\_vector\_set = constructPathsCheckInterval(\\ constructed\_vectors, velocity\_vector, target \\ ) \quad (5.9)$$

The function mentioned in Equation 5.9, uses almost the same logic with the Equation 5.1. The difference between two function implementations results from the distinct models. First difference is the selection of $constructed\_vectors$. The domain has been constructed with the use of MONOPOD model, and the established clouds can be observed from Figures 4.5 and 4.9.

As mentioned in the Equation 4.3, the domain's datum has an additional value of α, which has been used to specify the amount of applied torque. Therefore, the end to end connection of the reachable vectors and the goal domains are required to control not only the touchdown angle (θ_{td}), but also the applied torque over time(α).

The execution tasks clarified in the sub-section 5.1.1.1 also applies in this sub-section, too. Therefore, appending and implementing the explained differences construct valid paths for the MONOPOD model.

5.1.2.2 Backwards planning based on distance

This sub-section is basically the combination of sub-sections 5.1.1.2 and 5.1.2.1. As mentioned in the previous part, input constructed vector in Equation 5.6 has been selected from the preparation of the MONOPOD model. In addition to this, the path construction part with respect to the acceptable region controls the touchdown angle(θ_{td}) and applied torque(α), respectively.

The path construction policy can also be selected as step. However, as it will be the same implementation with Section 5.1.1.3, it is not necessary to reexplain it.

5.2 Offline Implementation

Similarly in Section 5.1, Offline process can be executed in constant energy and torque actuated with damping model. The logic of the benefited functions are fundamentally the same, but the implementations are different.

Each distinct planning in this section has been carried out in between the preparation phase and the beginning of the simulation. Therefore, this part can be considered as modifying the results achieved in Chapter 4, and resolving them in

the simulation. Equivalently in Section 5.1, the constructed paths are also based on the improved domain sets, whose affecting high error values are eliminated. Based on the constructed clouds in the previous section, the following function calculates all of the possible paths that can be constructed based on the provided input step size. The function is symbolically as follows;

$$[all\_possible\_paths] = constructAllPossibleWays(constructed\_vectors, level) \tag{5.10}$$

where:

$$\begin{aligned} all\_possible\_paths &= \textit{a huge sized array, based on the provided level} \\ level &= \textit{recursion level, can be considered as the step count} \end{aligned}$$

$$\begin{aligned} all\_possible\_paths.next = constructAllPossiblePathsRecursive(\\ constructed\_vectors, all\_possible\_paths(i) \\ ) \end{aligned} \tag{5.11}$$

where:

$$all\_possible\_paths(i) = \textit{i-th row of possible ways array}$$

The level parameter in Equation 5.10 can be indicated as the step count. It construct all possible paths based on the given step count. Assume that the level of the function is assigned as n, then the function recursively calls itself to construct paths starting from 1 to n. In other words, the function gathers all information, which consists the cluster of horizontal and vertical velocities, about a single stride, two step strides, three step strides, and n step strides.

After constructing the all of the possible ways, its straightforward to use it in the planning phase, because the algorithm has the opportunity to reach any

distance with respect to the initial touchdown state, without making any further calculation.

5.2.1 SLIP - Constant Energy - Model

5.2.1.1 Forward planning based on distance

Similar approach in Section 5.1.1.1, forward planning phase starts when the model is in initial touchdown state. The touchdown state's horizontal and vertical velocity has been given as an input the to calculation function. The mentioned function is symbolically as follows;

$$\begin{aligned}[best\_theta\_tds, best\_reach\_index, best\_vector\_field, best\_distance] = \\ findBestPath(\\ all\_possible\_paths, touchdown\_vector, target \\ ) \end{aligned} \quad (5.12)$$

where:

$best\_theta\_tds$ = *best touchdown angles*
$best\_vector\_field$ = *best vector set*
$best\_reach\_index$ = *best vector set's path indices*
$best\_distance$ = *calculated closest distance to achieve the goal*
$all\_possible\_paths$ = *all possible ways including n steps*
$touchdown\_vector$ = *current touchdown horizontal and velocity vector*
$target$ = *mainly includes the waypoints, error rates, max iteration count*

The resulting output variables are placed into the positional correction function, and with respect to the corrected angles, the constructed plan can be processed. Additionally, the policy type can also be selected as step, but its not separately mentioned as its indicated in Section 5.1.1.3.

5.2.2 MONOPOD - Torque Actuated with Damping - Model

Implementing the Equation provided in 5.10 with the results acquired from the MONOPOD part of the Chapter 4 constructs different possible paths for the torque actuated model. These paths can cover wider areas as there exists two control inputs, which are stated as the touchdown angle (θ_{td}) and the applied torque(α).

5.2.2.1 Forward planning based on distance

Forward planning on MONOPOD model applies the same procedure indicated in section 5.1.2.1. The resulted touchdown angles and torque values are provided from the best possible path to reach the goal. The same positional correction and last step correction function has been used to eliminate the positional velocity error. It is also important to note that, like previous parts, the policy can be selected as the minimum step count.

Chapter 6

Results

The results part consists of two sections. One of them demonstrates the constant energy model results, and the other one illustrates the torque actuated with damping results.

Obtained results are executed in the MATLAB environment [44]. The specifications of the simulation computer are 64 GB ram and Intel(R) Xeon(R) E-2176M CPU @2.70GHz, 2.71GHz.

6.1 SLIP - Constant Energy - Model Results

6.1.1 Generic Results

Each figure in this section follows the same plotting template, and consists of 3 rows and 5 columns. The illustration template can be examined in a detailed way.

- Row 1, Column 1: *Horizontal position change over time*

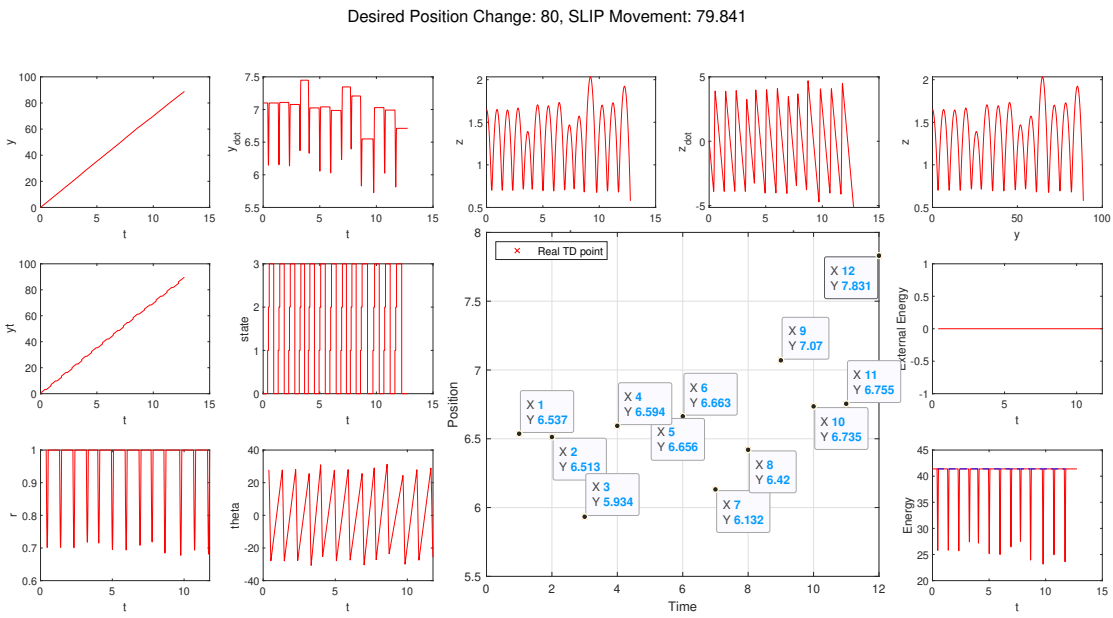

Figure 6.1: Online - Forward Planning - Minimum Distance

The positional change over time in Figure 6.1 demonstrates the position of the model body. The leg position is not included, because it's position alters back and forth in time, rather than increasing in a constant (very close to constant) way.

- Row 1, Column 2: *Horizontal velocity change over time*

The horizontal velocity changes in between the states. It's expected that the velocity should increase in the liftoff state, decrease in the touchdown state, and remains constant on apex and bottom states. When the demonstrated Figures are examined carefully, these velocity alterations can be observed.

- Row 1, Column 3: *Vertical position change over time*

Unlike horizontal position, the vertical position does not constantly increase. It basically represents the current height of the model, and expected to be maximum at apex, and minimum at the bottom states. Therefore, the each periodic

behavior can be considered as a single step.

- Row 1, Column 4: *Vertical velocity change over time*

Like Horizontal velocity, the vertical velocity also alters in between the states. At apex and bottom the vertical velocity is expected to be 0, minimum at touchdown state, and maximum in bottom state.

- Row 1, Column 5: *Vertical position change over horizontal position*

This plot is very similar to Row 1 and Column 3, the only difference only occurs on the plot's x axis. This is actually beneficial for the cases, where the horizontal position is not constantly increasing (Different ground types, outside effects).

- Row 2, Column 1: *Leg position change over time*

The graph of the leg position is almost identical to the horizontal position. The variation of the ragged view originated from the states.

- Row 2, Column 2: *State change over time*

As it can be understood from the name, the plot illustrates the current state change of the model over time. It follows a periodic change as long as the used model moves.

- Row 2, Column 5: *Supplied external energy change over time*

For the SLIP model, this part is always 0, because the model follows a constant energy approach. On the other hand, for the MONOPOD model this part becomes meaningful.

- Row 3, Column 1: Spring compression

It only changes between the beginning of the touchdown state and the end of bottom phase. Therefore, in between these states, the values if smaller than 1, and for the remaining phases, it becomes 1.

- Row 3, Column 2: *Leg angle change over time*

This plot is one of the most important one among the others. It demonstrates the current leg angle, and the touchdown state angles can be smoothly observed.

- Row 3, Column 5: *Total energy over time*

Total energy over time plot is also illustrated in the general figure template. Just like external energy, for the SLIP model it's supposed to be a constant, and for the MONOPOD model, it's expected as a combination of increasing and decreasing values.

- Row 2,3 Column 3,4: *Positional Change in between the Touchdown states over time*

The footstep positions are illustrated in the bigger part of the generic figure. Visualizing the implemented model's behavior on each stride is important to observe the behavior of the model.

Figure 6.1 shows an example of constructed planning for the SLIP model. The waypoint values are provided as $\begin{pmatrix}19\,\mathrm{m} & 45\,\mathrm{m} & 80\,\mathrm{m}\end{pmatrix}$. Therefore, the model has to step on the provided array in a connected manner. The provided figure's first three strides are $\begin{pmatrix}6.537\,\mathrm{m} & 6.513\,\mathrm{m} & 5.934\,\mathrm{m}\end{pmatrix}$, and the addition of these positions is $\begin{pmatrix}18.984\,\mathrm{m}\end{pmatrix}$. As this value is very close to the first waypoint, it can clearly be stated that the first goal has been reached. For the second goal

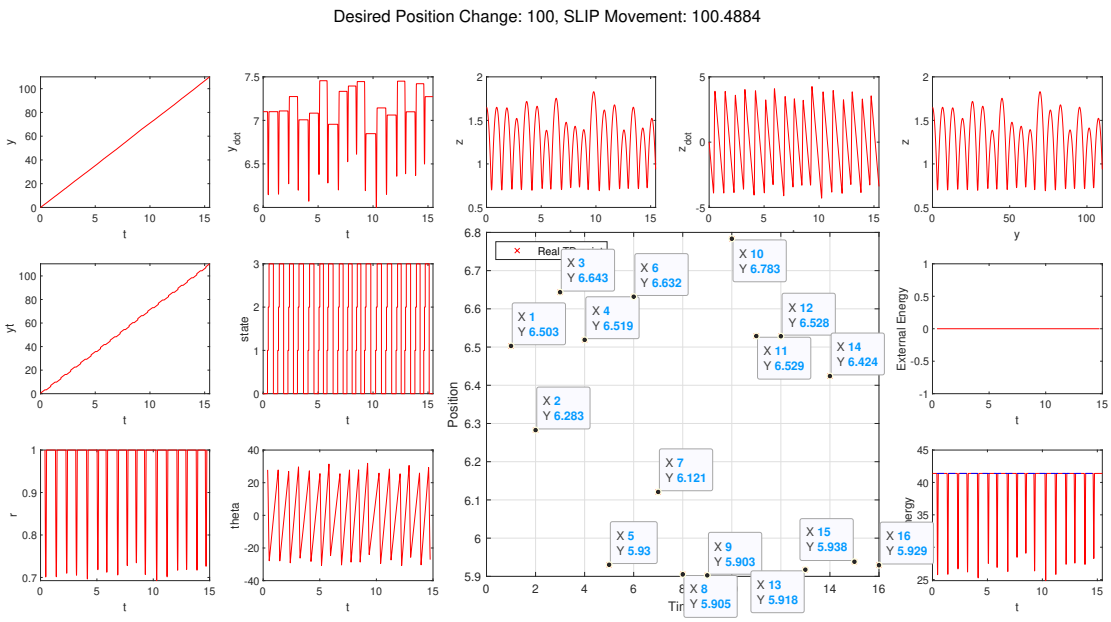

Figure 6.2: Online - Forward Planning - Minimum Distance

point, which is $\left(45\,\mathrm{m}\right)$, the model successfully jumps for four more steps. The positional differences of these steps are $\left(6.594\,\mathrm{m} \quad 6.656\,\mathrm{m} \quad 6.663\,\mathrm{m} \quad 6.132\,\mathrm{m}\right)$, respectively. The total positional difference from the initial touchdown state becomes $\left(45.032\,\mathrm{m}\right)$, so the second waypoint has been reached. Finally, for the last waypoint or the latter goal position, the remaining steps are $\left(6.42\,\mathrm{m} \quad 7.07\,\mathrm{m} \quad 6.735\,\mathrm{m} \quad 6.755\,\mathrm{m} \quad 7.831\,\mathrm{m}\right)$, jointly. The total value of the last steps is $\left(34.811\,\mathrm{m}\right)$, Therefore, the total positional difference from initial touchdown state to goal position becomes $\left(34.811\,\mathrm{m} + 45.032\,\mathrm{m}\right) = 79.843\,\mathrm{m}$. As a result, the first waypoint is reached in 3 strides, second one in 7 strides and the last one in 12 strides. The planning has been successfully executed with an error value of 0.0019914, or in other words, the deviation is 0.19%.

