Chapter 1

Introduction

There exist organisms of many different shapes and body configurations both in nature and media. Creating realistic looking character locomotion for real-time applications remains a demanding task especially in the absence of motion data.

In this work, we present a simple and robust approach to generating gait patterns in real-time. Full-body character dynamics are difficult and high dimensional, instead we use a simplified dynamics model to estimate the trajectory of the character's center of mass. To achieve realistic-looking balancing behaviour, we adapt the character's planned footstep locations and body trajectory using *Support Polygon* optimisation. Our approach is novel in that the footfall pattern is computed continuously instead of in phases; it instantly reacts to changes in user input and environment.

The system can be split into three different sections: *Trajectory*, *Footstep* and *Support Polygon* optimisations. The trajectory is given by the dynamics of the *Inverted Pendulum on a Cart* (IPC) model. Each simulation step of the trajectory has a corresponding point on the support polygon which is spanned by the current footstep locations. Using *Single Shooting Trajectory Optimisation*, we optimise all three sections so that these two sets of points get as close to each other as possible while the dynamics of the pendulum are still satisfied.

Chapter 2

Related Work

There are many approaches to physically simulated characters, several involve simplified dynamics models to satisfy real-time requirements. The inverted pendulum is often used to model pelvis movement. An inverted pendulum fixed to a foot location reasonably approximates the trajectory of the pelvis around that foot. This can be used to compute a good long-term strategy to be enhanced by rigid-body simulation and full-body motion control [ZCY+15]. Two possible expansions are using a *Spring-Loaded Inverted Pendulum* (for knee bending behaviour) and by moving the location of the center of pressure (CoP) over time [MdLH10].

Conversely, the *Inverted Pendulum on a Cart (IPC)* model can be used to approximate the center of mass trajectory, not by fixing the CoP to a footstep location but by simulating its movement over time. It realistically models both wind-up and wind-down, and provides a rotation for character leaning. The resulting trajectory is translated into character locomotion using additional dynamics such as rigid-body simulation [KLVDP20] or motion data [KH10].

For a character to keep balance, their *Center of Mass* (CoM) should be inside the beforementioned *Support Polygon*, i.e. the convex hull spanned by their contact points on the ground. One reasonable implementation of the *Support Polygon* is to define the CoP as the linear combination of all contact points similar to barycentric coordinates [Zim13]. This makes sure the footstep locations are computed in a way that the CoP remains in the area of support (instead of the CoM, which accounts for momentum changes).

Chapter 3

IPC Dynamics

We model the locomotion trajectory as an *Inverted Pendulum on a Cart* (IPC), where the two degrees of freedom along the ground plane (x and y) are computed independently. The IPC model is enhanced with two additional models: The pendulum orientation and variable height.

3.1 Linearised Dynamics

We can simplify the pendulum dynamics of a pendulum $\mathcal{P}$ as a linearised model where s denotes the current system state, y the current force response, and A, B the dynamics of the system

$$\dot{s} = A \cdot s + B \cdot y. \tag{3.1}$$

This linearised model allows us to use *LQR Optimisation* to compute the force response y similar to [KH17]. The matrices A and B are *block diagonal*, thus equation (3.1) can be split into three independent dynamics models for the pendulum P (one for each dimension x and y), rotation R, and height H[1]:

$$A = \begin{pmatrix} P_A & 0 & 0 & 0 \\ 0 & P_A & 0 & 0 \\ 0 & 0 & R_A & 0 \\ 0 & 0 & 0 & H_A \end{pmatrix}, \; s = \begin{pmatrix} s_x \\ s_y \\ s_r \\ s_h \end{pmatrix}, \; B = \begin{pmatrix} P_B & 0 & 0 & 0 \\ 0 & P_B & 0 & 0 \\ 0 & 0 & R_B & 0 \\ 0 & 0 & 0 & H_B \end{pmatrix}, \; y = \begin{pmatrix} u_x \\ u_y \\ u_r \\ u_h \end{pmatrix}$$

The state vector s defines multiple dynamic properties of the pendulum; namely position, velocity, and angle for both axis, as well as rotation around

[1] these models are explained in more detail in Appendix IPC

the pendulum axis and distance to the ground (i.e. height).

u_x and u_y are the force responses applied to the pendulum base, u_r is the torque applied along the pendulum axis, and u_h is the force applied against the ground. It has to be noted that we still assume constant pendulum length, i.e. we ignore changes in pendulum dynamics due to changes in height as to not introduce non-linearities to A and B. Consequently, we assume the height to remain close to the pendulum length L. This turned out to be sufficient while walking on a flat plane. In case of significantly uneven terrain, Zimmermann discusses a possible solution using power series [Zim13]. A similar approach could be chosen to allow for variable A and B.

