Universität des Saarlandes
Fakultät für Mathematik und Informatik
Fachrichtung Informatik

Masterarbeit

Natural and Accurate Stair Walking Synthesis for Character Control using Environment Information

vorgelegt von

Rolando Morales Suárez

am Oktober 2021

Begutachtet von:

Prof. Dr.-Ing. Philipp Slusallek

Prof. Dr. Christian Theobalt

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Enverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Stair walking is a very basic human locomotion that people perform everyday. However, trying to emulate this is not a trivial task, as it is often seen in video games. There, the stair walking action is not correctly addressed offering an unnatural looking locomotion, especially when a user needs to control the movements of a character responsively. Since this issue still poses a challenge for researchers, this thesis explores whether a supervised deep learning motion synthesis model can perform such task properly.

To address the task, a time-series supervised deep learning model that takes into account the feet as well as environment information is developed. This model estimates the pose of the character along with the foot step goals and the feet trajectories for the next frame. These results are corrected and then fed back as inputs for the next iteration. Also, in order to train the model, a dataset was recorded using a motion capture suite and later on processed by adding systematic variation.

There are two proposed correction stages before feeding back the estimated outputs related to the feet: a foot step goal correction and a foot trajectory correction. The first correction places the foot step goals in the center of a tread so that the character tries to center the feet on each step. The second correction changes the estimated trajectory of the feet depending on the corrected foot step goals. The evaluation shows that such changes can improve the performance on the problem in comparison to a baseline model. Also, the proposed changes in the virtual character still keep a natural looking synthesized body motion.

Furthermore, since the stair walking task usually requires to have some knowledge about the geometry of the stairs in order to correctly place the feet, a heightmap sensor is tested and compared against volumetric sensors. For the stair walking task, a heightmap sensor offers memory advantages for the training data generation as well as for the trained model over the other sensors. Also, it boosts the performance on the metrics.

Contents

Chapter 1
Introduction

1.1 Motivation

Humans are very good at learning tasks, and once a task is learned, it can be performed unconsciously. That is the case for stair walking, which is a basic locomotion for humans that is performed sometimes in a daily basis and it happens mechanically (it is performed without thinking about it). But these sort of tasks, which are commonly trivial in the real world, are hard to transfer into virtual applications because computers do not think and process information as humans do.

One of the main reasons why someone would like to transfer the stair walking locomotion into a virtual world are video games. The video games industry generates billions of dollars in revenue ($155 billion U.S. dollars in 2020 [5]) and is a part of the life of millions of people around the globe. Also, according to analysts, this revenue is predicted to grow in the upcoming years reaching $260 billion U.S. dollars in 2025 [47].

When it comes to video games, many people want to have the most entertaining and realistic experience possible. There are different aspects in a video game that make it look real, among others: graphics, gameplay, sound and animation. Each of those aspects might be independent, but when it comes to realism, all of them add a very important part to the final result, e.g. a video game with really good graphics but a very poor character animation is likely to discourage the players to continue with the game which could be disastrous for the game producer.

As it was mentioned, animation is an important aspect of the whole gaming experience, and it has gained much importance in the last decades that even very well known companies like Ubisoft and EA have research departments that have a focus in motion synthesis, making contributions in the area [45, 46, 24, 25].

(a) (b)

Figure 1.1: Character walking through stairs in the video game *Grand Theft Auto V*. Both figures show examples of wrong foot placement. Images taken from [19].

In video games, it is very common to use humanoid characters, and usually such characters need to traverse the virtual world terrain. In urban environments stairs are a very important aspect of this terrain, thus the stair walking locomotion needs to be addressed to keep the realism of the video game. To solve this problem in the early years of 3D video games, stair walking as such was not even really considered. Instead, hand-made pre-computed walking animation was played while the character was walking through stairs, and the character was moved vertically on a slope. Since this kind of solution does not care about the shape of the stairs, it usually offers a non realistic final result. One common artifact that this solution presents is that it is possible to notice how the character shifts up and down and also sinks within the stairs. This can be further improved by using inverse kinematics (IK), but this requires more resources for processing.

Later on, video games started to use motion capture instead of the hand-made movements for a more realistic character animation yielding an improved quality. For example, in the very well known video game *Grand Theft Auto V* (GTA V) motion capture was used to record the animation for the characters, which conveys realism to the spectators. Nevertheless, when performing stair walking, some artifacts can be seen, like sinking into the stairs or colliding with the treads as it can be seen in figures 1.1a and 1.1b. Also, in the full animation, it can be noticed that the character shifts over the treads of the stairs after performing a step. Those artifacts are very likely to happen if there is a limited number of stair walking recordings. On the other hand, having many stair walking recordings to cover a large amount of different staircase sizes would be memory inefficient. Thus, there is always a trade-off between memory and possible motions available. So, even though stair walking is an easy task for humans, it still poses a challenge to transfer this type of locomotion properly to a virtual character.

In order to immerse the audience into the virtual world and not to break the illusion, virtual characters that are ascending or descending stairs need to look as real as possible. A very important part to preserve such realism is performing an action naturally, or more

precisely, to have a correct foot placement of the character while keeping the overall locomotion of the body smooth and steady. Here, the game producers need to balance between one or the other, and as it was previously mentioned for GTA V, smooth body locomotion is preferred over natural foot placement. For example, motion capture recordings are combined with other tricks to perform the stair walking action, but it produces the sink and collision artifacts. Also, stair walking needs to be synthesized immediately depending on the user's input, otherwise it would disrupt the gameplay: it would not be pleasant to provide a command with a gamepad and wait a longer time for the response. Also, the synthesized stair walking locomotion should adapt to the current character's environment, or in other words, it should be robust to traverse through different stair sizes while keeping a natural looking animation. The ideal scenario would be if all of the aforementioned features are performed by using a non memory consuming method. In other words, instead of using many recordings to perform the action for different stair sizes, use one method that covers them all.

Currently, state-of-the-art algorithms [6, 25, 46] for character control with environment interaction do not offer a correct solution for the stair walking synthesis. Usually, the algorithms either synthesize a good overall body locomotion with wrong foot placement or they offer a good foot placement with a unnatural body locomotion. For example, the reinforcement learning method of Bergamin et al. [6] is really good at adapting to the character's environment which lets them place their feet correctly onto the ground, but the character is also trying to balance himself, thus yielding a non natural looking body pose (the character shakes a lot and does not look steady). Another example is the method of Holden et al. [25] that is very reactive to the input of the user, but it is not really aware of the geometry of stairs, yielding a non natural looking foot placement animation.

There is also the possibility to synthesize animation by using footstep goals, just as it was done by Peng et al. [39]. The problem with that approach is that it does not look natural and the character does not offers good responsiveness.

1.2 Objectives

The **primary objective** of this thesis is to develop a motion synthesis algorithm for character control that shows a natural looking locomotion for stair walking, i.e. that the virtual character should correctly place its feet onto the treads of a staircase while keeping a natural body pose. After some literature review, which is presented in chapter 2, some ideas were gathered in order to achieve the main objective: the use of a supervised learning motion synthesis model along with foot placement goal and foot trajectories.

A supervised learning motion synthesis model is considered because it has been shown that this approach offers a good motion quality together with nice character responsiveness as well as compact memory requirements (trained models of tens or hundreds of megabytes can learn multiple locomotions) [26, 56, 44]. Also, the training time of supervised methods is usually shorter than the time it takes to train other approaches, like reinforcement learning. For example, it may take around 30 hours to have a decent result in a reinforcement learning method [6], whereas a supervised method may take around 20 hours [44].

The idea of using a foot placement goal can be traced back to different works published before neural networks were widely used for motion synthesis [4, 51]. Thus, coupling a supervised learning model with foot step goal seems to be helpful. There is already a reinforcement learning publication that used foot placement goals as part of the walking synthesis showing that even though the bipedal character could not follow the goals precisely (some times the goals were even impossible to reach) they helped with the overall walking locomotion [39]. In this thesis, the idea is to let the supervised model estimate the foot placement goals and afterwards correct such goals so that they match the desired positions. The desired positions are going to be regarded as placing a foot completely over a tread of a staircase, as it is commonly done by people. This foot placement goal correction is performed because the supervised model often places the goals in positions that might lead to inaccuracies (e.g. near the edge of a tread). The proposed foot placement goal correction is simple, just translate the estimated position to the center of the tread that the supervised model is already targeting. The corrected footstep goal is then used as input for the next iteration.

The idea of the foot trajectory was based on the root trajectory estimation and correction of the neural state machine (NSM) supervised learning model [44]. The results of the NSM showed that it can predict the future root trajectory of a character and correct it using the user's input. The corrected root trajectory result is then fed back as input for the next iteration. In this thesis, the idea is to let the supervised model estimate the future foot trajectories that the character should follow. Once the future foot trajectory is estimated, it is corrected by using the desired footstep goal positions. The proposed method to correct the future foot trajectories computes the vector between the desired footstep goal and the closest point in the trajectory. This vector is applied along with a weighting function to all points in the future estimated foot trajectory. The corrected foot trajectory is used as input for the next iteration.

Also, part of the main objective is to find out how well the proposed method in this thesis performs in comparison with other alternatives. Thus, this thesis contains an assessment of the differences in performance between the proposed method, the NSM and the PFNN. Furthermore, in this thesis an ablation study was conducted, analyzing the influence of the footstep goal correction and foot trajectory correction procedures.

Additionally, there is a **secondary objective** in this thesis, namely the evaluation of a proper sensor for stair walking. This is something important because a motion model needs to be aware of the surroundings of the character so that it can predict more accurately the next pose. There are different kinds of possible sensors that can be used, but in this thesis the use of a heightmap sensor is suggested due to the fact that it offers a good description of the ground (as it was shown in [39]) without the need of many samples. The so-called volumetric sensors are not really the best option for stair walking because they waste many unnecessary samples by sensing above the stairs, which is useless for this (for stair walking, a description of the to surface of the ground is what matters when it comes to foot placement, not what lies above it). A heightmap sensor yields a lighter training data and a more compact trained model in comparison with the volumetric sensors of the NSM.

1.3 Outline

The remainder of this thesis is structured as follows. Chapter two provides some background on stairs and human walking synthesis. It also presents some related work leading to the approach, which is presented in chapter three. Chapter four explains the procedure for the data capturing and processing for stair walking. Chapter five has a evaluation of the proposed algorithm compared against a baseline method. It also contains an ablation study of the different components. Chapter six concludes with the observations and limitations of the proposed algorithm. It also presents some interesting directions for future work.

Chapter 2
Background

2.1 General Information about Stairs

Since this thesis deals with stair walking motion synthesis, it is useful to know some information related to staircases. The most important vocabulary words that are needed are the *tread*, which is the flat surface where a person places his feet, and the *riser*, which is the height between consecutive treads. These parts can be seen in Figure 2.1.

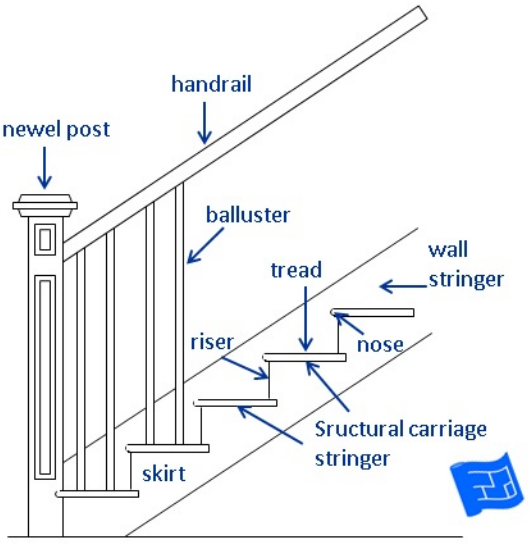


Figure 2.1: Parts of a staircase [15].

When building a staircase, there is a rule of thumb for its dimensions [16], namely the formula

$$S = t + 2r,$$

where S is the step length, t is the tread depth and r is the riser height. Some common values for tread and riser vary from 26 to 32 cm and 14 to 19 cm respectively. t = 29cm and r = 17cm are considered ideal. When it comes to step length, values from 59cm and 65cm are decisive [16].

Usually, many papers [51, 11, 26, 6, 25] develop methods to synthesize general walking and showcase that the published method is also applicable for stair walking. Thus, in the next section there is a compilation of some of the most relevant works related to general human walking synthesis.

2.2 Human Walking Synthesis

Human walking synthesis refers to the generation of human walking locomotion and has been studied for decades in computer graphics. Human walking is defined as a cycle composed of phases that contain foot strikes (foot touching the ground) and takeoffs (foot that left the ground). The phase in which the foot is in contact with the ground is also called *stance phase* and the phase when the foot is on the air is also referred as *swing phase* [36]. The different aforementioned phases of walking motion are shown in figure 2.2. Stair walking is fundamentally the same as general human walking (the main difference is that instead of walking on a planar surface, a character has to walk on stairs).

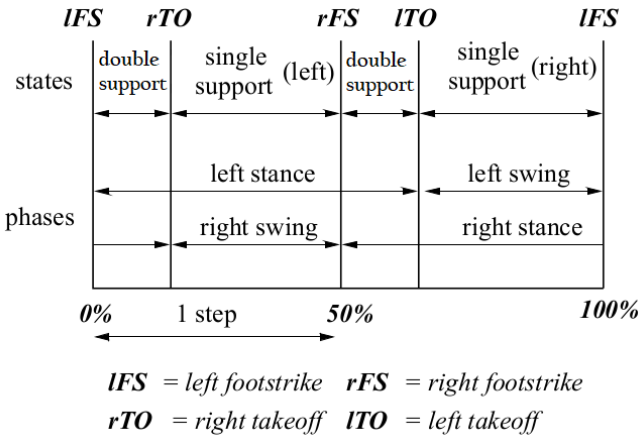


Figure 2.2: Phases of a walking motion [36].

As it was already mentioned, the graphics community usually considers stair walking as a sub-problem of the general human walking synthesis, thus the published works commonly report models that solve the walking animation and, if possible, also deal with stair walking. In the next sections, a general overview of the *classical* and *neural network* methods for human walking synthesis is presented.

2.2.1 Classical Methods

Traditionally, the locomotion animation was done by using a set of so called *keyframes*, which specify the positions of the joints of the virtual character. The joint positions for the in-between frames were commonly generated via interpolation. Thus, in order to have a realistic walking animation with the traditional animation procedure, a good choice of keyframes was needed (key frames could also be generated by a motion model [36]). In order to improve the quality of animation, new methods were developed, which are going to be referred in this thesis as classical methods (methods that do not use deep learning). There are three main categories of classical motion synthesis techniques when it comes to human walking locomotion, namely procedural methods, physics-based methods and example-based methods [49]. Also, to get the best of different worlds, hybrid methods were developed as well. Each one of them has advantages and disadvantages and is utilized depending on the end-use of the motion model. These categories are described next.

Procedural Methods

Procedural methods generate locomotion by using a set of equations or constraints based on biomechanical or empirical concepts [49]. One example is the model of Boulic et al. [7], where forward kinematics are used to generate keyframes, and after a prediction is done, the result is corrected using inverse kinematics to ensure a more realistic contact with the ground.

Van de Panne [51] presented a procedural solution that creates a center of mass (COM) trajectory given a footstep plan and viceversa. This trajectory is optimized using penalization terms that take into account Newton's second law. After the trajectory optimization is finished, the legs are later corrected using inverse kinematics. This method can compute a global optimized trajectory offline but it also has the possibility to interactively create a footstep plan on the go. However, this interactive option is applied to a planar surface, and more investigation is needed to see if it is also applicable for stair walking. Nevertheless, Van de Panne showed that by using just two footstep goals it is possible to generate a natural looking locomotion.

Chung and Hahn [11] proposed a hierarchical control system that uses the support foot as kinematic constraint for the pelvis trajectory optimization. The swing foot is later defined according to the generated trajectory using collision constraints to account for uneven terrain. This is also a global method because it needs a given trajectory to perform the pelvis trajectory optimization.

The main advantage of the procedural methods is the low computational cost. The most costly part would be the inverse kinematics solution, but since it is usually done just for the legs, it is almost negligible. The main disadvantage comes from the interpolation, where the in-between frames show non realistic movements due to the filtering of intrinsic motion dynamics [36].

Physics-Based Methods

Physics-based methods take into account the *forces* to generate the desired locomotion. One example is the work of Badler and Ko [2], where they extend a kinematic model by using dynamics as a constraint to correct the generated motion. Also, Coros et al. [12] presented a strategy that tries to follow footsteps according to simulated motion dynamics.

Other physics-based models use *PD-controllers* to generate the forces and torques that are then applied to the joints of the virtual character for a desired motion [36]. A common PD-controller is an equation of the form

$$f = -k_p(p - p_d) - k_v(\dot{p} - \dot{p_d}),$$

where $(p, \dot{p})$ is the current state (position and velocity respectively), $(p_d, \dot{p_d})$ is the desired state, k_p is the proportional gain and k_v is the derivative gain. Such controllers can be added to joints in order to generate the forces to move them to the next target joint positions, as it is done by Van de Panne et al. [**?**].

Since these methods take forces into account, the generated locomotion might useful for cases when a virtual character needs to traverse through uneven terrain. The main disadvantage of these kind of methods is the high computational cost. Another problem is that in order to have an accurate and good looking result, a good model of the system is needed because such model needs to realistically emulate the movements of the virtual character. Also, in the case of controllers, it is needed to tune the parameters to have a nice output, which is often problematic [36].

Reinforcement learning is an area of machine learning that has been applied along with physics-based methods. The foundations of reinforcement learning methods are the Markov decision processes (MDPs), which are models of the form (S, A, R, T), where S is the set of possible states, A is the set of possible actions, R is a real valued reward function that depends on the current $(State, Action)$ pair and T is the description of the effects of an action on each state. Such models have the Markov property, namely the effects of an action only depend on the current state, not on the past ones. For some kind of problems, the main goal is to learn a policy π, which is a mapping from S to A, or in other words, the policy decides which action to take depending on the current state [20].

Reinforcement learning methods were introduced in computer graphics by Lee et al. [31] as a precomputation of the state-space search on the motion graph and posterior tabulation of control policies for interactive character animation. Treuille et al. [48] later showed an enhanced method by using linear function approximators to encode the policies for an improved memory efficiency.

Reinforcement learning methods can have a nice interactivity with the environment, but designing reward functions for these methods is usually a non trivial task [44].

Example-based Methods

Example-based methods use previously recorded data from the real world in order to synthesize new data. There are different motion capture techniques, but for this thesis a XSense intertia MVN Awinda motion capture suit. The suit contains inertial sensors should be located in specific parts of the body of a person to record joint information. Once the data is recorded, it is possible to work on the motion synthesis. There are two main approaches: motion warping and motion blending [36].

Motion warping techniques create new motion by modifying keyframes or motion capture trajectories [36]. One example of this kind of technique is the work of Witkin and Popovic [53], in whose work a reference motion trajectory is tuned by scaling and shifting selected keyframes from it.

For a better foot placement, van Basten et al. [49] proposed the step space, which is a data structure of the parameterized stepping animations extracted from motion clips. For the step synthesis, a search through the data structure is performed in order to select the best step parameters according to a distance metric. These parameters are then warped in order to reach the desired target. The follow-up work of Egges and van Basten [13] extended the step space with footstep planning given a desired trajectory. The motion warping methods using the step-space only showed applications for planar locomotion and, as stated by the authors, more investigation is needed to see if the method also works for stairs. Also, it is hard to guarantee a realistic motion by using motion warping in general, just as it is with procedural methods [36].

Motion blending techniques, on the other hand, use a database of recorded motions and interpolate between their parameters to synthesize new motion [36]. A very popular example are *motion graphs*. This technique consists on playing a recorded motion clip from a motion capture database and depending on the user input it switches to another clip that has a close similarity according to some distance metric. For the switching, transition points are needed and these are created according to some transition scheme [30]. Later on, Heck and Gleicher [22] presented the *parametric motion graphs*, which instead of switching between clips of motion, they switch between motion spaces, which are subsets of motion clips with a similar action (e.g. walking, running, etc.). The problem with motion graph methods is that in order to have a rich repertoire of movements there needs to be a huge motion database which is not memory efficient.

Van Basten et al. [50] also experimented with a combination of the step space and motion blending, where instead of using warping for synthesizing motion, candidates were chosen an then blended together for a more accurate result. The problem with this work is that it only showed applications for planar locomotion and more investigation was needed to see if the method would work on uneven terrain (like stairs).

Beacco et al. [4] also used parametric spaces of stepping animations extracted from motion clips. These spaces are queried online and depending on the current state of the character the three best candidates are chosen and blended together. The weights for the blending are given by barycentric coordinates of the target within the triangle of candidates. The result is also corrected using IK to account for uneven terrain. In figure 2.3, the framework of Beacco et al. can be seen. Nowdays, most of the parts used

in this framework are replaced by neural networks in newer locomotion models (e.g. the motion database is learned and a new character pose is given by a neural network), however this method can serve as a guideline to choose correct inputs for a deep learning method as it is going to be explained in chapter 3. This method is a good compromise between character control, accurate foot stepping and natural locomotion.

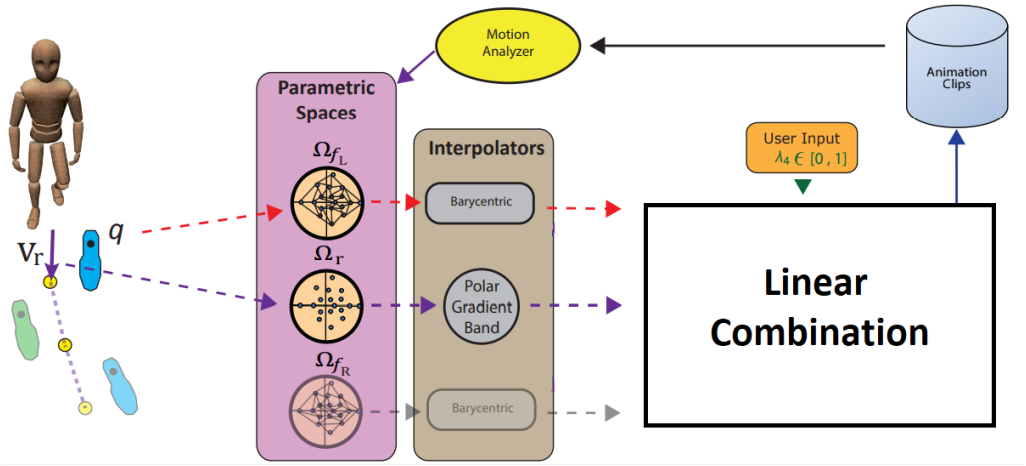


Figure 2.3: Footstep parameterized motion blending framework. Figure from [4].

Hybrid methods

Since every method has advantages over one another, hybrid methods combine different solutions to take the best of different worlds. Hybrid methods use motion capture data to have the natural looking locomotion of real subjects and further process it with other of the aforementioned methods.

Motion fields is a technique proposed by Lee et al. [33] that represents motion capture data as a motion space and uses it to query next frame candidates as actions via k Nearest Neighbors (kNN). The current state of the character and the actions are then used with RL to choose the next best character pose. The kNN query was very costly, thus motion matching was proposed by Büttner and Clavet [9], which is regarded as a greedy approximation of motion fields. Instead of using RL, they choose the next candidate with a lookup algorithm. Although motion matching is used in production for some video games, its core limitation is the memory usage [25].

2.2.2 Deep Learning Methods

Before diving into the deep learning motion synthesis methods, a brief introduction to deep learning is presented to the reader.

Deep Feedforward Networks

In the last decade, deep neural networks (DNN) have been widely used in many different fields, and motion synthesis is no exception.

Deep feedforward networks, also known as feedforward networks (FFN) or multilayer perceptrons (MLP's), are function approximators, i.e. if there is a function of the form $y = f(x; \theta)$, they find the parameters θ such that it approximates y [21].

In these kind of networks, the flow of information goes from the input x through several in-between operations up to the output y without any sort of feedback information or loop [21].

These networks are typically composed of functions in a chain, i.e. a function could be composed of several other functions, e.g. $f(x) = f^{(2)}(f^{(1)}(x))$. The component functions of this chain are called **layers**. The number of layers gives the **depth** of the network whereas the size of a layer gives the **width**. In figure 2.4 there is an example of a basic DNN[21].

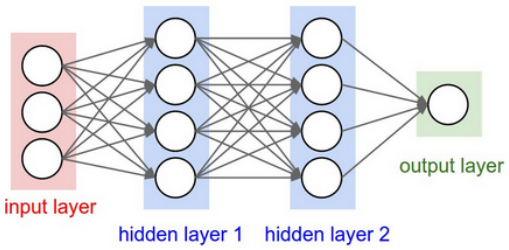


Figure 2.4: A simple Feedforward Network [34].

The simplest **unit** of a network's layer is a **neuron**, and as the name suggests, it is named after the resemblance of the human brain's cells (figure 2.5 shows the similarity between them). A network's neuron is basically a mathematical function that consists of several weights and a bias that are applied to its inputs, i.e. a dot product of the form $\sum_i w_i x_i + b$. Also, it contains a non linear function, or activation function (e.g. sigmoid function σ) that serves as a trigger to that specific neuron [34].

After all neurons of a layer apply its mathematical function to their inputs, their results are passed into the next layer. This process repeats until the last layer, or **output layer** (also known as **feedforward computation**)[34].

In order to evaluate the predictions of a network, a loss function is applied at the end (e.g. $L2$ norm). Afterwards, the optimization part begins and it is known as **backpropagation**

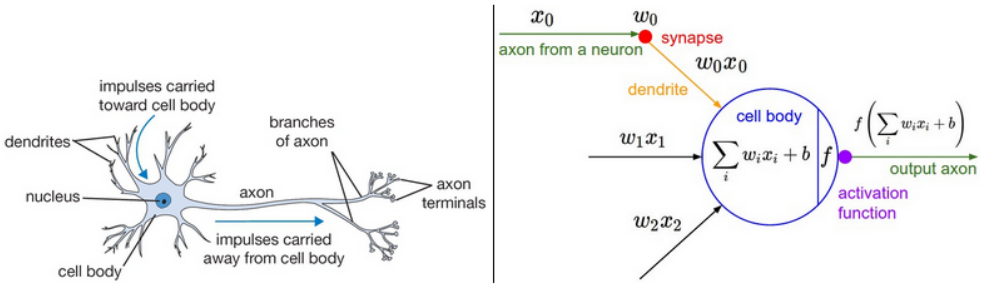


Figure 2.5: Left: biological neuron. Right: mathematical model of the neuron [34].

[21]. During backpropagation, the network tries to optimize the weights via a gradient-based algorithm. The most basic of this type of algorithms is the gradient descent method

$$w^{(k+1)} = w^{(k)} + \alpha^{(k)} \Delta w^{(k)},$$

where $\alpha^{(k)}$ is the learning rate and $\Delta w := -\nabla f(x)$. In other words, the parameters are updated in the negative direction of the derivative weighted by the learning rate [8].

It has been shown that updating the learning rate parameter during training improves the learning performance, yielding new algorithms such as Adam, where the learning rate changes w.r.t. some estimation of the first and second gradient's moments [29]. Further improvements are made in AdamWR, where a warm restart to reset the learning parameters using a schedule speeds up the learning even more [35].

To decrease the co-adaptation of neurons of a layer and reduce overfitting (a common problem in FFN), **dropout** is usually used. This technique drops units of a layer randomly during training which is similar as having many different smaller networks. At test phase, it is possible to have an average of such smaller networks by evaluating a network with all units active and smaller weights [43].

Deep learning is a powerful tool that has been widely used in the last decade and it has been also introduced in some of the categories of motion synthesis like physics-based, example-based as well as hybrid methods as it is presented in the next sections.

Physics-Based Methods using Deep Learning

Deep reinforcement learning (DRL) was born thanks to the advent of neural networks. The main difference here is that instead of learning the policies using tabulation or linear function approximators, as it was previously done, it is possible to learn policies by using a neural network, which is regarded as a more powerful function approximator. DeepLoco is an example of DRL methods, where a hierarchical control strategy was implemented [39]. In the DeepLoco framework, a high level controller receives the input of a terrain sensor and outputs a footstep plan for a low level controller. This other controller takes the generated footsteps to output the joint targets for a PD-controller.

DRL approaches are a promising solution for character locomotion due to their physical interactivity with the surroundings, but currently they do not offer a natural looking body locomotion. For example, the strategy used in DeepLoco can be useful to traverse some terrain and reach a goal position target, nevertheless the locomotion of the bipedal character used looks robotic, it does not resemble a human locomotion. Also, the design of reward functions for specific goals is a hard task and training times are usually long [44, 6].

Example-Based Methods using Deep Learning

Recurrent neural network (RNN) is a network architecture that is suitable to deal with sequential data [42, 52]. Fragkiadaki et al. [18] proposed an encoder-recurrent-decoder (ERD) to reproduce locomotion which consist in a RNN with long short-term memory (LSTM) [23] combined with representation learning. Lee et al. [32] showed an improved RNN model that also receives character control inputs and has some environment interactions. However, LSTM models are usually very hard to train [3] and they showed problems when it comes to user response (Character takes too long to perform an action after it was given by the user) [44].

Convolutional autoencoder [40] architectures have also been utilized by Holden et al. [27] to map high-level user instructions (e.g. a trajectory to follow) to character locomotion. Later on, Holden et al. [26] proposed the *phase-functioned neural network* (PFNN), a real-time method for character control using a phase value to blend motions in the right timing. Zhang et al. [56] published a similar approach using a mixture of experts (section 2.3), where the experts and the gate are deep feedforward networks, for controlling a quadruped character. The follow-up work, the *neural state machine*, reduced the foot skating and also added sensors that let the character interact with the environment [44]. These methods provide a very nice character control and the added sensors to the neural state machine can come handy when it comes to tasks where the character needs to plan an action ahead of time, like in the case of foot placement.

Starke et al. [46] proposed the neural animation layering, a framework for synthesizing novel character poses given some control trajectories. This is analogous to classic blending techniques, but instead of using a step space or something alike to blend candidates, it uses a motion generator network. This framework needs the control signals in order to generate the unseen motions, which is one of the objectives of the current thesis (generate natural looking locomotion for stair walking through feet trajectory estimation).

Hybrid methods using Deep Learning

To overcome the memory problems of motion matching, Holden et al. [25] presented learned motion matching (LMM), which has a compressor and decompressor neural networks to alleviate the memory requirements. This method also uses inverse kinematics to place the feet of the character over the stepping surfaces. However, it is not possible to control the foot placement, as it can be seen in figure 2.6 where LMM fails to correctly place the foot plants of the character over the treads of the staircase which yields a non-natural looking stair walking animation.

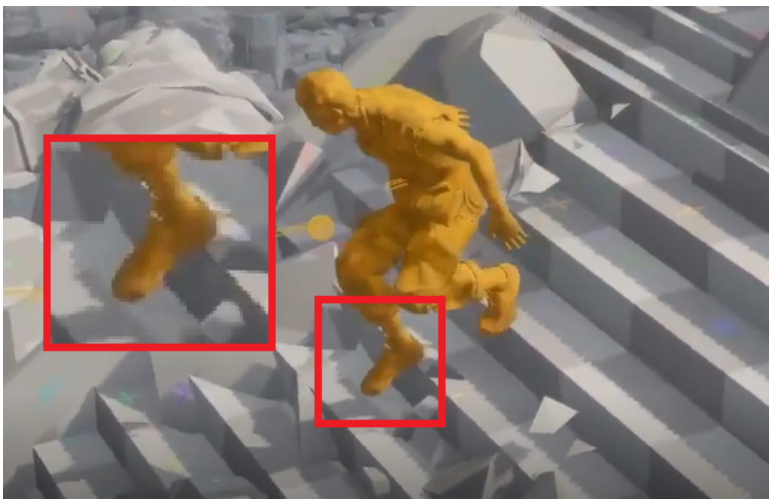


Figure 2.6: Learned motion matching fails to correctly place the feet of the character onto the treads of a staircase. Figure taken from supplementary material of [25].

Other hybrid methods use RL along with motion capture data. That is the case of DeepMimic, a RL framework that tries to imitate recorded data to perform a task and also uses PD-controllers to drive the joints of the characters [38]. Park et al. [37] proposed a method that combines a RNN and RL to learn tasks from unorganized motion data. DReCon is another method that uses RL but in combination with motion matching. This method also uses terrain sensors to plan the foot placement ahead of time [6]. The problem with these methods is usually the long training time, the not very good character controllability compared with other methods and the non natural looking locomotion while traversing some terrains (as it can be seen in the supplementary materials of [38, 37, 6]).

2.3 Mixture of Experts

Mixture of experts (MoE) is an ensemble method that follows the design paradigm of divide and conquer [28]. This method has several experts that learn a specific part of a problem and a gate that decides which parts of the expert opinion's to trust. The gate is responsible for assigning a probability to blend the outputs of the experts. MoE has been recently used in visual tasks with promising results [10, 14].

In figure 2.7, a representation of a MoE can be seen. There, it is possible to notice that the same input is given to all expert networks, as well as the gating network. Later on, the gating network outputs the probabilities for each expert and these are used to blend the final output.

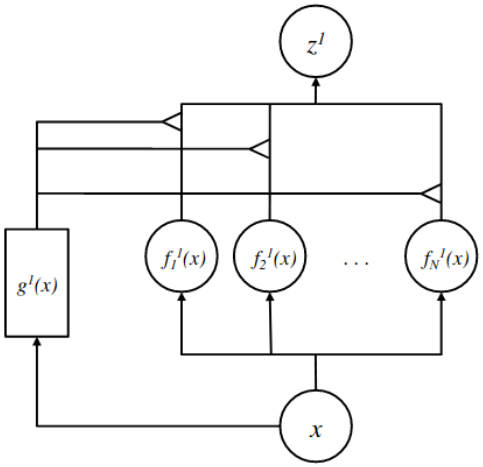


Figure 2.7: Representation of a mixture of experts. x is the input, $f_i(x)$ are the expert networks, $g(x)$ is the gating network and z is the output. Image from [14].

MoE has been used in motion synthesis for character control using FFN's as experts and gate, showing a nice character controllability [56, 44]. Apart from the controllability, these methods differ from other MoE models because, instead of blending the outputs of an ensemble, the blending occurs at feature level. This means that the gating network outputs the probabilities for the weights of an expert network, and then these weights are blended together to form a *compound network* that outputs the final result.