Figure 6.2 demonstrates the same logic with different waypoint set. The waypoint set is randomly selected as $\left(32\,\mathrm{m} \quad 63\,\mathrm{m} \quad 100\,\mathrm{m}\right)$. First 5 strides of the model are $\left(6.503\,\mathrm{m} \quad 6.283\,\mathrm{m} \quad 6.643\,\mathrm{m} \quad 6.519\,\mathrm{m} \quad 5.93\,\mathrm{m}\right)$, so the total positional difference is $\left(31.978\,\mathrm{m}\right)$. Next 5 steps are

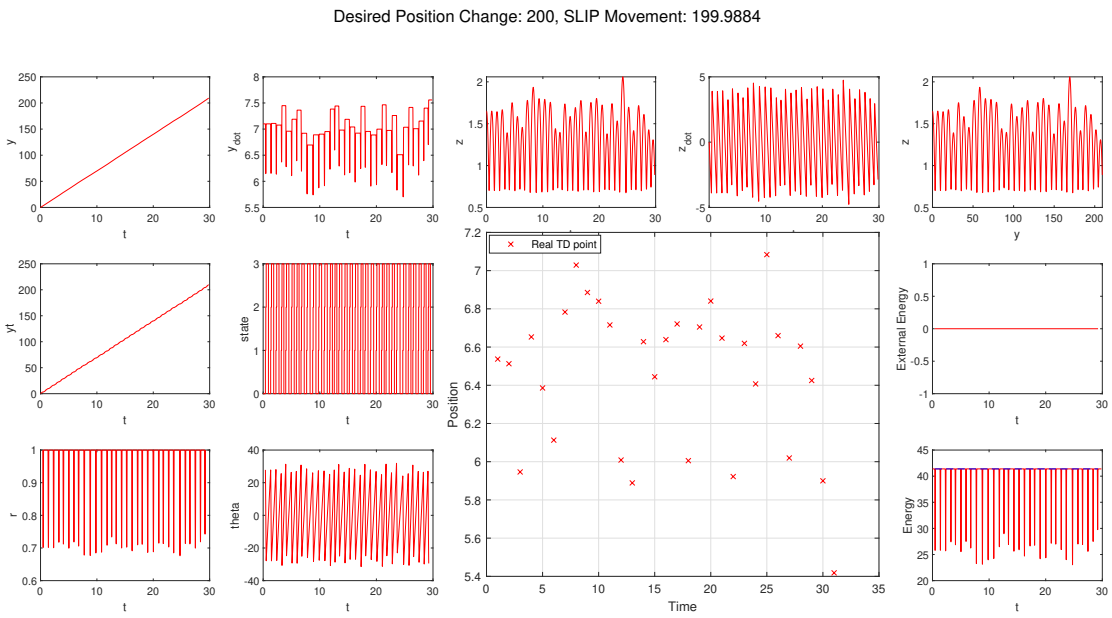

Figure 6.3: Online - Forward Planning - Minimum Distance

$\left(6.632\,\mathrm{m} \quad 6.121\,\mathrm{m} \quad 5.905\,\mathrm{m} \quad 5.903\,\mathrm{m} \quad 6.783\,\mathrm{m}\right)$ ($\left(63.222\,\mathrm{m}\right)$), and the in remaining strides are in total $\left(100.4884\,\mathrm{m}\right)$. Hence, from the initial planning phase the first waypoint has been reached in 5 steps, next one in 10 steps, and the final one in 16 steps. Also, the error percentage is 0.48%.

In Figure 6.3, the waypoint count has been increased to 5, and the distance final goal distance incremented by two times. The algorithm successfully planned the footstep, with respect to the provided waypoints. The final positional change is $\left(199.9884\,\mathrm{m}\right)$, the number execution steps are 5, and the error percentage is 0.058%.

6.1.2 Comparison - Online & Offline Implementation

This subsection fundamentally compares online and offline implementation results. As the algorithmic difference almost the same as mentioned in Sections 5.1.2.1 and 5.2.2.1, the main resulting difference is occurred from the amount of

time to construct a plan.

The randomly acquired goal distances of Figures 6.4 and 6.5 are equivalent. Therefore, they represent the results of a two step planning. The executed plans are very similar to each other (i.e. $\left(13.0005\,\mathrm{m}\right)$ and $\left(13.008\,\mathrm{m}\right)$). However, the positional difference occurs from the implementation characteristics. The offline implementation accommodates every possible planning scenario, whereas the online version reserves a portion of it (at most 3000 appropriate paths). The reason behind holding a part is to avoid waiting a lot in the planning phase. The execution time of the offline planning took approximately 10 ms, whereas the construction of possible online paths occupy 0.76 s.

Differently from Figures 6.5 and 6.4, Figures 6.7 and 6.6 illustrate an example of 4 step planning, where the desired distance is assigned as $\left(26\,\mathrm{m}\right)$. The results of both implementation methods are very close to the goal position ($\left(26.0041\,\mathrm{m}\right)$ and $\left(26.0003\,\mathrm{m}\right)$, respectively). However, the difference between plan construction times increases exponentially when the step count is incremented. Therefore, offline planning took approximately 20 ms, and the online one occupied 7.45 s.

Final Figures in this section (6.8 and 6.9) demonstrates results of a 5 stride footstep plan. The construction times are 25 ms and 14.3 s, respectively.

6.1.3 Comparison - Forward & Backwards Planning Implementation

The specifications of the forward and backwards planning implementations are mentioned at the Sections 5.1.1.1 and 5.1.1.2. Therefore, this subsection only emphasizes the results and their corresponding comparisons.

Figures 6.10 and 6.11 demonstrates the results of the forward and backwards planning, with respect to a random goal distance. Both of the methods successfully reached very close positions to the goal destination, and their execution

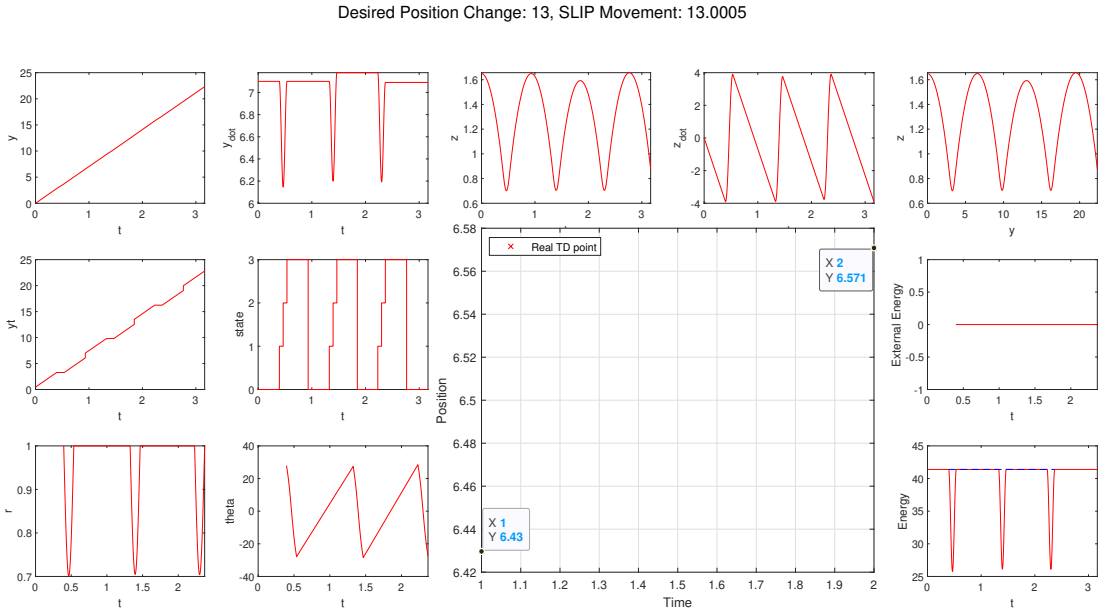

Figure 6.4: Target position = 13 m - Offline - Forward Planning - Minimum Distance

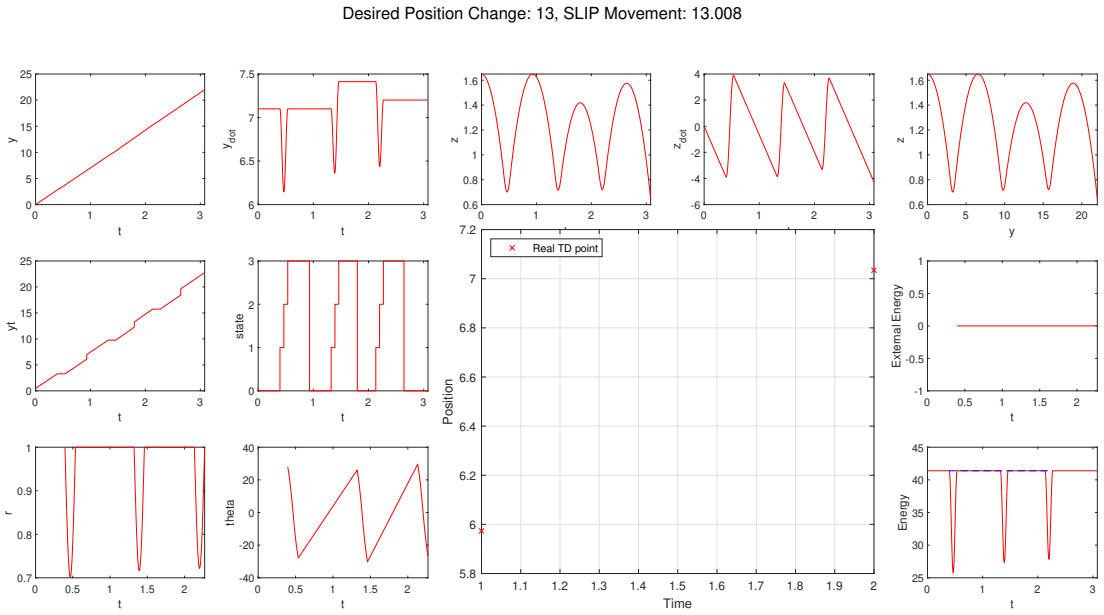

Figure 6.5: Target position = 13 m - Online - Forward Planning - Minimum Distance