3.2 Force Response

Before we can compute the final force response y, we first compute the desired force response f using *LQR Optimisation*:

$$f = K \cdot (s - q), \tag{3.2}$$

where K is the feedback from *Infinite-horizon LQR*[Clo97] for our dynamics system equation (3.1) and target state q. We define the cost function

$$J = \int_0^\infty ((s-q)^T Q(s-q) + u^T R u)dt,$$

which minimises the difference between the pendulum state s and target q according to some weight matrix Q. It also keeps the force response low with regulariser R.

Undesirably, this model allows the CoM to be pulled towards the floor[2] and conversely permits forces to be applied while in mid-air. We solve both of these issues by adding a non-linearity to the force response. From the LQR optimisation, u_h will always be optimised to pull the height z towards L (i.e. upwards). Therefore, it is sufficient to prevent any application of force when the character is in flight. We define a sigmoid function $g(z) \approx 0$ for $z > L$ with steepness τ:

$$g(z) = 1 - \frac{1}{1 + e^{\tau(1-\frac{z}{L})}} \tag{3.3}$$

As a multiplier, this prevents any reaction forces when the pendulum is too far off the ground. It is also smooth for differentiability. With some offset W, the final reaction forces are computed as

[2] legged creatures are generally not equipped with suction cups

$$y = (f + W) \cdot g(z).$$

A non-zero function $W(s, f)$ can add dynamics to the system that depend on both the desired force f and the current state s. One possible implementation is height contraction when a planar force is applied to the pendulum while at an angle (θ_x, θ_y) with

$$w_h = \theta_x \cdot f_{p_x} + \theta_y \cdot f_{p_y}.$$

This leads to the pendulum trajectory not only swinging back and forth, but also up and down as illustrated in figure 3.1.

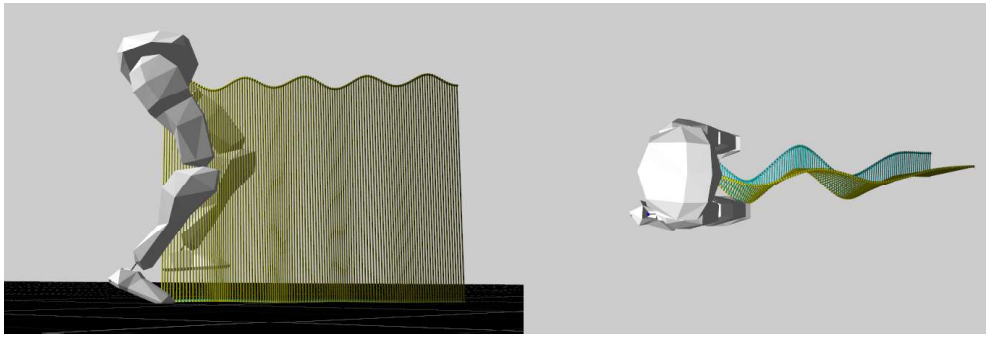

Figure 3.1: The pendulum height gets contracted as the pendulum swings back and forth.

The additional dynamics and constraints are not considered by the LQR optimisation as they are non-linear. Instead we rely on the feedback control to react to these perturbations retroactively. For anticipatory behaviour, the target state q is computed by *Single Shooting Trajectory Optimisation* (discussed in optimisation chapter).

3.3 Discretisation and Differentiability

The actual simulation is computed in discrete time steps, which gives us from equation (3.1)[3]:

$$s_{t+1} = (A \cdot t + \mathbb{I}) \cdot s_t + (B \cdot t) \cdot y_t \tag{3.4}$$

The whole system is differentiable for the target state q_t at the given time t. More specifically, one can see together with equation (3.2) that differentiating (3.4) for q_t will yield a sum between a term that depends on $\frac{ds_t}{dq_t}$ and one that does not:

[3] K also changes with the LQR optimisation now being discrete-time

$$\frac{ds_{t+1}}{dq_t} = \Psi \cdot \frac{ds_t}{dq_t} + \Phi$$

for some Ψ and Φ. This can be used to recursively compute the first derivative for all states to then be used during optimisation.