Chapter 3
Method

3.1 Method Overview

In this thesis, a recurrent supervised learning method for walking locomotion is proposed, to not only walk on a flat surface but to ascend and descend stairs as well. An overview of the proposed method can be seen in figure 3.1. The method consists in two networks: the motion prediction network and the gating network. The motion prediction network contains different sets of network weights (experts) and takes as input the pose of the character, the goal of the task it is performing and a description of the surrounding ground. The gating network receives as input a subset of the input of the motion prediction network along with the user input. This gating network is in charge of blending the features of the experts, so that the blended result outputs the pose of the character for the next frame. The motion prediction network estimates the future root trajectory, the footstep goals and the feet trajectories in a one second window in the future. These estimations are corrected before they are used in the next frame and also fed back as input for the next iteration. In the next sections, all parts of the method are explained in more detail.

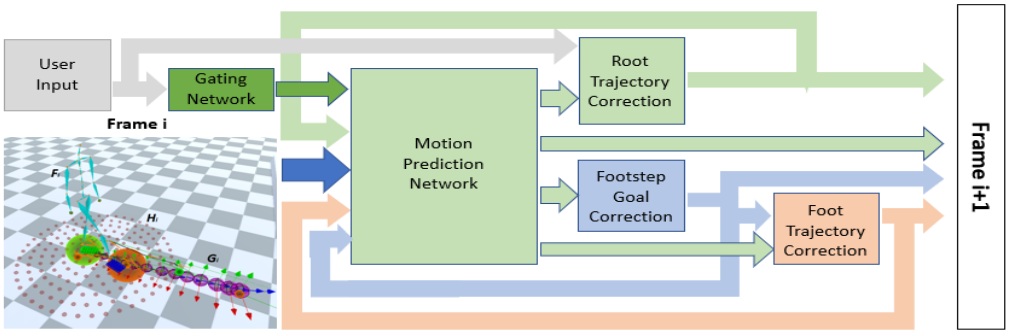


Figure 3.1: Overview of the method.

3.2 Neural State Machine

The aim of this thesis is to synthesize natural a accurate stair walking for character control, and for that a motion synthesis framework is needed. Also, since this work deals with stairs, the selected motion model needs to have an understanding of the character terrain. A good candidate for this is the *neural state machine* (NSM) [44], a supervised learning framework that offers natural looking locomotion for planar surfaces along with character-scene interactions thanks to sensors that describe the surroundings.

The NSM is a **time-series model**, meaning that it estimates the pose of the character in the current frame given the information of the past frame. In figure 3.2, a representation of such a model can be seen. The time window spanned by the NSM is two seconds (one second in the past and one in the future). The information of the past in given by the already recorded data while the data of the future is an estimation depending on the current inputs.

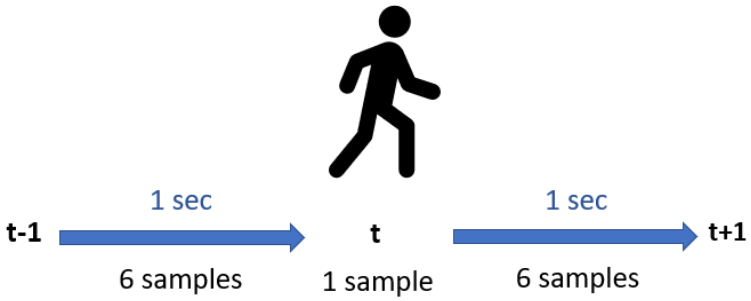


Figure 3.2: Time series model representation of the neural state machine.

As many other motion models [26, 56, 4], the NSM uses the *root* (projection of the hips onto the ground) information to control the character. Thus, in order to generate a trajectory for the character, it takes samples of the root information along the time window. In total there are 13 samples: 6 samples that span a 1 second window in the past, the current present sample and 6 samples of the future estimated 1 second time window.

3.2.1 Network Architecture

The NSM consists in two main parts, the motion prediction network and the gating network, which are described next.

Motion Prediction Network

This network is in charge of computing the pose of the character in the current frame given the past information. It receives as input three different components: frame input (F), goal input (G) and heightmap input (H). The *frame input* is composed of the character pose in the last frame (joint positions, rotations and velocities relative to the root), and the past/future root trajectory (positions and forward directions) in 13 sampled points

that span a 2 second window (-1s to 1s relative to the root coordinate of the last frame). It also contains the continuous action labels of the trajectory points as well as the feet trajectory positions in the 13 sampled points. The *goal input* is composed of the 3D root goal positions and 2D directions in the horizontal plane of all 13 sampled trajectory points relative to the root of the past frame. It also has the one-hot action labels at the goal and the footstep goals for each foot. The *heightmap input* is a representation of the surrounding floor of the character. All of these components are the inputs of an **encoder module** $H(X;\beta)$, which consists of 3 three-layer fully-connected encoder networks ($\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, respectively to the inputs), and is described as (equations have same notation as in [44]):

$$H(X;\beta) = \{\mathcal{F}(F_i;\beta^{\mathcal{F}}), \mathcal{G}(G_i;\beta^{\mathcal{G}}), \mathcal{H}(H_i;\beta^{\mathcal{H}})\}, \tag{3.1}$$

where β are the parameters of the networks. The outputs of these networks are given to the **prediction module**. This module is a mixture of experts (section 2.3), composed of 10 different sets of network weights (three-layer networks) which are in charge of becoming experts in different modes of motion. Each of these experts have the following structure:

$$\Theta(X;\alpha_i,\beta) = W_{i2}\ ELU(W_{i1}\ ELU(W_{i0}\ H(X;\beta) + b_{i0}) + b_{i1}) + b_{i2}, \tag{3.2}$$

where $\alpha_i = \{W_{i0}, W_{i1}, W_{i2}, b_{i0}, b_{i1}, b_{i2}\}$, is composed of the i-th expert's weights and biases, which are later blended to give the final network output $y = \{\alpha_1, ..., \alpha_{10}\}$ according to

$$\alpha = \sum_{i=1}^{K=10} \omega_i \alpha_i, \tag{3.3}$$

where w_i are the blending coefficients given by the gating network and K is an hyperparameter to select the number of experts (which is originally selected to be 10).

The ELU is the exponential linear unit and it is used as activation function. It is defined as

$$ELU(x) = max(x, 0) + exp(min(x, 0)) - 1. \tag{3.4}$$

Gating Network

The **gating network** is the one that outputs the blending weights that are used to combine the result of the experts of the motion prediction network. It is a fully connected network with three layers, and as it was briefly mentioned before, it outputs the weights $\omega = \{\omega_1, ..., \omega_{10}\}$ that are used to blend the expert's weights and biases. The input $\hat{X}_i$ of this network is denoted as

$$\hat{X}_i = P_i \otimes X'_i, \tag{3.5}$$

where $\otimes$ is the Kronecker product, P_i is a 2D *phase* vector denoted by

$$P_i = \{sin(p_i), cos(p_i)\}, \tag{3.6}$$

where p_i is a scalar phase value, which is a so called **latent variable**, defined as an estimation of unobservable variables that cannot be directly measured [17]. This variable has a global effect in the network inference and indicates the current phase of an action by giving it a value, e.g. in figure 3.3, an incrementing scalar value in the range of $0 \leq p_i \leq 2\pi$ is assigned to the walking action depending on the feet velocities. Since the gating network needs differentiable functions for the training, the trigonometric functions are applied on top of p_i to form P_i.

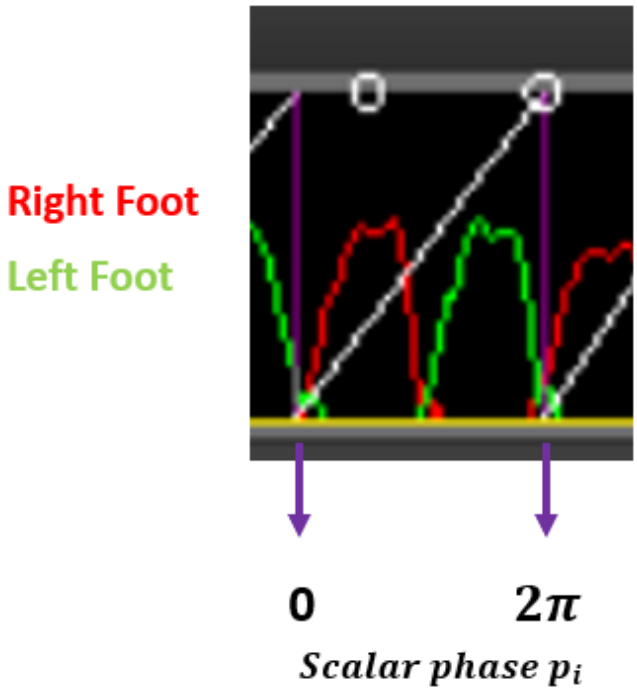


Figure 3.3: Scalar phase value p_i for the walking action. The red and green lines are the feet velocities, whereas the white line is the incrementing scalar value from 0 to 2π.

The last part of the input in equation 3.5 (X_i') is denoted as

$$X_i' = \{t_{i-1}^a, g_{i-1}^a, \delta \cdot g_{i-1}^a, \theta \cdot g_{i-1}^a\}, \tag{3.7}$$

where t_{i-1}^a is the current action, g_{i-1}^a is the action at the goal, δ is the distance scalar to the goal position and θ is the angular scalar to the goal orientation.

The gating network configuration is similar to the ones from the experts, namely a three layer fully connected network denoted as

$$\omega = \Omega(\hat{X}; \mu) = \sigma(W_2' \, ELU(W_1' \, ELU(W_0' \, \hat{X} + b_0') + b_1') + b_2') \tag{3.8}$$

where ω represents the blending coefficients for the motion prediction network, $\mu = \{W_0', W_1', W_2', b_0', b_1', b_2'\}$, which are the gating network's weights and biases, and $\sigma(\cdot)$ is the softmax function to normalize the network's outputs, so that they sum up to one.

3.3 Locomotion Model for Stair Walking

The NSM was made to deal with actions that happen over a plane, in other words, the character could not move on different heights. Thus, the original NSM had to be modified and extended in order to account for uneven terrain (which is a requirement for the stairs). But there was a question that needed to be addressed, namely how to modify and extend the NSM such that it can perform a natural looking stair walking?

In the following sections, a deep explanation addressing the stair walking problem is presented. First, the inputs and outputs required in the locomotion model are explained in more detail. Second, the footstep plan approach used is presented. Third, the feet trajectory correction that follows the footstep plan is shown at the end of this chapter.

3.3.1 Network Parameters and Input Format

In order to be able to perform the stair walking action using the NSM, some new information was needed, namely a footstep plan to have a goal for the feet (just as it was done in [4]) and also feet trajectory estimation and correction as some sort of control signal to drive the motion of the feet (similar to [44]). All this newly provided information is explained in more detail in later sections.

Motion Prediction Network Inputs

The inputs of the extended motion prediction network are the following (same notation as in [44]):

- **Frame input** $F_i = \{j^p_{i-1}, j^r_{i-1}, j^v_{i-1}, t^p_{i-1}, t^d_{i-1}, t^a_{i-1}, f^{(left)}_{i-1}, f^{(right)}_{i-1}\}$, composed of:
  - *Character pose in frame* $i-1$: $j^p_{i-1} \in \mathbb{R}^{3j}, j^r_{i-1} \in \mathbb{R}^{6j}, j^v_{i-1} \in \mathbb{R}^{3j}$ are the joint positions, rotations and velocities, respectively, relative to the root coordinate in last frame $(i-1)$. j denotes the joints of the character (in this case 23; it can be seen in figure 3.4).
  - *Past/Future root trajectory*: $t^p_{i-1} \in \mathbb{R}^{2t}, t^d_{i-1} \in \mathbb{R}^{2t}$ are the root coordinate positions and forward directions in the 2 second window (-1 second to 1 second relative to the root coordinate of the last frame i-1). This window is sampled with t (t=13) trajectory points (6 samples in the past, 6 in the future and 1 present sample). The continuous action labels (with values from 0 to 1) of each of the t sampled points are denoted by $t^a_{i-1} \in \mathbb{R}^{4t}$. This auto-regressive vector describes the current state of the character and has four different types: idle, walk, ascend and descend.
  - *Past/Future feet trajectories* : $f^{(left)}_{i-1} \in \mathbb{R}^{3t}, f^{(right)}_{i-1} \in \mathbb{R}^{3t}$ are the sampled ankle trajectory points of the left and right foot, respectively, relative to the root coordinate in the last frame (i-1).

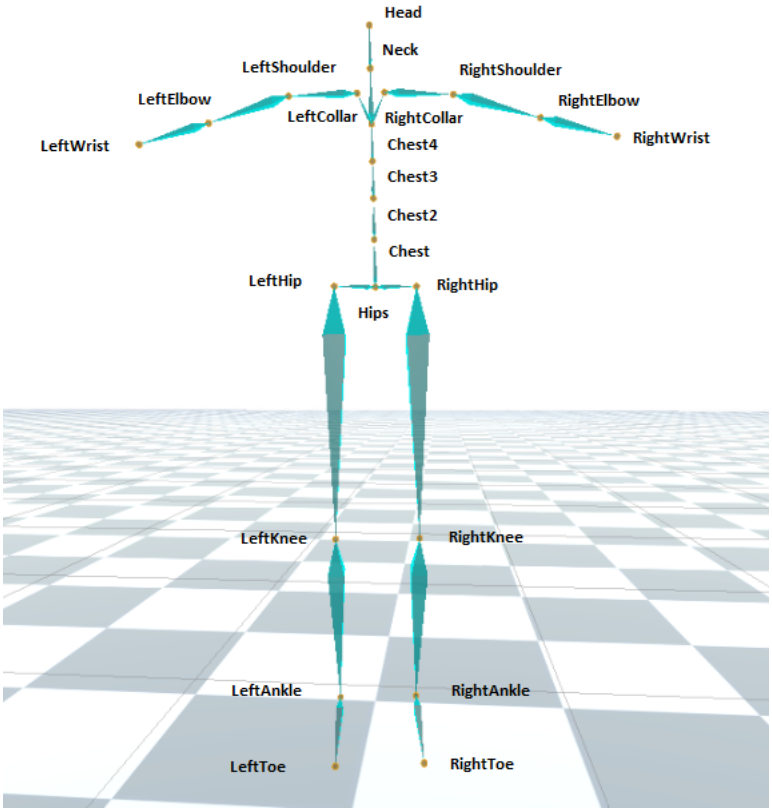


Figure 3.4: Visualization of the skeleton used in the framework containing 23 joints.

- **Goal input** $G_i = \{g^p_{i-1}, g^d_{i-1}, g^a_{i-1}, g^{(left)}_{i-1}, g^{(right)}_{i-1}\}$, composed of:
  - *Goal positions and orientations* : $g^p_{i-1} \in \mathbb{R}^{3t}, g^d_{i-1} \in \mathbb{R}^{2t}$ are the 3D coordinate goal positions and the 2D directions in the horizontal plane, respectively. These are sampled along the 13 trajectory points and computed relative to the root coordinate of the last frame (i-1). Unlike the original NSM, here only a low-level control is used, meaning that the user is the one that controls the locomotion of the character. In this type of control mode, each goal sample is located 1 second in the future with respect to each trajectory sample, which means that there is a window that covers 2 seconds in the future that starts at the present frame.
  - *Action at the goal* : $g^a_{i-1} \in \mathbb{R}^{2t}$ is composed of one-hot action labels that should be launched when reaching the goal. Here, only the idle and walk actions are used since it is not trivial to specify the ascend and descend actions to happen at the correct timing.
  - *Footstep goals* : $g^{(left)}_{i-1} \in \mathbb{R}^3, g^{(right)}_{i-1} \in \mathbb{R}^3$ are the 3D coordinate goal positions of the left and right foot, respectively, relative to the root coordinate frame of the last frame (i-1). During training, these goals are recorded using a 1 second time window in the future with respect to the current frame.
- **Heightmap input** $H_i \in \mathbb{R}^{528}$ is a representation of the surrounding floor below the character (section 3.3.4).

Table 3.1: Summary of inputs of the motion prediction network. $j = 23$ and $t = 13$.

Type	Name	Dimension	Total
Frame inputs	j^p_{i-1}	$3j$	69
	j^r_{i-1}	$6j$	138
	j^v_{i-1}	$3j$	69
	t^p_{i-1}	$2t$	26
	t^d_{i-1}	$2t$	26
	t^a_{i-1}	$4t$	52
	$f^{(left)}_{i-1}$	$3t$	39
	$f^{(right)}_{i-1}$	$3t$	39
Goal inputs	g^p_{i-1}	$3t$	39
	g^d_{i-1}	$3t$	39
	g^a_{i-1}	$2t$	26
	$g^{(left)}_{i-1}$	3	3
	$g^{(right)}_{i-1}$	3	3
Heightmap	H_i	$3 * 176$	528

A summary of all the inputs of the motion prediction network can be seen in table 3.1.

Motion Prediction Network Outputs

The outputs of the extended motion prediction network, which is denoted as $Y_i = \{j^p_i, j^r_i, j^v_i, \tilde{j}^p_i, f^{(left)}_i, f^{(right)}_i, t^p_i, t^d_i, t^a_i, \tilde{t}^p_i, \tilde{t}^d_i, \dot{g}^p_i, \dot{g}^d_i, \dot{g}^a_i, g^{(left)}_i, g^{(right)}_i, c_i, \dot{p}_i\}$, are the following (same notation as in [44]):

- **Predicted character pose in egocentric coordinate system** : $j^p_i \in \mathbb{R}^{3j}, j^r_i \in \mathbb{R}^{6j}, j^v_i \in \mathbb{R}^{3j}$ are the estimations of the joint positions, rotations and velocities, respectively, with respect to the root coordinate of the current frame (i).
- **Predicted character joint position in the future egocentric coordinate system** : $\tilde{j}^p_i \in \mathbb{R}^{3j}$ represents the estimated joint positions in the egocentric coordinate system 1 second in the future.
- **Future feet trajectories in egocentric coordinate system** : $f^{(left)}_i \in \mathbb{R}^{3t'}, f^{(right)}_i \in \mathbb{R}^{3t'}$ are the estimated feet trajectory positions with respect to the root coordinate of the current frame (i).
- **Future root trajectories in egocentric coordinate system** : $t^p_i \in \mathbb{R}^{2t'}, t^d_i \in \mathbb{R}^{2t'}$ are the estimated root coordinate positions and forward directions, respectively, of the $t' = 6$ future trajectory points with respect to the root coordinate. $t^a_i \in \mathbb{R}^{4t'}$ represents the continuous actions labels in a window of 1 second in the future.
- **Future root trajectories in goal-centric coordinate system** : $\tilde{t}^p_i \in \mathbb{R}^{2t'}, \tilde{t}^d_i \in \mathbb{R}^{2t'}$ are the same as the root trajectories in egocentric coordinates, but these are defined with respect to the goal-centric coordinate system.
- **Goal output** : $\dot{g}^p_i \in \mathbb{R}^{3t}, \dot{g}^d_i \in \mathbb{R}^{3t}, \dot{g}^a_i \in \mathbb{R}^{2t}$ are the updated positions, forward directions, and goal actions, respectively.

Table 3.2: Summary of outputs of the motion prediction network. $j = 23$ and $t' = 7$. The entries in blue are the added outputs.

Name	Dimension	Total
j_i^p	$3j$	69
j_i^r	$6j$	138
j_i^v	$3j$	69
$\tilde{j}_i^p$	$3j$	26
$f_i^{(left)}$	$3t'$	21
$f_i^{(right)}$	$3t'$	21
t_i^p	$2t'$	14
t_i^d	$2t'$	14
t_i^a	$4t'$	28
$\tilde{t}_i^p$	$2t'$	14
$\tilde{t}_i^d$	$2t'$	14
$\dot{g}_i^p$	$3t$	39
$\dot{g}_i^d$	$3t$	39
$\dot{g}_i^a$	$2t$	26
$g_i^{(left)}$	3	3
$g_i^{(right)}$	3	3
c_i	4	4
$\dot{p}_i$	t'	7

- **Footstep goals** : $g_i^{(left)} \in \mathbb{R}^3, g_i^{(right)} \in \mathbb{R}^3$ are the estimated footstep goals of the left and right foot, respectively.
- **Contact labels** : $c_i \in \mathbb{R}^4$ denotes the contact labels of the key joints. The four key joints used are the left ankle, right ankle, left wrist and right wrist (the wrist contacts are not used but could be helpful for handrails in future work).
- **Update in phase** : $\dot{p}_i \in \mathbb{R}$ is the update of the angular phase.

In table 3.2, a summary of the outputs of the motion prediction network is shown.

Motion Prediction Network Architecture

The overall architecture of the motion prediction network can be seen in figure 3.5. All layers in the network are fully-connected. The outputs of the encoder module are concatenated and passed as inputs to the prediction module, which consists of 10 different sets of weights (experts). The features of the experts are blended together using the outputs of the Gating Network. The blended features give the final inference output Y. This final output Y is used to animate the character at the current frame. Also, some of the outputs are first corrected as it is explained in later sections.

Auto-regression

As it was mentioned in section 3.2, the NSM is a time-series model, thus it needs the past information to make an estimation of the next time step. Thus, the predicted information is fed back as input to the network for the next estimation. The current output of the

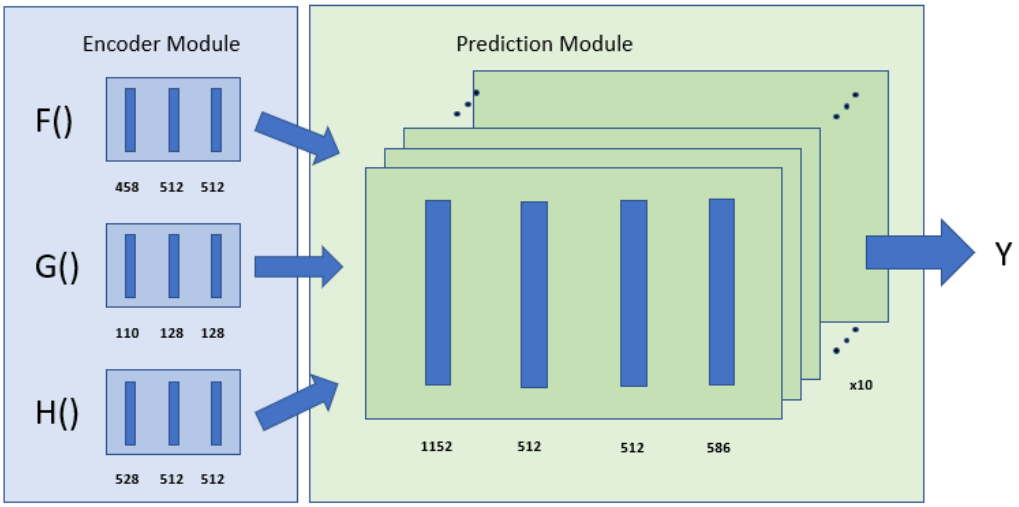


Figure 3.5: Architecture of the motion prediction network. The blue bars represent network layers. The numbers are the units for every specific layer.$F()$ is the frame input, $G()$ is the goal input and $H()$ is the heightmap input. All layers are fully-connected.

network is used to animate the character and some of these outputs are directly fed back as inputs, whereas the root trajectory outputs are blended with the user input. This blended result serves as a control signal for the facing direction of the virtual character. In the case of the feet of the character, a control signal is also needed for the correct foot placement, therefore the footstep goals and foot trajectory estimations are utilized. These corrections, just as with the root trajectory correction, are also fed back as input for the next iteration (more details are presented in sections 3.3.2 and 3.3.3). The contact labels can be used as post-processing step by using some IK solver. The phase is passed to the gating network after it is updated.

Gating Network Architecture

The inputs of the gating network are the same as they were mentioned in section 3.2.1, since there are only two possibilities for actions at the goal, namely the idle and the walk actions.

The architecture of the gating network can be seen in figure 3.6. All the layers of the network are fully connected. The outputs are used as weights to blend the parameters of the motion prediction network, as it was mentioned before.

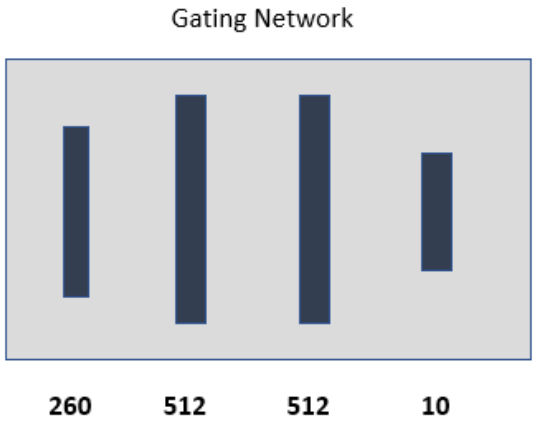


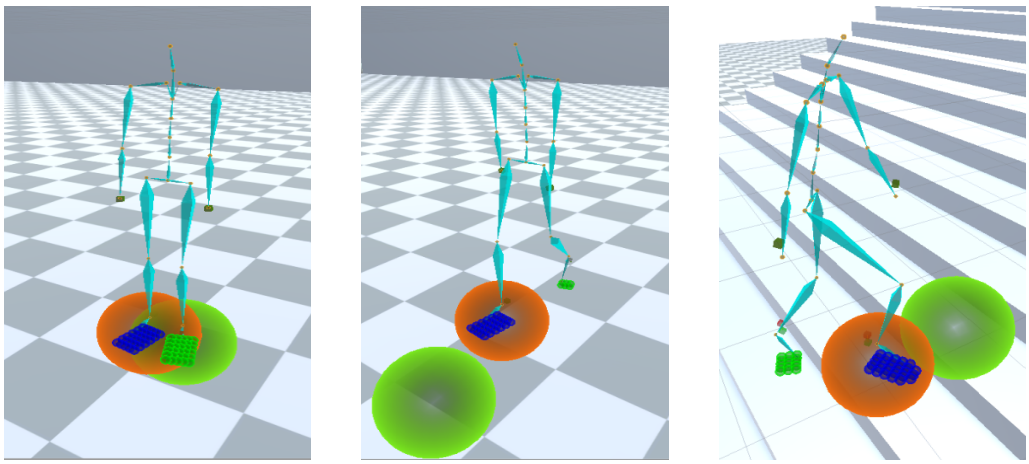
Figure 3.6: Architecture of the gating network. The dark bars represent network layers. The numbers are the units for every layer. All layers are fully-connected.

3.3.2 Footstep Plan

The original NSM was not completely suitable for stair walking because it needed a mean to control the foot placement. Thus, a natural question was: how to modify the NSM so that it is possible to perform stair walking properly? The work of Beacco et al. [4] already showed that by using footstep goals it is possible to obtain good foot placement. Therefore, a footstep goal for each foot is considered in this thesis. Also, it has been previously shown by van de Panne [51] that a natural locomotion can be generated by using only two footstep goals, so, there is no need to plan more than two steps ahead of time.

In figure 3.7, the footstep goals for different actions are shown. These figures are from the training data. The footstep goals for the Idle action are located under each foot of the character (figure 3.7a). For the walk and stair walking actions (figures 3.7b, 3.7c respectively), there is a foot step goal for the stance foot (orange sphere) and there is another foot step goal for the swing foot (green sphere). These footstep goals are taken from a one second window ahead of the current frame.

When the character performs stair walking during online inference, the network estimates by itself the next footstep goals. These footstep goals are not necessarily placed in the desired locations in order to have a natural looking locomotion. For example, a footstep goal might be on the edge of a tread, which may lead to placing the ankle of the stance foot over that edge leaving the rest of the foot on the air (as it was shown in figure 2.6). To avoid or reduce such cases, it is possible to correct the footstep goal to a desired position. Since having most of the foot surface over a tread is considered to be a natural foot placement, a good choice for a footstep goal is the center of a tread. Thus, once the character is walking through stairs, the next footstep goal is placed at the center of the next tread. With this, it is possible to ensure that the footstep goals are being placed in a desired position. This corrected footstep goal is then fed back as input for the next iterations until it is reached. Once a footstep goal is reached, the next footstep goal for the other foot is corrected in the same manner, and the process repeats.



(a) (b) (c)

Figure 3.7: Footstep goals for different actions. Left: footstep goals for idle action. Middle: footstep goals for walk action. Right: footstep goals for ascend action. The orange sphere is the footstep goal of the right foot, whereas the green sphere is the footstep goal of the left foot.

3.3.3 Trajectory Correction

Correcting the footstep goal estimations from the network can improve the result of stair walking, but usually that is not enough to correctly place a foot of the character over a tread. To improve upon that, it is possible to also correct the estimated future trajectory for the feet, since this estimation may not necessarily be accurate enough. In order to correct the foot trajectories, it is possible to use the corrected footstep goals to guide the correction.

The procedure to correct the trajectories of the feet is as follows:

1. The network estimates the footstep goals and the feet trajectories (figure 3.8a).
2. The footstep goals are fixated in the desired positions (center of tread), as it was mentioned in section 3.3.2.
3. There is going to be a distance threshold that determines when the trajectory correction can be applied. This threshold parameter was chosen to be 50 cm during tests (figure 3.8b).
4. If a point of the trajectory lies within the distance threshold, then the *target vector* (vector between such point and the target goal) is added so that the point in the trajectory matches the desired footstep position (figure 3.8c).
5. That same vector is applied to the other points in the trajectory, but a weighting function is also used to reduce the weight of the vector the farther away the points are (figure 3.8d).
6. This corrected trajectory and footstep goals are then fed back as input to the network for the next iteration.

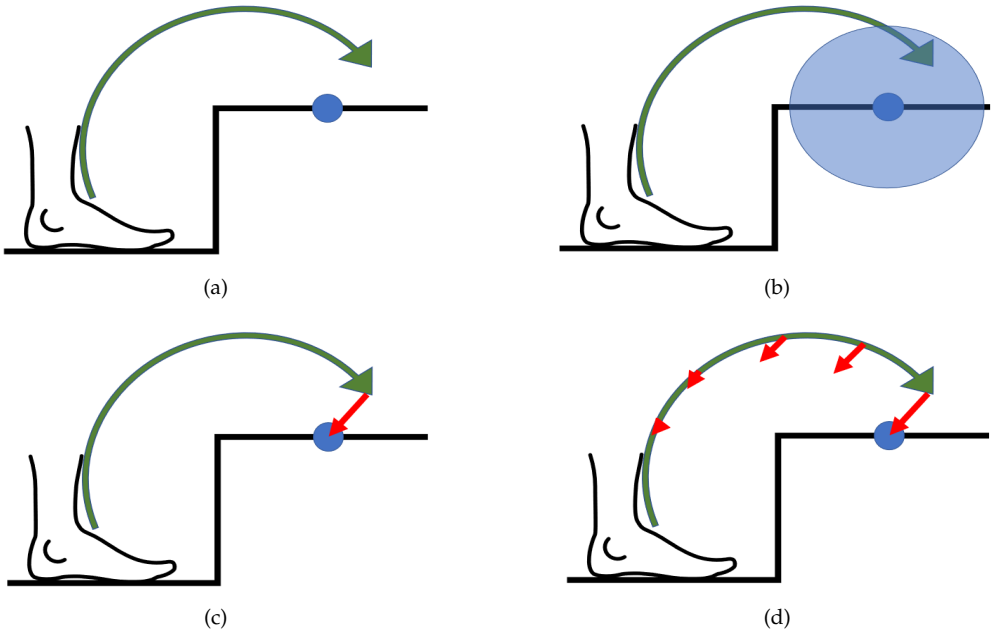


Figure 3.8: Foot trajectory correction. Top Left: foot trajectory estimation of Motion Prediction Network. Top Right: distance threshold of desired footstep goal. Bottom Left: matching closest trajectory point with desired goal. Bottom Right: applying vector to trajectory using weighting function.

The weighting function used to reduce the influence of the target vector is given by

$$p_i = \hat{p}_i + (1 - M * W) * O, \tag{3.9}$$

where p is the corrected position, $\hat{p}$ is the estimated position given by the network and O is the offset vector given by the difference between the step goal and the closest goal in the estimated trajectory (this would be represented by the longest red arrow in figure 3.8d). The term M is a constant multiplier given by

$$M = \frac{0.5}{K_c - P_k}, \tag{3.10}$$

where $K_c = 13$ is the key count number (number of key points of the trajectory) and $P_k = 6$ is the pivot key (current frame). Initially, the value in the numerator was chosen to be 1, but after tests, 0.5 offered a better performance. The last term W is given by

$$W = |i_o - i|, \tag{3.11}$$

where i_o represents the index in the trajectory of the offset vector and i is the index of the p_i position that is being computed. In other words, the corrected trajectory point is given by the estimated point plus the offset vector weighted by the index position in the trajectory (the closer the index i to the index i_o, the stronger the influence of the offset vector).

3.3.4 Heightmap Sensor

In this thesis, correct foot placement is something important, thus a mean to describe the surrounding ground of the character is needed. There are different sensors used in the literature and each sensor offers advantages or disadvantages depending on the use case. For example, it is possible to use shoot rays in the facing direction of the character to check the ground ahead [6, 26], but this kind of sensor is too narrow to capture the description all the surroundings. There are also more complex sensors, like the volumetric sensors, and these can offer a good description of the surrounding environment (not just the ground) [44]. For stair walking it is only needed to describe the surrounding ground of the character, not what lies above it, thus a *heightmap* is a good choice just as it was used by Peng et al. [39]. A heightmap would not waste samples, since it is not a volumetric sensor, and would also give a good representation of the ground.

A heightmap can be created by shooting rays around the character and taking the hit points over the ground surface. Those points are then computed with respect to the root of the character. A representation of the used heightmap can be seen in figure 3.9.

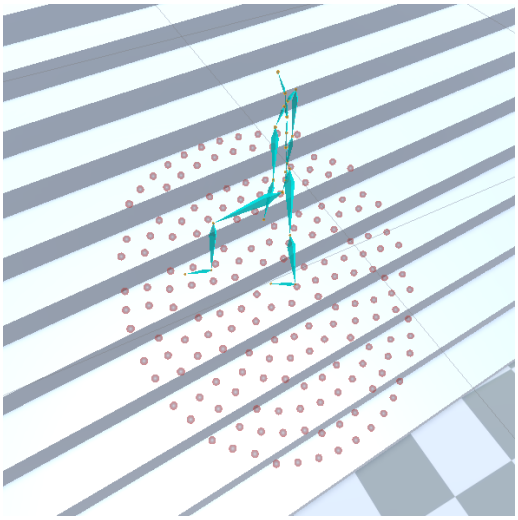


Figure 3.9: Heightmap sensor. The red dots around the character represent the hits of the shot rays (the height values are recorded in those places).

Chapter 4
Data Capturing

4.1 Motion Capture

For the specific task of stair walking, new data was required. All this new data was recorded using a XSense intertia MVN Awinda Motion Capture (MoCap) System [54], which can be seen in figure 4.1a. The recordings took place in a 18 steps staircase that can be seen in figure 4.1b.

(a)

(b)

Figure 4.1: Left: XSense MoCap suite. Right: 18 steps stairs used for data recording.