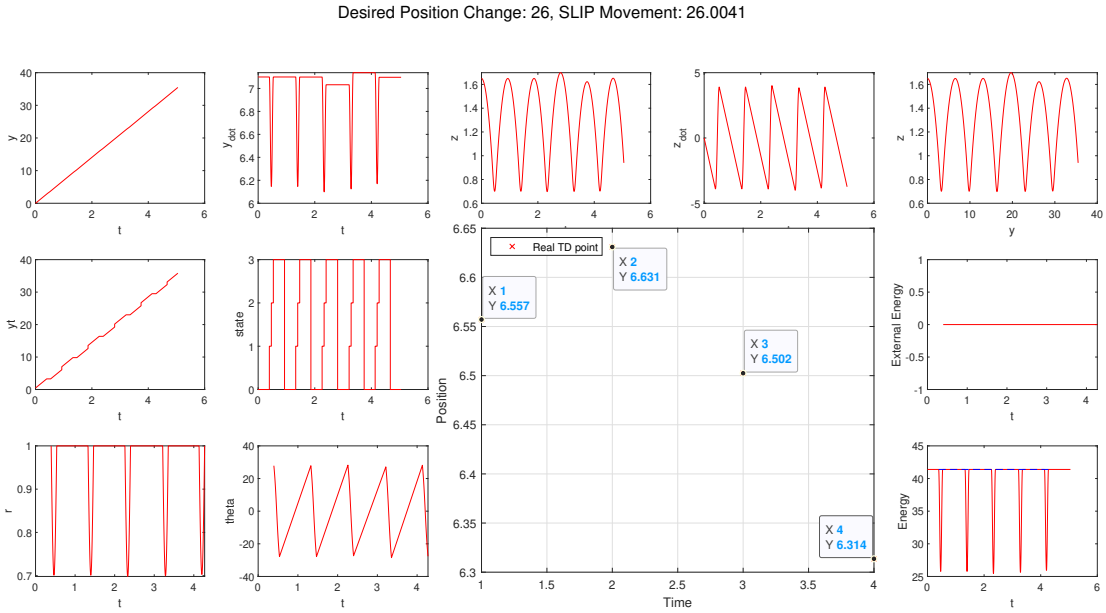

Figure 6.6: Target position = 26 m - Offline - Forward Planning - Minimum Distance

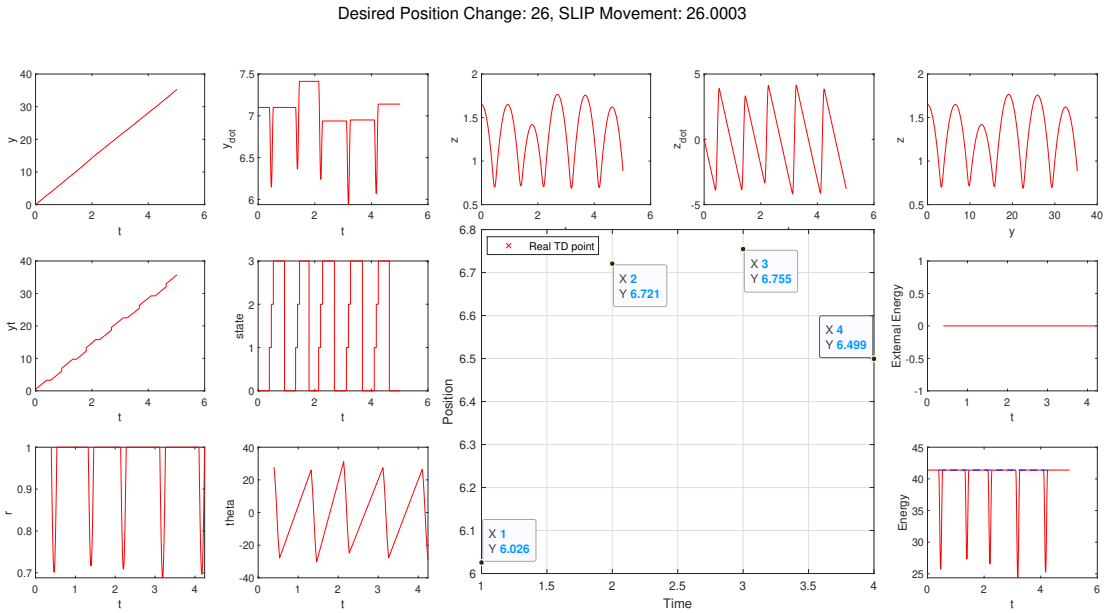

Figure 6.7: Target position = 26 m - Online - Forward Planning - Minimum Distance

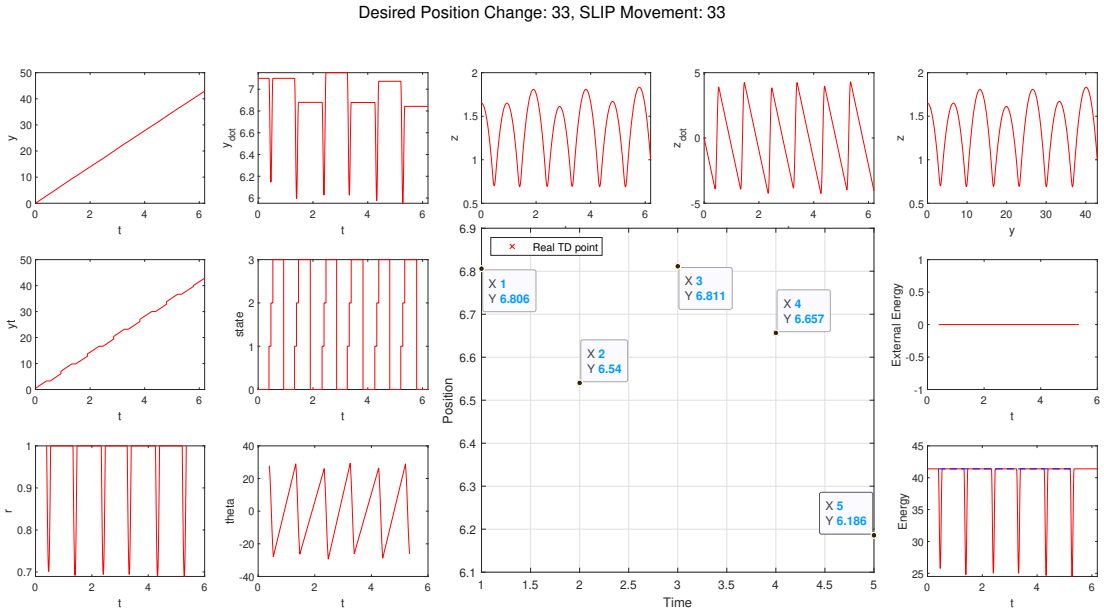

Figure 6.8: Target position = 33 m - Offline - Forward Planning - Minimum Distance

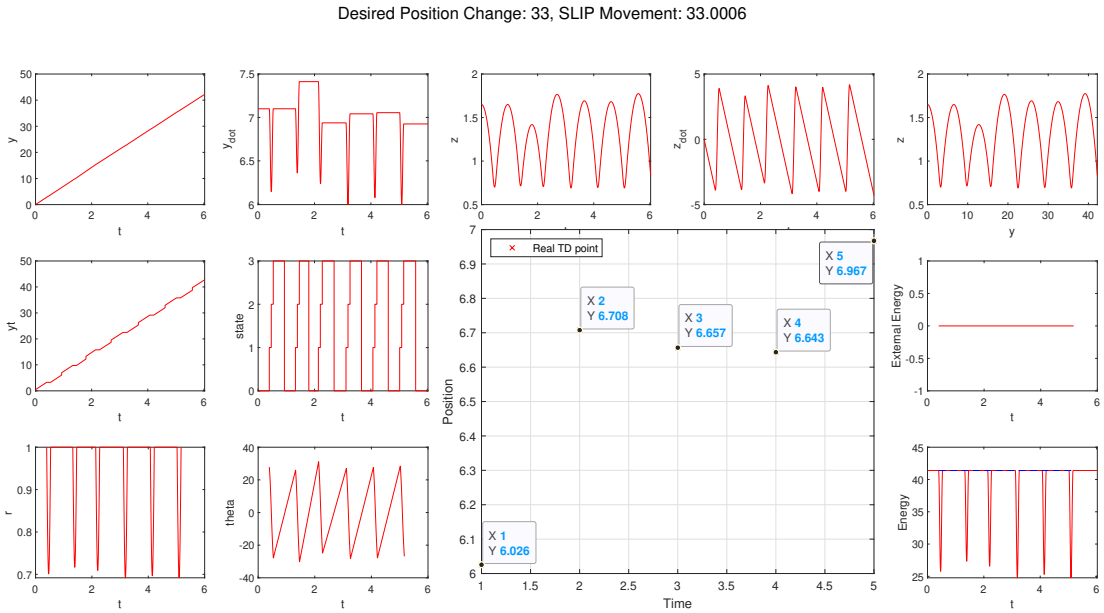

Figure 6.9: Target position = 33 m - Online - Forward Planning - Minimum Distance

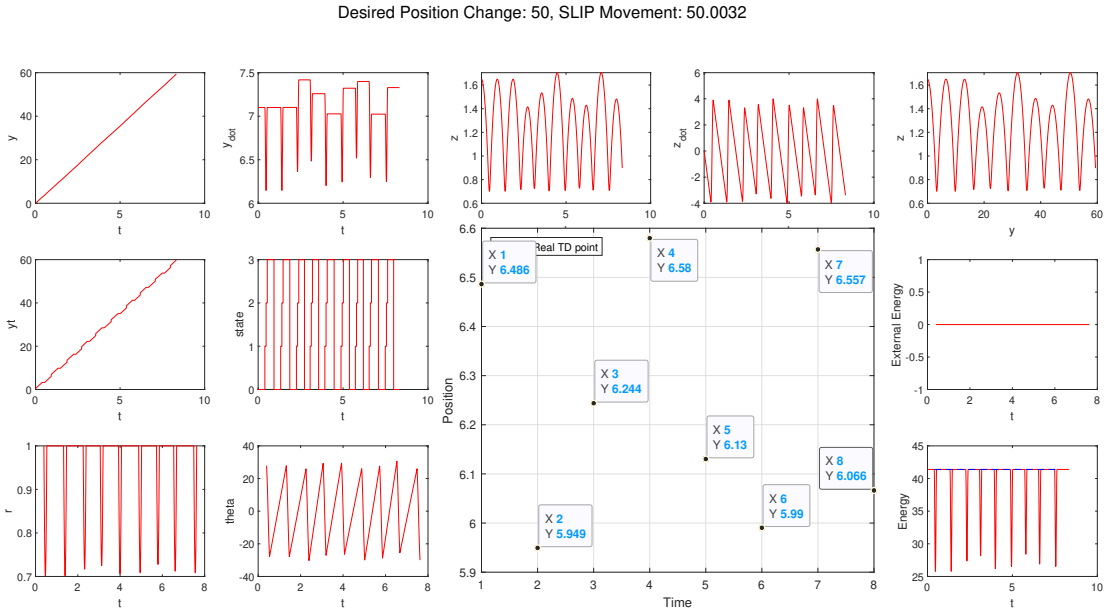

Figure 6.10: Target position = 50 m - Online - Forward Planning - Minimum Distance

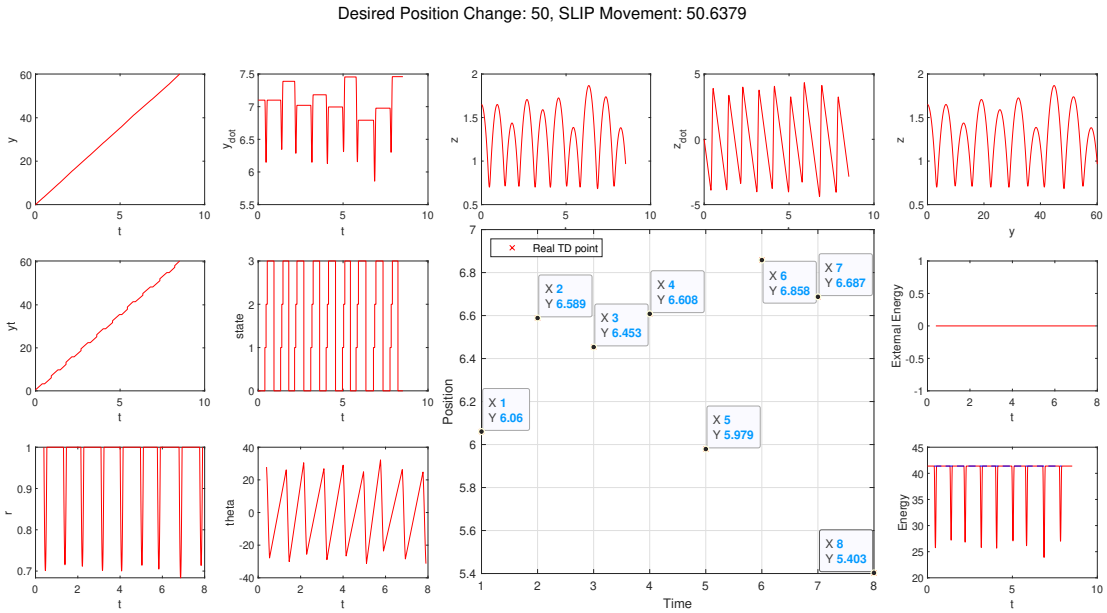

Figure 6.11: Target position = 50 m - Online - Backwards Planning - Minimum Distance

times are not much different from each other. According to the provided Figures, the number of steps can be counted as 8, and the plan construction times are 40.26 s (for forward) and 58.04 s (for backwards), respectively.