It has to be noted that a solution exists with differentiable $A(t)$ and $B(t)$ retaining the same properties. As mentioned before, with a power series approximating the pendulum dynamics A and B, one can easily account for changes in pendulum length to the pendulum dynamics. Unfortunately, this comes at the cost of more matrix operations. Experiments have shown that for flat ground, the difference in motion is negligible while significantly hurting runtime performance.

Chapter 4

Footstep Generation

4.1 Transform Data

For each foot, we define a footstep matrix V where each column v_i describes a footstep at a specific location and time. Our goal is to define a sampler that can give us the final location and rotation (transform) of this foot at any given time t. We define v_i as

$$v_i = \begin{pmatrix} x & y & \varphi & \mu & \sigma & \varepsilon \end{pmatrix}$$

where x, y, φ the location and rotation of the foot in floor space. μ describes the central time and σ the duration after and before the foot is on the ground. ε describes the duration before and after μ when the step is still relevant, i.e. after the previous step ends and before the next step starts. For a steepness parameter τ, we define a weight function w as

$$w(t) = \frac{1}{1 + e^{((t-\mu)^2 - \sigma^2)\cdot\tau}} + \frac{1}{1 + e^{(\frac{t-\mu}{\varepsilon})^2}}. \tag{4.1}$$

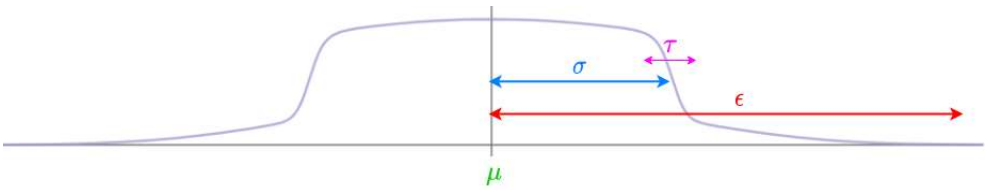

Figure 4.1: Step sampling with parameters from equation (4.1)

We define ε big enough for each step so that $\sum_i^n w_i > 0$ for all n steps. From this, we can always get a weighted average $\bar{v} = \frac{\sum w_i \cdot v_i}{\sum w_i}$ for the step transform

at time t. It is important to note that due to the second term in equation (4.1) there is always a smooth transition between steps as illustrated in figure 4.2. Later, optimisation and motion synthesis can make use of the total weight $\bar{w}$ to determine the distance of the foot to the ground and whether it is touching the ground (e.g. if $\bar{w} > 1$).

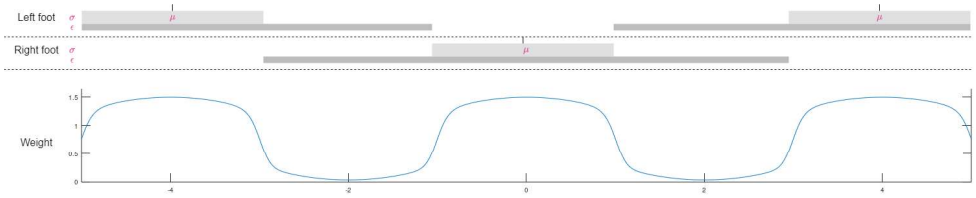

Figure 4.2: Footstep pattern on top with total weight $\bar{w}$ on the bottom.

4.2 Objectives

We don't only want to optimise for realistic foot trajectories, but also have demands on the footstep locations. Trivially, one can define safe-zones on the terrain using a differentiable 2D function $\mathbb{R}^2 \to \mathbb{R}$ as a measure for unsafety. This is sufficient for our purposes. Although where multiple situations have to be considered, we could generate multiple motion plans and pick the best one [ZCY+15]. Introducing safe-zones increases the likelihood of steps landing close to each other or legs intersecting. We define another error function $e : \mathbb{R}^2 \to \mathbb{R}$ on all step locations as

$$e(x, y) = e^{\tau(x-y^2)},$$

where x, y is a foot location relative to another as follows: The coordinate system is aligned with the two steps in the reference pose, scaled by their distance and the origin in between. Adding up this error function between all foot combinations as illustrated in figure 4.3 produces satisfactory results for any leg configuration. Steps are prevented from getting too close while not crossing over each other.