The user of the MoCap suite had to follow different plans that were design to capture as many transitions as possible. This plans are called *dance cards* [55]. One of them can be seen in figure 4.2, which shows a plan to follow:

1. Start with *Idle* state.
2. *Walk* 3 steps starting with left foot.
3. *Ascend* stairs 5 steps starting with right foot.
4. Stop ascending and keep the feet on different treads.
5. Continue ascending 5 steps starting with left foot.
6. Stop ascending and keep the feet on different treads.
7. Continue the steps 3-6 until the end of the stairs.
8. Walk straight.
9. Stop.

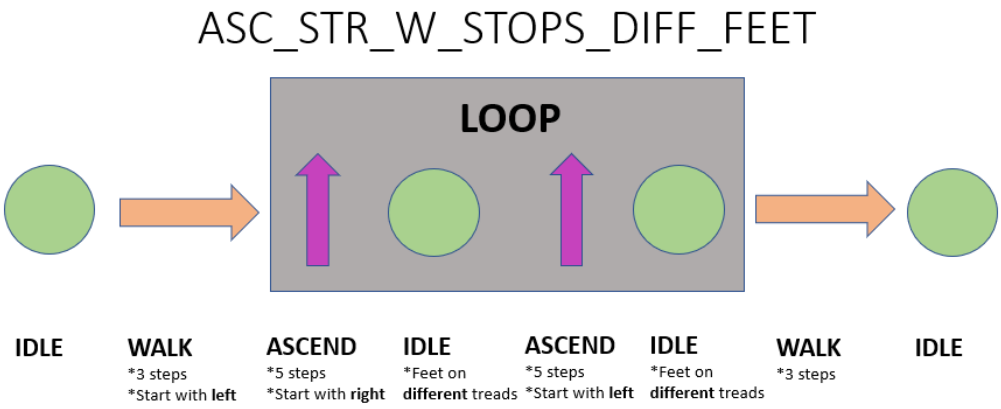


Figure 4.2: Example of a dance card used for data capturing.

The main idea of this plan is to capture the the transition between the walk-ascend-walk actions as well as the ascend-idle-ascend actions. All the other dance cards used for the data capturing process are in the appendix A.

4.2 Data Labelling

Since in this thesis a supervised learning model is used, the training data needs to be labeled accordingly. In order to process the data, a MoCap editor for labelling is used (the MoCap editor is publicly available in the supplementary material of [44]). The labelling process is simple: The MoCap data needs to be imported and afterwards labeled by using the tools of the editor. There are different annotations that have to be done, namely the phase value, the action labels, the goal labels and the contact labels.

4.2.1 Phase Value

As it was previously mentioned in section 3.2.1, the phase value indicates the current stage of an action that is being performed. For the case of basic stair walking, it is somewhat simple to define the start and the end of an action since walking is a cyclic movement. Thus, the starting of a phase is given by the moment when the right foot changes from the *stance phase* to the *swing phase* (foot takeoff), and the end of the phase is then when this transition happens again (end of a walking cycle).

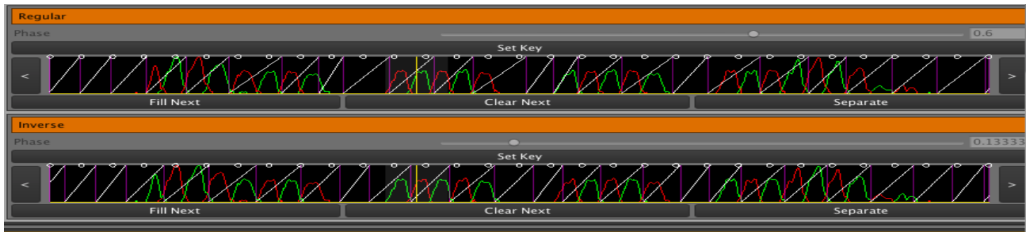


Figure 4.3: Phase labeling of a BVH file. The red line represents the velocity of the right foot, the green line represents the velocity of the left foot, the white diagonal lines are the increasing scalar values from 0 to 2π and the vertical yellow line represents the current frame. *Regular* is the original MoCap data and *inverse* is the mirrored data.

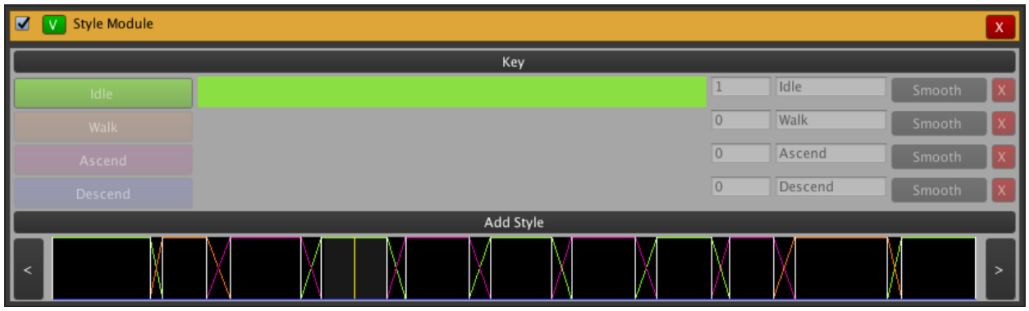
In figure 4.3, the assigned phase value to one of the recordings can be seen. The red and green curves represent the velocity magnitude of the right and left foot, respectively. The purple vertical lines are the start and end of a cycle and the white diagonal lines represent the increasing phase value. Those phase values are the ones resulting after following the dance card in figure 4.2 (Idle -> Walk 3 steps -> Ascend 5 steps -> etc.).

4.2.2 Action Labels

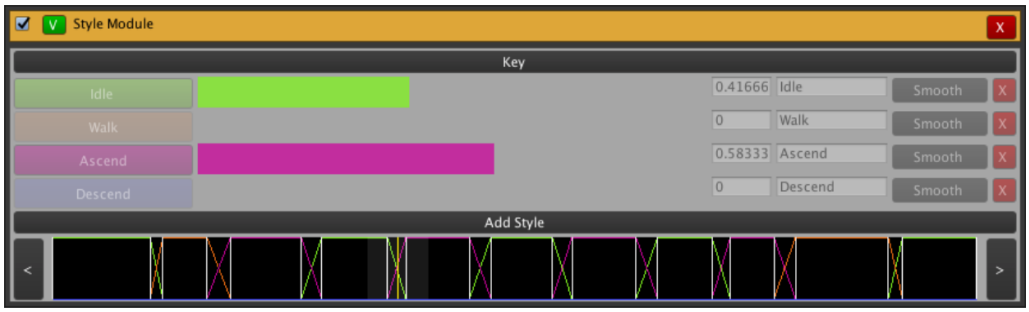
Each frame was annotated with one of 4 different action labels ("idle", "walk", "ascend", "descend") describing the current frame's action. Figure 4.2.2 shows the transition between *idle* and *ascend* actions. The reason to split the stair walking action in 2 (*ascend, descend*) was to avoid difficulties during training due to data imbalance because having different labels allows different experts to learn a specific action and perform better. In table 4.1, an overview of the training dataset is presented. It is important to point out that the values in the table represent the data without mirroring and at the original frame rate (60 hz).

Table 4.1: Overview of the training dataset for stair walking.

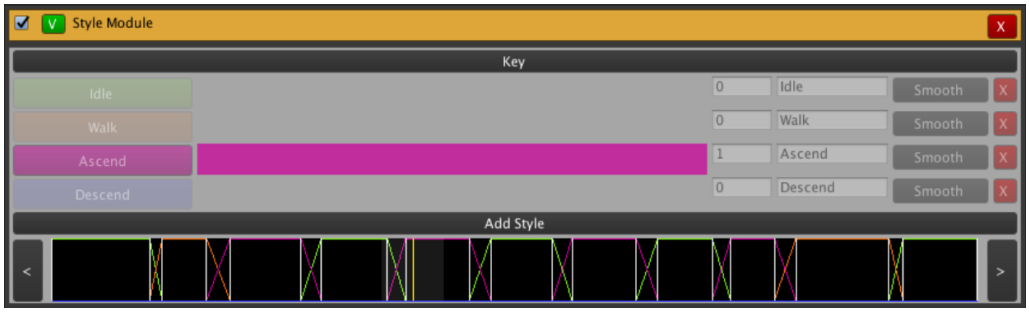
Motion type	time (sec)	frames	ratio(%)
idle	132.12	7927	34.75
walk	111.55	6693	29.34
ascend	70.2	4212	18.46
descend	66.35	3981	17.45



(a)



(b)



(c)

Figure 4.4: Transition of actions. The color bars represent the current action value and the black graph at the bottom of each figure shows the actions per frame for all the current MoCap file (the yellow vertical line is the current frame). Top: character is in idle state. Middle: character is transitioning to ascend action. Bottom: character is ascending the stairs.

4.2.3 Goal Labels

The goal labels represent the goal actions for each frame and they are the action labels of the future frames. In figure 4.5, the goal actions of a MoCap file are shown. There are only two different goal action labels ("idle", "walk") because, same as it was reported by Starke et al. [44], "it is difficult to specify the climb to happen at the right timing". In other words, the user input determines the future goal (pressing a movement key triggers the walking action; no key triggers the idle action; other keys may trigger other actions) and it is really hard to trigger the stair walking action in the right time by pressing a key, thus that action was not taken into account in the future goals. What happens here is that the character still has the walking action as future goal when performing stair walking, but the style changes depending on the preset style label.

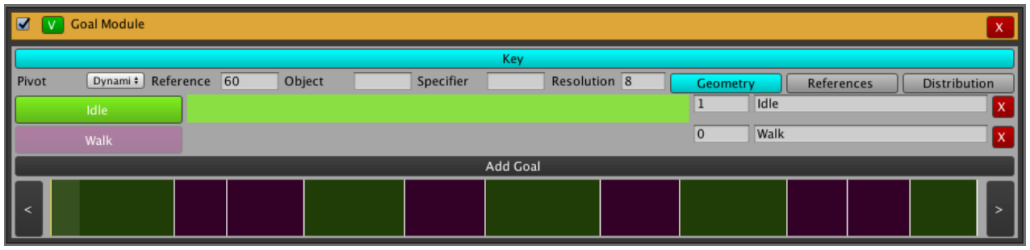


Figure 4.5: Goal labels. The color bar represents the future goal for that frame. The graph at the bottom shows all the future actions of the whole MoCap file.

4.2.4 Contact Labels

The contact labels represent the contact of the joints of a character that interact with the environment. An example of these labels can be seen in figure 4.6, where the green bars represent when a target joint should be in contact with a surface. In this case, the contact labels are given just to the feet (since only simple stair walking locomotion is represented) because these joints interact with the treads of the stairs.

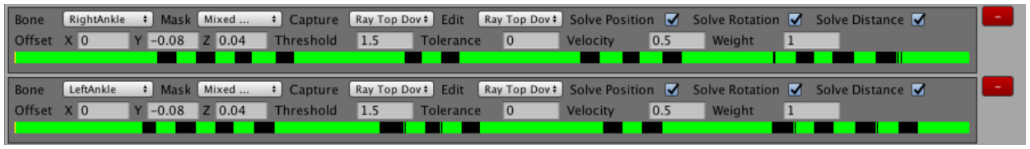


Figure 4.6: Contact labels. The green bars represent a contact with the environment of the corresponding joint. Here, a walking pattern can be seen (when one foot is in contact with the ground, the other one swings).

The contact labels have the purpose of modifying the pose of the character when applying noise to the interaction objects in order to add variation to the training data. The process of data variation is explained in the next section.

4.3 Environment Fitting and Data Augmentation

For the data augmentation, two different techniques were applied (just as in [44]). The first one is just doubling the data by mirroring. The second one is to edit the motion and the geometry such that the context is preserved in order to avoid training data size explosion. The geometry in this case is the stairs, which are manually created using the MoCap data. There are 5 steps in the context preserving data augmentation scheme:

1. **Environment fitting:** Fit the original stairs with the MoCap data using frames when the character is static with respect to a tread.
2. **Creating contact points and modifying key joint trajectories:** Contact points of the joints and the treads are created using a distance threshold and then the key joints are redefined with respect to these newly created points.
3. **Transforming environment:** When the objects are transformed, the contact points follow the applied transformation. For the stairs, the tread and the riser were modified by scaling.
4. **Updating key joint trajectories:** Key joint positions are updated depending on the modifications applied to the contact points.
5. **Full-body pose update:** the pose of the character is updated as well with full-body IK.

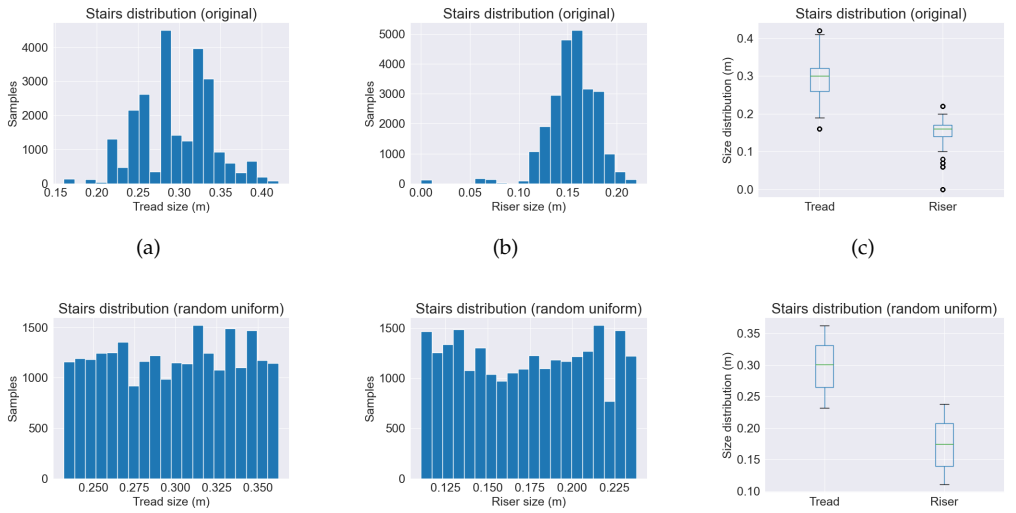
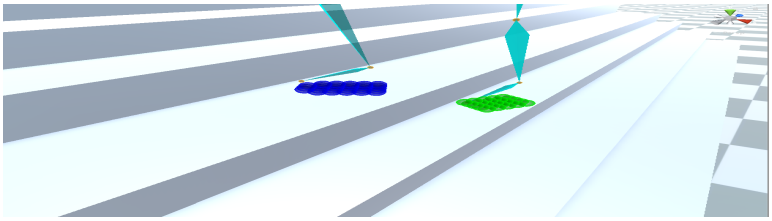


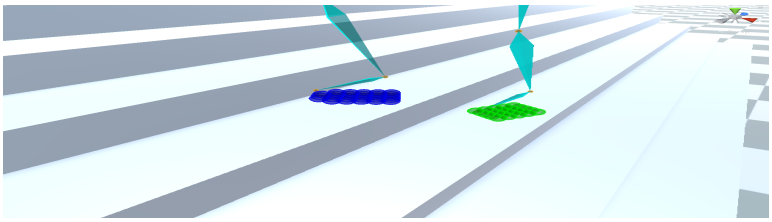
Figure 4.7: Stairs distribution. Top left: histogram of the treads with the original stairs distribution. Top middle: histogram of the risers with the original stairs distribution. Top right: box plot of the original stairs distribution. The stairs should have the same size but the recording technique adds noise. Bottom left: histogram of the treads with the uniform distribution. Bottom middle: histogram of the risers with the uniform distribution. Bottom right: box plot of the stairs with random uniform distribution.

4.4 Stairs Scaling and Data Warping

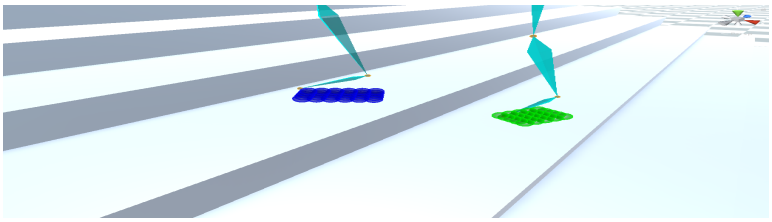
The MoCap solution used to record the data (inertial sensors) inherently accumulates error over time. If the stairs are fitted to this original data, the treads would have different dimensions as well as the risers, since the cumulative error propagates in all directions. To alleviate that, it is possible to transform the stairs such that all treads and risers share the same size. Once all steps of the stairs have the same sizes, it is easier to control the scaling for the data augmentation step mentioned in section 4.3.



(a)



(b)



(c)

Figure 4.8: Scaling of stairs. Top: original tread sizes. Middle: even sized treads (29 cm). Bottom: tread sizes upscaled after they had even size.

In Figure 4.7a, the recorded tread sizes are shown. It is possible to see that there is a bias towards the real tread sizes (29 cm), but there is a lot of noise. Thus, all treads are first scaled to a single size (29 cm, just as the real size) and then scaled using a random uniform distribution. That result can be seen in figure 4.7d. There, it is possible to see that the tread sizes are more evenly distributed. The rescaling procedure can be seen in Figure 4.8. In Figure 4.8a, the original sizes of the treads are shown and it is possible to see that the tread under the foot with the green grid of points is shorter than its neighboring treads. In figure 4.8b, all treads have been scaled to the same real size (tread = 29 cm, riser = 17 cm). In figure 4.8c, the treads have been upscaled. It should be noted that the relative position of the feet w.r.t. the treads is kept, and also the pose of the character is updated accordingly. Also, same scaling procedure is applied to the risers.

If the stairs are transformed, the foot data also needs to be updated accordingly, otherwise the character would not be stepping onto the treads correctly. In order to correct the foot and trajectory positions depending on the applied scale, such points were projected onto the ground and then the relative positions to the corresponding tread were captured. In other words, if a projected point lies on the edge of a tread, if the tread is scaled to twice its original size, the projected point should still lie on the edge of the scaled tread. This is represented in figure 4.9, where the positions of the foot and the trajectories relative to the treads are kept after the transformation is applied. In figure 4.10, the warping is being applied to the trajectory of the foot over the blue grid of points.

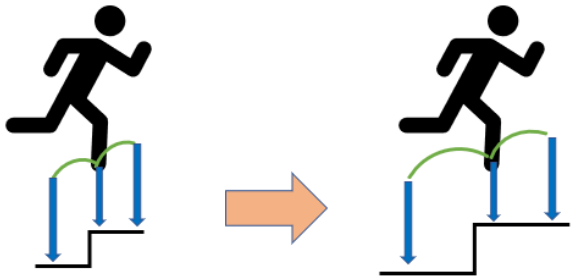
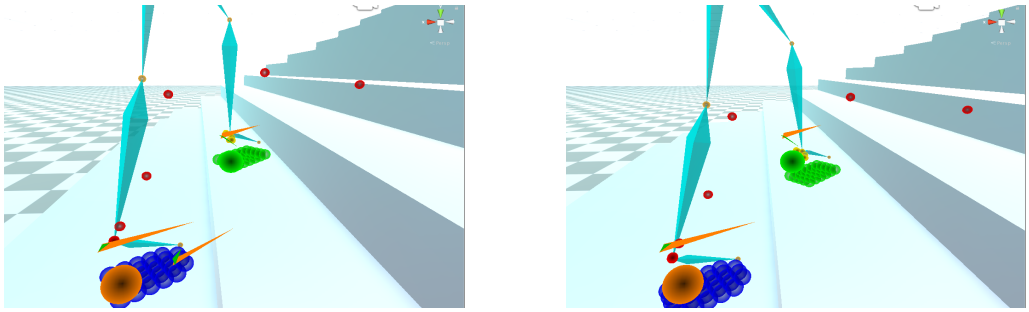


Figure 4.9: Data projection and warping. Left: original data. Right: warped data. The green lines represent the foot trajectories and the blue arrows represent the projections. The trajectory points are first projected onto the treads and then captured. These are later used to warp the data after scaling is applied to the stairs.



(a) (b)

Figure 4.10: Foot trajectory warping. Left: original trajectory. Right: warped trajectory. The red dots represent the future trajectory of the foot over the blue grid of points.

Chapter 5
Evaluation

5.1 Network Training

In order to train the network, the data must be processed and then saved into files. The processing is done as it was reported in chapters 3 and 4. The data is then saved into input and output files in the form $X = [x_1, x_2, ...]$, $Y = [y_1, y_2, ...]$ for every frame. Also, before training, the recorded data is normalized to have a mean of 0 and standard deviation of 1.

The training is done in python using tensorflow [1]. The is code provided by the supplementary material of the NSM [44]. It consists of a regression task, where a given set of inputs X should produce a close result with respect to the outputs Y using the mean squared error

$$Cost(X, Y; \alpha, \beta, \mu) = \|Y - \Theta(H(X; \beta), \Omega(\hat{X}; \mu); \alpha)\|_2^2, \tag{5.1}$$

where X and Y are the inputs and outputs respectively, α, β and η represent the weights of the expert networks, the encoder networks and the gating network respectively, Ω and Θ denote the gating network operation and the motion prediction network operation respectively.

The optimization algorithm used is the stochastic gradient descent with the warm restart technique of AdamWR [35]. This technique uses two parameters T_i and T_{mult} which control the restart and decrease of the weight decay rate λ and the learning rate η. These values are initialized with $\lambda = 2.5 \cdot 10^{-3}$ and $\eta = 1.0 \cdot 10^{-4}$. T_i represents the total number of epochs within the restart i and it is initialized with 10. Every time there is a restart of parameters, T_i is multiplied by $T_{mult} = 2$. Since the total number of epochs is set to be 150, there are 3 restarts at epoch 11, 31 and 71.

The training is done in mini-batches of size 10 and samples are randomly chosen. Also, dropout with retention probability of 0.7 is used (in general, the same hyperparameter settings as in [44] were used). The full training takes around 4.5 hours on a NVIDIA RTX 2070 GPU.

5.2 Metrics

In order to compare the methods, some metrics are proposed: average stepping distance to ground, stepping accuracy and number of steps taken. These metrics are described next.

5.2.1 Average Stepping Distance to Ground

This metric measures the average distance (in cm) between the center of the sole of the corresponding foot and the stepping surface (treads of stairs) while performing a step and it is computed as

$$Avg = \frac{\sum_{i=1}^{i=T} |y_{desired} - y_i|}{T}, \tag{5.2}$$

where Avg is the arithmetic mean, $y_{desired}$ represents the desired height value while stepping (which is 0), y_i is the ith height value considered in the sum and T is the total number of values. The center of the sole (figure 5.1b) is regarded as a point contained in the plane spanned by the toe and the projection of the ankle (blue line in figure) which is located in the middle of those two (the red line represents the normal to the center point).

(a) (b)

Figure 5.1: Left: foot distance w.r.t. treads. Right: foot center point.

The distance values taken into account for this metric are those that are present when the velocity magnitude of a foot is below some threshold (in this case 0.5 was chosen). This metric is useful to describe how close to the treads are the feet of the character while performing stair walking. It also gives an idea of how much is a character sinking or floating around the stairs. For example, if a character is sinking into the stairs, the distance error would increase. The same would happen if the opposite is true (foot floating above treads). This is depicted in figure 5.1a. The ideal step would be if the foot center point is at a 0 distance w.r.t. the treads (sole parallel and coincident to tread).

5.2.2 Stepping Accuracy

This metric measures how centered a foot is being planted on a tread while walking. A rectangular grid of 40 points (5x8) is projected under each foot onto the ground. The height is recorded for each one of the points. Then, all the points on the grid which have the same height as the foot center point are regarded as *correctly placed* (points lie in the same tread as the center of the foot). The ratio between the correctly placed points and the total number of points yields the accuracy. This is represented in figure 5.2.

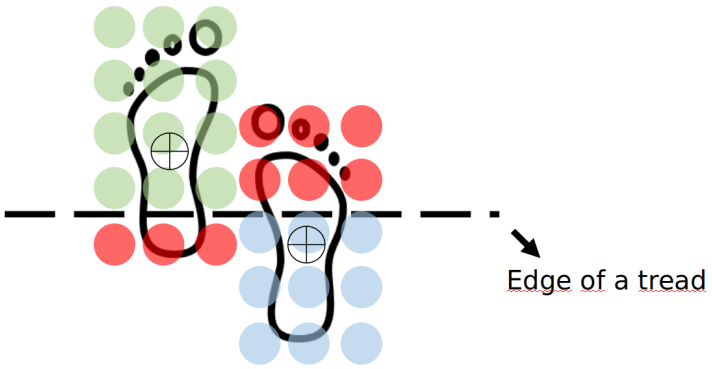


Figure 5.2: Stepping accuracy. A grid of points is projected under each foot. Red points are regarded as incorrectly placed. The cross represents the center of the foot. The green and blue points represent correctly placed points of the left and right foot respectively.

This metric is useful to check if the character is centering its feet on the treads or just places them in a random position. Also, this metric would not discard foot placements that could be considered correct. For example, in figure 5.3, it is possible to see that the tiptoe of the support foot is "flying" but still most of the sole is located on a single tread.

Figure 5.3: A person walking downstairs. Notice that the tiptoe of the support foot is not located above the same tread as the center of the foot. Image taken from [41].

5.2.3 Number of Steps

This metric just measures the number of steps taken while the character walks through stairs. This is helpful to know whether or not the character takes the desired number of steps. Ideally, the character should take as many steps as there are treads on the stairs. If the character deviates much from the desired number of steps, that would mean that the model is not really aware of the geometry of the stairs.

In some cases overstepping (as well as understepping) might be considered natural, for example if the treads of a staircase are really long a person should take more than one step to reach the next tread (and viceversa). Nevertheless, the scope of this thesis is being restricted to sizes of treads and risers that are considered common. Thus, all tread and riser sizes used in the tests are around the ideal values for a staircase, just as it was reported in section 2.1.

5.3 Evaluation Scenarios

Since one of the goals of this thesis is to have a robust stair walking framework for character control, different walking scenarios were created to test the different models using various metrics. The scenarios consist of stairs and a trajectory that the character needs to follow. The tests are performed ten times and after every restart the initial position of the character with respect to the stairs is moved so that the result is not biased towards some convenient start point.

The test scenarios consist of stairs with a fixed number of treads (20). As baseline, the ideal size of real stairs was used (Tread = 29 cm, Riser = 17 cm; this size will be referred as 29x17). After the character walks through the stairs, the character resets to its starting position and the stairs change to a different dimension so that it is possible to measure how well a model performs with different stair sizes. First, the risers are kept fixed (17 cm) and the treads vary from 23 to 35 cm with increments of 3cm. Then, the treads are kept fixed (29 cm) and the risers vary from 11 to 23 cm with increments of 3 cm. Thus, the character walks through 9 different staircases in total. Once the character is finished with those stairs, that is considered a *run*. So, in order to check the reliability of a model, 10 runs are performed and for each one of them the character is changed to a new initial position (10 cm away from the previous one).

The tested methods in these scenarios are NSM_Steps, PFNN and NSM. NSM had to be tested with a smaller variation during training data (Treads: 26cm-32cm, Risers: 14cm-20cm) because when the model was trained with the same variations as the NSM_Steps (Treads: 23 cm-35 cm, Risers: 11 cm-23 cm) there was a *bouncing artifact* that made the model really unstable and the results on the metrics were not comparable at all. By reducing the variation this bouncing artifact is less present and the method is more comparable.

5.3.1 Descending Straight

For the descending straight scenario, the character stands on top of the stairs and is facing downwards. Then, it walks straight and descends. Once the character reaches the bottom ground, the scenario resets to its starting position and another stairs with different sizes are placed. This is done for 10 runs and all the previously mentioned stair sizes. In addition, as it was mentioned in section 5.3, the starting position of the character is changed w.r.t. the first tread in increments of 10 cm.

5.3.2 Descending Diagonal

For the descending diagonal scenario, the character stands on top of the stairs and is facing downwards in diagonal. Then, it walks straight in the initial diagonal direction across the stairs and descends. Once the root of the character reaches the bottom ground, the scenario resets to its starting position and another stairs with different sizes are placed. This is done for 10 runs and all the previously mentioned stair sizes. In addition, the starting position of the character is changed w.r.t. the first tread in increments of 10 cm.

5.3.3 Ascending Straight

For this scenario, the character stands at the bottom of the stairs and is facing upwards. Then, it walks straight and ascends. Once the root of the character reaches the end of the stairs, the scenario resets to its starting position and another stairs with different sizes are placed. This is done for 10 runs and all the previously mentioned stair sizes. In addition, the starting position of the character is changed w.r.t. the first tread in increments of 10 cm.

5.3.4 Ascending Diagonal

For this scenario, the character stands at the bottom of the stairs and is facing upwards in diagonal. Then, it walks straight in the initial diagonal direction across the stairs and ascends. Once the root of the character reaches the end of the stairs, the scenario resets to its starting position and another stairs with different sizes are placed. This is done for 10 runs and all the previously mentioned stair sizes. In addition, the starting position of the character is changed w.r.t. the first tread in increments of 10 cm.

5.4 Results

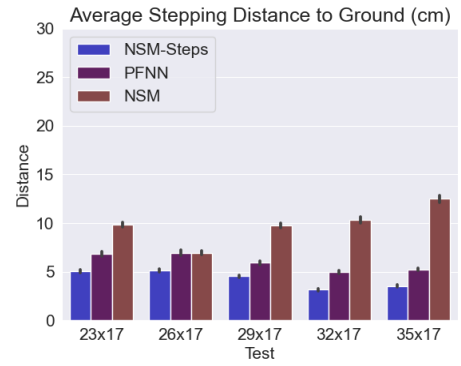
In this section, the results of the comparison between different methods is presented. The method of this thesis (NSM_Steps) is compared against the original NSM. Both are trained with the same parameter configuration unless said otherwise. However, as it was mentioned in section 5.3, the training data variation in NSM was reduced so that it could be compared with the other methods. Also, the original PFNN trained with its original data is also included in the comparison.

In figure 5.4 (left), the results of the metrics in the scenario of descending straight with tread variation are presented. It is possible to see in figure B.2a that NSM_Steps keeps the feet of the character closer to the treads compared to the other two methods. PFNN also does a slightly worse job even with a far greater amount of training data. NSM is less stable and the character bounces, which yields a higher error. Figure B.3a presents a similar accuracy for all methods and it is possible to notice that the greater the treads, the higher the accuracy. Figure B.4a shows NSM_steps is closer than the other two methods to the desired number of steps. PFNN is far behind in this metric since it is not really trained for stairs (it is trained to work in uneven terrain).

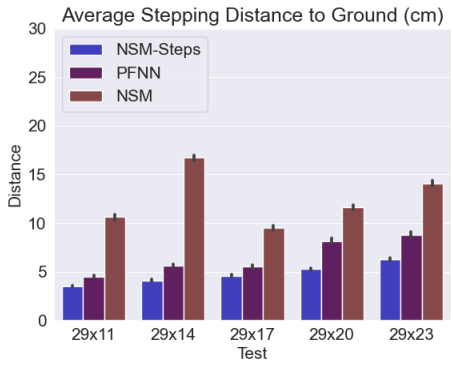
In figure 5.4, the results of the metrics in the scenario of descending straight with riser variation are presented. It can be seen in figure B.2b that NSM_Steps still outperforms the other two methods, even having a considerable difference at larger risers (20 cm and 23 cm). Both NSM_Steps and NSM again fall somewhat behind when it comes to stepping accuracy, as it can be seen in figure B.3b, but at larger riser sizes NSM_Steps takes the lead. Figure B.4b shows that NSM_Steps is closer than the other two methods to the desired 20 steps, taking even the exact number at the largest riser sizes.

In Figure 5.5 (left), the results of the metrics in the scenario of descending diagonal with tread variation are shown. NSM_Steps keeps the feet of the character considerably closer to the ground compared to the original NSM and still beats the PFNN for all stair sizes (approximately 4 cm closer w.r.t. PFNN and 10 cm difference w.r.t. NSM). Regarding the stepping accuracy and number of steps taken though, NSM is a little better than NSM_Steps for small tread sizes, but still closely similar. For large tread sizes, NSM takes many more steps as desired, while NSM_Steps keeps notably closer to the 20 steps.

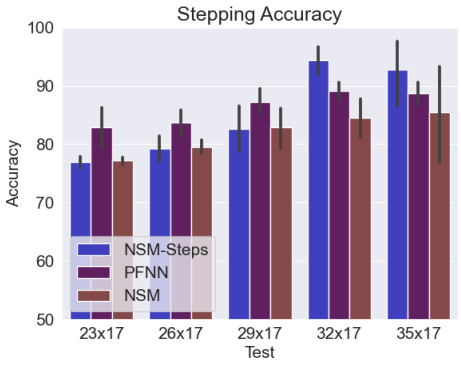
In figure 5.5 (right), the results of the metrics in the scenario of descending diagonal with riser variation are presented. For this tests, NSM_Steps performs better than the other two methods in the average stepping distance metric, as it is shown in figure B.5b, where differences up to 3 cm w.r.t. PFNN and up to 14 cm w.r.t. NSM can be seen. Surprisingly, for the other two metrics presented in figures B.6b and B.7b, NSM performs slightly better than NSM_Steps.