In addition to this, Figures 6.12 and 6.13 represents the results when the goal destination is increased to 58.04 m. The number of executed strides can be counted as 9, and the execution times are 36.26 s (for forward) and 64.04 s (for backwards), respectively. Comparing with the previous Figures, Although the stride count has been increased, for the forward case, the amount of time to construct the plan seems like decreased. The reason behind this situation can be interpreted as the goal distance is in a better reachable interval than the previous one.

The very last Figures of this section consists of 6.14 and 6.15. The number of step count can be regarded as 11, and the plan establishment times are 43.46 s (for forward) and 71.61 s (for backwards), respectively.

6.1.4 Comparison - Policy Type - Step & Distance Implementation

Figures 6.16 and 6.17 demonstrate the obtained results with respect to minimum distance and minimum step count. For the first figure, the policy type is selected as minimum distance, so there are 13 steps were required to reach the target position. However, second illustration shows the same position can be reached with lesser steps, i.e. 12, with a slightly increased error value. The error values of the provided plots are 0.09%, and 0.1%.

6.1.5 Comparison - Different Initial Touchdown States

Previous Subsections demonstrate the results based on the same initial touchdown state, which is $\begin{pmatrix} y_{horizontal} & 7.1000\,\mathrm{m/s} & 0.8842\,\mathrm{m} & -3.8763\,\mathrm{m/s} & y_{horizontal} + \theta_{td} \end{pmatrix}$.

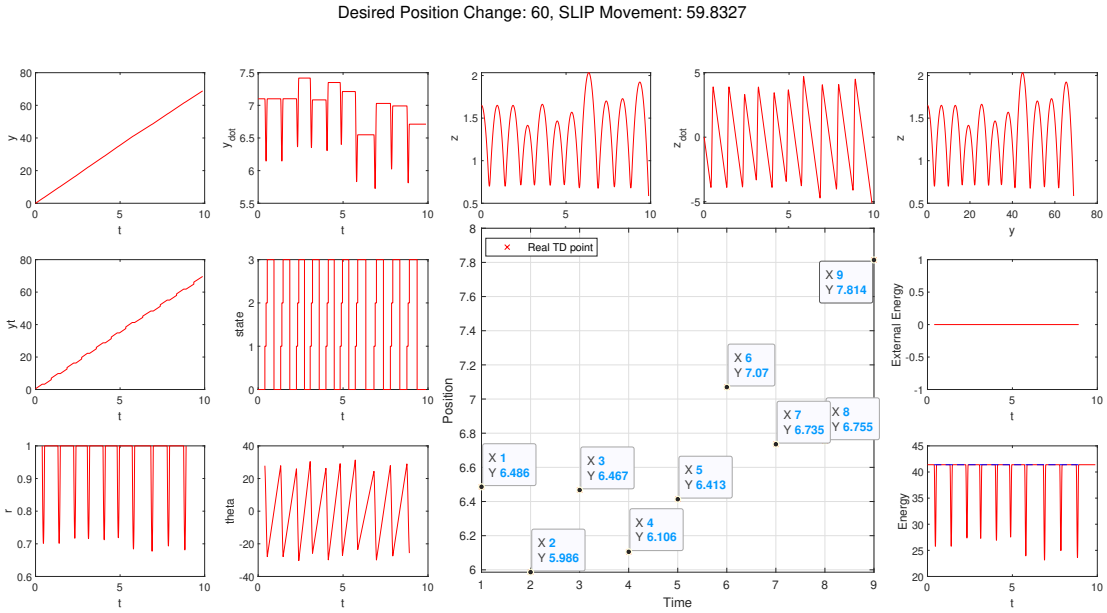

Figure 6.12: Target position = 60 m - Online - Forward Planning - Minimum Distance

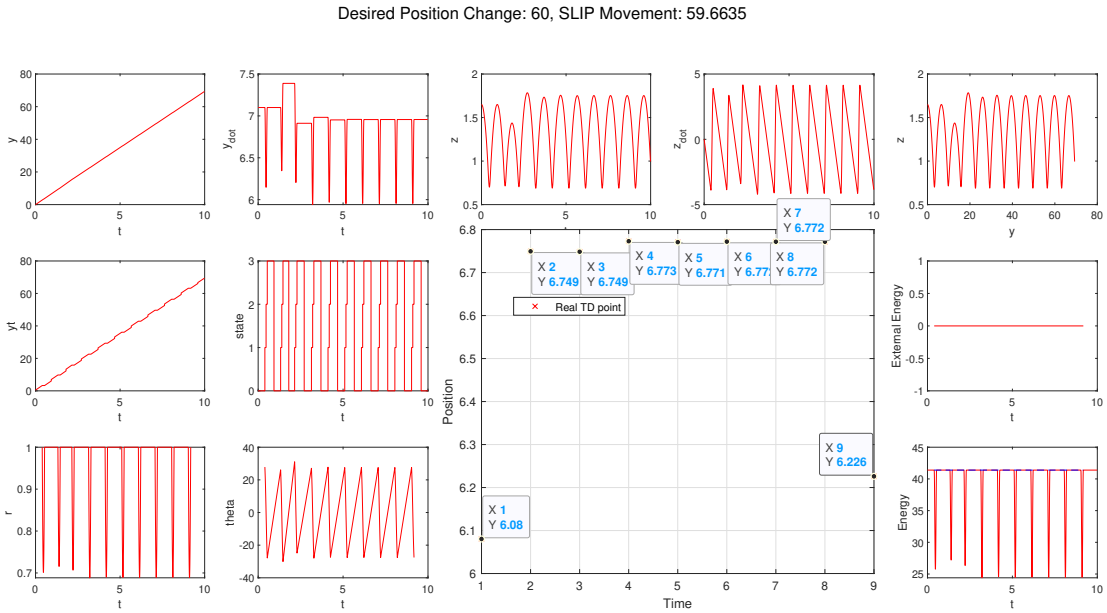

Figure 6.13: Target position = 60 m - Online - Backwards Planning - Minimum Distance

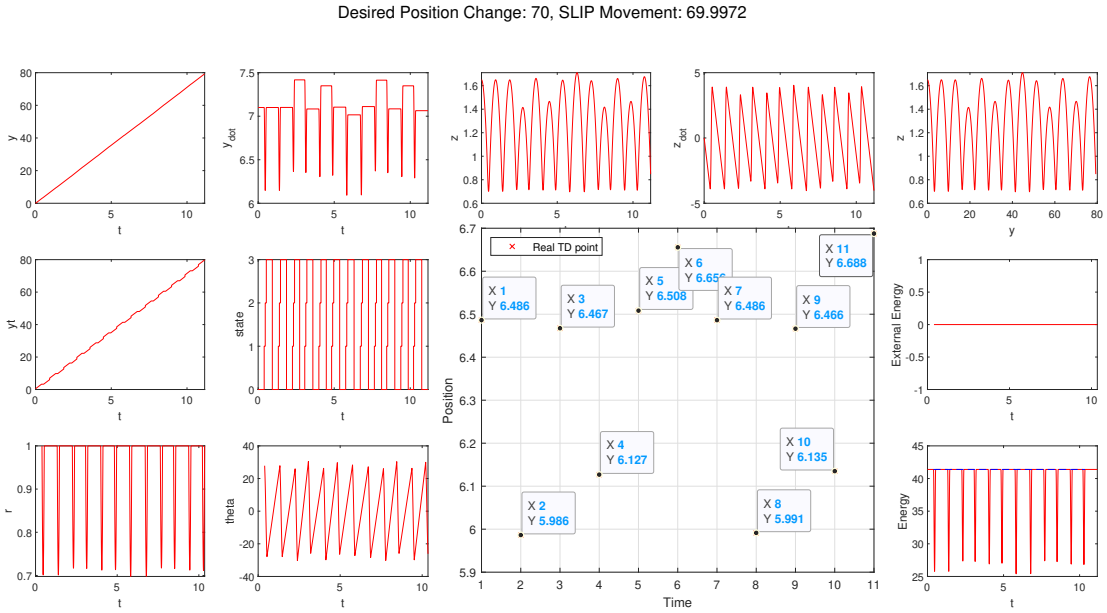

Figure 6.14: Target position = 70 m - Online - Forward Planning - Minimum Distance

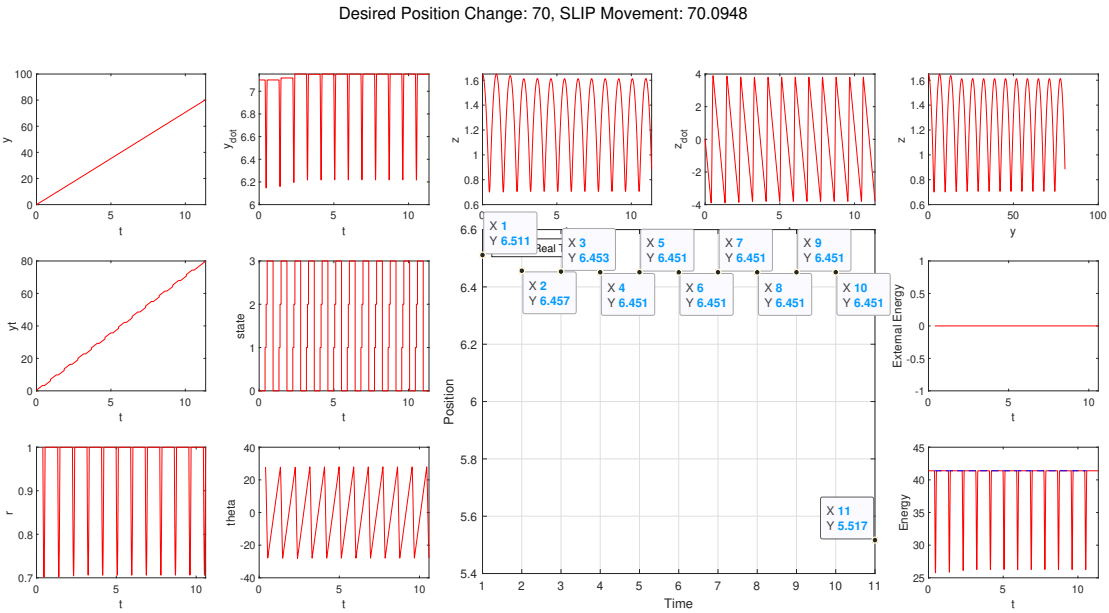

Figure 6.15: Target position = 70 m - Online - Backwards Planning - Minimum Distance

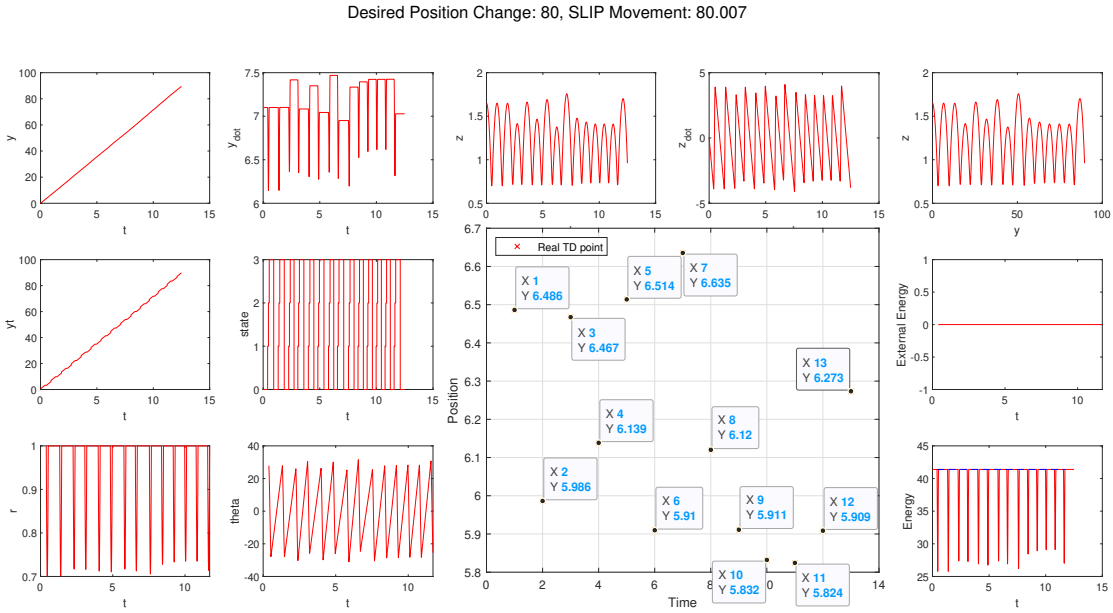

Figure 6.16: Target position = 80 m - Online - Forward Planning - Minimum Distance