4.3 Step Fixation

All steps are continuously moved to minimise the objective functions. This is undesirable for steps where the character's foot is currently on the ground. To mitigate this, we fix the current foot location of the character to a given location during ground phase. The weight function deciding if the foot is on the ground is smooth, thus it is non-trivial to find the fixed location. Thresholding, for example, either makes the foot slide backwards before

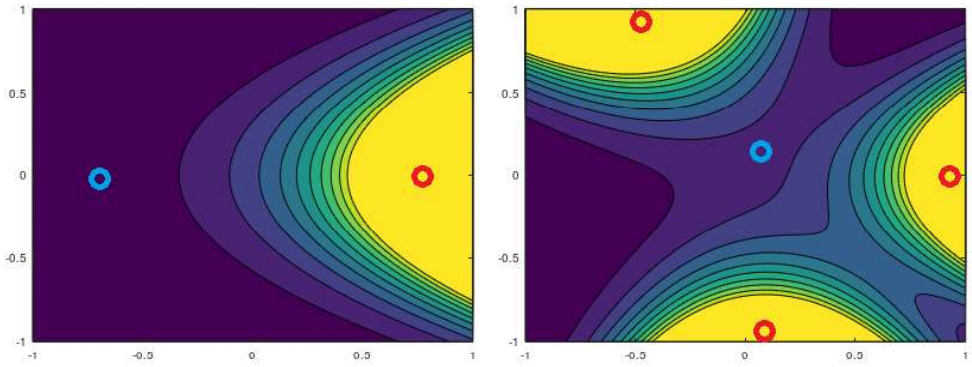

Figure 4.3: Foot configuration of bipedal character to the left and quadrupedal to the right. Blue is the current step and red the steps we are comparing against.

it hits the ground if too high, or creates erratic movement if too low. We use the weight function $w(t_0)$ at the current simulation time t_0 to smoothly interpolate the fixed step location for each foot. This works in most cases, but requires manual tweaking depending on the input parameters to prevent foot sliding.

Chapter 5

Optimisation

With optimisation, we want to achieve realistic motion where the *Center of Pressure* (CoP) of the character stays within their area of support. One or more footsteps on the ground and their contact points span the *Support Polygon*. Using a simplified definition of the support polygon, we establish a relationship between the footsteps V and the pendulum $\mathcal{P}$. The goal is to make sure that the pendulum state locations stay inside of the support polygon.

5.1 Support Polygon

Every i-th foot defines a set of vertices D_i local to its transform. Given multiple sets of these vertices transformed by their foot transform, we get a (not necessarily convex) set of n vertices $D = (D_1|D_2|\dots)$ that span the support polygon. With vertex activations $R_t \in \mathbb{R}^n$ and vertex weights $P_t \in \mathbb{R}^n$ at a given time t, we define

$$p_t = W \cdot diag(R_t) \cdot P_t \tag{5.1}$$

where p denotes the desired CoP location inside the support polygon. For each step, the components of R_t are given by the foot's total sampled weight $\bar{w}$ as described in the chapter Footstep Generation, meaning they are at least 1 when the foot is on the ground and close to 0 when in mid-air. While not all vertices in W are guaranteed to be part of the convex hull, if we enforce the constraint

$$R_t^T P_t = 1,$$

the equation (5.1) inexpensively approximates a convex combination. With this, p_t is either inside or reasonably close to the support polygon. Given the pendulum mechanics from equation (3.1) in chapter IPC Dynamics, we describe an optimisation problem where we're looking for optimal footsteps V,

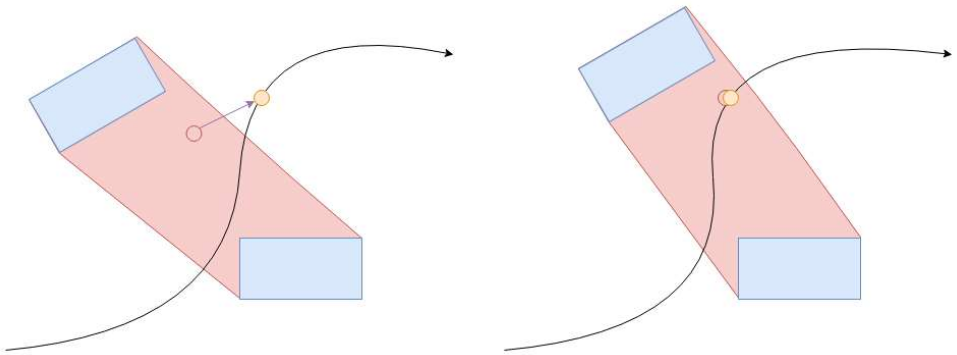

Figure 5.1: Optimisation of the trajectory (orange), foot transform (blue) and convex combination (red). Showcase of possible states before update step (left) and after (right). All three systems move at once.