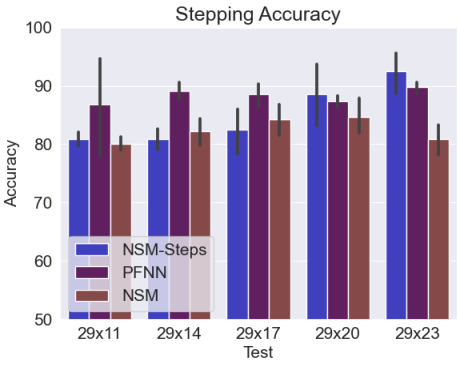
(a) Tread variation



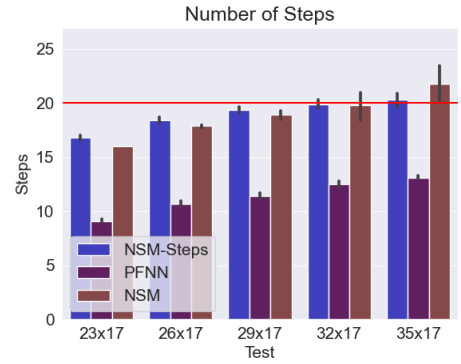
(b) Riser variation



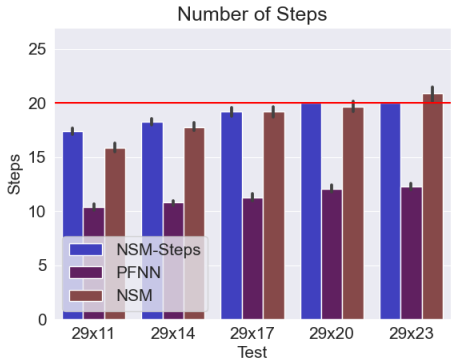
(c) Tread variation



(d) Riser variation

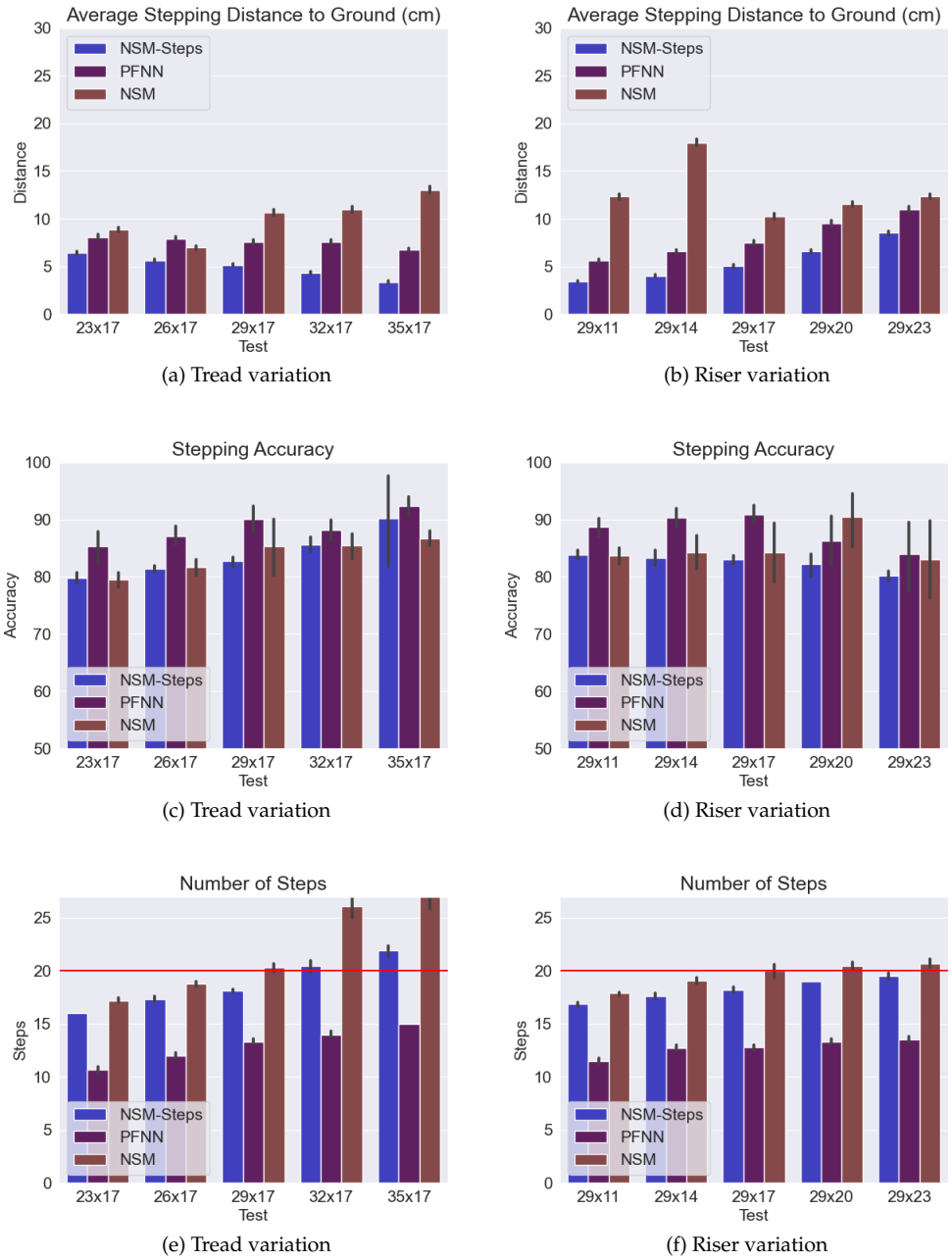


(e) Tread variation



(f) Riser variation

Figure 5.4: Overview of results of the *descending straight* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in appendix B in figures B.2, B.3 and B.4.



(a) Tread variation

(b) Riser variation

(c) Tread variation

(d) Riser variation

(e) Tread variation

(f) Riser variation

Figure 5.5: Overview of the results of the *descending diagonal* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in appendix B in figures B.5, B.6 and B.7.

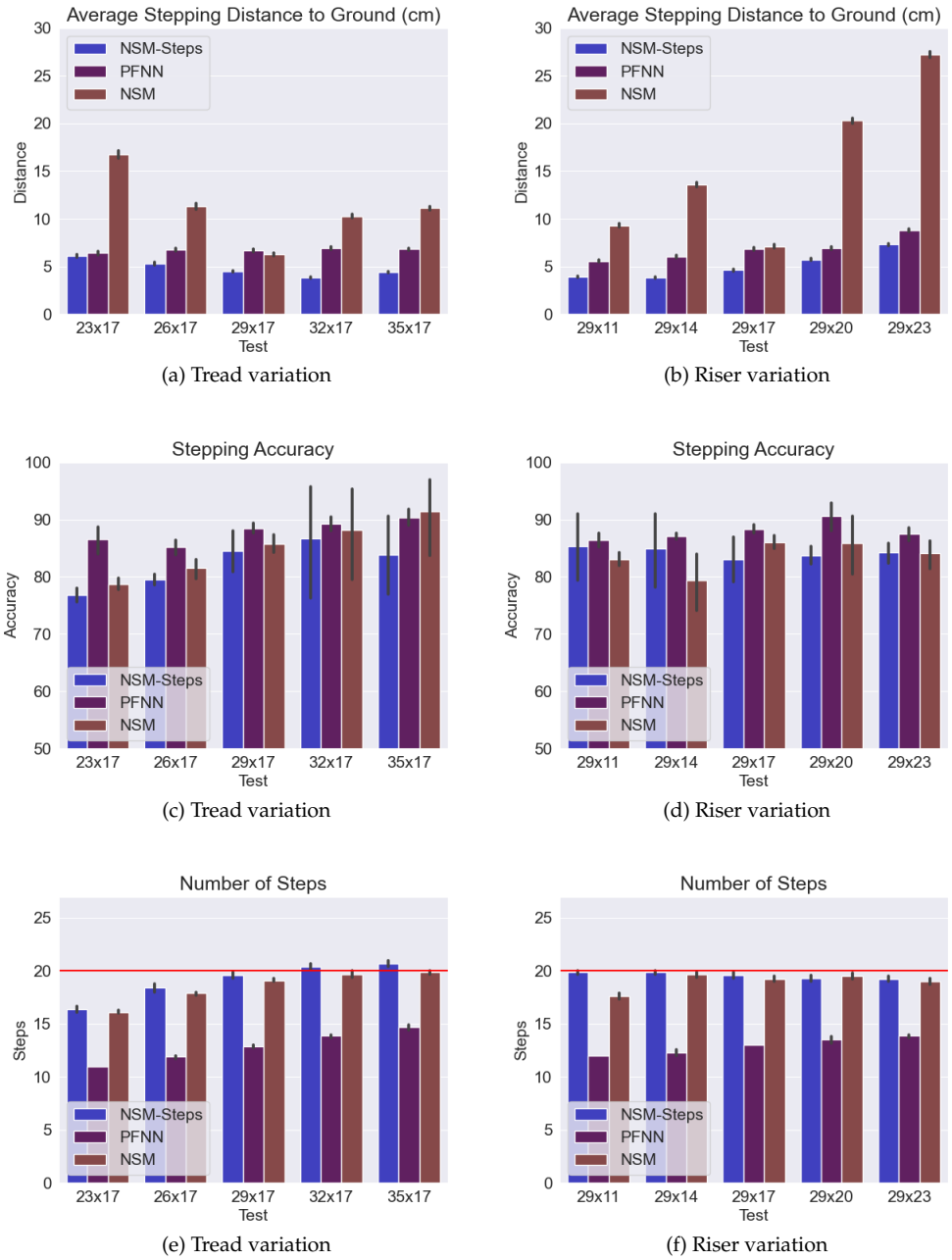
In figure 5.6 (left), the results of the metrics in the scenario of ascending straight with tread variation are presented. It is possible to see in figure B.8a that NSM_Steps still performs better than the other two methods for all tread sizes when it comes to placing the feet closer to the treads. However, figure B.9a shows that the stepping accuracy falls shortly behind NSM. In the third metric presented in figure B.10a, it can be seen that NSM_Steps has a better performance than the other two methods for almost all the sizes, except the largest one, where it has a similar performance to NSM.

In figure 5.6 (right), the results of the metrics in the scenario of ascending straight with riser variation are presented. It can be seen in figure B.8a that NSM_Steps has the lead in all different riser sizes. There, the error of NSM is much more noticeable, compared with the other methods, as the character bounces a lot, especially for the largest sizes. It is also interesting to note in figures B.9a and B.10a that even though NSM bounces a lot, it seems to only affect the character in the vertical direction, since it still has a comparable accuracy with the other methods and a very similar number of steps taken to NSM_Steps, but this last one still takes the lead when it comes to the desired 20 steps.

In figure 5.7 (left), the results of the metrics in the scenario of ascending diagonal with tread variation can be seen. The MSE metric in figure B.11a shows that PFNN performs slightly better than NSM_Steps in these tests, but the latter has a closely similar performance. Regarding the stepping accuracy, PFNN performs better than the other two methods. NSM_Steps and the original NSM perform really close in this metric. Regarding the number of steps taken, as it can be seen in figure B.13a, NSM_Steps is notably better than the other methods.

The results of the metrics in the scenario of ascending diagonal with tread variation are contained in figure 5.7 (right). As it can be seen in figure B.11b, NSM_Steps performs closely similar to PFNN. The character using the original NSM model bounces a lot for large riser sizes. When it comes to the stepping accuracy, shown in figure B.12b, NSM_Steps performs better than NSM, but falls behind PFNN. Regarding the number of steps, which are presented in figure B.13b, NSM_Steps surpasses the other two methods. However, none of them takes the desired 20 steps.

Regarding the runtime, the whole system runs at around 30 frames per second in Unity. The animation part takes approximately 5 ms; the foot placement correction and trajectory correction parts run at approximately 0.1 ms.



(a) Tread variation

(b) Riser variation

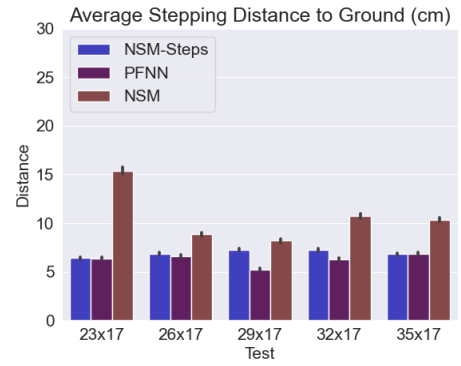
(c) Tread variation

(d) Riser variation

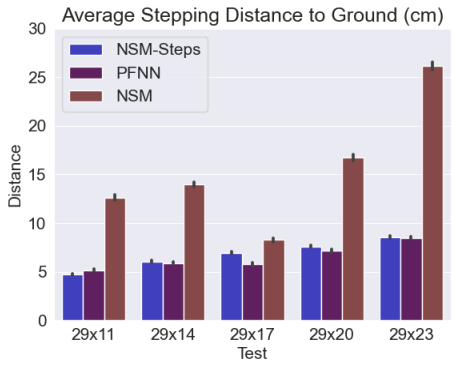
(e) Tread variation

(f) Riser variation

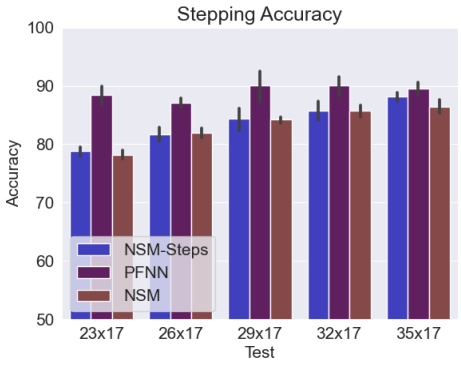
Figure 5.6: Results of *ascending straight* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in appendix B in figures B.8, B.9 and B.10.



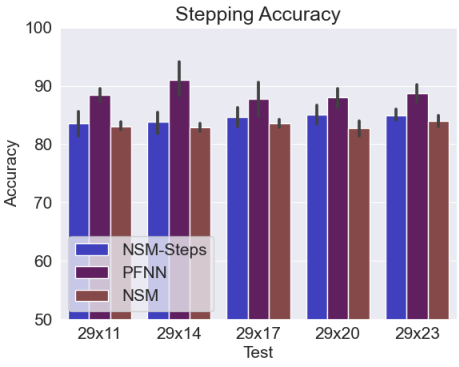
(a) Tread variation



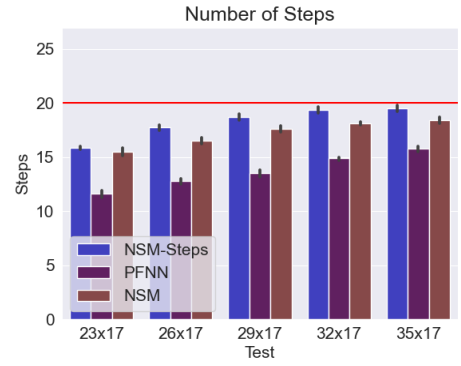
(b) Riser variation



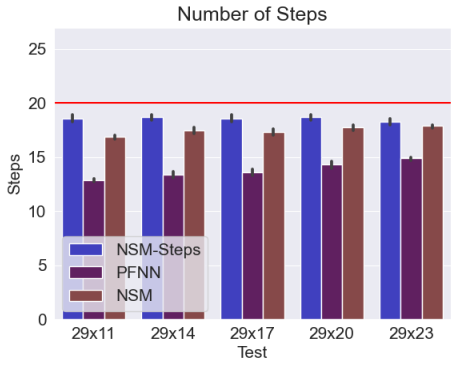
(c) Tread variation



(d) Riser variation



(e) Tread variation



(f) Riser variation

Figure 5.7: Results of *ascending diagonal* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in appendix B in figures B.11, B.12 and B.13.

5.5 Ablation study

In this section, the results of the ablation study for the NSM_Steps model are presented. For the results shown here, the same test scenarios used in section 5.3 were utilized as well (unless said otherwise).

First the influence of the foot trajectory correction in shown, then the effect of the footstep goal correction and at the end, the impact of the heightmap sensor.

5.5.1 Influence of the Foot Trajectory Correction

In order to check the influence of the trajectory correction, the NSM_Steps model was tested several times with some of its added parts deactivated. There are in total 4 different tested models:

- *NSM_Steps*: Same model shown in section 5.4.
- *NSM_Steps_No_Traj*: Same as NSM_Steps, but with the foot trajectory correction deactivated.
- *NSM_Steps_No_Goal*: Same as NSM_Steps, but with the footstep goal correction deactivated.
- *NSM_Steps_No_Corrections*: Same as NSM_Steps, but with the foot trajectory and the footstep goal corrections deactivated.

The results of the models with the corresponding deactivated parts for the *descending straight* test scenario are shown in figure 5.8. For the stepping distance to ground metric, there is some advantage by using the full model compared to the other ones. It can be seen in figures B.14a and B.14b that NSM_Steps beats the other methods (approximately 1 cm in most cases, but around 3 cm in the largest tread size B.14b). However, it is important to notice that the two methods that have the trajectory correction deactivated (NSM_Steps_No_Traj and NSM_Steps_No_Corrections) have a higher average distance to the ground w.r.t. the methods that have it activated. Even though the differences are very similar for the average stepping accuracy, the NSM_Steps method shows a slight numerical improvement of the number of steps taken.

The results of the models for the *ascending diagonal* test scenario are shown in figure 5.9. There, more interesting differences are clearly visible for the distance metric in figures B.17a and B.17b, where the average stepping distance to ground of the models with the foot trajectory correction deactivated explodes dramatically (approximately 60 cm), whereas the distance from the other two methods with foot trajectory correction still keep distance values closely similar to the ones from the descending straight test scenario previously presented (for consistency and for an easier comparison with the other results, the same axis limits were kept). Interestingly, this seems to only affect the result w.r.t. the height, as it can be seen in the rest of the figures.

Regarding the *descending diagonal* and the *ascending diagonal* test scenarios, similar trends follow as well. There are similar results for all methods descending stairs, but the result of the distance metric for the methods with the deactivated foot trajectory correction in the ascending scenario explodes dramatically once again. For the interested reader, such results can be found in the appendix (section B).

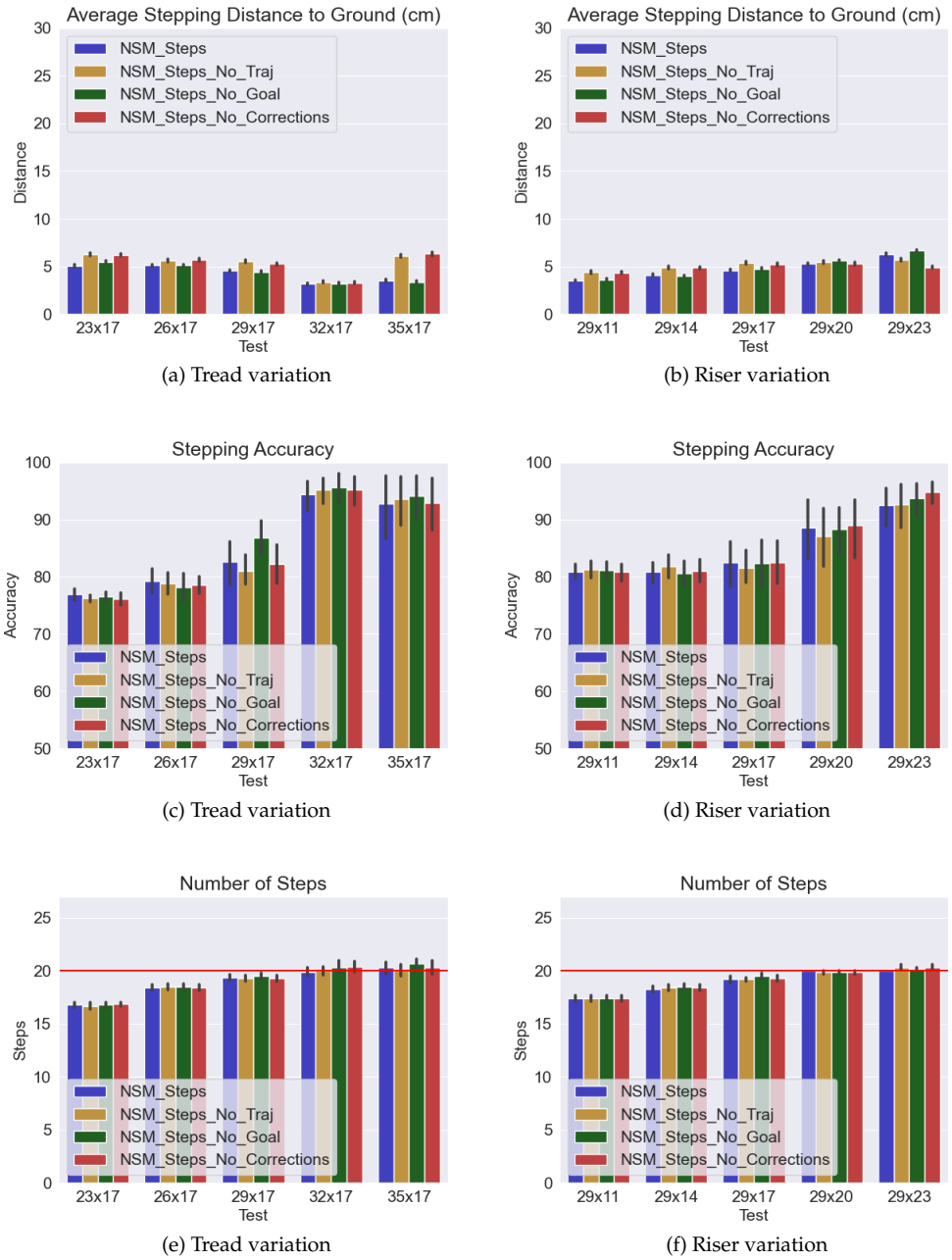
5.5.2 Influence of the Footstep Goal Correction

In order to check the influence of the footstep goal correction, it is not enough by just having a look at the figures 5.8, 5.9 and the appendix (Section B, figures B.20 and B.24), because, on the contrary to the foot trajectory correction, there is not a noticeable difference between NSM_Steps and NSM_Steps_No_Goal (results are virtually equal; for some cases one method performs better than the other in some metrics, but the difference is negligible).

During the tests, by visual inspection, some difference was present when the character was walking through the stairs, thus, some other test scenarios were created too see more clearly if this correction was certainly working. Such scenarios consist of the same procedure as the tests in section 5.5, but the difference is that a staircase of 40 steps is used instead of one with 20 steps.

The results of the average stepping distance to ground metric with the 40 steps staircase varying treads are presented in figure 5.10. Figure 5.10c is the only one of the figures that show almost no difference between the two methods, but still our approach takes the lead.

It possible to see that there is an advantage of NSM_Steps over the method without the footstep goal correction (up to 8% accuracy improvement in foot placement shown in figure 5.10d). It is important to notice that this trend is only visible for large tread sizes (32 cm and 35 cm). For the smallest tread sizes (23 cm, 26 cm and 29 cm) not much advantage was noticeable (results were approximately the same), thus such graphs were omitted. Also, for these tests, the results of the two aforementioned methods in the other two metrics are as well basically the same, and therefore are also skipped. The same holds for the diagonal walking tests (no improvement was found, hence the results of such tests were also neglected).



(a) Tread variation

(b) Riser variation

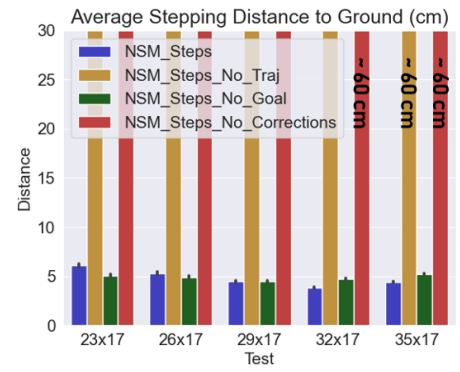
(c) Tread variation

(d) Riser variation

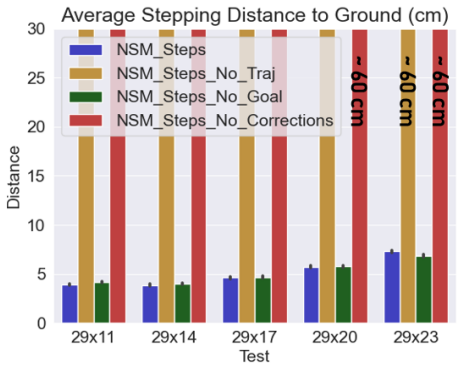
(e) Tread variation

(f) Riser variation

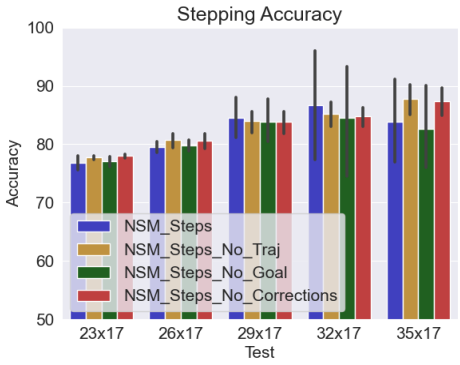
Figure 5.8: Results of *descending straight* test scenario for the ablation study. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in appendix B in figures B.14, B.15 and B.16.



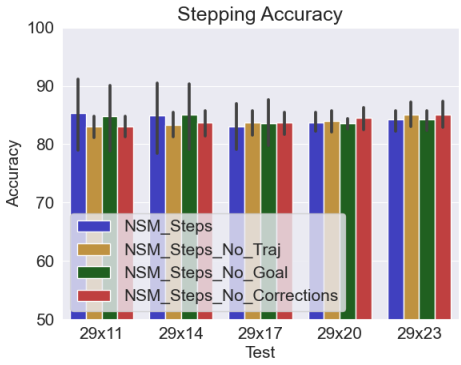
(a) Tread variation



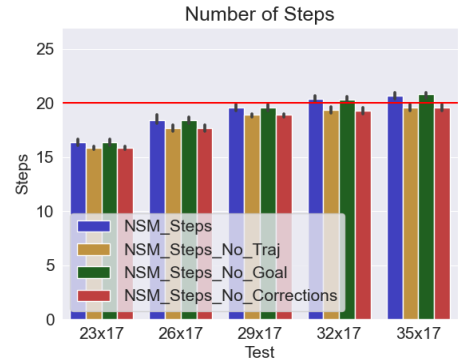
(b) Riser variation



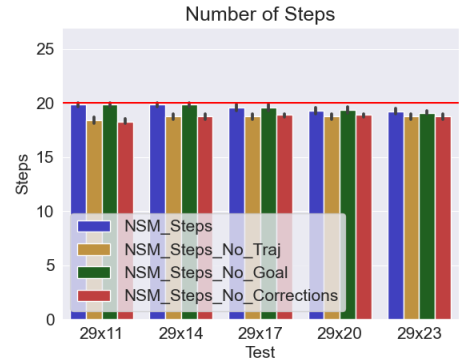
(c) Tread variation



(d) Riser variation

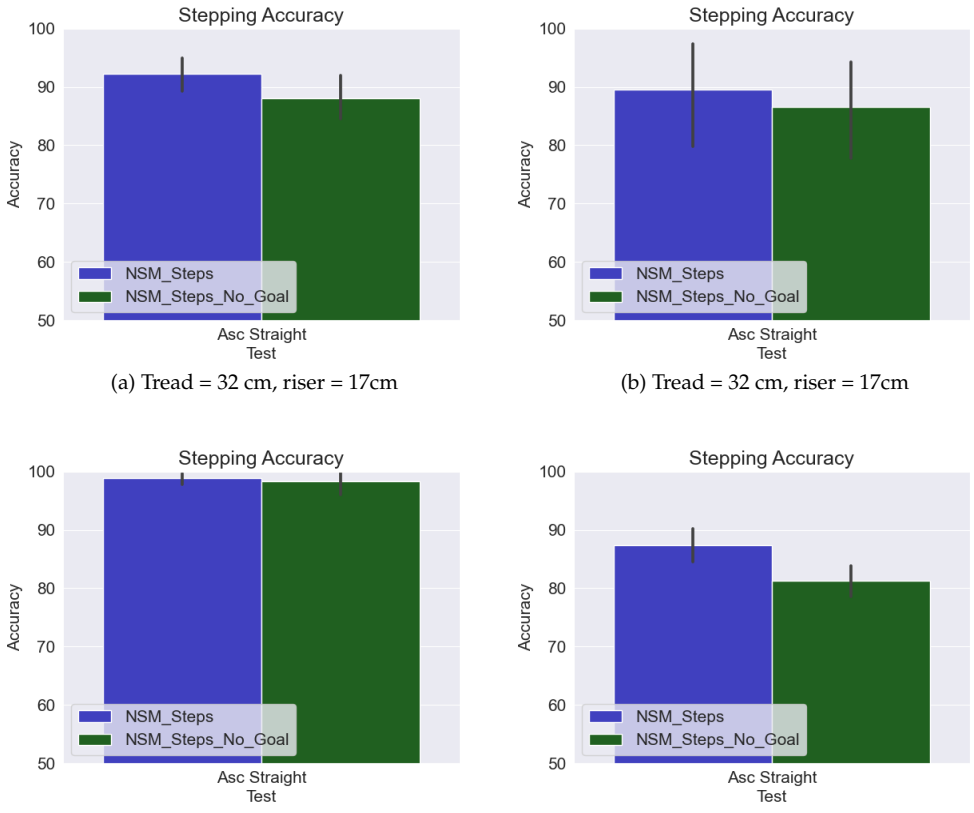


(e) Tread variation



(f) Riser variation

Figure 5.9: Results of *ascending straight* test scenario for the ablation study. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in appendix B in figures B.17, B.18 and B.19.



(a) Tread = 32 cm, riser = 17cm

(b) Tread = 32 cm, riser = 17cm

(c) Tread = 35 cm, riser = 17cm

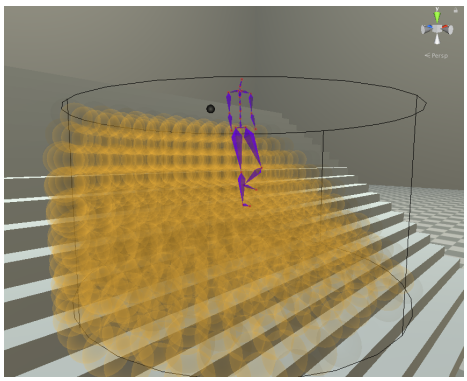
(d) Tread = 35 cm, riser = 17cm

Figure 5.10: Results of the stepping accuracy for the footstep goal correction in large tread sizes (higher is better). Left: descending straight scenario. Right: ascending straight scenario. Top: 32x17 step sizes. Bottom: 35x17 step sizes. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

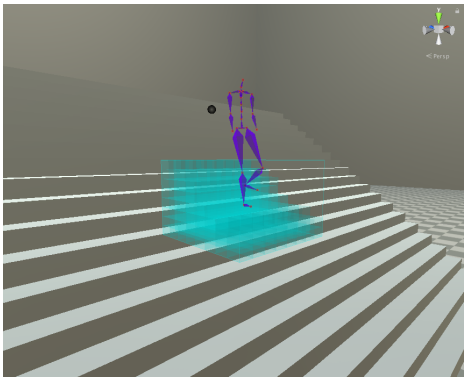
5.5.3 Influence of the Heightmap Sensor

The original NSM model has two sensors to describe the character's environment, namely the environment sensor and the interaction sensor. Both of them are volumetric sensors, meaning that they sense throughout a three dimensional shape. In the case of the environment sensor, a cylindrical shape as seen in figure 5.11a; in the case of the interaction sensor, a cubic shape as seen in figure 5.11b.

For the specific task of stair walking, the most important is to know the shape of the stairs so that the character can act accordingly. Thus, there is no need to sense inside or above the stairs, otherwise many samples are wasted. That problem is avoided by using the heightmap sensor, since the samples taken are only those over the treads of the stairs. This brings to advantages in memory requirements of the training data, because as it was mentioned in chapter 4, the data is generated per frame and stored for later training, thus having a smaller number of samples directly impacts the size of the training data. The data generated for NSM_Steps weights 356.3 MB, whereas the size of the data for the



(a) Environment sensor

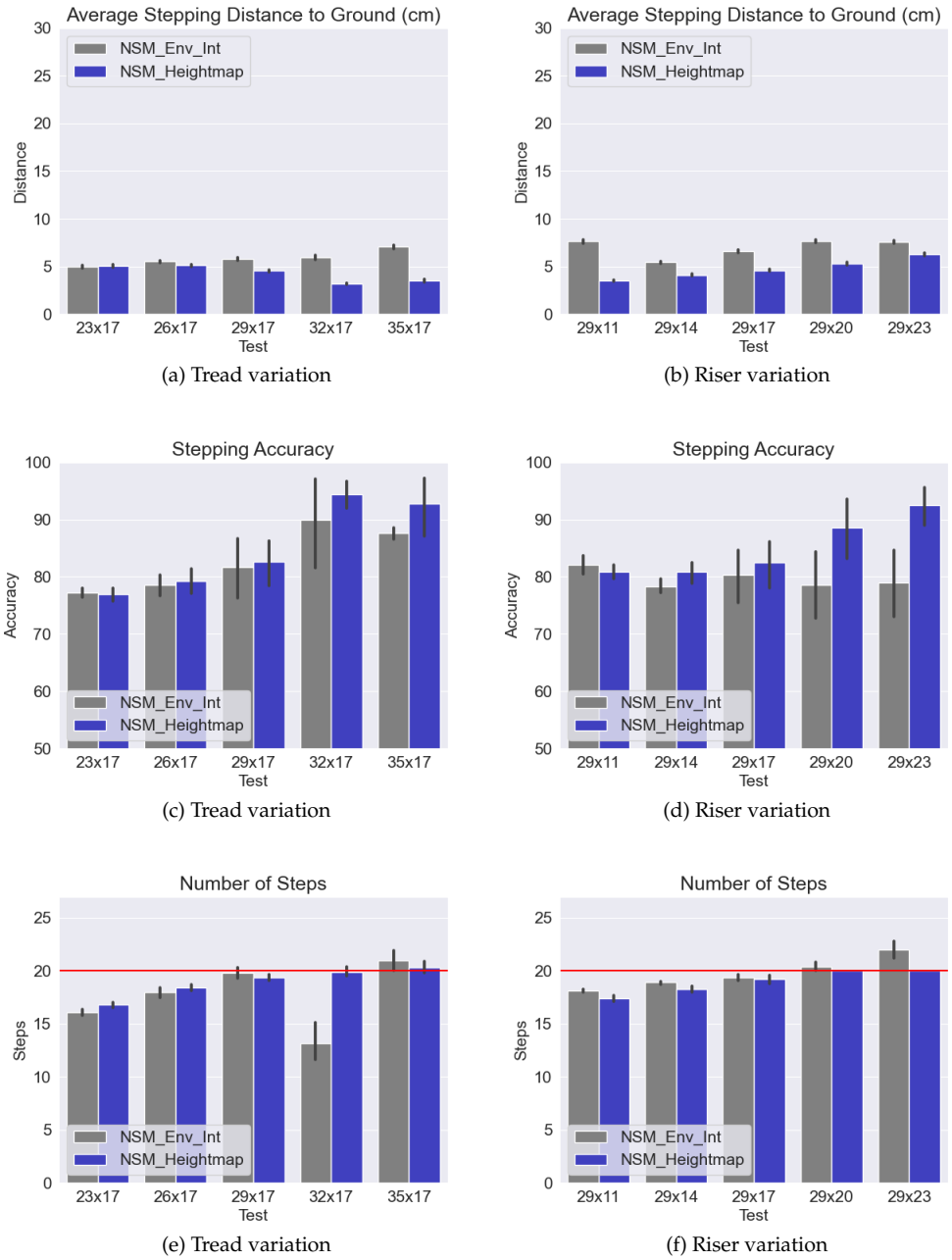


(b) Interaction sensor

Figure 5.11: Original volumetric sensors of the NSM. Left: environment sensor. Right: Interaction sensor.

NSM is 1.0 GB (that is approximately a 65% relative improvement). Also, NSM_Steps uses fewer encoder networks, which yields a more compact trained model with a size of 52.1 MB (NSM has a size of 71 MB, that is almost a 30% relative improvement).