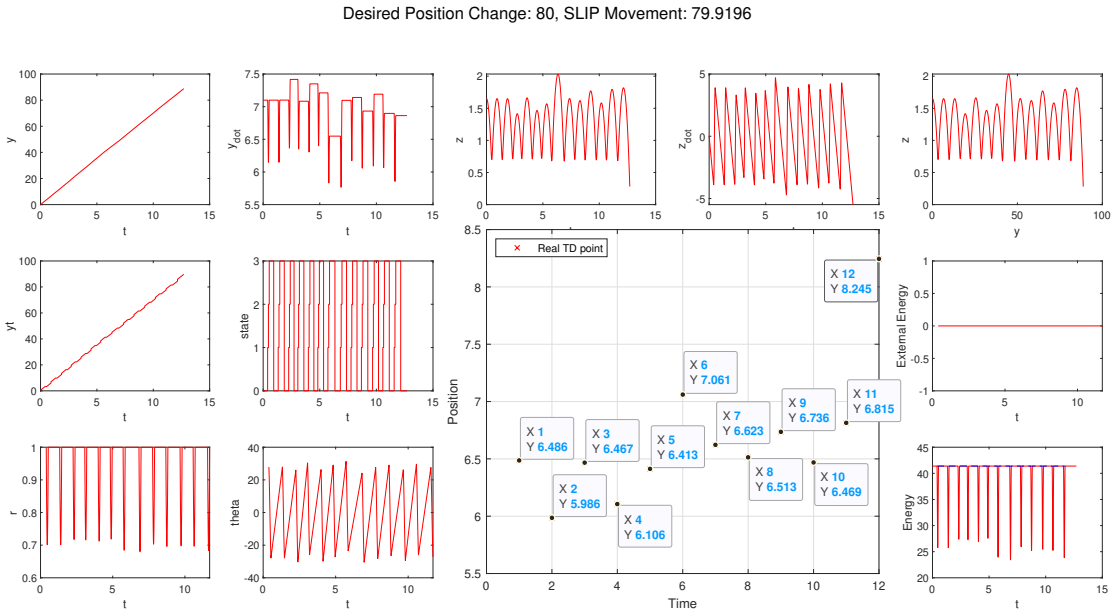

Figure 6.17: Target position = 80 m - Online - Forward Planning - Minimum Step Count

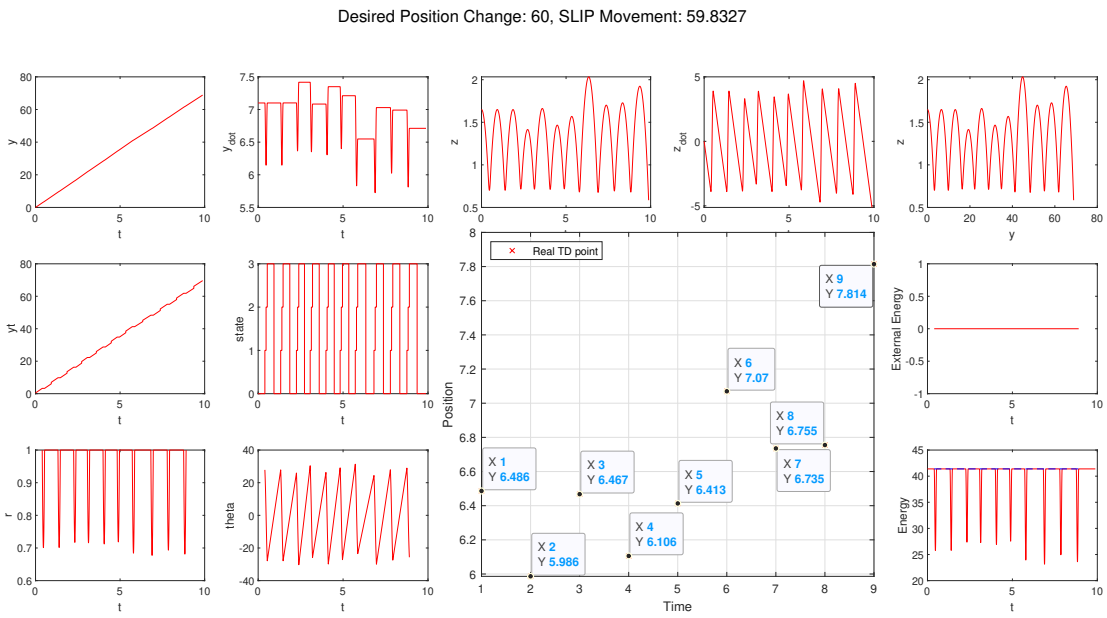

Figure 6.18: Target position = 60 m - Online - Forward Planning - Minimum Distance - $vel_{horizontal} = 7.1000\,\mathrm{m/s}$, $vel_{vertical} = -3.8763\,\mathrm{m/s}$

The value of $y_{horizontal}$ does not mean anything as the algorithm considers it equal to 0. The reason behind this consideration is that it is actually the point where planning started to be constructed. Second parameter in the matrix is the horizontal velocity, third one is the current height (vertical position), fourth one is the vertical velocity and the last one is the leg position. Therefore, different initial state results can be observed from the following figures.

Figure 6.19 demonstrates the result where the goal distance is selected as 60 m. Positional leg differences can be discovered by comparing it with Figure 6.18.

Similarly in the previous Figure 6.19, algorithm worked perfectly, and constructed the planning based on the given initial velocity of $vel_{horizontal} = 7.7000\,\mathrm{m/s}$, $vel_{vertical} = -3.9121\,\mathrm{m/s}$.

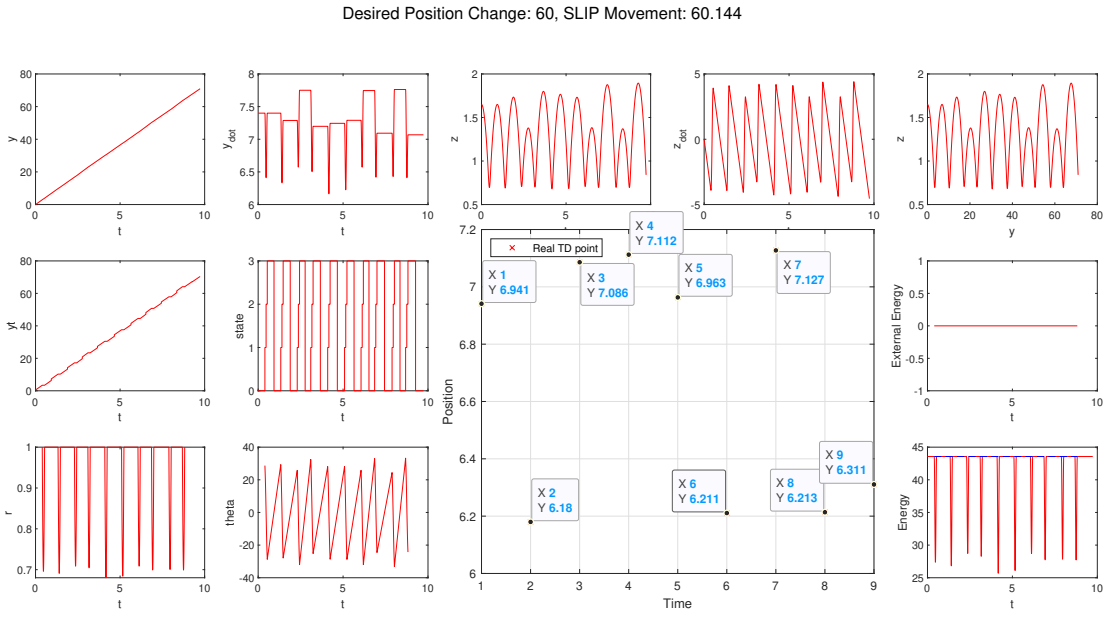

Figure 6.19: Target position = 60 m - Online - Forward Planning - Minimum Distance - $vel_{horizontal} = 7.4000\,\mathrm{m/s}$, $vel_{vertical} = -3.8942\,\mathrm{m/s}$

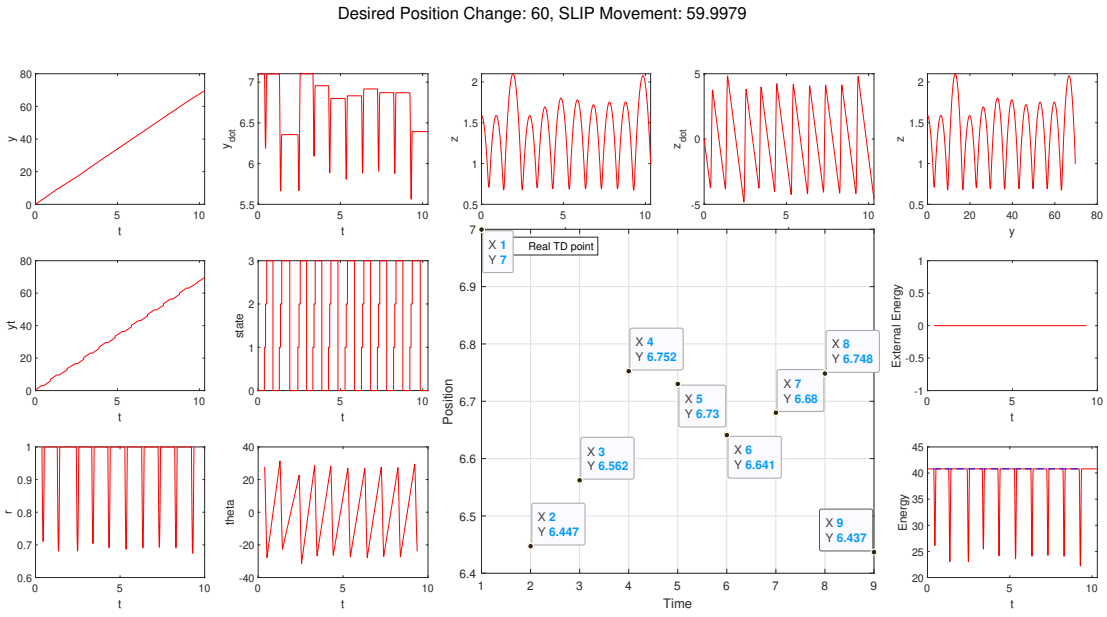

Figure 6.20: Target position = 60 m - Online - Forward Planning - Minimum Distance - $vel_{horizontal} = 7.7000\,\mathrm{m/s}$, $vel_{vertical} = -3.9121\,\mathrm{m/s}$

6.2 MONOPOD - Torque Actuated with Damping - Model Results

6.2.1 Comparison - Online & Offline Implementation

Comparably with the previous section, the MONOPOD model's online and offline implementation is also satisfactory for reaching the desired goal positions. The fundamental difference occurs from the time constraints.

According to the results illustrated in Figures 6.21 & 6.22, the algorithm successfully reached the provided goal distance. The step size in this part is relatively smaller than the one proposed in the SLIP results section. Therefore, the number of strides to execute a plan is also increased, respectively. Both of the model implementations successfully completed the task by stepping 12 times, and the error percentages resulted as 0.0017% and 0.0074%. On the other hand, for the offline case, the path construction process took 30 ms, and for the online case, it took 18 s.

In addition to this, Figures 6.23 and 6.24 illustrates the results when the target distance is selected as 45 m. Both of the implementation types consistently jumped for 19 times. Error rates are 0.24% and 0.0026%, respectively. Finally, the path construction took 35 ms and 39.29 s.

6.2.2 Comparison - Forward & Backwards Planning Implementation

Figures 6.25 and 6.26 shows the results of forward and backwards implementation on the torque actuated with damping model. The goal distance is selected as 20 m, and execution error has came up as 0.008% and 0.004, respectively. The amount of time to find the best path is much bigger than the offline case, but very similar to each other. In total, the algorithm has computed to reach the goal by 8 steps.