In order to compare the performance between the heightmap sensor and the original sensors, NSM_Steps was compared with the same method but using the original sensors instead. In this section, NSM_Steps is going to be referred as NSM_Heightmap. Also, NSM_Steps with the original environment and interaction sensors is going to be referred as NSM_Env_Int. The results of the metrics in the descending straight scenario are shown in figure 5.12. Surprisingly, there is a clear advantage in all metrics just by switching to the heightmap sensor. The results for all the other test scenarios can be found in appendix B (figures B.28, B.29, B.30).



(a) Tread variation

(b) Riser variation

(c) Tread variation

(d) Riser variation

(e) Tread variation

(f) Riser variation

Figure 5.12: Results of *descending straight* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

Chapter 6
Conclusion

6.1 Discussion

In this thesis, a model to synthesize natural stair walking for character control was presented. This was achieved by sensing the surroundings of a virtual character and also by correcting the inference results of the footstep and the foot trajectories predicted by the network model.

The results of the evaluation showed that the proposed changes have an positive impact on the metrics. The foot trajectory correction has a strong influence on the final result keeping the character close to the treads of the stairs and reducing by a large margin the unwanted bouncing artifacts. PFNN and NSM yield to undesired artifacts: PFNN sinks the feet of the character into the stairs every time it takes a step and NSM is unstable and bounces up and down. The footstep goal correction improves the accuracy of the foot placement over longer animation sequences in comparison with NSM. The heightmap sensor reduces the memory requirements since it requires less data to have a good description of the surrounding ground. Also, less data yields to a more compact trained model. And on top of that, the performance might even improve for some tests. For the tests where the performance does not improve, it is still comparable. This might be due to the large amount of inputs given by the original sensors, which makes it harder for the network to learn the desired locomotions.

The presented method is suitable for real-time applications, e.g. video games. As it was reported in section 5.4, the overall framework runs at approximately 30 frames per second. The proposed correction parts take approximately 0.1 ms to run, which is not much compared to the 5 ms it takes to compute the animation.

Some important point that should be mentioned is that in the original NSM there is a velocity parameter that needs to be set depending on the action that the character is going to perform and it is not included as a learned parameter. There is no explanation

as how this value is chosen in the original work. Thus, in this thesis some velocity values were captured heuristically (using the NSM_Steps method) by traversing the stairs using different tread sizes, and then this captured data was used to fit a velocity curve. The created velocity profile was used in both NSM_Steps and NSM during runtime.

Another important observation is that, while testing the NSM_Steps and NSM methods, it was observed that the models performed better by decreasing the velocity going upstairs. The reason for this behavior is likely to be that in the training data, going upstairs is slower than going downstairs (approximately 25% slower), meaning that the network captured this as well.

Another observation is that, when comparing the descending and ascending test scenarios, the first ones usually perform better than the latter ones. This observation is more obvious in the ablation study, where the results of the metrics for the model without any kind of correction was still comparable to the others while descending stairs. This could mean that going downstairs is an easier task than going upstairs. While ascending the stairs, the estimated trajectory of the feet usually ended up slightly above the treads, and if this was not corrected for the next iterations the error kept accumulating and the pose of the character also shifted with it ending up way above the stairs. On the other hand, the trajectory estimation while descending was generally comparable with the methods that use corrections.

When it comes to the test scenarios varying tread sizes, all methods usually perform better for the largest sizes. One possible reason of this behaviour is that it is easier for the network to deal with large tread sizes. This also might have a relation with the density of the heightmap, because the smaller the treads, the less samples per tread are taken. This is true for all methods, since all of them use some sort of sensor to describe the ground.

In the *stepping distance to ground* metric, NSM commonly performs better with the tread of 29 cm and the riser of 17 cm, which is actually where the data variation was centered and also the original size of the stairs where the data was recorded. For NSM_Steps this is not the case, performing consistently for all tread sizes.

PFNN usually preforms better than NSM_Steps and NSM methods in the *stepping accuracy* metric and the reason for that is most likely to be that the original skeleton of PFNN was used for the tests. This skeleton differs from the one used in NSM and NSM_Steps, and has smaller feet (the area of the foot projection grid is smaller), thus making it more likely to land the feet correctly over the treads of the stairs.

In the *number of steps* metric, PFNN is the worst among all methods. This is because it was not really trained for normal stair walking, but for uneven terrain walking. The reason for including the original PFNN in the evaluation is that since the original NSM performs better than PFNN (as it was reported in [44]) in this thesis it was more interesting to compare NSM_Steps against a model that has a considerable different training size (PFNN: 1 hour @ 60 fps, NSM_Steps: approximately 5 min @ 60 fps). Surprisingly, NSM_Steps has even a better performance in the *distance to ground metric* than PFNN. Also, NSM_Steps performs visually good in both normal stairs and uneven terrain, whereas PFNN struggles to offer a nice stair walking synthesis even with larger training data. These results were omitted because they were out of the scope of the current work.

6.2 Limitations

The most obvious limitation of the method presented in this thesis is the footstep placement, because even though the accuracy improves over the original NSM, it is still not precise enough to place the feet of the character exactly on the goals. The footstep correction helps just slightly and its influence is not as strong as expected. This is not good for applications were precise foot placement is a must, like the so called *stepping stones* problem, where a virtual character should traverse the environment placing its feet exactly in the desired locations. For example, if a character is supposed to cross a river by placing its feet on dry spots (stones).

Another limitation would be to determine the velocity parameter depending on the tread size, which can be somewhat tedious. For example, as it was already mentioned, the velocity was chosen heuristically by changing the velocity value while traversing the stairs with a fixed tread size. Once a good velocity value was found, it was recorded. This procedure was repeated for all five selected tread sizes (ranging from 23 cm to 35 cm). Finally, all the recorded velocity values were used to fit a curve, which is used during runtime. However, this problem is inherent from the original NSM model.

Also, as it was mentioned in section 3.3.3, there is a distance parameter that was tuned. Adding parameters always adds complexity, and also the chosen value might not work for other purposes. Nevertheless, the selected parameter seems to work fine for different stair sizes, as shown in the evaluation.

6.3 Future Work

There are different directions that could be explored based on the results. One direction could be the extension of the method presented in this thesis by including the interaction with handrails. Adding handrail interaction can increase the realism of the synthesized motion even more. This extension could be potentially straightforward because the same principle that was used for the feet could very likely be used for the hands as well. First of all, some data including handrail interaction should be recorded. Then, this recorded data would have to be processed in the same way as with the feet: the data has to be labeled and it would be probably enough to keep the same ascending and descending labels for the handrail interaction as well. For the hands, only the future hand trajectory should be taken into account for the estimation. The hand goal can be discarded because that would be non trivial to define a goal in the runtime setting. Also, it would be useful to add some sort of variation to the handrails so that the model learns to reach handrails at different positions from the ground and with different arm extensions. The only useful kind of variation that should be applied to the handrails would be translation and some length scaling depending on the size of the stairs, since the handrail should cover the entire length of the stairs. Changing the width scale of a handrail does not make sense because it can not vary much, e.g. it would not be a noticeable difference to vary the width of the handrail because the range of the variation would be very small. Once the variation on the handrails is applied, it would be necessary to just export the training data and use it to train a new model. In the runtime script in Unity, it would not make much sense to set a *desired* hand goal position because it is not as straightforward as with

the feet (the handrail has no separations like the treads of a staircase, it is completely flat), thus it would only be possible to perform a hand trajectory correction. This hand correction should be enough to keep the hands of the character close enough to the handrail improving the overall quality of the synthesized motion.

Another part that should be explored is related to the used sensor. In the evaluation, it was shown that the heightmap sensor reduces the memory requirements and offers some performance boost compared to the volumetric sensors utilized in NSM. However, there is still no answer as to how much effect the density of points in the heightmap has over the final outcome. Thus, it could be insightful to test the different scenarios with different variations in the density of samples of the heightmap. This sample density variation could show if there is an influence on the foot placement accuracy. Also, this could show how many samples are really needed to have a decent motion quality for stair walking.

The proposed method in this thesis was based upon NSM, improving the results w.r.t. the original model. Hence, the proposed method should be analyzed with a different model in order to see its behavior in different settings. In other words, another method should be extended using the required inputs and outputs and the foot placement goal correction and foot trajectory correction should be applied as well.

Another aspect that should be explored is the impact of the training data on the final result. It is still not clear whether more data can improve the results of the overall synthesized motion. Also, if the performance improves, it would be useful to know if the trade-off between training data size and performance boost really pays off.

In this thesis, only tests with homogeneous step sizes were performed. It would be interesting to check the behavior of the method if also treads are skipped during training in order to see if the method could actually overstep small stairs in a natural way during runtime.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). `https://www.tensorflow.org/` Software available from tensorflow.org.

[2] Norman I. Badler and Hyeongseok Ko. 1996. Animating Human Locomotion with Inverse Dynamics. *IEEE Computer Graphics and Application* 6, 2 (1996), 50–59. `DOI:http://dx.doi.org/10.1109/38.486680`

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *CoRR* abs/1803.01271 (2018). `http://arxiv.org/abs/1803.01271`

[4] Alejandro Beacco, Nuria Pelechano, Mubbasir Kapadia, and Norman I. Badler. 2015. Footstep parameterized motion blending using barycentric coordinates. *Computers & Graphics* 47 (2015), 105–112. `DOI:http://dx.doi.org/https://doi.org/10.1016/j.cag.2014.12.004`

[5] Andrew Beattie. 2021. How the Video Game Industry Is Changing. (2021). `https://www.investopedia.com/articles/investing/053115/how-video-game-industry-changing.asp` , last accessed on 26/10/2021.

[6] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 206 (Nov. 2019), 11 pages. `DOI:http://dx.doi.org/10.1145/3355089.3356536`

[7] Ronan Boulic, Ramon Mas, and Daniel Thalmann. 1996. A robust approach for the control of the center of mass with inverse kinetics. *Computers & Graphics* 20, 5 (1996), 693–701. `DOI:http://dx.doi.org/https://doi.org/10.1016/S0097-8493(96)00043-X`

[8] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[9] Michael Büttner and Simon Clavet. 2015. Motion Matching and The Road to Next Gen Animation. In *Proc. of Nucl.ai*. `https://www.youtube.com/watch?v=z_wpgHFSWss&t=658s` , last accessed on 30/04/2021.

[10] Xiaobin Chang, Timothy M. Hospedales, and Tao Xiang. 2018. Multi-Level Factorisation Net for Person Re-Identification. *CoRR* abs/1803.09132 (2018). http://arxiv.org/abs/1803.09132

[11] Shih-Kai Chung and J.K. Hahn. 1999. Animation of human walking in virtual environments. In *Proceedings Computer Animation 1999*. 4–15. DOI:http://dx.doi.org/10.1109/CA.1999.781194

[12] Stelian Coros, Philippe Beaudoin, Kang Kang Yin, and Michiel van de Panne. 2008. Synthesis of Constrained Walking Skills *(SIGGRAPH Asia '08)*. Association for Computing Machinery, New York, NY, USA, Article 113, 9 pages. DOI:http://dx.doi.org/10.1145/1457515.1409066

[13] Arjan Egges and Ben van Basten. 2010. One step at a time: animating virtual characters based on foot placement. *The Visual Computer* 26, 6 (01 Jun 2010), 497–503. DOI:http://dx.doi.org/10.1007/s00371-010-0443-0

[14] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. 2013. Learning Factored Representations in a Deep Mixture of Experts. (12 2013). http://arxiv.org/abs/1312.4314

[15] Meg Escott. 2021. Stair parts. (2021). https://www.houseplanshelper.com/stair-parts.html , last accessed on 20/02/2021.

[16] Federal Institute for Occupational Safety and Health. 2021. The Design of safe Stairs. (2021). https://www.baua.de/EN/Topics/Work-design/Workplaces/Safe-stairways/The-design-of-safe-stairs.html , last accessed on 20/02/2021.

[17] The Institute for Statistics Education. 2021. Latent variable. (2021). https://www.statistics.com/glossary/latent-variable/ , last accessed on 10/03/2021.

[18] Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. 2015. Recurrent Network Models for Kinematic Tracking. *CoRR* abs/1508.00271 (2015). http://arxiv.org/abs/1508.00271

[19] Frostbite Gaming. 2020. I Found a Secret Stairway to Heaven with Techno Gamerz in GTA 5. (2020). https://www.youtube.com/watch?v=1DcdJLrgcOE , last accessed on 10/02/2021.

[20] Bob Givan and Ron Parr. 2001. An Introduction to Markov Decision Processes. (2001). https://www.cs.rice.edu/~vardi/dag01/givan1.pdf , last accessed on 20/03/2021.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[22] Rachel Heck and Michael Gleicher. 2007. Parametric Motion Graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*. Association for Computing Machinery, New York, NY, USA, 129–136. DOI:http://dx.doi.org/10.1145/1230100.1230123

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation* 9 (12 1997), 1735–80. DOI:http://dx.doi.org/10.1162/neco.1997.9.8.1735

[24] Daniel Holden. 2018. Robust Solving of Optical Motion Capture Data by Denoising. *ACM Trans. Graph.* 37, 4, Article 165 (July 2018), 12 pages. DOI:http://dx.doi.org/10.1145/3197517.3201302

[25] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Trans. Graph.* 39, 4, Article 53 (July 2020), 13 pages. DOI:http://dx.doi.org/10.1145/3386569.3392440

[26] Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-Functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (July 2017), 13 pages. DOI:http://dx.doi.org/10.1145/3072959.3073663

[27] Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4, Article 138 (July 2016), 11 pages. DOI:http://dx.doi.org/10.1145/2897824.2925975

[28] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Computation* 3, 1 (1991), 79–87. DOI:http://dx.doi.org/10.1162/neco.1991.3.1.79

[29] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[30] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. *ACM Trans. Graph.* 21, 3 (July 2002), 473–482. DOI:http://dx.doi.org/10.1145/566654.566605

[31] Jehee Lee and Kang Hoon Lee. 2006. Precomputing avatar behavior from human motion data. *Graphical Models* 68, 2 (2006), 158–174. DOI:http://dx.doi.org/https://doi.org/10.1016/j.gmod.2005.03.004

[32] Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-Objective Control. *ACM Trans. Graph.* 37, 6, Article 180 (Dec. 2018), 10 pages. DOI:http://dx.doi.org/10.1145/3272127.3275071

[33] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010. Motion Fields for Interactive Character Locomotion. *ACM Trans. Graph.* 29, 6, Article 138 (Dec. 2010), 8 pages. DOI:http://dx.doi.org/10.1145/1882261.1866160

[34] Fei-Fei Li and Andrej Karpathy. 2016. CS231n: Convolutional Neural Networks for Visual Recognition. (2016). http://cs231n.stanford.edu/ , last accessed on 01/03/2021.

[35] Ilya Loshchilov and Frank Hutter. 2017. Fixing Weight Decay Regularization in Adam. *CoRR* abs/1711.05101 (2017). http://arxiv.org/abs/1711.05101

[36] Franck Multon, Laure France, Marie-Paule Cani-Gascuel, and Giles Debunne. 1999. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation* 10, 1 (1999), 39–54. DOI:http://dx.doi.org/https://doi.org/10.1002/(SICI)1099-1778(199901/03)10:1<39::AID-VIS195>3.0.CO;2-2

[37] Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. Learning Predict-and-Simulate Policies from Unorganized Human Motion Data. *ACM Trans. Graph.* 38, 6, Article 205 (Nov. 2019), 11 pages. DOI:http://dx.doi.org/10.1145/3355089.3356501

[38] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Trans. Graph.* 37, 4, Article 143 (July 2018), 14 pages. DOI: http://dx.doi.org/10.1145/3197517.3201311

[39] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.* 36, 4, Article 41 (July 2017), 13 pages. DOI:http://dx.doi.org/10.1145/3072959.3073602

[40] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. 2016. Variational Autoencoder for Deep Learning of Images, Labels and Captions. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/eb86d510361fc23b59f18c1bc9802cc6-Paper.pdf

[41] Rehab and Revive. 2020. Climb Down Stairs Correctly | How to Descend Stairs | Physical Therapy. (2020). https://www.youtube.com/watch?v=YpLHb3LIGmA&t=78s , last accessed on 20/10/2021.

[42] Paul Rodriguez, Janet Wiles, and Jeffrey L. ELMAN. 1999. A Recurrent Neural Network that Learns to Count. *Connection Science* 11, 1 (1999), 5–40. DOI:http://dx.doi.org/10.1080/095400999116340

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[44] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural State Machine for Character-Scene Interactions. *ACM Trans. Graph.* 38, 6, Article 209 (Nov. 2019), 14 pages. DOI:http://dx.doi.org/10.1145/3355089.3356505

[45] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Trans. Graph.* 39, 4, Article 54 (July 2020), 14 pages. DOI:http://dx.doi.org/10.1145/3386569.3392450

[46] Sebastian Starke, Yiwei Zhao, Fabio Zinno, and Taku Komura. 2021. Neural Animation Layering for Synthesizing Martial Arts Movements. *ACM Trans. Graph.* 40, 4, Article 92 (July 2021), 16 pages. DOI:http://dx.doi.org/10.1145/3450626.3459881

[47] Statista. 2021. Video gaming market size worldwide 2020-2025. (2021). https://www.statista.com/statistics/292056/video-game-market-value-worldwide/ , last accessed on 26/10/2021.

[48] Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-Optimal Character Animation with Continuous Control. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 7–es. DOI:http://dx.doi.org/10.1145/1275808.1276386

[49] Ben J. H. van Basten, P. W. A. M. Peeters, and Arjan Egges. 2010. The step space: example-based footprint-driven motion synthesis. *Comput. Animat. Virtual Worlds* 21, 3-4 (2010), 433–441. DOI:http://dx.doi.org/10.1002/cav.342

[50] Ben J. H. van Basten, Sybren A. Stüvel, and Arjan Egges. 2011. A hybrid interpolation scheme for footprint-driven walking synthesis. In *Proceedings of the Graphics Interface 2011 Conference, May 25-27, 2011, St. John's, Newfoundland, Canada*, Stephen Brooks and Pourang Irani (Eds.). Canadian Human-Computer Communications Society, 9–16. http://portal.acm.org/citation.cfm?id=1992920&CFID=25874310&CFTOKEN=65321882

[51] Michiel Van De Panne. 1997. From Footprints to Animation. *Computer Graphics Forum* 16, 4 (1997), 211–223. DOI:http://dx.doi.org/10.1111/1467-8659.00181

[52] Ronald J. Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Comput.* 1, 2 (June 1989), 270–280. DOI:http://dx.doi.org/10.1162/neco.1989.1.2.270

[53] Andrew Witkin and Zoran Popovic. 1995. Motion Warping. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. Association for Computing Machinery, New York, NY, USA, 105–108. DOI:http://dx.doi.org/10.1145/218380.218422

[54] XSense. 2021. MVN Analyze. (2021). https://www.xsens.com/products/mtw-awinda , last accessed on 05/05/2021.

[55] Kristjan Zadziuk. 2016. The Future of Gameplay Animation... Today. (2016). https://www.youtube.com/watch?v=KSTn3ePDt50&t=627s&ab_channel=KristjanZadziuk , last accessed on 20/02/2021.

[56] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. Graph.* 37, 4, Article 145 (July 2018), 11 pages. DOI:http://dx.doi.org/10.1145/3197517.3201366

Appendix A
Dance Cards

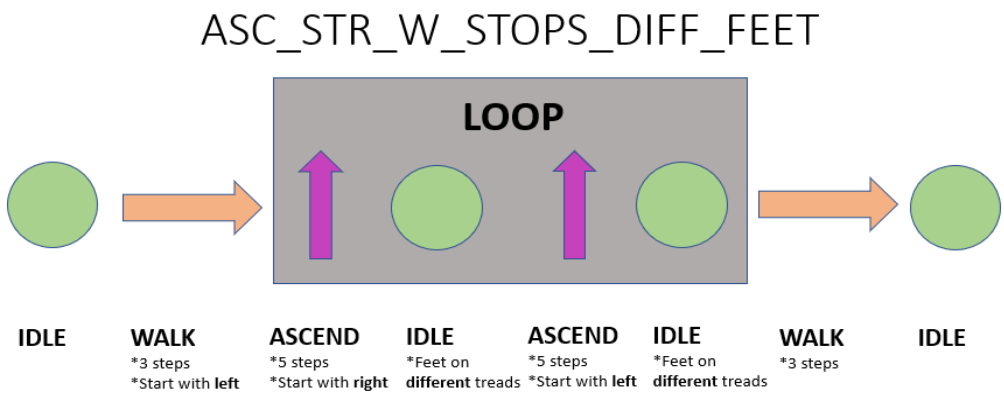


Figure A.1: Ascend straight with stops placing feet on different treads.

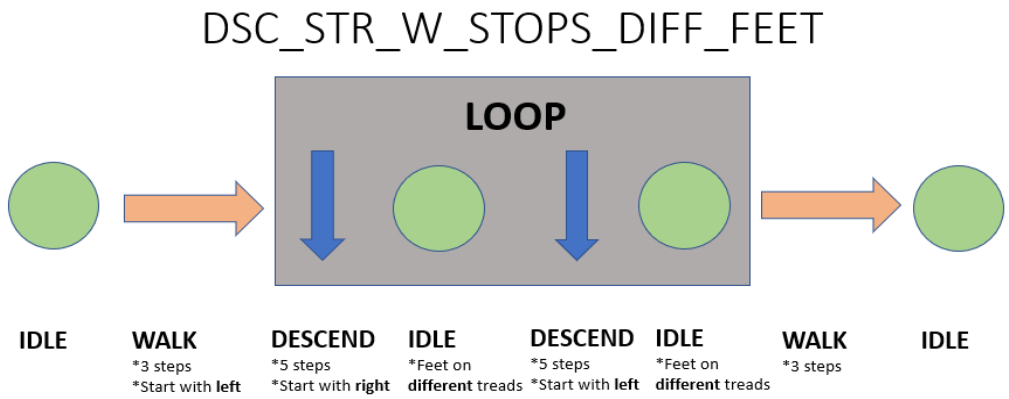


Figure A.2: Descend straight with stops placing feet on different treads.

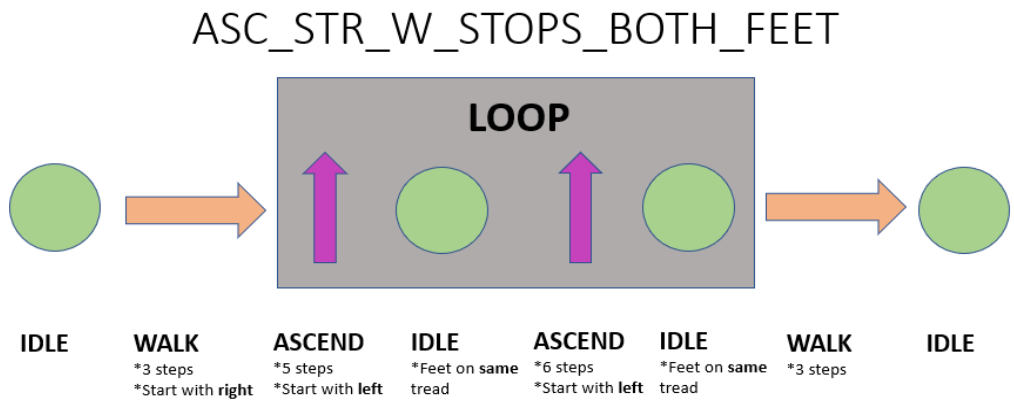


Figure A.3: Ascend straight with stops placing feet on same tread.

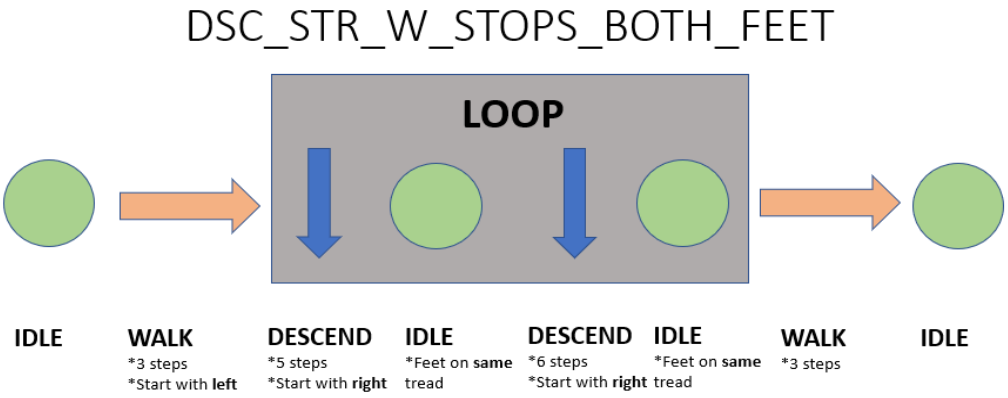


Figure A.4: Descend straight with stops placing feet on same tread.

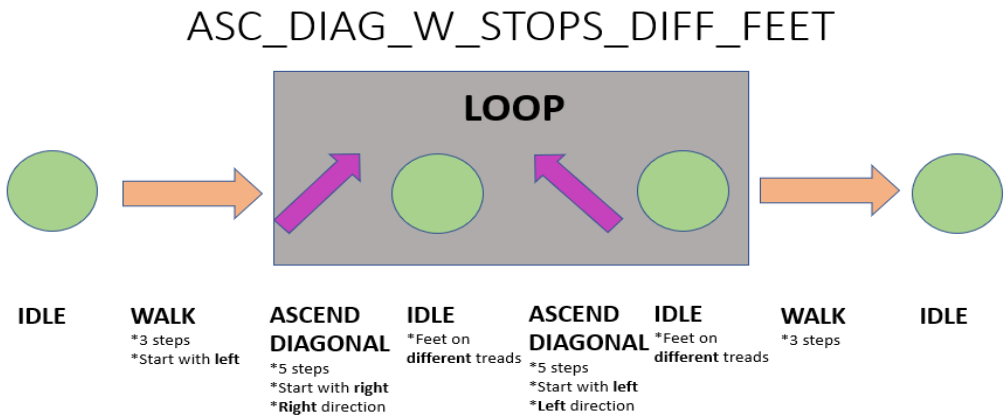


Figure A.5: Ascend diagonally with stops placing feet on different treads.

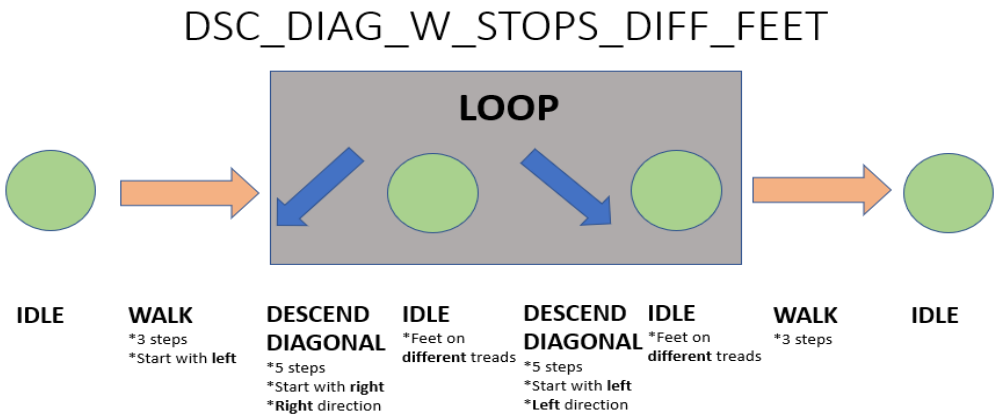


Figure A.6: Descend diagonally with stops placing feet on different treads.

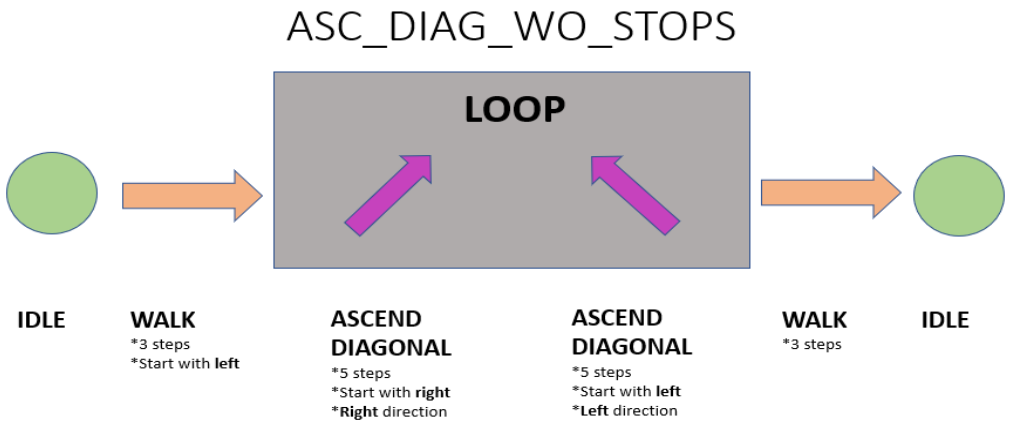


Figure A.7: Ascend diagonally without stops.

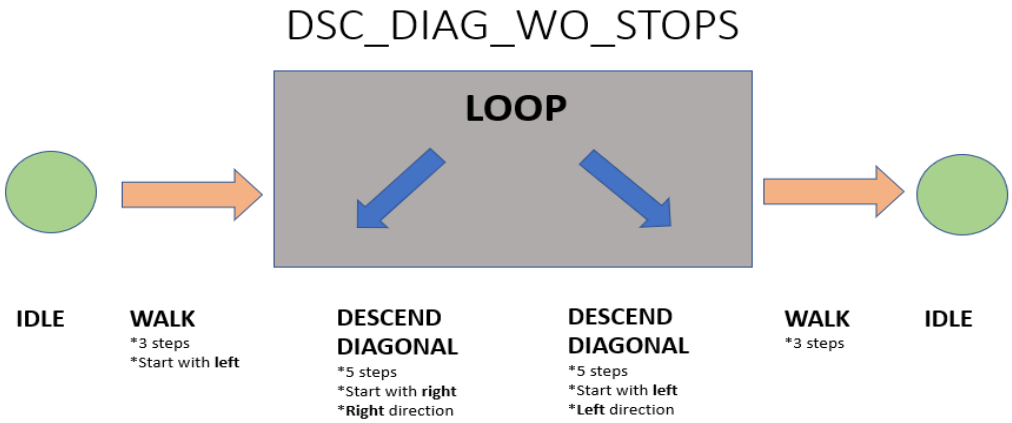


Figure A.8: Descend diagonally without stops.

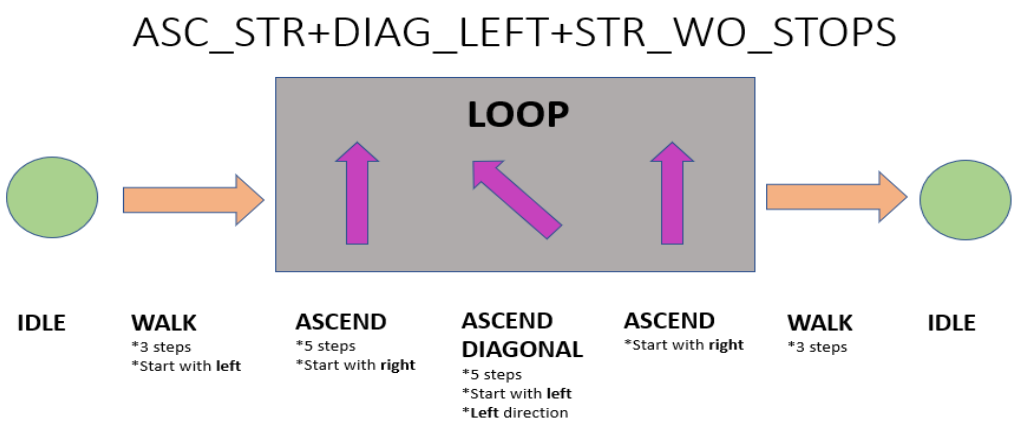


Figure A.9: Ascend straight + diagonally left + straight without stops.

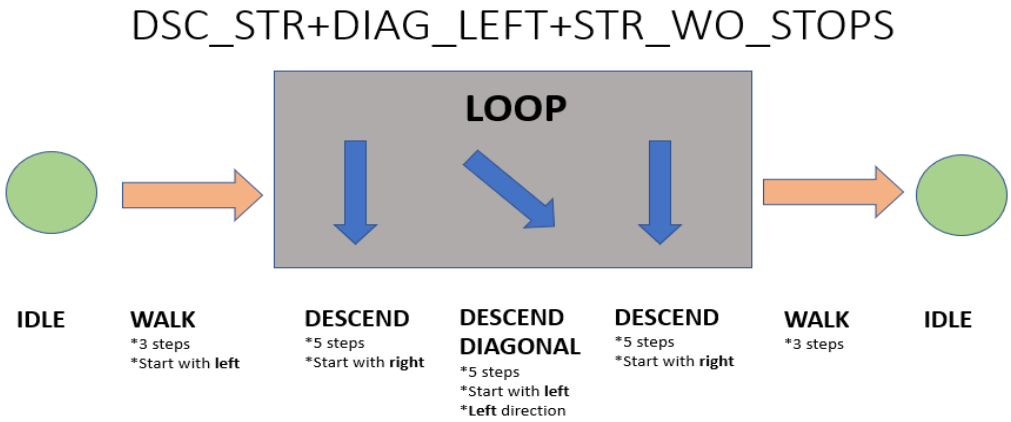


Figure A.10: Descend straight + diagonally left + straight without stops.

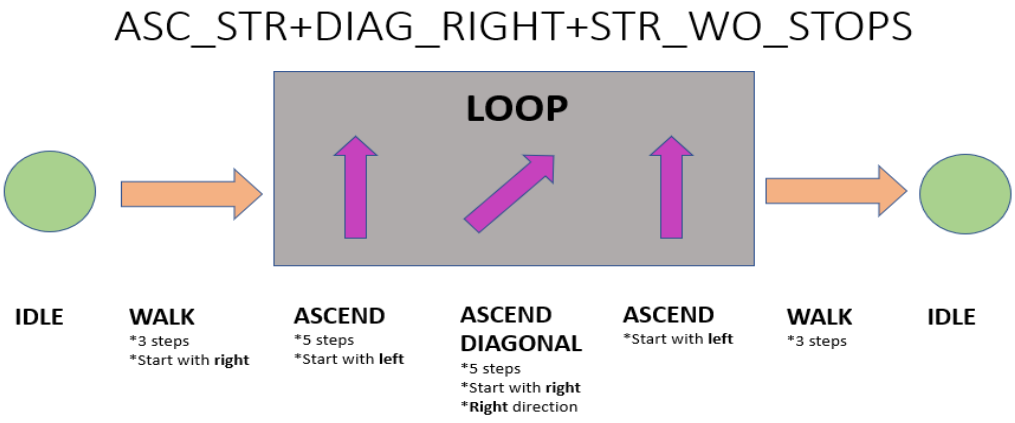


Figure A.11: Ascend straight + diagonally right + straight without stops.