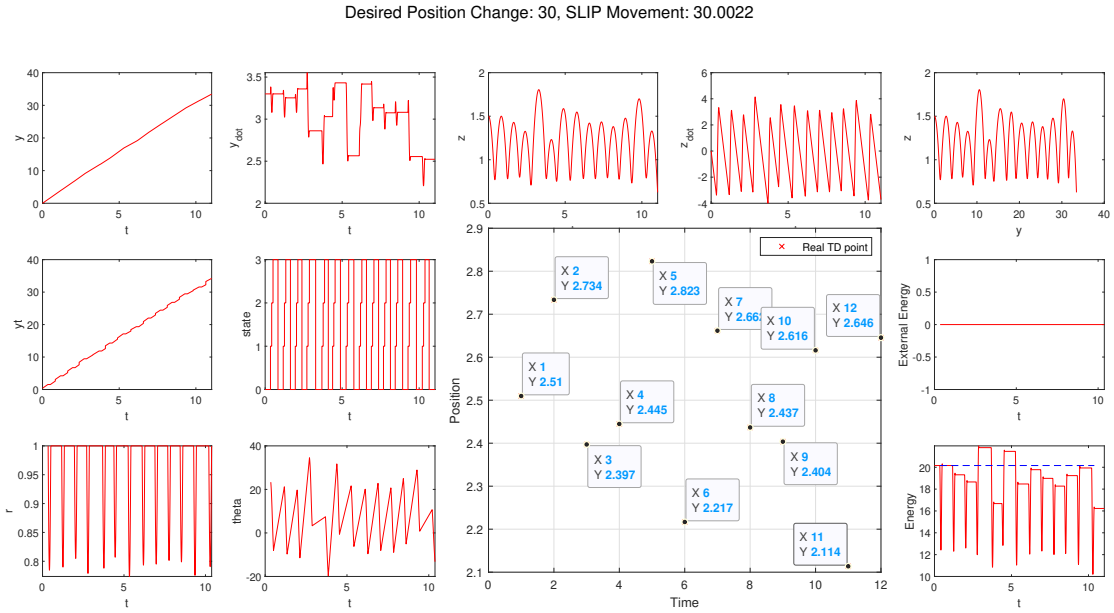

Figure 6.21: Target position = 30 m - Offline - Forward Planning - Minimum Distance

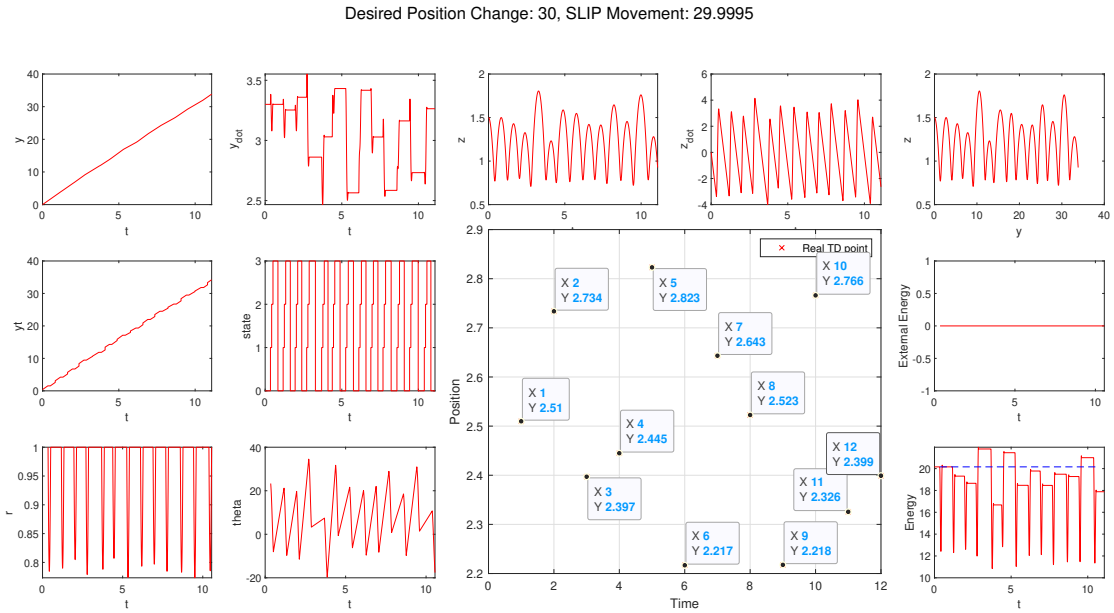

Figure 6.22: Target position = 30 m - Online - Forward Planning - Minimum Distance

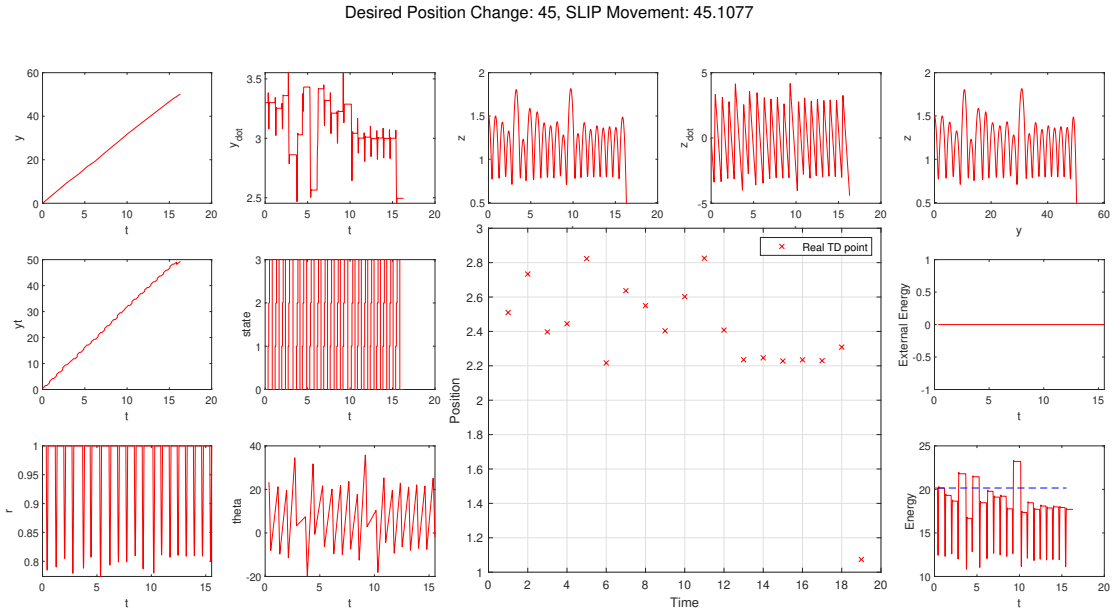

Figure 6.23: Target position = 45 m - Offline - Forward Planning - Minimum Distance

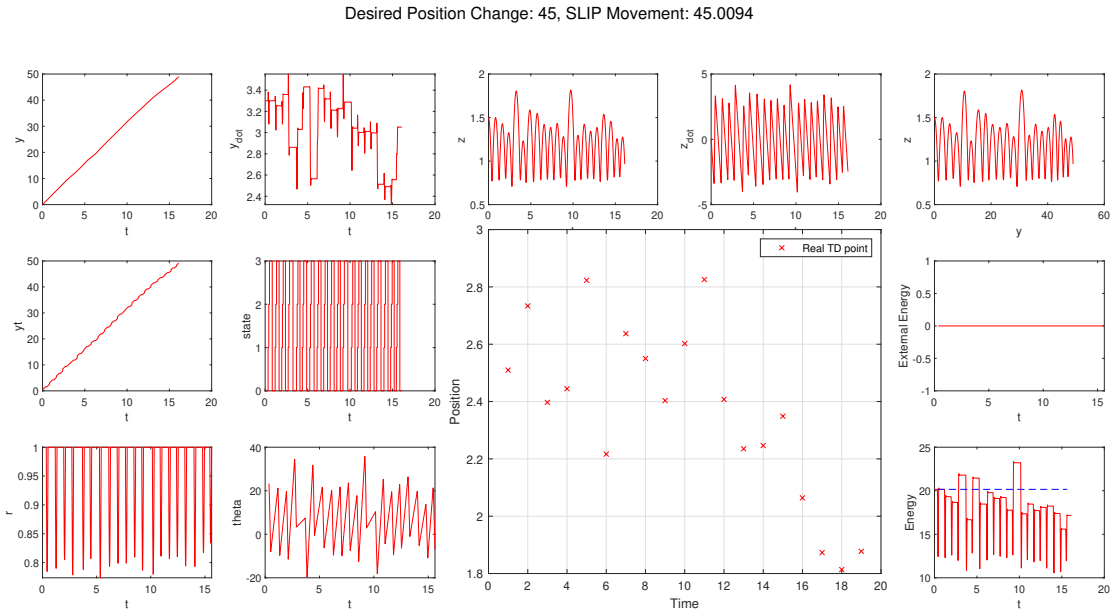

Figure 6.24: Target position = 45 m - Online - Forward Planning - Minimum Distance

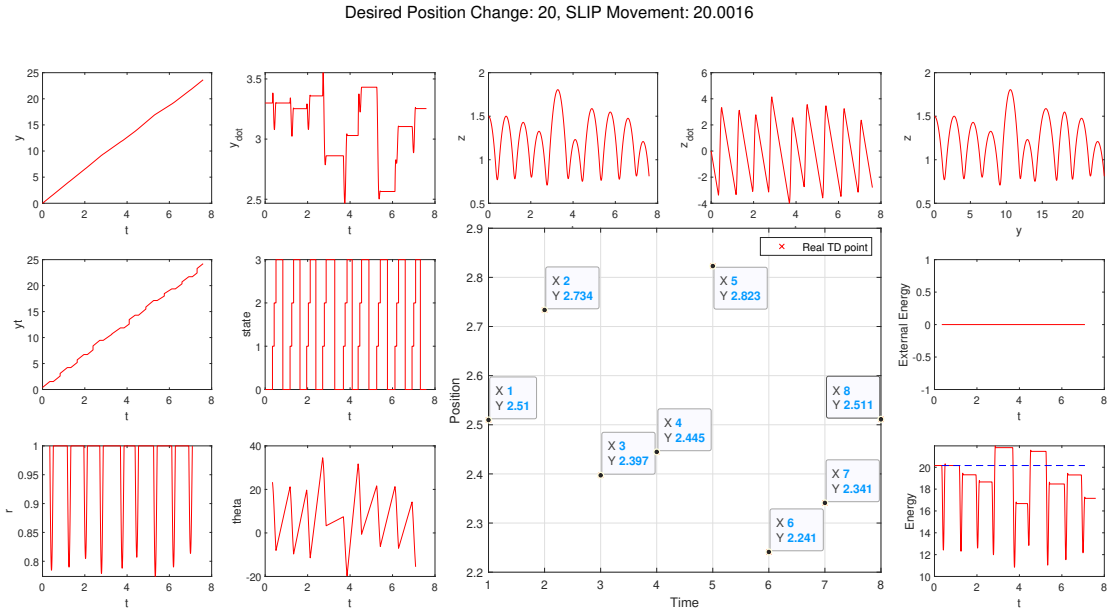

Figure 6.25: Target position = 20 m - Online - Forward Planning - Minimum Distance

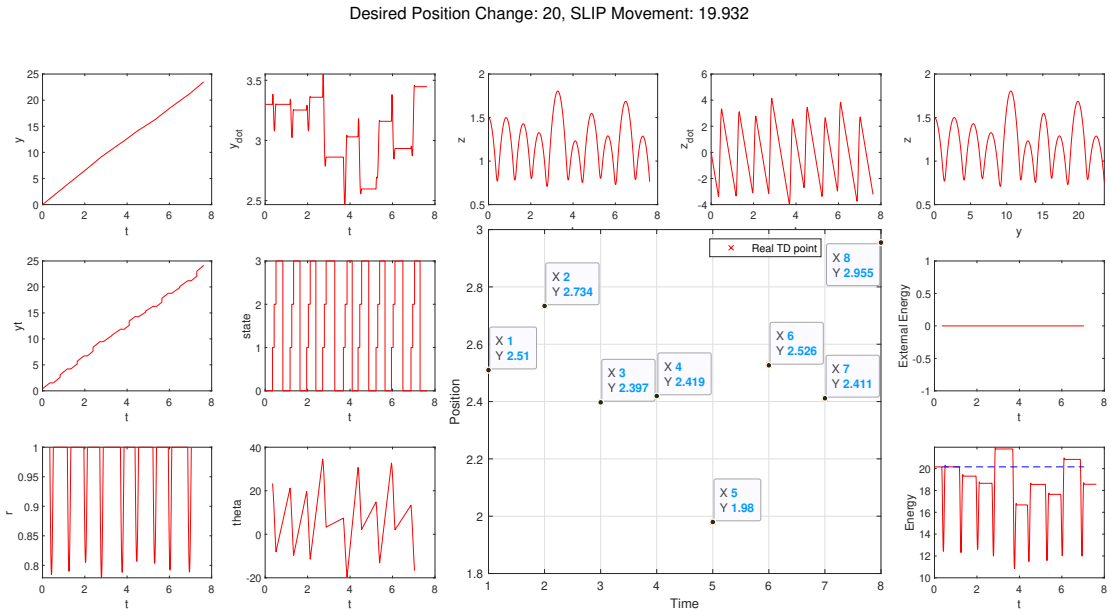

Figure 6.26: Target position = 20 m - Online - Backwards Planning - Minimum Distance