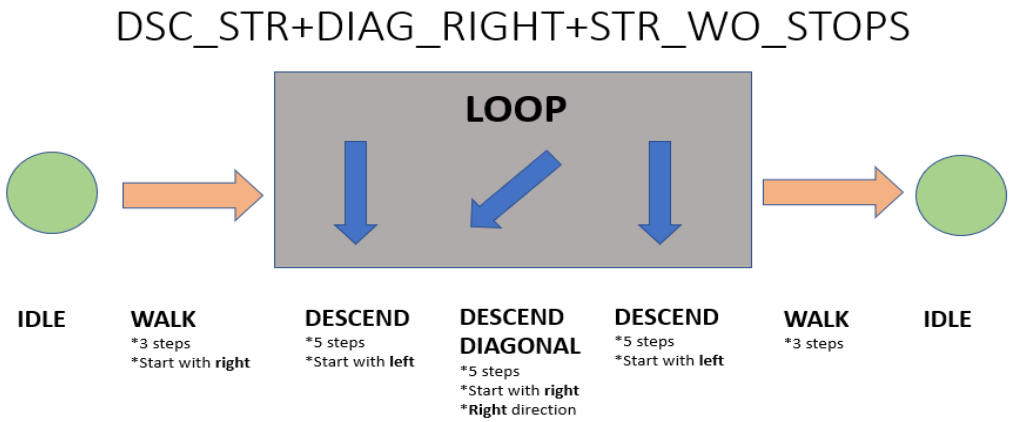
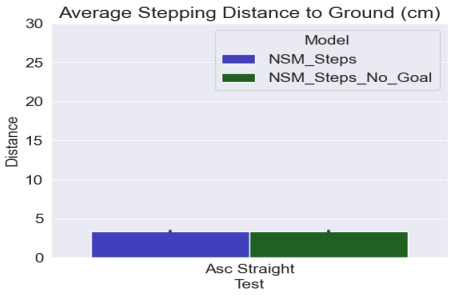
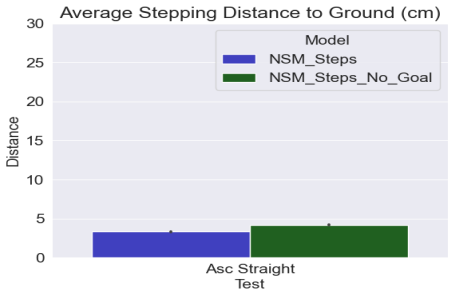


Figure A.12: Descend straight + diagonally right + straight without stops.

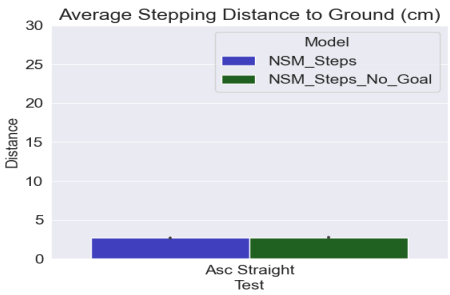
Appendix B
Additional Evaluation Data



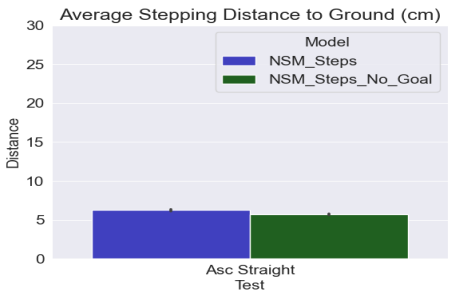
(a) Tread = 32 cm, riser = 17cm



(b) Tread = 32 cm, riser = 17cm

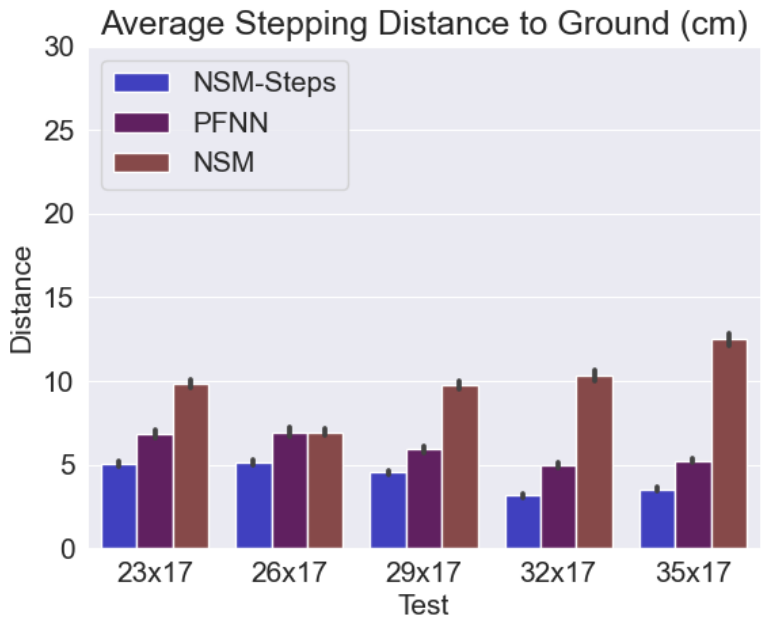


(c) Tread = 35 cm, riser = 17cm

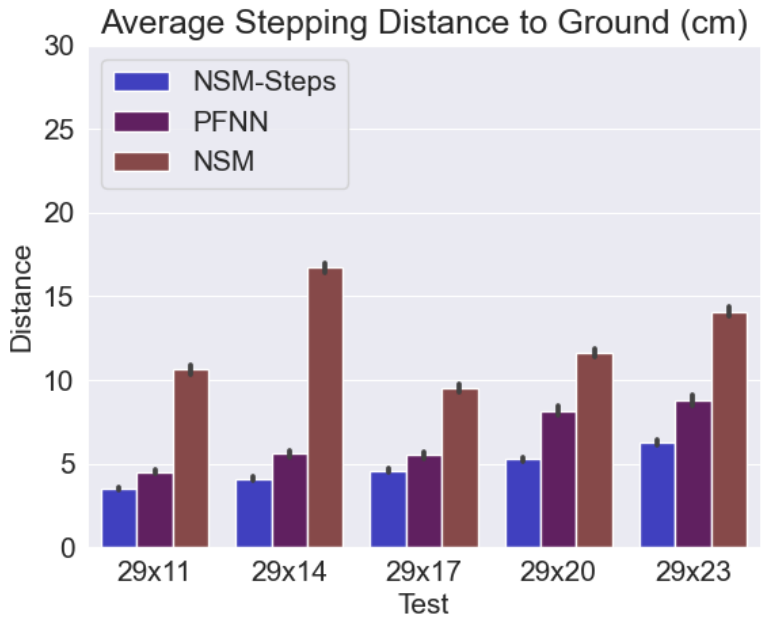


(d) Tread = 35 cm, riser = 17cm

Figure B.1: Results of the stepping distance to ground in a 40 steps staircase for the footstep goal correction in large tread sizes (higher is better). Left: descending straight scenario. Right: ascending straight scenario. Top: 32x17 step sizes. Bottom: 35x17 step sizes. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

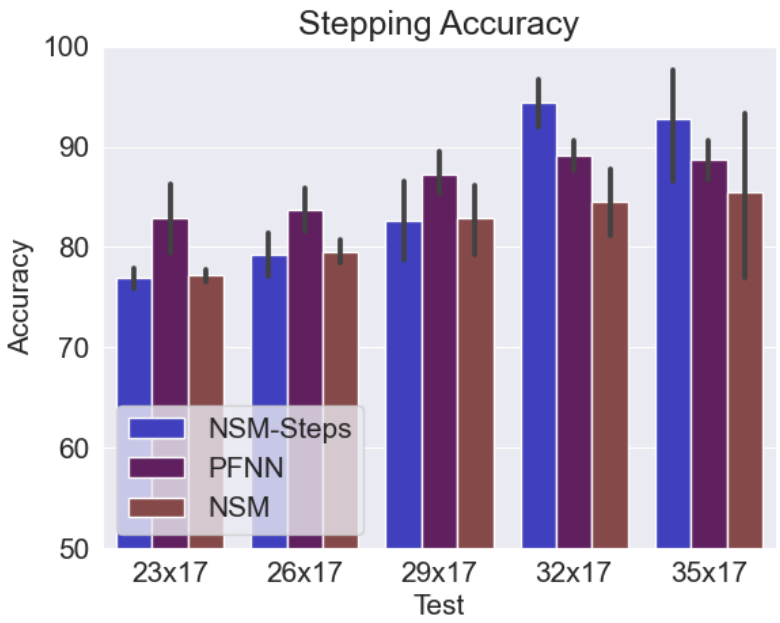


(a) Tread variation

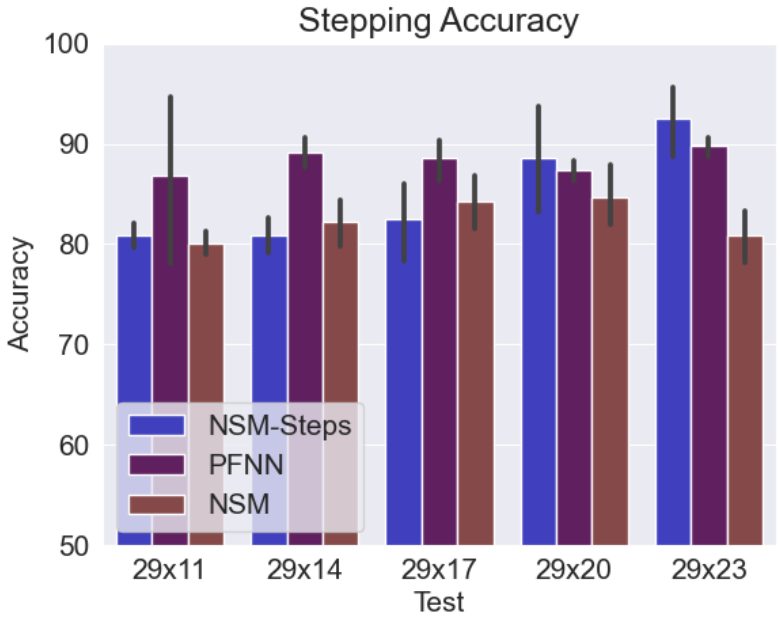


(b) Riser variation

Figure B.2: Average stepping distance to ground in cm (lower is better) of the *descending straight* test scenario. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

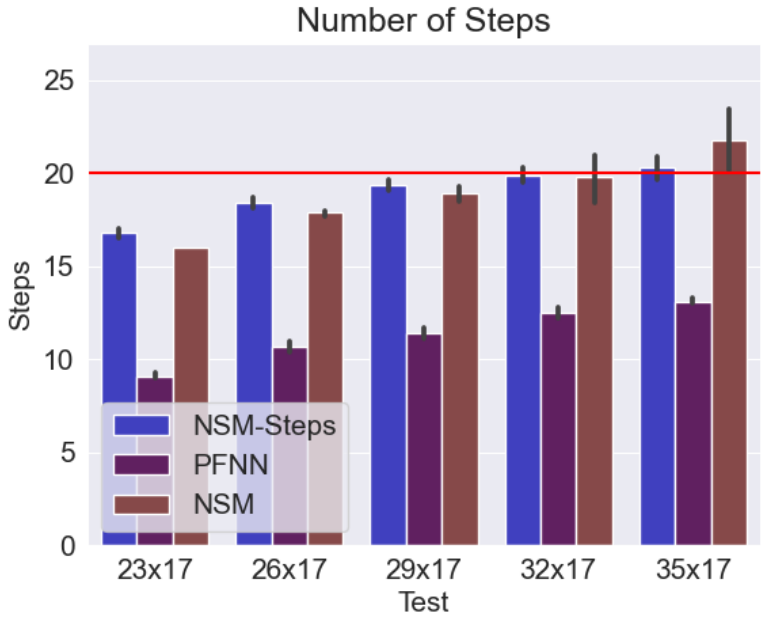


(a) Tread variation

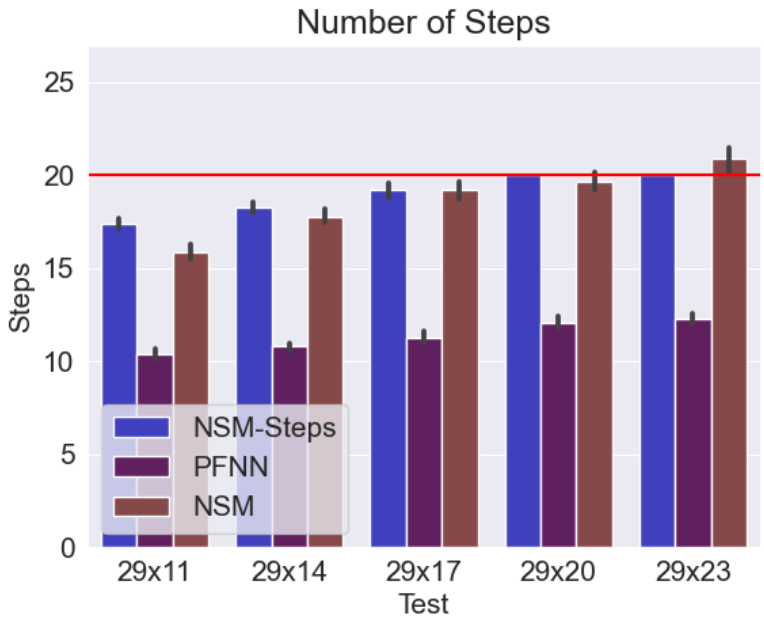


(b) Riser variation

Figure B.3: Stepping accuracy percentage (higher is better) of the *descending straight* test scenario. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

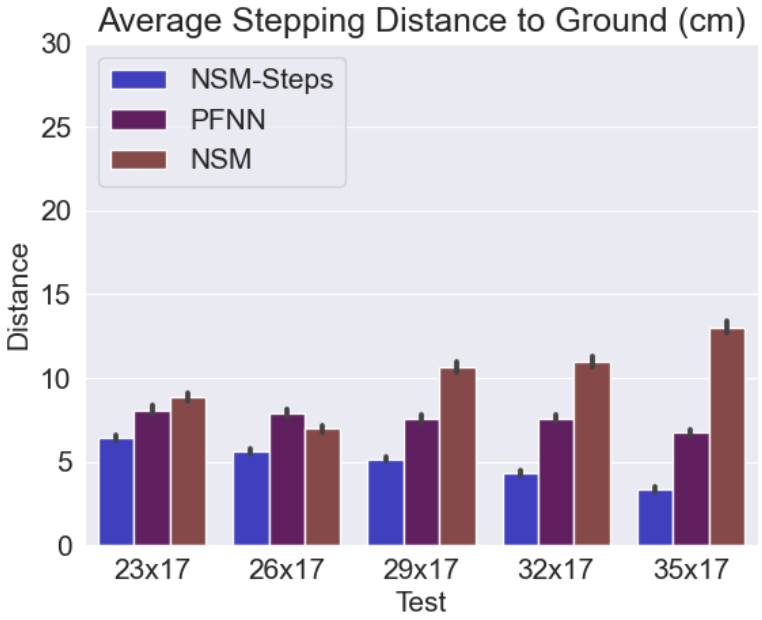


(a) Tread variation

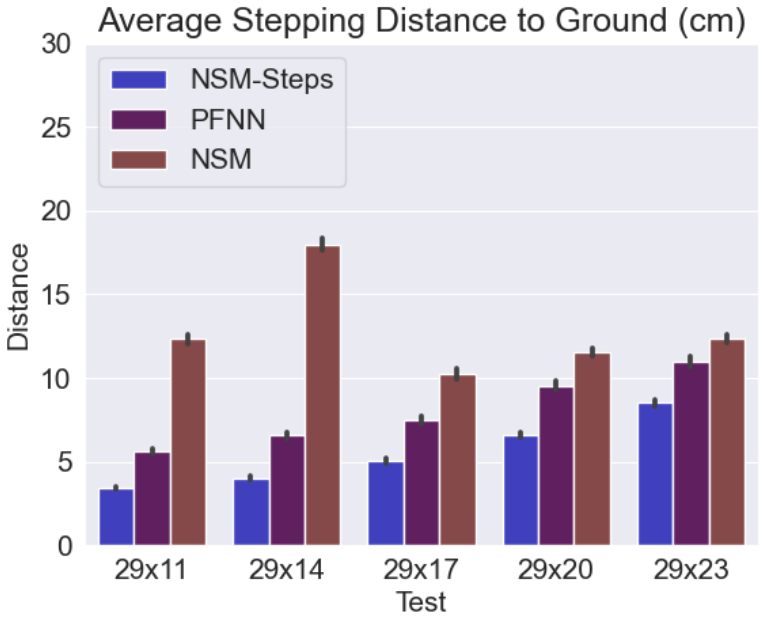


(b) Riser variation

Figure B.4: Number of steps taken of the *descending straight* test scenario. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better.The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

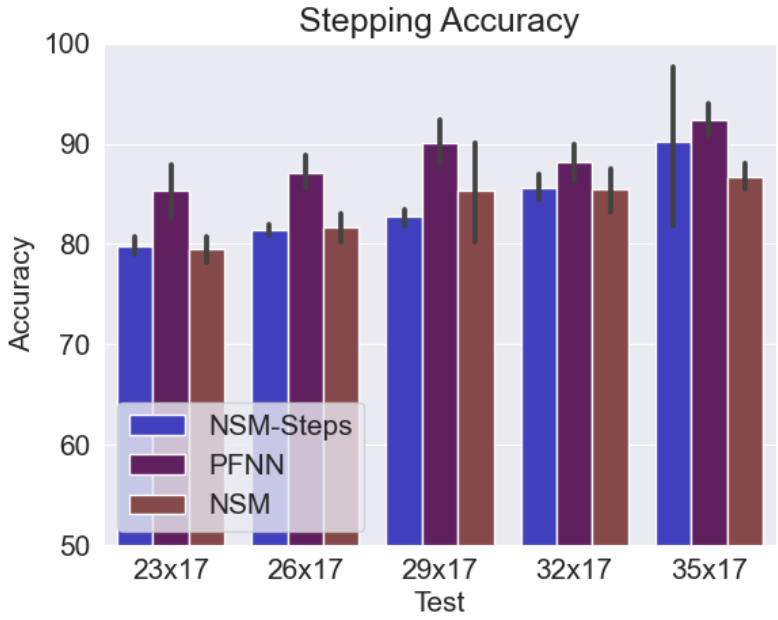


(a) Tread variation

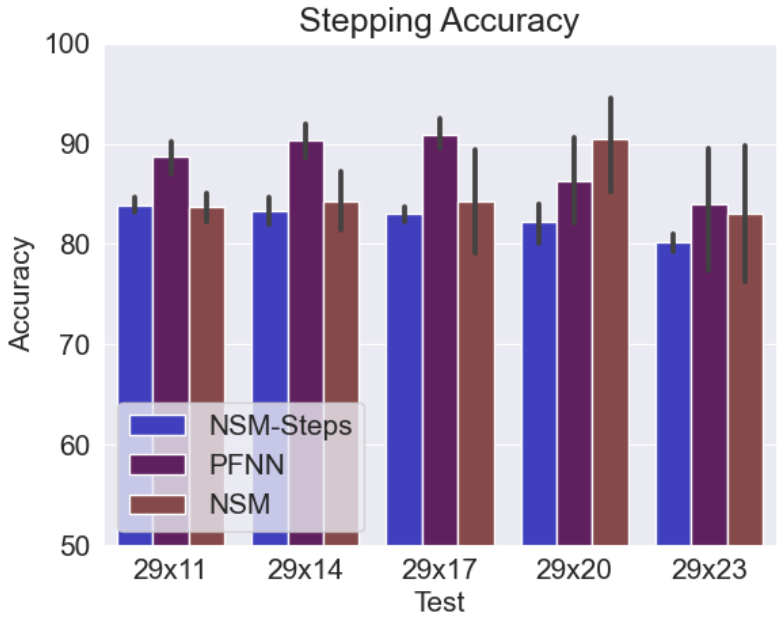


(b) Riser variation

Figure B.5: Average stepping distance to ground in cm (lower is better) of the *descending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

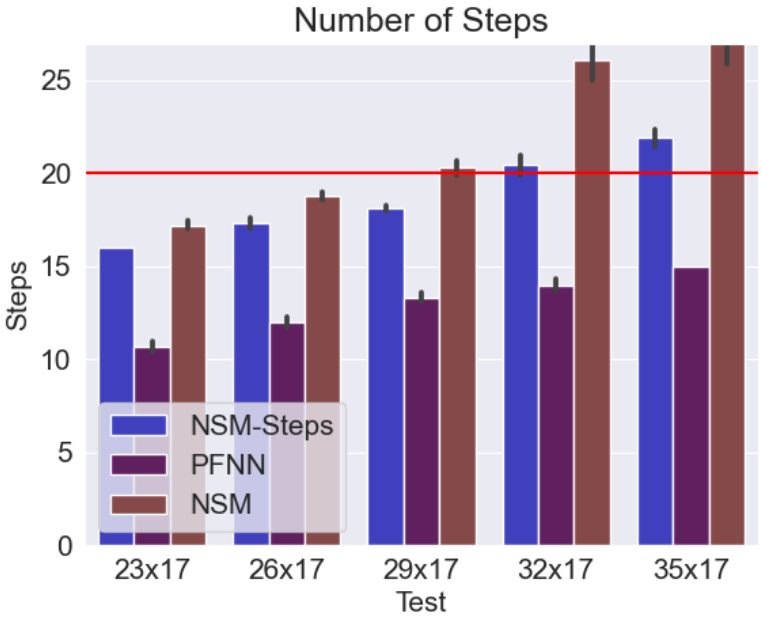


(a) Tread variation

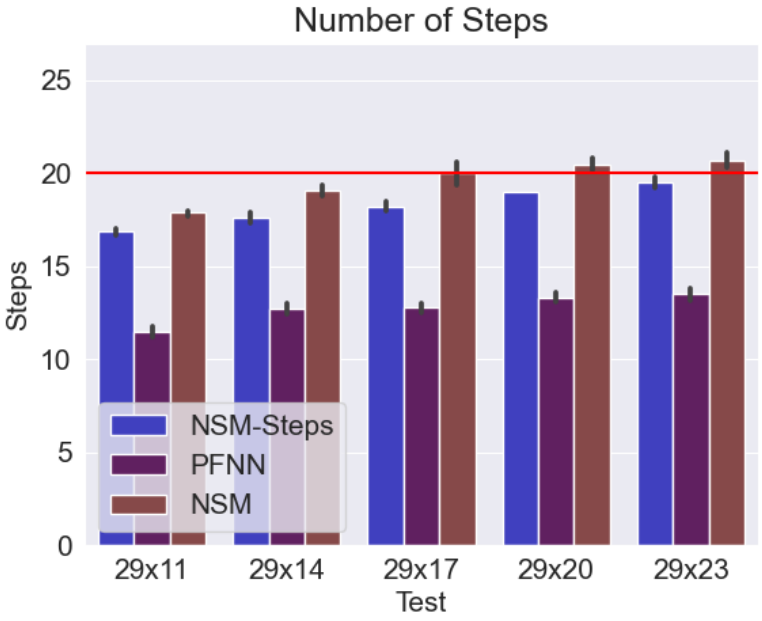


(b) Riser variation

Figure B.6: Stepping accuracy percentage (higher is better) of the *descending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

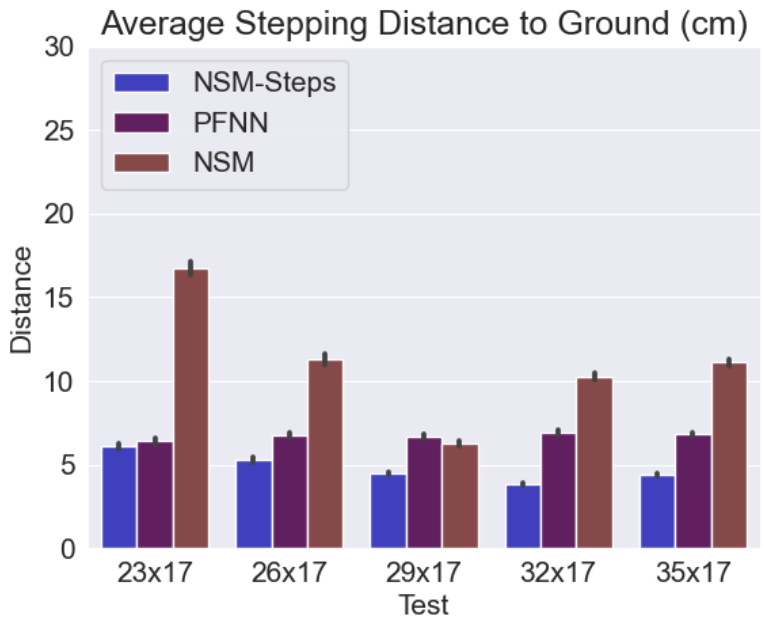


(a) Tread variation

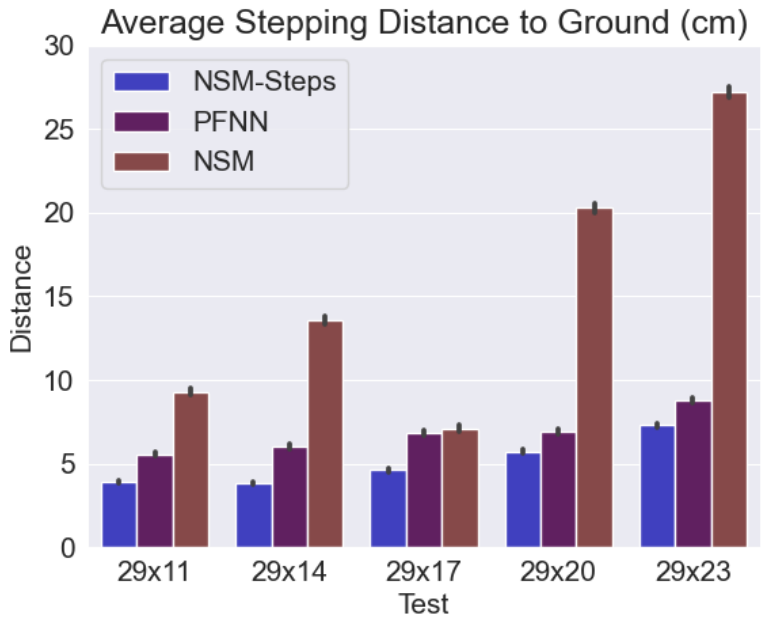


(b) Riser variation

Figure B.7: Number of steps taken of the *descending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

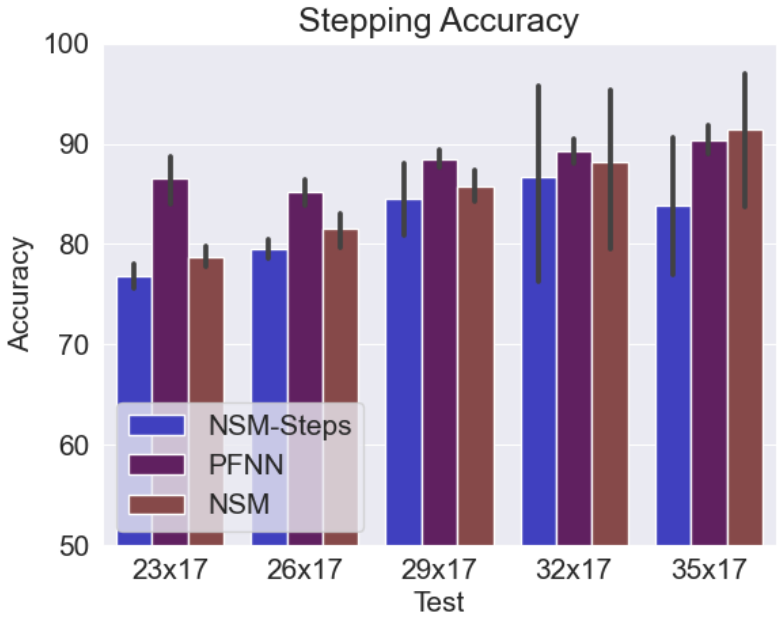


(a) Tread variation

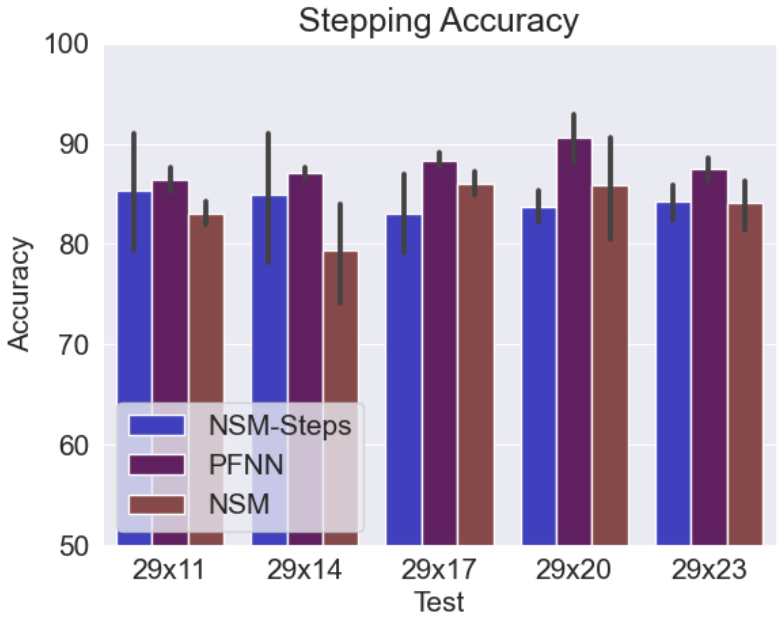


(b) Riser variation

Figure B.8: Average stepping distance to ground in cm (lower is better) of the *ascending straight* test scenario. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

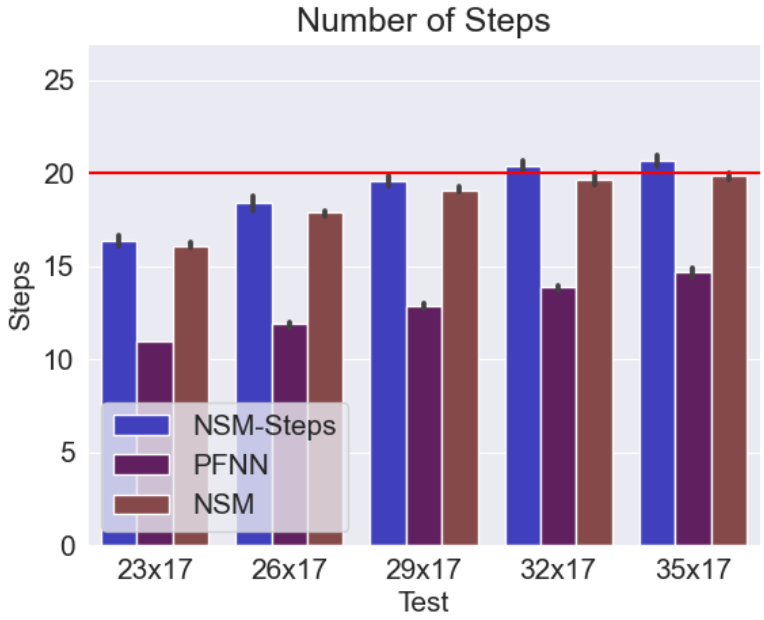


(a) Tread variation

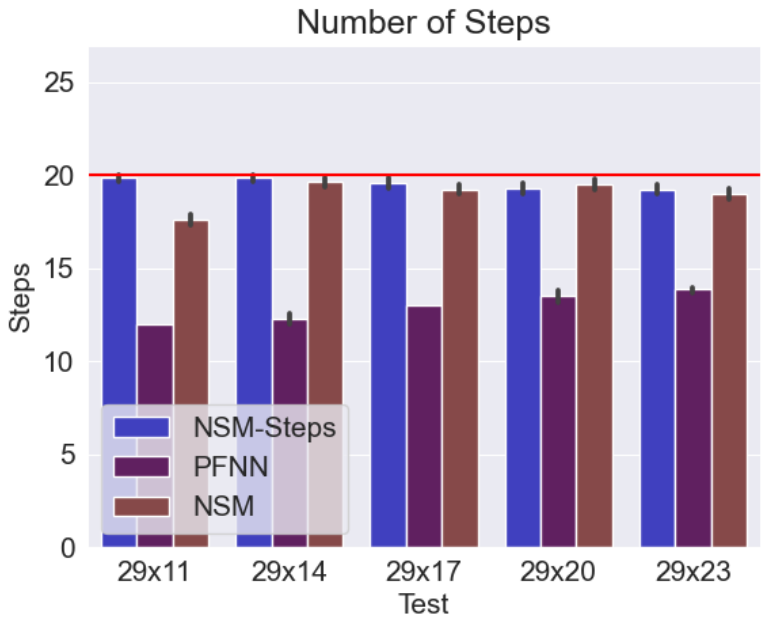


(b) Riser variation

Figure B.9: Stepping accuracy percentage (higher is better) of the *ascending straight* test scenario. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