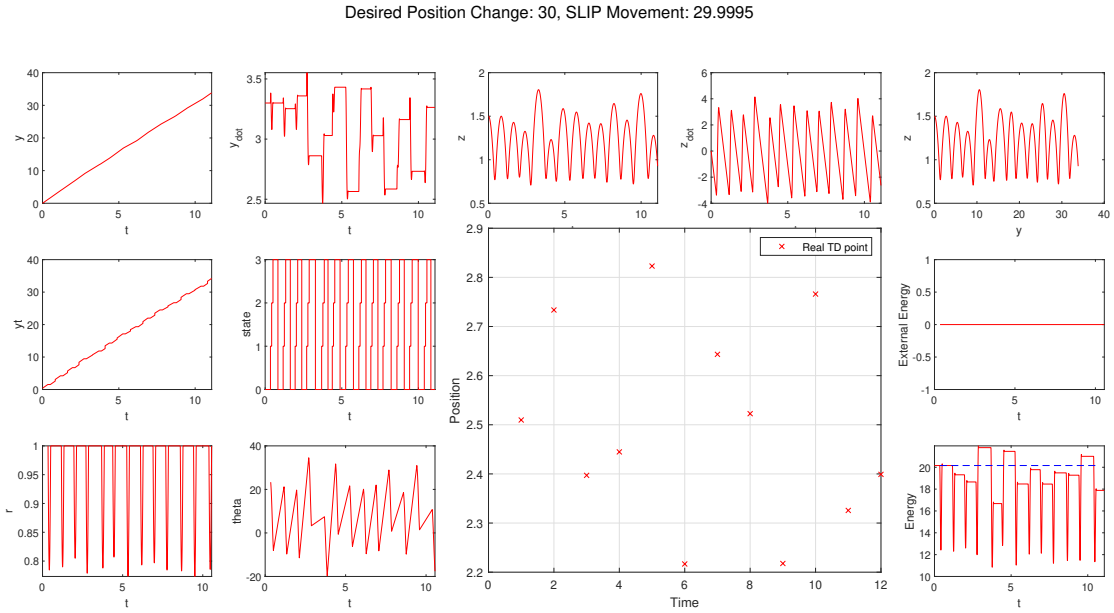

Figure 6.27: Target position = 30 m - Online - Forward Planning - Minimum Distance

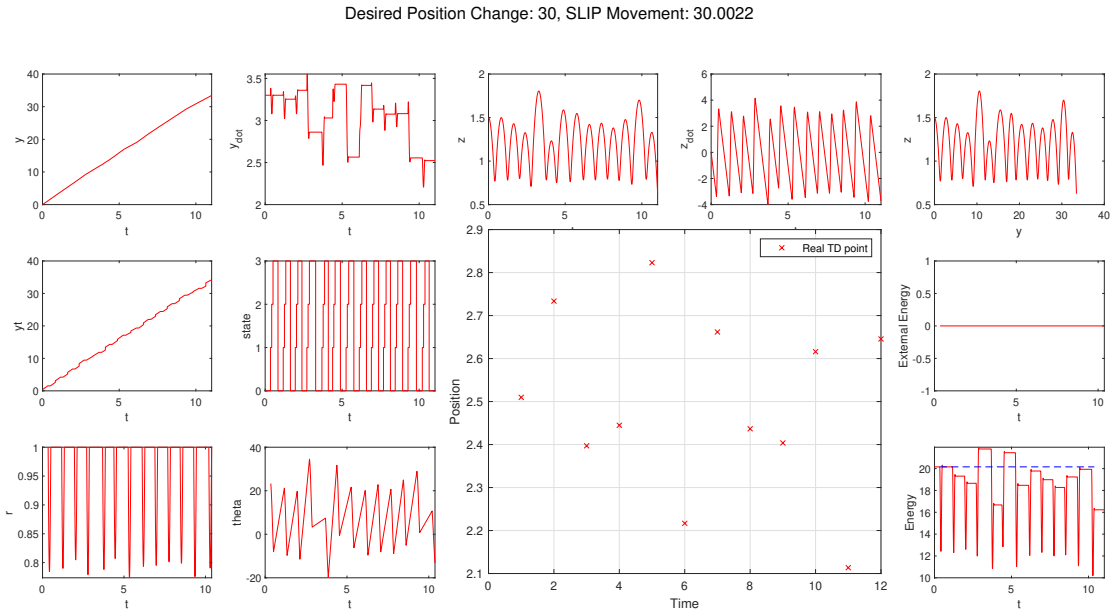

Figure 6.28: Target position = 30 m - Online - Backwards Planning - Minimum Distance

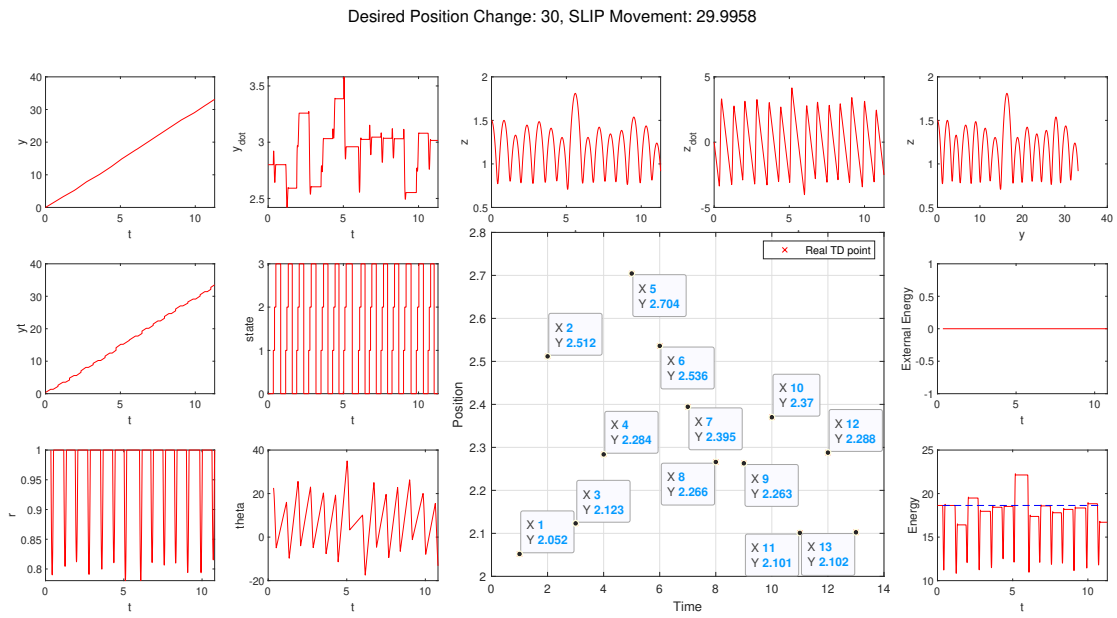

Figure 6.29: Target position = 30 m - Online - Forward Planning - Minimum Distance - $vel_{horizontal} = 2.8000\,\mathrm{m/s}$, $vel_{vertical} = -3.3640\,\mathrm{m/s}$

When the distance is incremented by 10 m, the MONOPOD model also increases its stride count by 4, and managed to reach that new goal position, with a very small error value. The horizontal and vertical position and the velocity can be observed from the Figures 6.27 and 6.28. The change of energy and the behavior can also be interpreted from the corresponding plots.

6.2.3 Comparison - Different Initial Touchdown States

In the previous MONOPOD results, the initial touchdown state was selected as $\begin{pmatrix} y_{horizontal} & 3.3000\,\mathrm{m/s} & 0.8842\,\mathrm{m} & -3.3784\,\mathrm{m/s} & y_{horizontal} + \theta_{td} \end{pmatrix}$. Therefore, in order to analyze the behavior on different initial touchdown state, this values are changed and results were obtained respectively.

In the Figure 6.29, the touchdown horizontal and vertical velocities are selected as $\begin{pmatrix} 2.8000\,\mathrm{m/s} & -3.3640\,\mathrm{m/s} \end{pmatrix}$. It's also clear the algorithm successfully executed the constructed planning with respect to a different initial touchdown state.

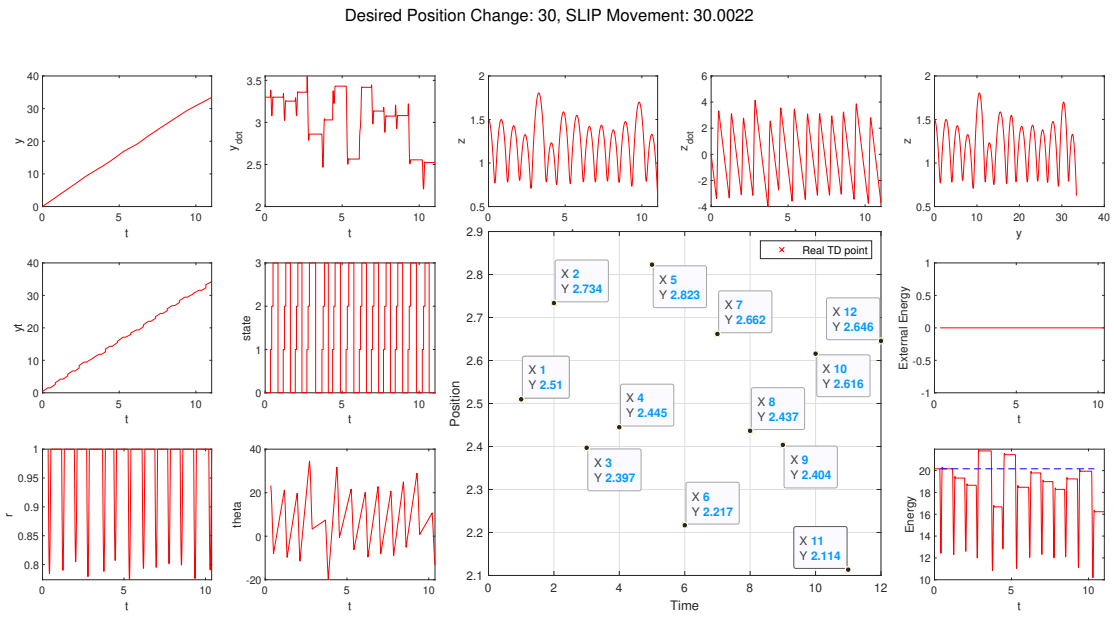

Figure 6.30: Target position = 30 m - Online - Forward Planning - Minimum Distance - $vel_{horizontal} = 3.3000\,\mathrm{m/s}$, $vel_{vertical} = -3.3784\,\mathrm{m/s}$

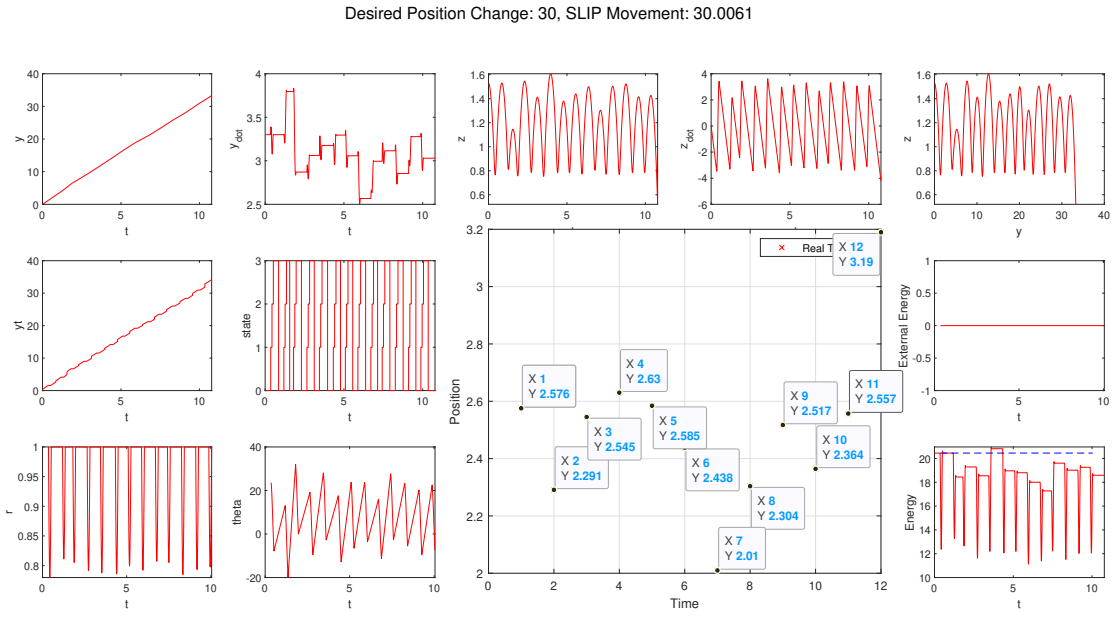

Figure 6.31: Target position = 30 m - Online - Forward Planning - Minimum Distance - $vel_{horizontal} = 3.3000\,\mathrm{m/s}$, $vel_{vertical} = -3.4690\,\mathrm{m/s}$

Similarly in Figure 6.31, changing the initial velocity vector on the opposite direction ($\begin{pmatrix} 3.3000\,\mathrm{m/s} & -3.4690\,\mathrm{m/s} \end{pmatrix}$), also resulted as a success. The planning is executed and the concluding error occurred as infinitesimal.