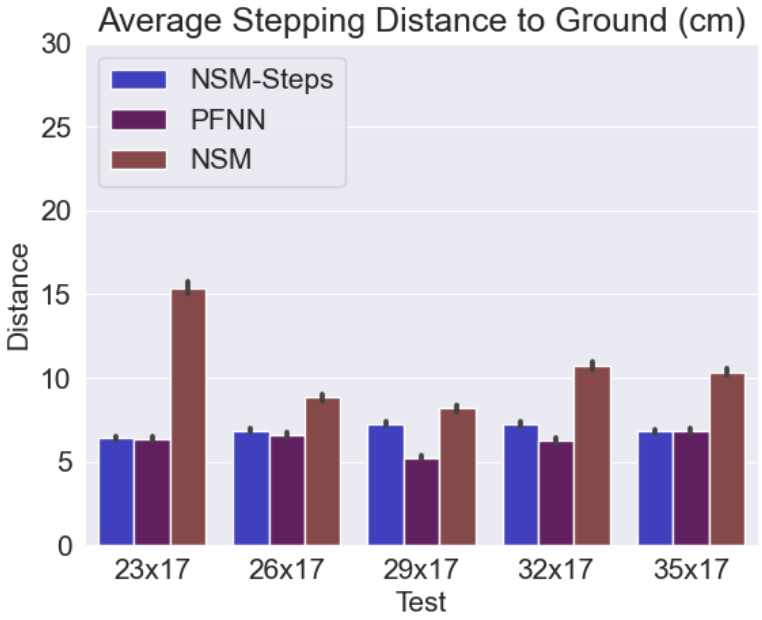


(a) Tread variation

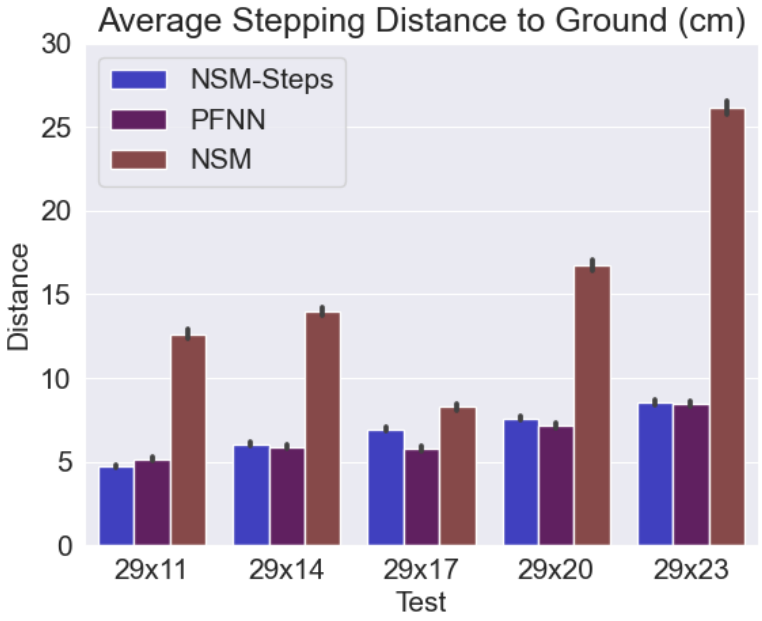


(b) Riser variation

Figure B.10: Number of steps taken of the *descending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

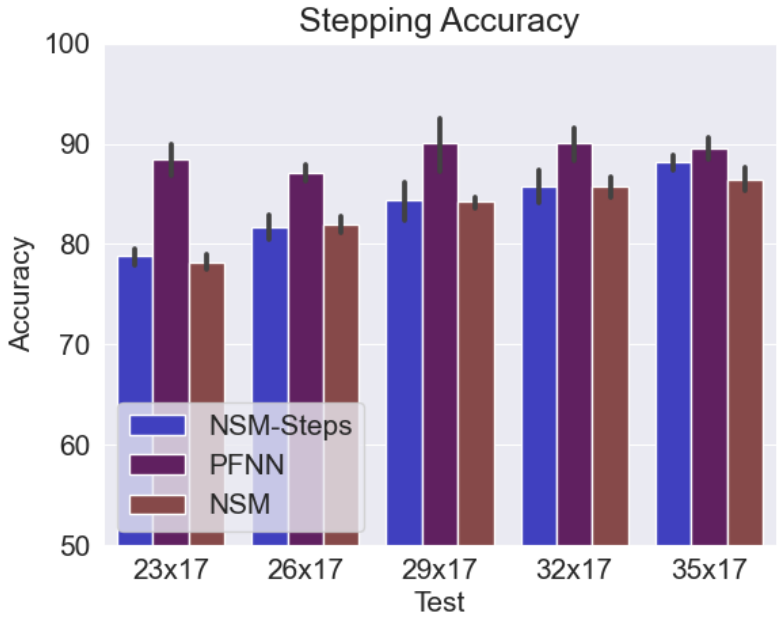


(a) Tread variation

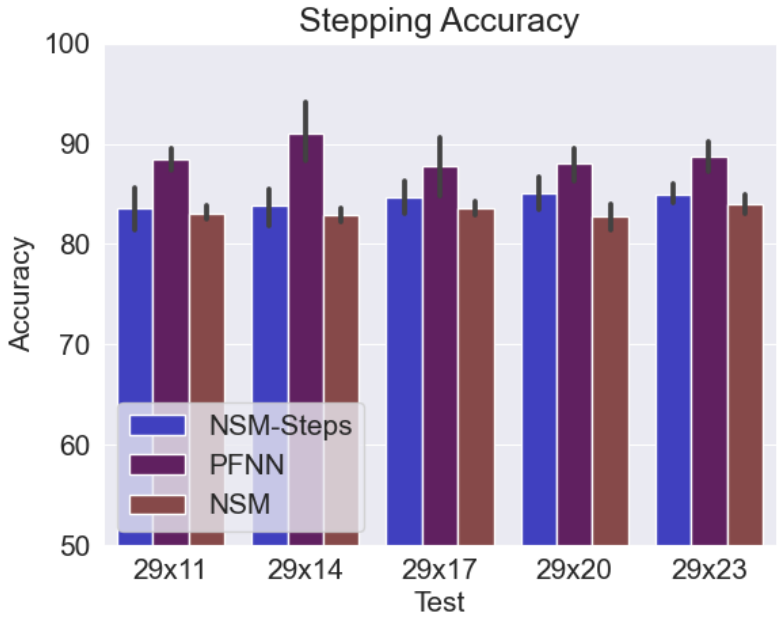


(b) Riser variation

Figure B.11: Average stepping distance to ground in cm (lower is better) of the *ascending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

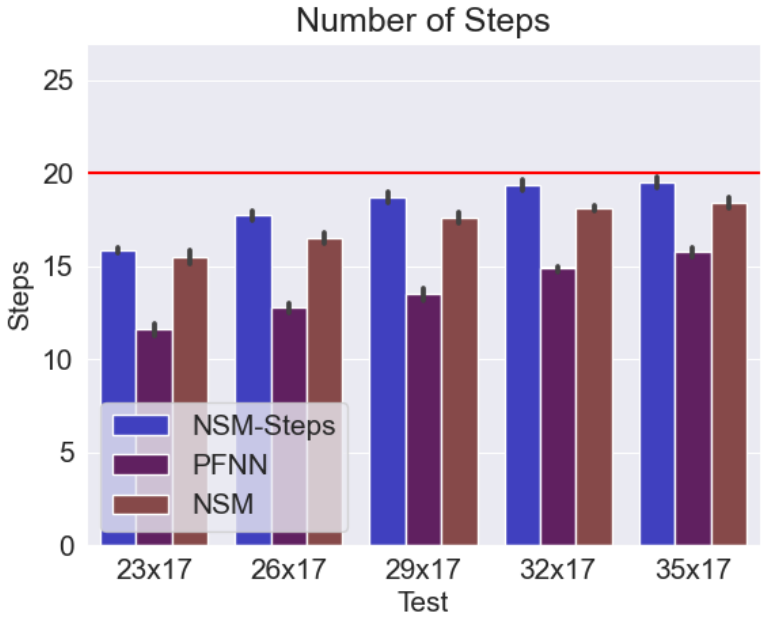


(a) Tread variation

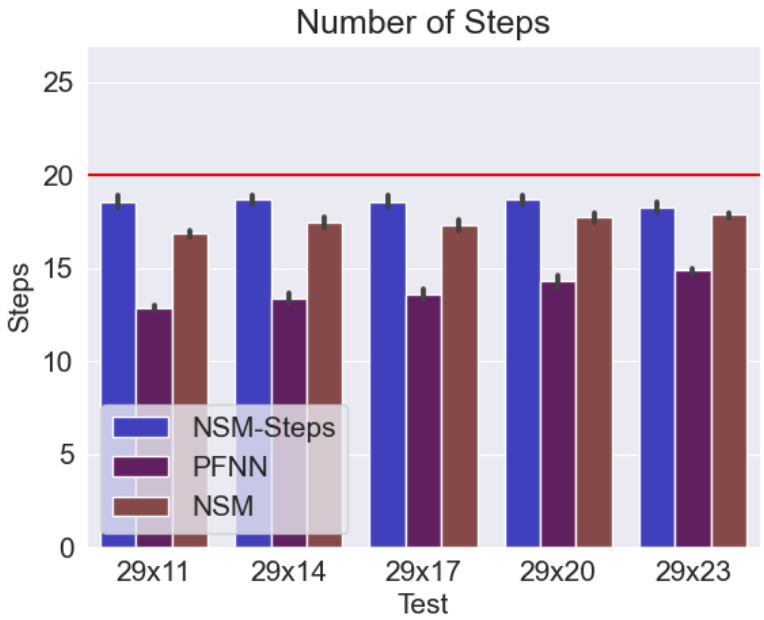


(b) Riser variation

Figure B.12: Stepping accuracy percentage (higher is better) of the *ascending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

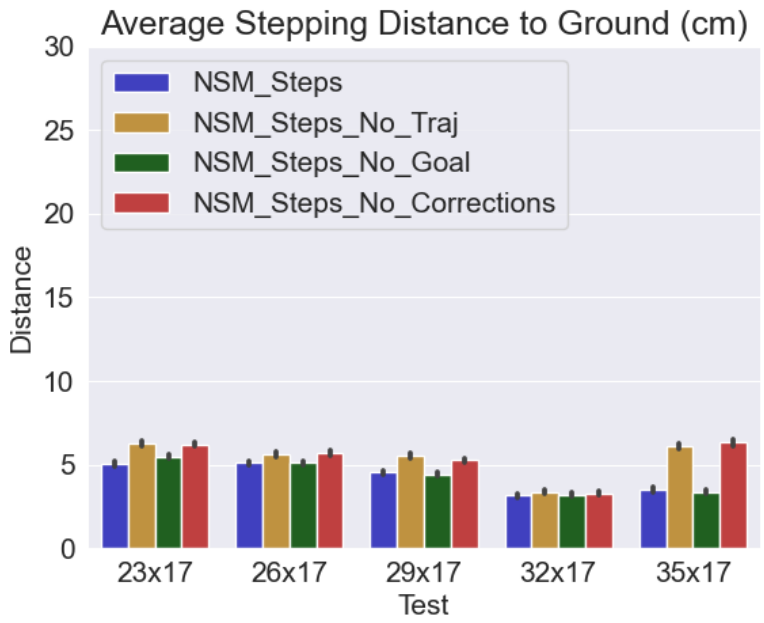


(a) Tread variation

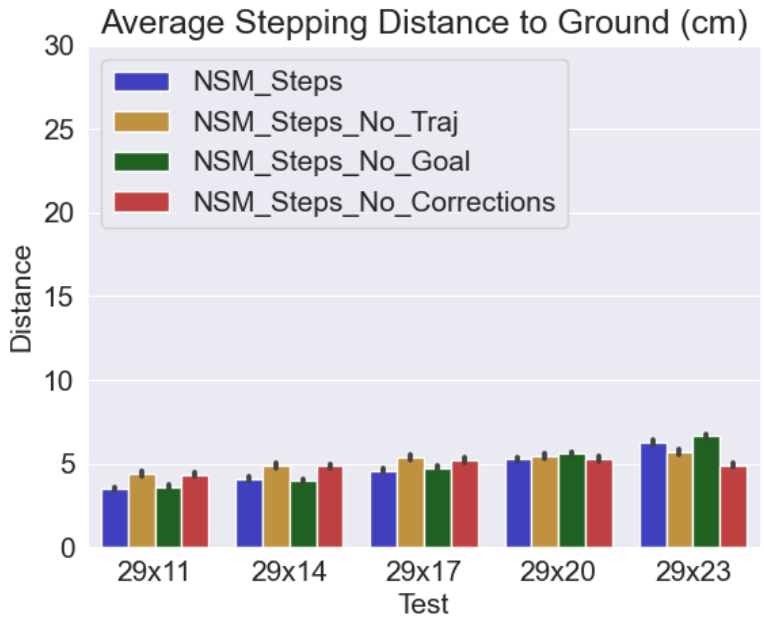


(b) Riser variation

Figure B.13: Number of steps taken of the *descending diagonal* test scenario. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

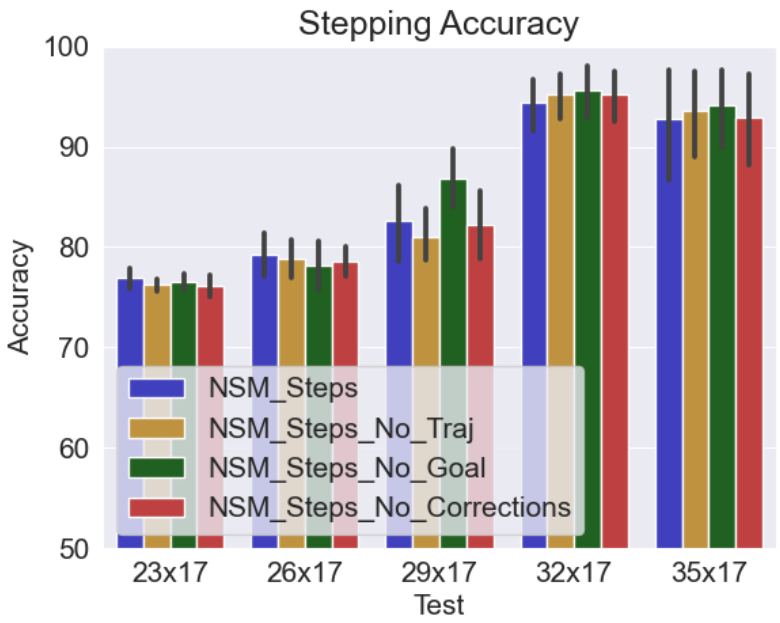


(a) Tread variation

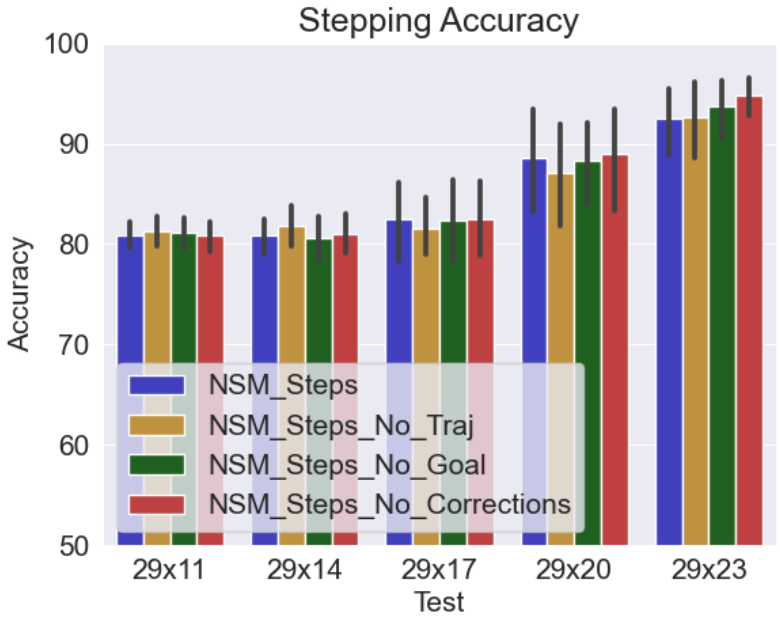


(b) Riser variation

Figure B.14: Average stepping distance to ground in cm (lower is better) of the *descending straight* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

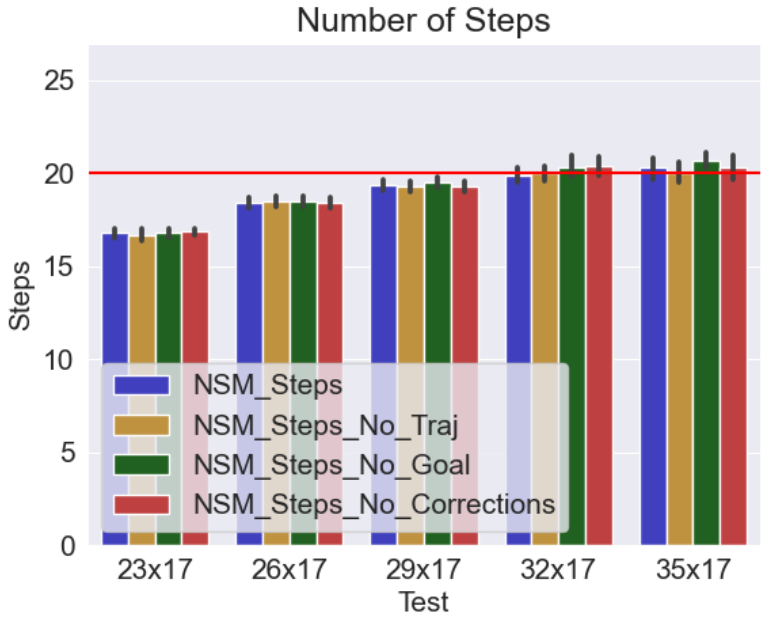


(a) Tread variation

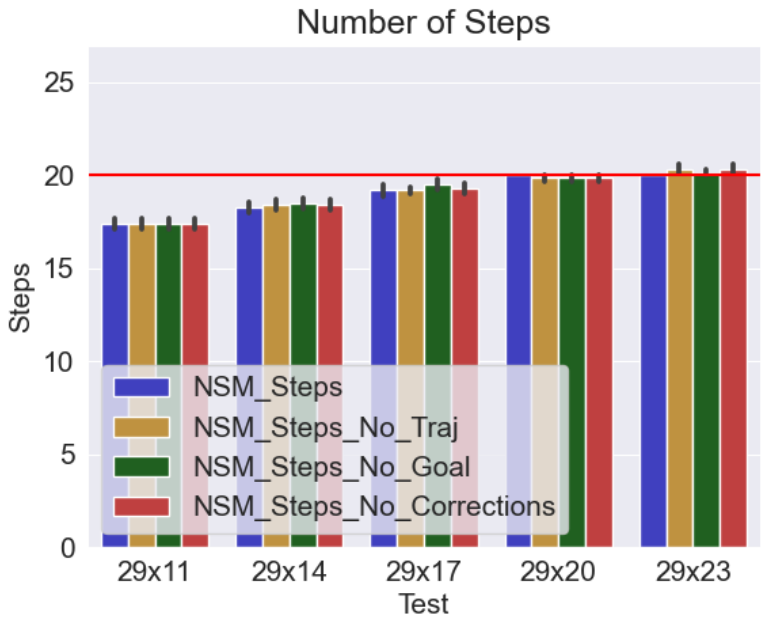


(b) Riser variation

Figure B.15: Stepping accuracy percentage (higher is better) of the *descending straight* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

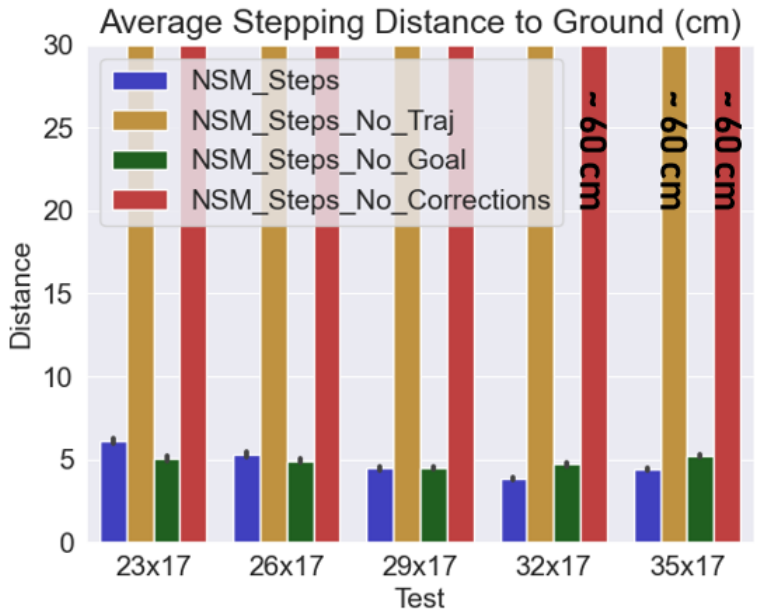


(a) Tread variation

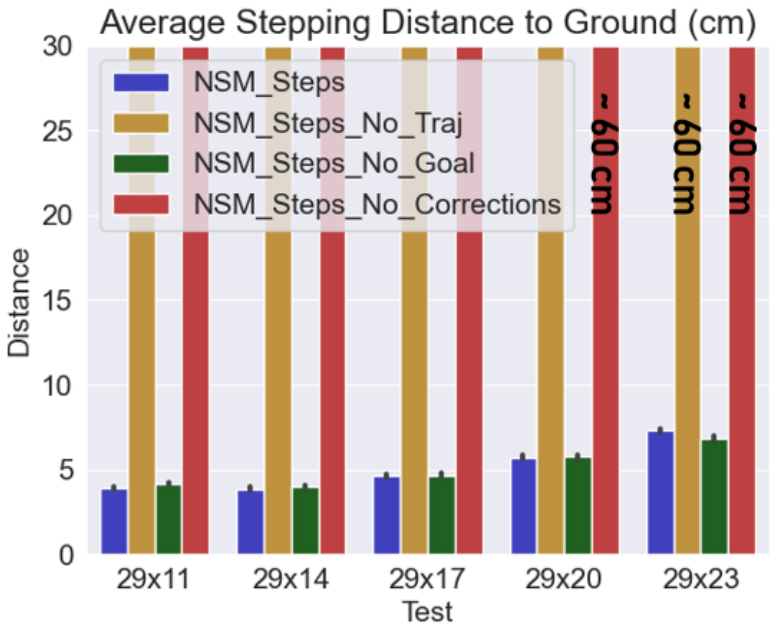


(b) Riser variation

Figure B.16: Number of steps taken of the *descending straight* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

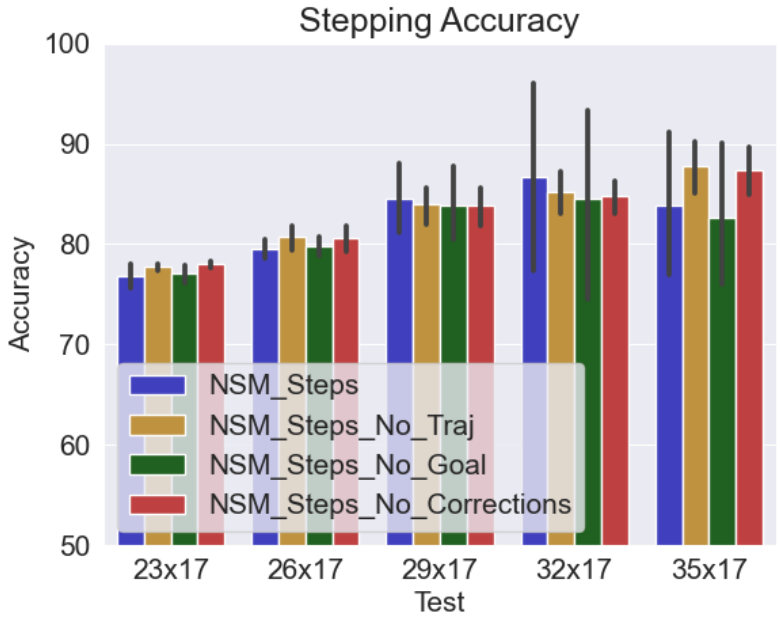


(a) Tread variation

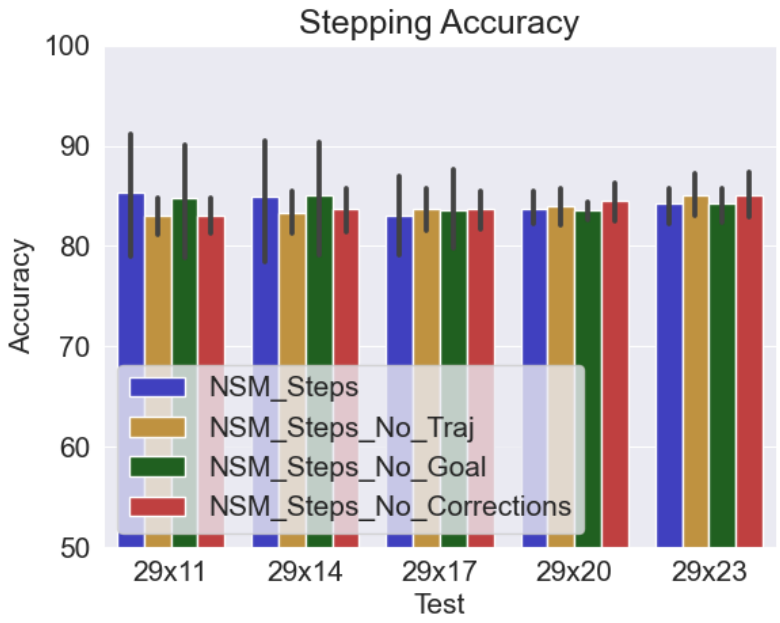


(b) Riser variation

Figure B.17: Average stepping distance to ground in cm (lower is better) of the *ascending straight* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

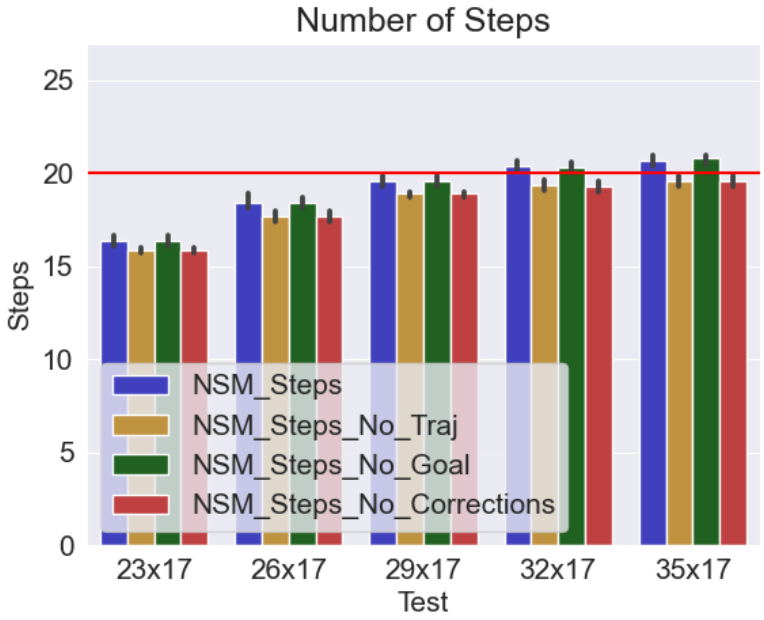


(a) Tread variation

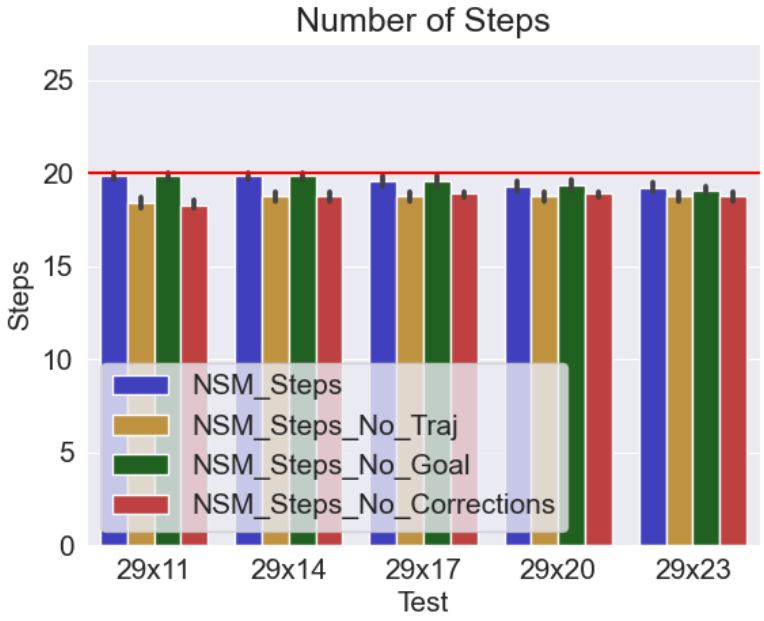


(b) Riser variation

Figure B.18: Stepping accuracy percentage (higher is better) of the *ascending straight* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

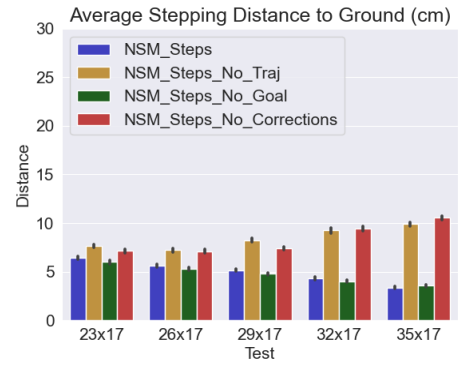


(a) Tread variation

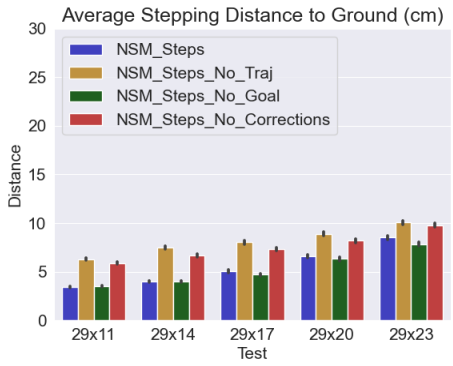


(b) Riser variation

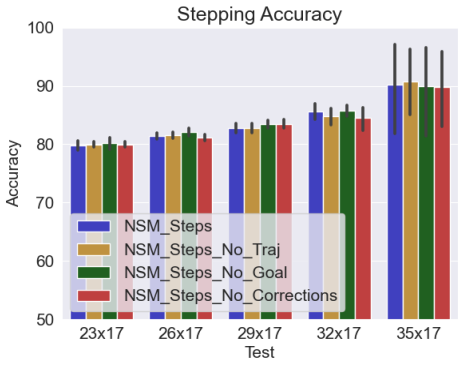
Figure B.19: Number of steps taken of the *ascending straight* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.



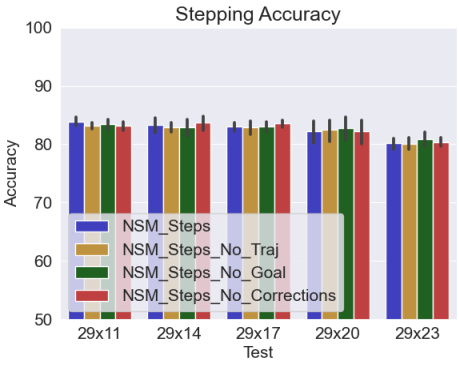
(a) Tread variation



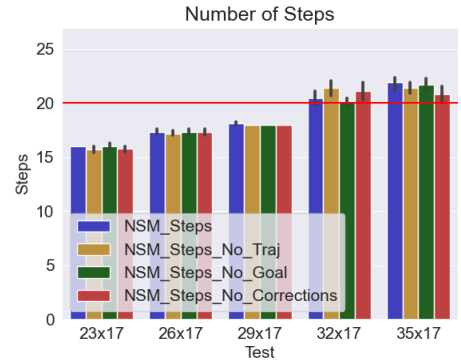
(b) Riser variation



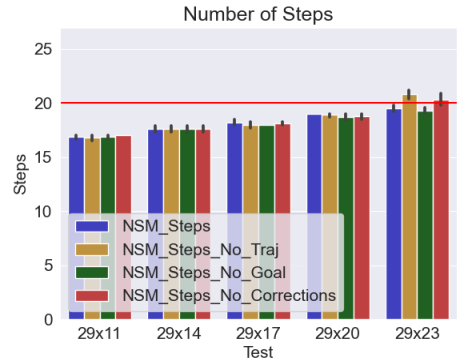
(c) Tread variation



(d) Riser variation

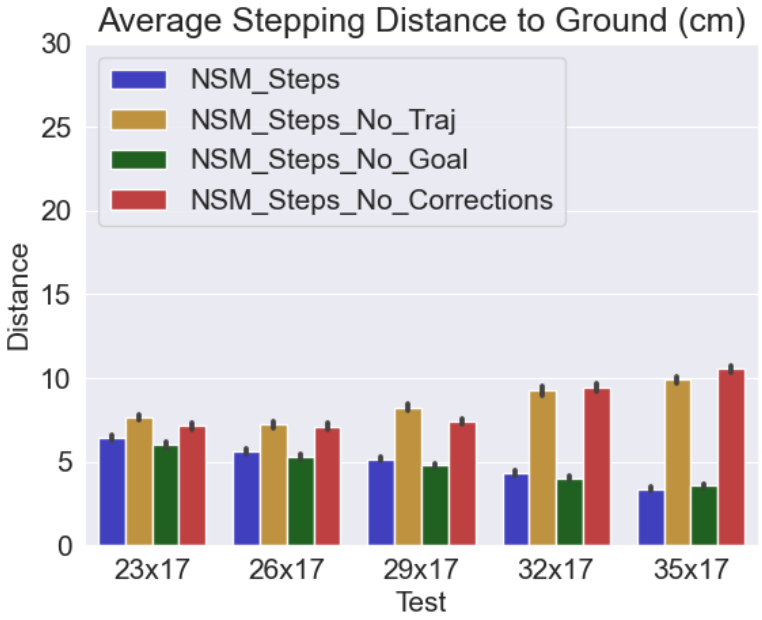


(e) Tread variation

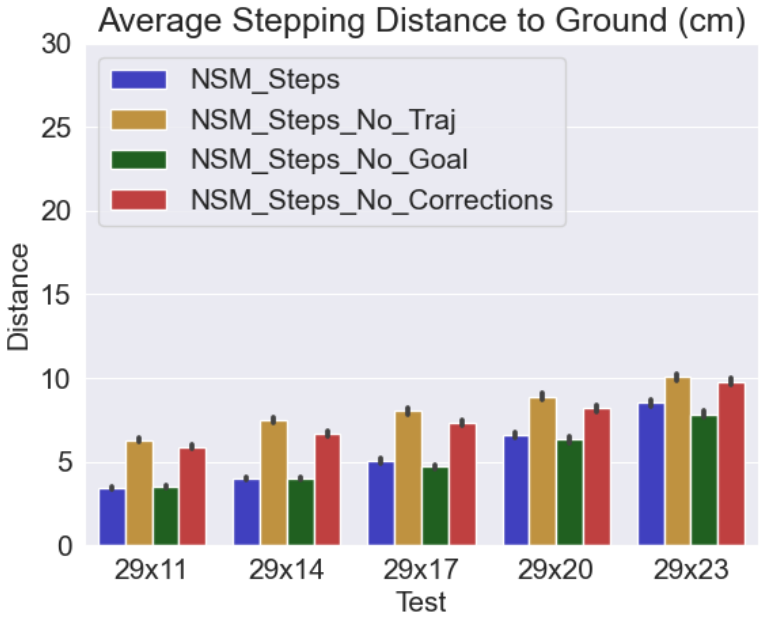


(f) Riser variation

Figure B.20: Results of *descending diagonal* test scenario for the ablation study. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in figures B.21, B.22 and B.23.

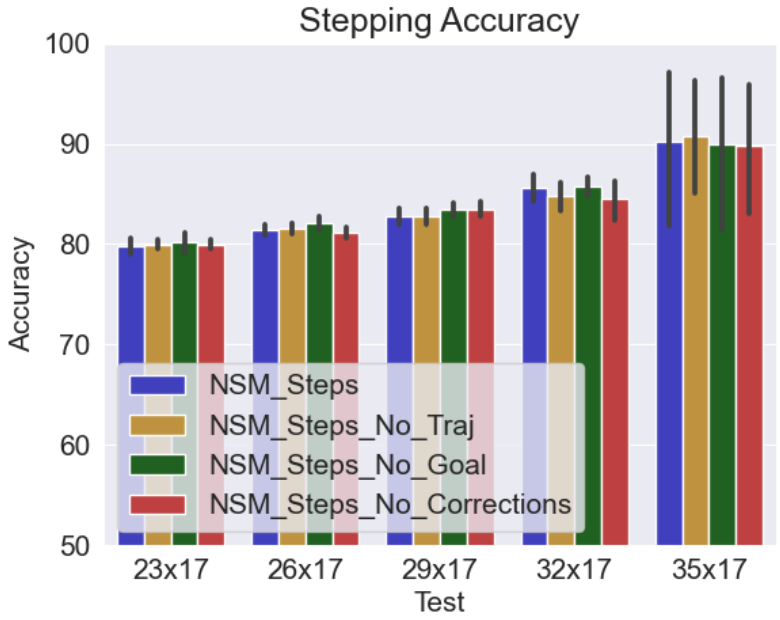


(a) Tread variation

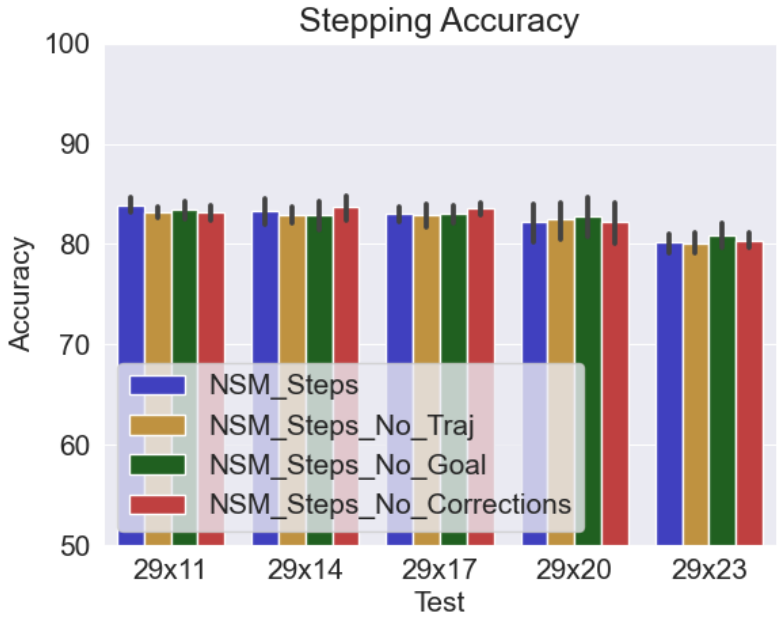


(b) Riser variation

Figure B.21: Average stepping distance to ground in cm (lower is better) of the *descending diagonal* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

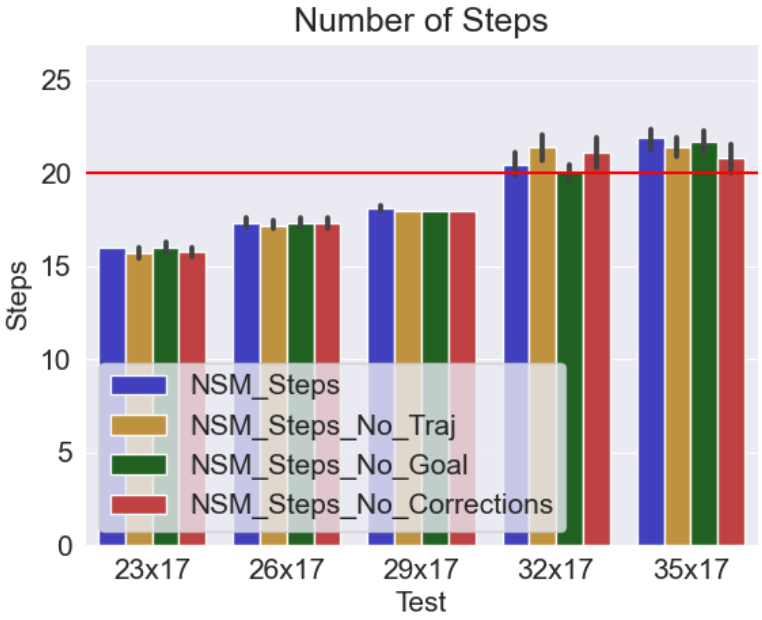


(a) Tread variation

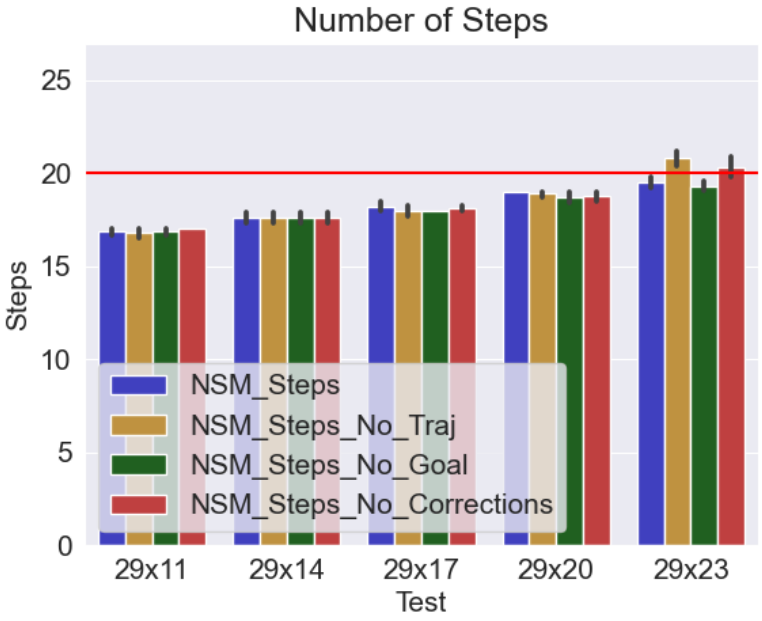


(b) Riser variation

Figure B.22: Stepping accuracy percentage (higher is better) of the *descending diagonal* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

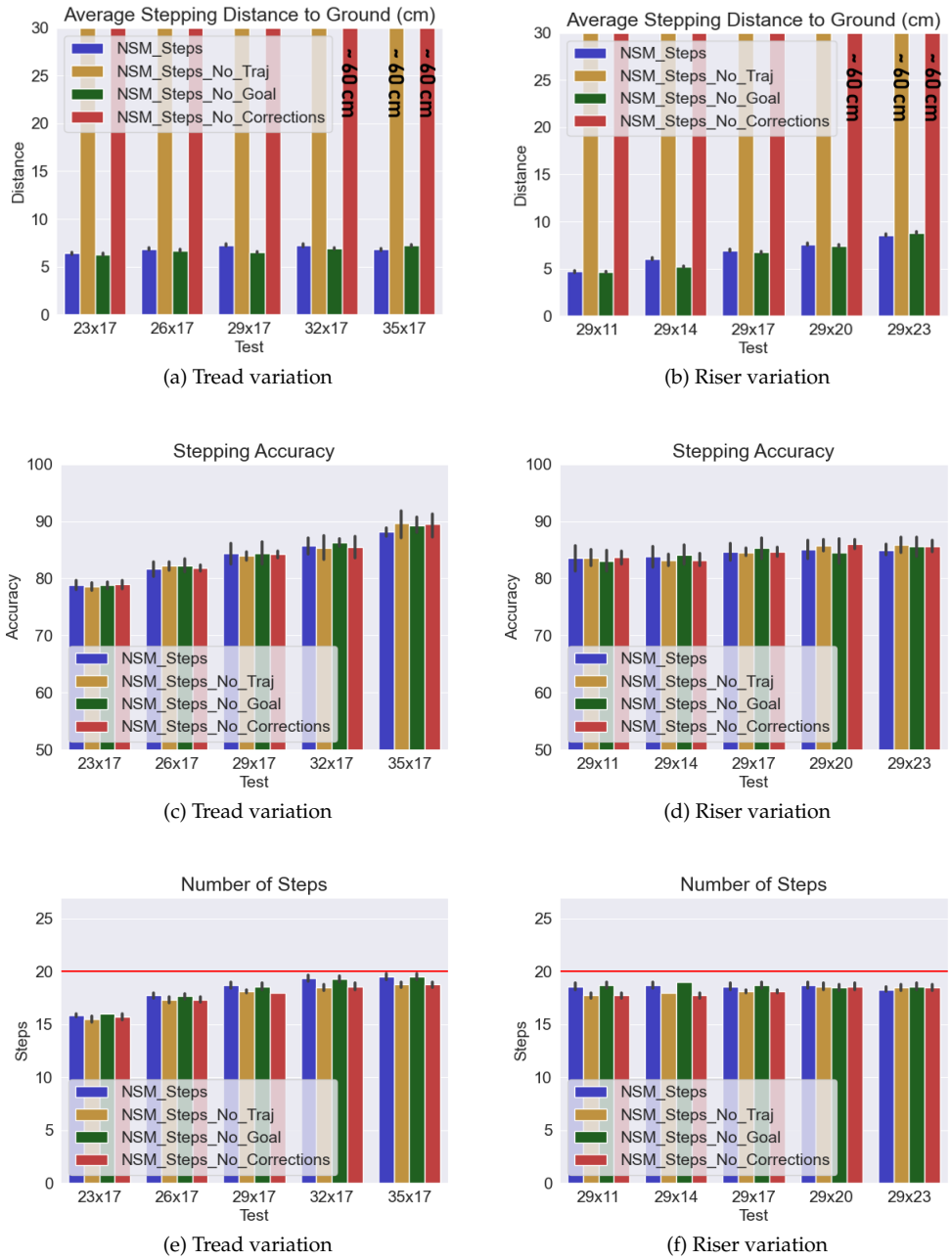


(a) Tread variation



(b) Riser variation

Figure B.23: Number of steps taken of the *descending diagonal* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.



(a) Tread variation

(b) Riser variation

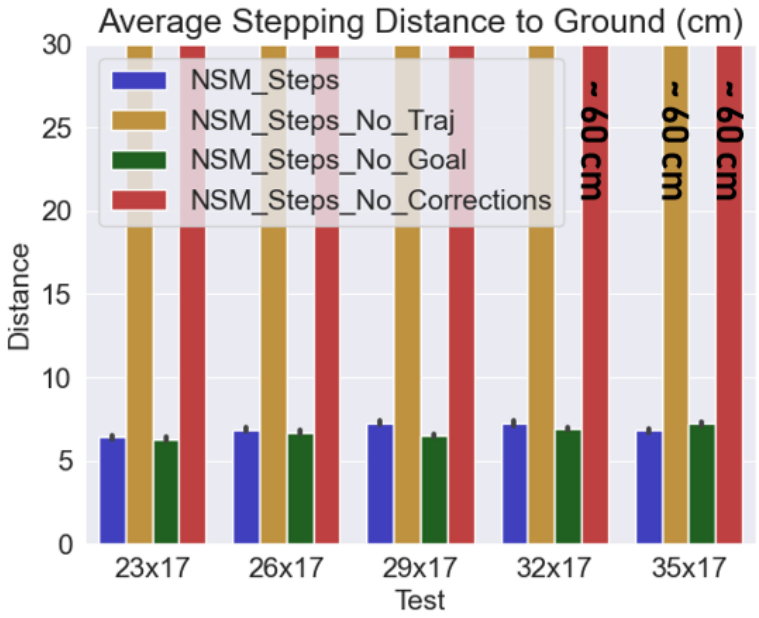
(c) Tread variation

(d) Riser variation

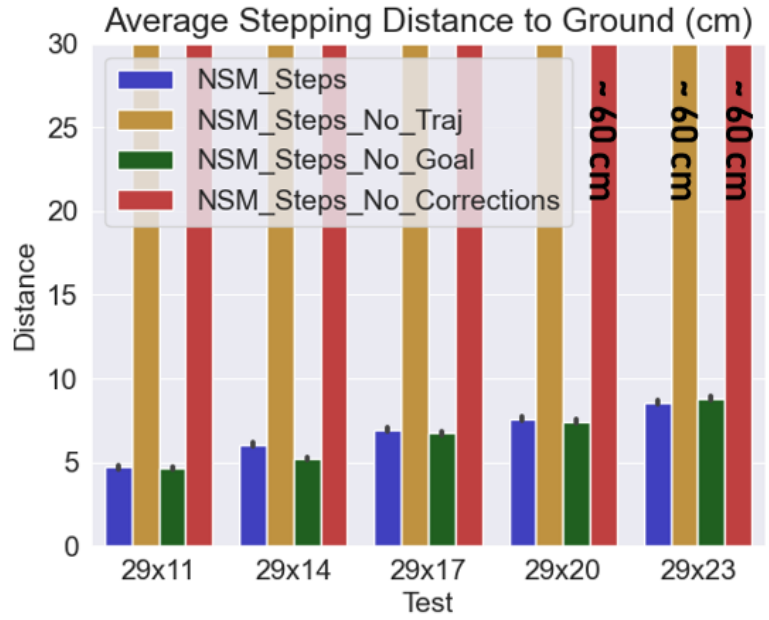
(e) Tread variation

(f) Riser variation

Figure B.24: Results of *ascending diagonal* test scenario for the ablation study. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average. A close-up of these results can be found in figures B.25, B.26 and B.27.

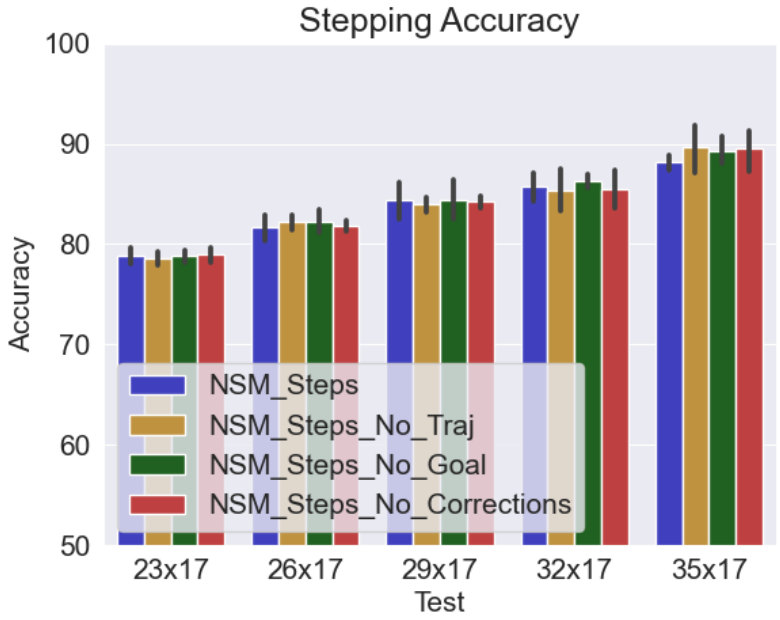


(a) Tread variation

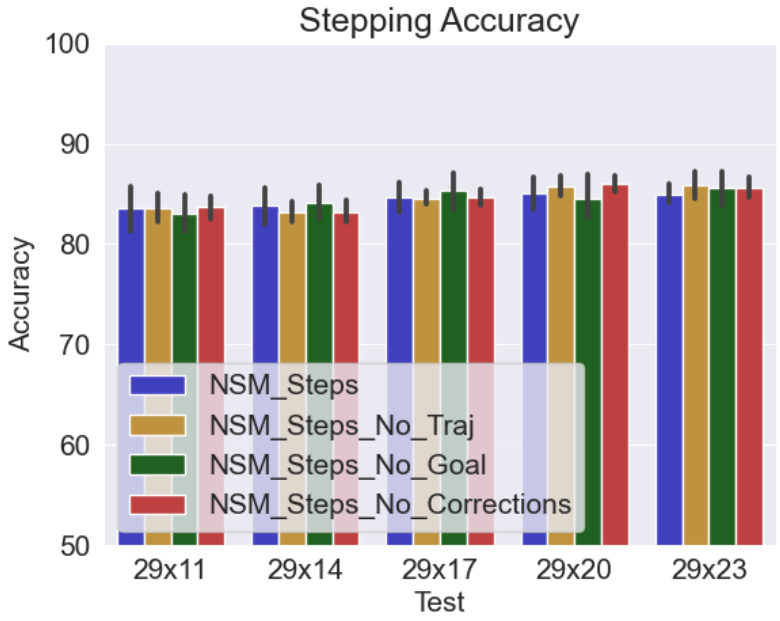


(b) Riser variation

Figure B.25: Average stepping distance to ground in cm (lower is better) of the *ascending diagonal* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

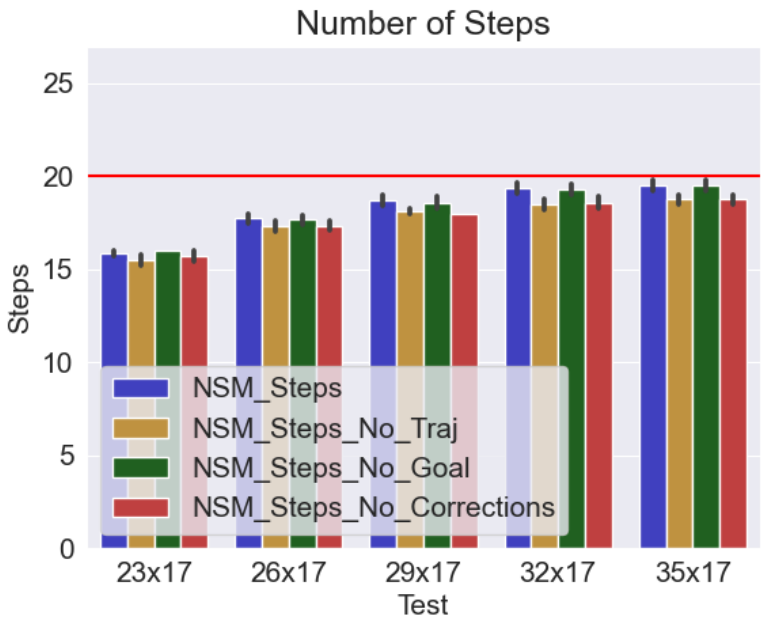


(a) Tread variation

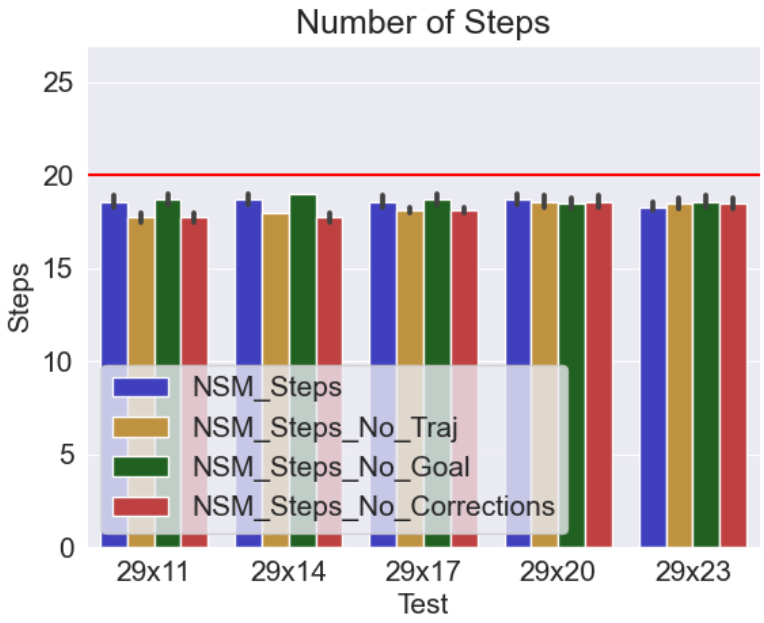


(b) Riser variation

Figure B.26: Stepping accuracy percentage (higher is better) of the *ascending diagonal* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. Up: average stepping distance to ground in cm (lower is better). The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.

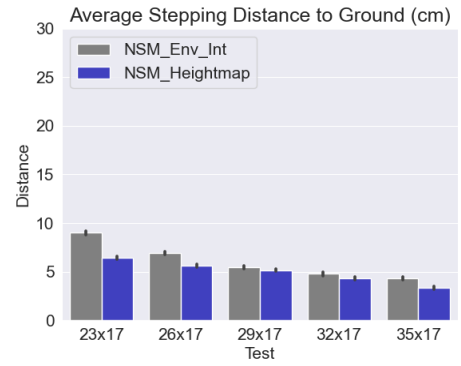


(a) Tread variation

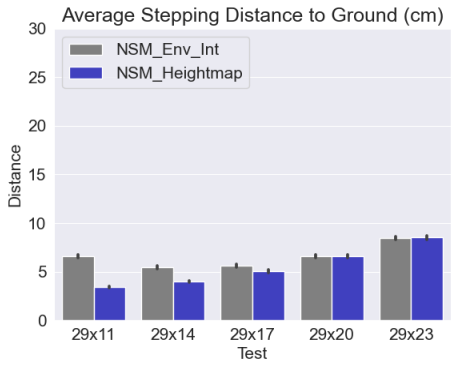


(b) Riser variation

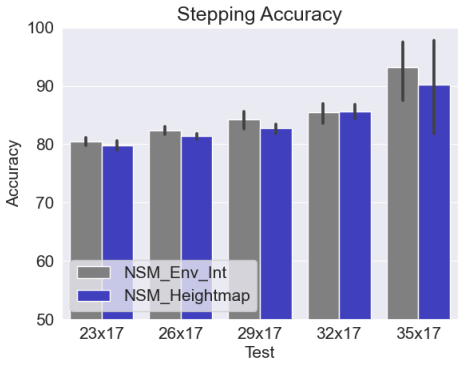
Figure B.27: Number of steps taken of the *ascending diagonal* test scenario for the ablation study. Top: results with tread variation. Bottom: results with riser variation. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.



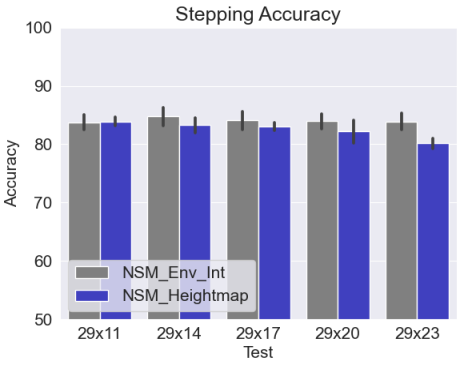
(a) Tread variation



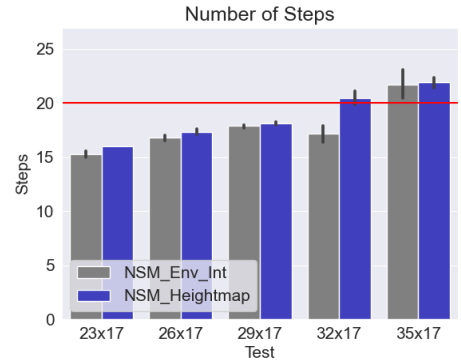
(b) Riser variation



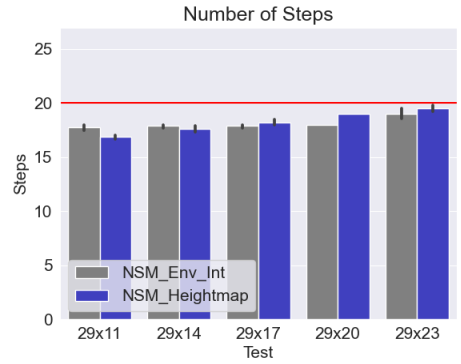
(c) Tread variation



(d) Riser variation

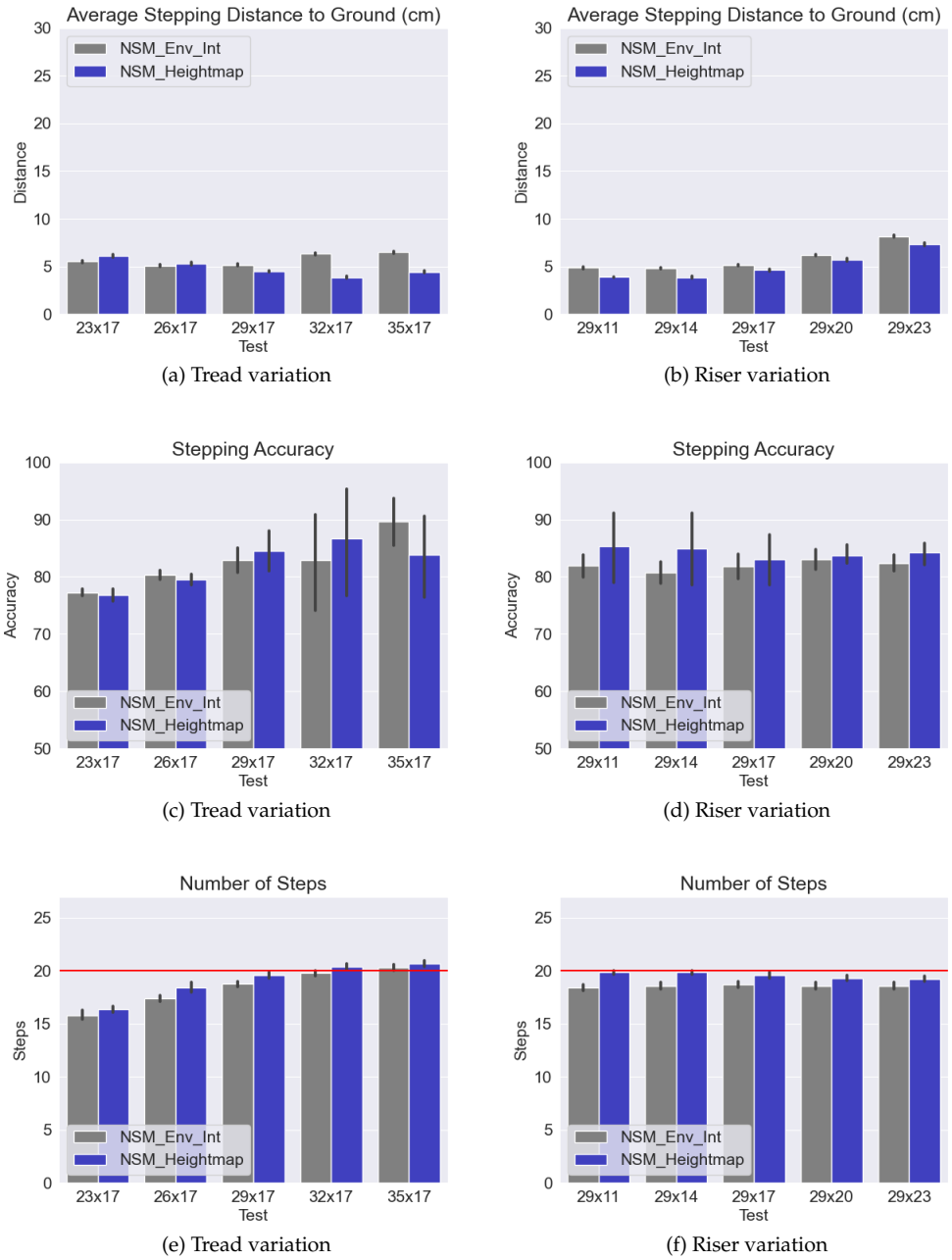


(e) Tread variation



(f) Riser variation

Figure B.28: Results of *descending diagonal* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.



(a) Tread variation

(b) Riser variation

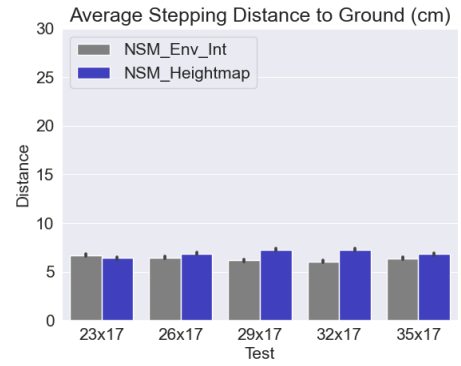
(c) Tread variation

(d) Riser variation

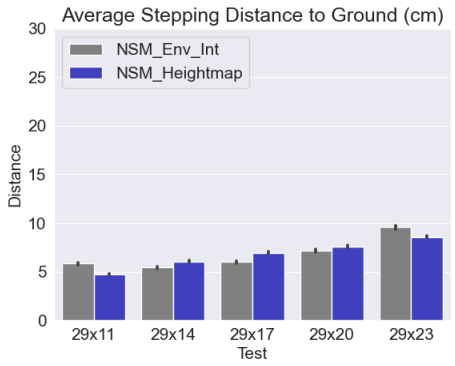
(e) Tread variation

(f) Riser variation

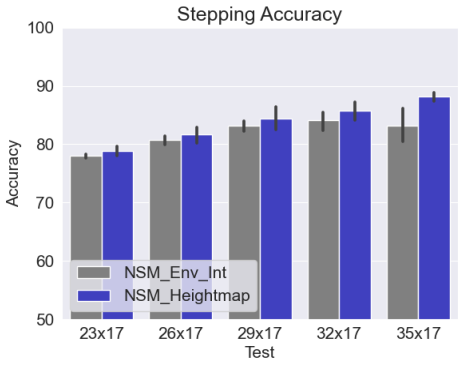
Figure B.29: Results of *ascending straight* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.



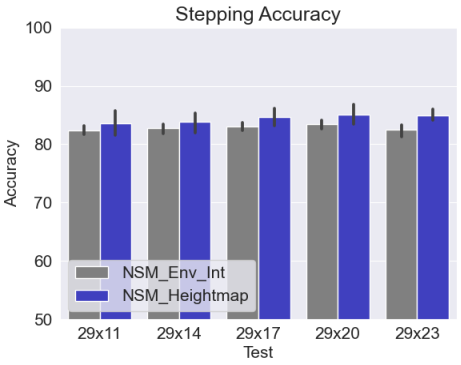
(a) Tread variation



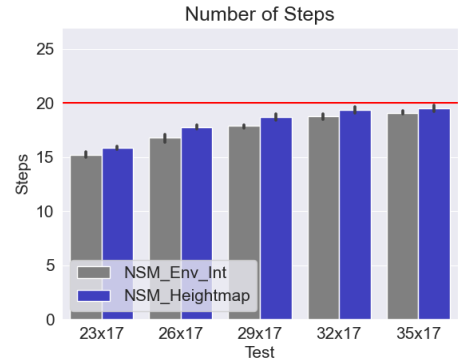
(b) Riser variation



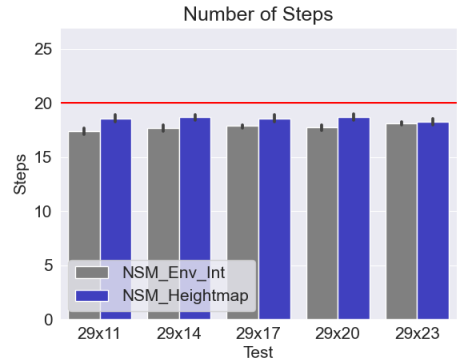
(c) Tread variation



(d) Riser variation

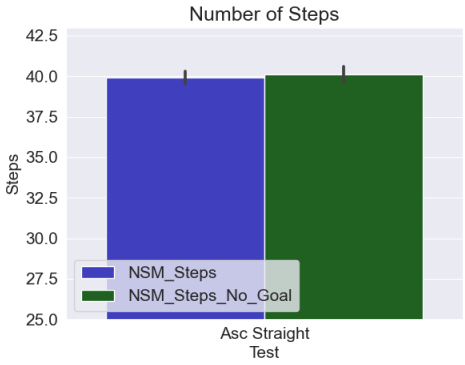


(e) Tread variation

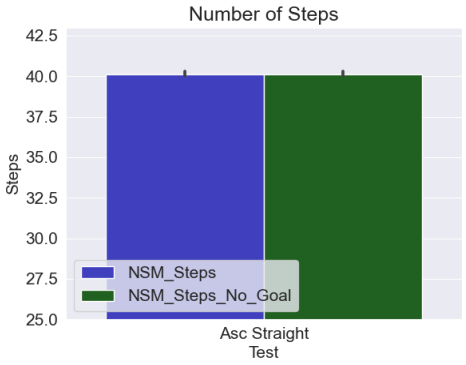


(f) Riser variation

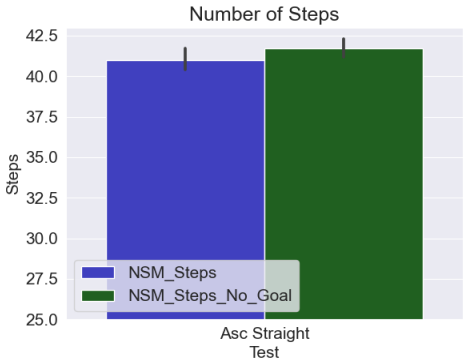
Figure B.30: Results of *ascending diagonal* test scenario. Left: results with tread variation. Right: results with riser variation. Up: average stepping distance to ground in cm (lower is better). Middle: stepping accuracy percentage (higher is better). Bottom: number of steps taken. The red horizontal line represents the number of steps of the staircase. Thus, the closer to this line, the better. The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.



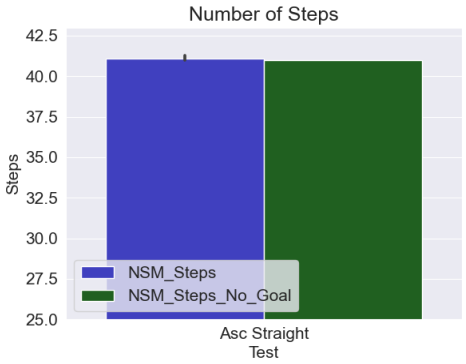
(a) Tread = 32 cm, riser = 17cm



(b) Tread = 32 cm, riser = 17cm



(c) Tread = 35 cm, riser = 17cm



(d) Tread = 35 cm, riser = 17cm

Figure B.31: Results of the number of steps in a 40 steps staircase for the footstep goal correction in large tread sizes (higher is better). Left: descending straight scenario. Right: ascending straight scenario. Top: 32x17 step sizes. Bottom: 35x17 step sizes . The error bars (black lines in the middle of the bars) represent the 95% confidence intervals around the average.