Chapter 7

Conclusion and Future Work

This thesis presented a new approach for the footstep planning on Spring Loaded Inverted Pendulum(SLIP) and Torque actuated Dissipative SLIP (TD-SLIP) model. The results consisted of online and offline as the construction type of the planning, forward and backwards as the algorithm type, and based on minimum distance and based on minimum step count as the policy type. The combination of these options provided wider range of results. Therefore, implementing the algorithm for both SLIP and TD-SLIP model provided much informative derivations.

The results Chapter starts with the waypoint implementation on both models. The algorithm options were provided as online, forward, and minimum distance, among all of the figures in that section. In Figure 6.1, the simulated models positional results were illustrated, according to a provided array of horizontal positions. The model successfully reach all of the provided array distance with less than 1% error (The specified error is actually smaller than that, but in order to acquire more realistic results, the error indication value is selected as 1%). Additionally, for the Figure 6.2, another randomly selected distance array was provided. Similarly with the first one, the algorithm successfully reached all of the waypoints with an error value smaller than 1%. Finally, the waypoint count has increased to 5, and the final waypoint distance has doubled. Not surprisingly,

the algorithm achieved the reach all of the positions with a very small deviation rate.

In addition to this, comparison between the online and offline execution type has a similar distance error, but huge time characteristics. The figures provided in Section 6.1.2 and 6.2.1 illustrate the comparison results. Among all of the results, the waypoint count is selected as 1. This implies that, the analysis of each distinct leg position can be achieved in a better way. According to the randomly selected distance positions, both of the methods succeeded to reach their goal distances. However, as all of the paths were constructed before the simulation for the offline case, the amount of time it takes to find the best path is enormously smaller than the online one. Constructing the paths before the simulation seems to be an important advantage, but unfortunately it has a fundamental problem, which is the size of the constructed mat file. Although, the size differs according to the recursion level (step count) in the Equation 5.10, for a 5 step case, the size of the mat file was 12GB. Therefore, generation the mat file requires a high amount of RAM, and the loading part takes few minutes.

Comparison between the forward and backwards algorithm type is also implemented for SLIP and TD-SLIP models. Other options were selected as online and minimum distance. Forward construction and backwards construction can be considered as very close to each other. According to the obtained results, they can both reach the goal distance, (the error rates of forward planning is a little bit better), and their time constraints are also very similar (forward is approximately 1.3 times faster). Additionally, selecting the policy type as based on minimum step count contributes similar results, when it compares it with the minimum distance. If energy is a constraint for the system, then outputting according to minimum step count becomes a very logical option.

Starting with different initial touchdown states are another important aspect of the obtained results. Previous results were consisted of the same initial touchdown states, so that the comparisons become much more informative. The different initial touchdown states in Sections 6.1.5 and 6.2.3 are selected from the constructed cloud domains. Same algorithm options are used and the same target positions

are selected, for all cases. In the light of these, the results contained very small error values, and demonstrated that the algorithm also provides satisfactory and acceptable results, regardless from the initial state.

Possible future extensions of the work done can start with increasing the cloud domain size. Among this thesis, the sample count of certain number is used. Investigating the results of increasing this count, by providing and checking identical test cases, would be an efficient analysis. After that, analysing the effect of enlarging the safe guard and outer cloud goal domain regions would also contribute the work to a different level. Additionally, changing the simulation environment from MATLAB, which is a JAVA based application to another C or C++ based application should decrease the time constraint in a better way. Also, the ground was selected as a flat surface across among of the simulation results. Changing the ground type from flat to another ones (most probably rough terrains) affect on the results is also seemed like a great extension. Last but not least, for the offline phase, the gathering all possible paths procedure takes an enormous amount of space on PC. Therefore, trying to find ways to decrease the size would also be a solid development so that greater step levels can be achieved on smaller disk, or may even be faster. Finally, another extension of the work would be trying everything on a 3D-SLIP environment. As the algorithm works on a 2D one, some of the real life scenarios are neglected. Including these to this work can be a satisfactory and immense effort.

Bibliography

[1] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.

[2] R. Blickhan and R. Full, "Similarity in multilegged locomotion: bouncing like a monopode," *Journal of Comparative Physiology A*, vol. 173, no. 5, pp. 509–517, 1993.

[3] C. T. Farley, J. Glasheen, and T. A. McMahon, "Running springs: speed and animal size," *Journal of experimental Biology*, vol. 185, no. 1, pp. 71–86, 1993.

[4] R. M. N. Alexander, "Principles of animal locomotion," *Princeton University Press, NJ*, 2006.

[5] P. Gregorio, M. Ahmadi, and M. Buehler, "Design, control, and energetics of an electrically actuated legged robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 4, pp. 626–634, 1997.

[6] M. H. Raibert, "Legged robots," *Communications of the ACM*, vol. 29, no. 6, pp. 499–514, 1986.

[7] U. Saranli, M. Buehler, and D. E. Koditschek, "Rhex: A simple and highly mobile hexapod robot," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 616–631, 2001.

[8] R. Alexander *et al.*, "Three uses for springs in legged locomotion," *International Journal of Robotics Research*, vol. 9, no. 2, pp. 53–61, 1990.

[9] W. J. Schwind, *Spring loaded inverted pendulum running: A plant model.* PhD thesis, University of Michigan, 1998.

[10] M. Srinivasan and P. Holmes, "How well can spring-mass-like telescoping leg models fit multi-pedal sagittal-plane locomotion data?," *Journal of theoretical biology*, vol. 255, no. 1, pp. 1–7, 2008.

[11] B. Brown and G. Zeglin, "The bow leg hopping robot," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, vol. 1, pp. 781–786, IEEE, 1998.

[12] M. M. Ankarali and U. Saranli, "Stride-to-stride energy regulation for robust self-stability of a torque-actuated dissipative spring-mass hopper," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 20, no. 3, p. 033121, 2010.

[13] U. Saranlı, Ö. Arslan, M. M. Ankaralı, and Ö. Morgül, "Approximate analytic solutions to non-symmetric stance trajectories of the passive spring-loaded inverted pendulum with damping," *Nonlinear Dynamics*, vol. 62, no. 4, pp. 729–742, 2010.

[14] H. Fang, C. Wang, S. Li, K. Wang, and J. Xu, "A comprehensive study on the locomotion characteristics of a metameric earthworm-like robot," *Multibody System Dynamics*, vol. 35, no. 2, pp. 153–177, 2015.

[15] H. Yu, M. Li, P. Wang, and H. Cai, "Approximate perturbation stance map of the slip runner and application to locomotion control," *Journal of Bionic Engineering*, vol. 9, no. 4, pp. 411–422, 2012.

[16] I. Uyanik, "Identification of legged locomotion via model-based and data-driven approaches," *arXiv preprint arXiv:1710.04275*, 2017.

[17] W. C. Martin, A. Wu, and H. Geyer, "Experimental evaluation of deadbeat running on the atrias biped," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1085–1092, 2017.

[18] A. Wu, *The theory, implementation, and evaluation of spring mass running on ATRIAS, a bipedal robot.* PhD thesis, Carnegie Mellon University, 2017.

[19] İ. Uyanık, U. Saranlı, M. M. Ankaralı, N. J. Cowan, and Ö. Morgül, "Frequency-domain subspace identification of linear time-periodic (ltp) systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 6, pp. 2529–2536, 2018.

[20] İ. Uyanık, U. Saranlı, Ö. Morgül, and M. M. Ankaralı, "Parametric identification of hybrid linear-time-periodic systems," *IFAC-PapersOnLine*, vol. 49, no. 9, pp. 7–12, 2016.

[21] I. Uyanık, M. M. Ankaralı, N. J. Cowan, Ö. Morgül, and U. Saranlı, "Toward data-driven models of legged locomotion using harmonic transfer functions," in *2015 International Conference on Advanced Robotics (ICAR)*, pp. 357–362, IEEE, 2015.

[22] I. Uyanık, M. M. Ankaralı, N. J. Cowan, U. Saranlı, and Ö. Morgül, "Identification of a vertical hopping robot model via harmonic transfer functions," *Transactions of the Institute of Measurement and Control*, vol. 38, no. 5, pp. 501–511, 2016.

[23] S. A. Burden and S. S. Sastry, "Reduction and identification for hybrid dynamical models of terrestrial locomotion," in *Micro-and Nanotechnology Sensors, Systems, and Applications V*, vol. 8725, p. 87251B, International Society for Optics and Photonics, 2013.

[24] D. Logan, T. Kiemel, and J. J. Jeka, "Using a system identification approach to investigate subtask control during human locomotion," *Frontiers in computational neuroscience*, vol. 10, p. 146, 2017.

[25] C. M. Pinto, "Stability of quadruped robots' trajectories subjected to discrete perturbations," *Nonlinear Dynamics*, vol. 70, no. 3, pp. 2089–2094, 2012.

[26] M. Golubitsky, I. Stewart, P.-L. Buono, and J. Collins, "Symmetry in locomotor central pattern generators and animal gaits," *Nature*, vol. 401, no. 6754, pp. 693–695, 1999.

[27] J. J. Collins and I. N. Stewart, "Coupled nonlinear oscillators and the symmetries of animal gaits," *Journal of Nonlinear Science*, vol. 3, no. 1, pp. 349–392, 1993.

[28] A. Sato and M. Buehler, "A planar hopping robot with one actuator: design, simulation, and experimental results," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 4, pp. 3540–3545, IEEE, 2004.

[29] İ. Uyanık, Ö. Morgül, and U. Saranli, "Experimental validation of a feedforward predictor for the spring-loaded inverted pendulum template," *IEEE Transactions on robotics*, vol. 31, no. 1, pp. 208–216, 2015.

[30] R. Altendorfer, U. Saranli, H. Komsuoglu, D. Koditschek, H. B. Brown, M. Buehler, N. Moore, D. McMordie, and R. Full, "Evidence for spring loaded inverted pendulum running in a hexapod robot," in *Experimental Robotics VII*, pp. 291–302, Springer, 2001.

[31] I. Poulakakis and J. W. Grizzle, "The spring loaded inverted pendulum as the hybrid zero dynamics of an asymmetric hopper," *IEEE Transactions on Automatic Control*, vol. 54, no. 8, pp. 1779–1793, 2009.

[32] G. Piovan and K. Byl, "Enforced symmetry of the stance phase for the spring-loaded inverted pendulum," in *2012 IEEE International Conference on Robotics and Automation*, pp. 1908–1914, IEEE, 2012.

[33] G. Secer and U. Saranli, "Control of hopping through active virtual tuning of leg damping for serially actuated legged robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4556–4561, IEEE, 2014.

[34] J. Schmitt and J. Clark, "Modeling posture-dependent leg actuation in sagittal plane locomotion," *Bioinspiration & biomimetics*, vol. 4, no. 4, p. 046005, 2009.

[35] F. Peuker, A. Seyfarth, and S. Grimmer, "Inheritance of slip running stability to a single-legged and bipedal model with leg mass and damping," in *2012*

4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob), pp. 395–400, IEEE, 2012.

[36] Ö. Arslan and U. Saranli, "Reactive planning and control of planar spring–mass running on rough terrain," *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 567–579, 2011.

[37] Ö. Arslan, U. Saranli, and Ö. Morgül, "Reactive footstep planning for a planar spring mass hopper," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 160–166, IEEE, 2009.

[38] D. E. Koditschek and M. Buehler, "Analysis of a simplified hopping robot," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 587–605, 1991.

[39] H. Geyer, A. Seyfarth, and R. Blickhan, "Spring-mass running: simple approximate solution and application to gait stability," *Journal of theoretical biology*, vol. 232, no. 3, pp. 315–328, 2005.

[40] W. J. Schwind and D. E. Koditschek, "Approximating the stance map of a 2-dof monoped runner," *Journal of Nonlinear Science*, vol. 10, no. 5, pp. 533–568, 2000.

[41] J. D'Errico, "fminsearchbnd, fminsearchcon." `https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd-fminsearchcon`, Sept. 2020. MATLAB File Exchange.

[42] H. Hamzacebi, *Analysis and control of periodic gaits in legged robots*. PhD thesis, Bilkent University, 2017.

[43] Ö. Arslan, "Model based methods for the control and planning of running robots," Master's thesis, Bilkent University, 2009.

[44] The Mathworks, Inc., Natick, Massachusetts, *MATLAB version 9.8.0.1396136 (R2020a) Update 3*, 2020.