pendulum force response q_t, and vertex weight P_t for all simulation steps t. It is defined such that the sum of all square distances of the resulting convex combination p_t and the position of the pendulum state s_t are minimal.

5.2 Gradient Descent

We optimise the target pendulum state q of $\mathcal{P}$, footsteps V, and the support polygon weights P using *Stochastic Gradient Descent* (SGD). As described above, we want to optimise for the pendulum trajectory to match the convex combination of the support polygon. Our Pendulum dynamics equation (3.4) and equation (5.1) give rise to the objective function for all simulation states

$$E = \sum_t ||p_t - s_t||^2.$$

This results in an update step that only depends on the first derivative of our systems

$$g = -\lambda \begin{pmatrix} \frac{dE}{dq} & \frac{dE}{dV} & \frac{dE}{dP} \end{pmatrix}$$

with some stepsize λ that is determined through line search. Higher order derivatives would be possible (e.g. Newton's method), but would also result in derivatives that depend both on V and q which are hard and costly to compute. Non-smooth convex hull and foot step constraints are enforced after every optimisation step. While this enforcement slows down descent, the overall optimisation still reliably converges to a local minimum within the constraints. New simulation steps are added continuously, which means for

a big trajectory window each point becomes relevant only after many simulation steps. Due to this continuity, we only need to do very few updates each simulation step.

5.3 Base Pendulum

Thanks to this continuity, our optimisation is robust and fast, but also reacts slowly to input changes from the user. We introduce a *base pendulum* $\mathcal{P}_c$ whose trajectory we use as the base for both the pendulum trajectory optimisation and footstep generation. For the trajectory, we define q from the IPC equation LQR as a *transformation* of a local state q' by a state s_c from $\mathcal{P}_c$ with

$$q = s_c + R(\varphi_c, q')$$

where R rotates directional properties of q' such as position, velocity, and pendulum angle according to its orientation φ_c. Having a target state q' local to a different pendulum has the advantage that characteristics of that optimisation persist through changes of the base pendulum $\mathcal{P}_c$. This base pendulum in turn is very flexible to inputs from the user. Additionally, q' remains relatively small and responds well to regularisation, i.e. gradient descent is less likely to get stuck in undesirable local minima.

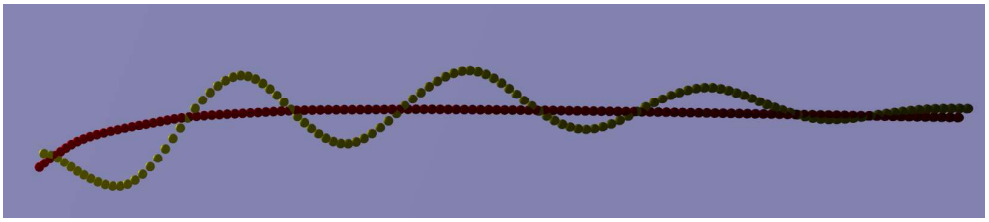

Figure 5.2: The base pendulum (red) transforms a local force response to the final pendulum trajectory (yellow).

5.3.1 Footstep Base

Footsteps, like the pendulum, can be transformed by a base $c_i = \begin{pmatrix} x_c & y_c & \varphi_c \end{pmatrix}$ describing the base location and rotation. Note that these values can come from the same base pendulum $\mathcal{P}_c$ discussed above. The number of footsteps (columns) in V is significantly smaller than the number of simulation steps in the pendulum. For that reason, we instead consider a four-dimensional B-spline that is fit to the pendulum states with *Smallest Square Distance*. Consider the spline knots C_m where for the pendulum states $S_c = (s_1|s_2|\ldots)$ we have

$$C_m = \underset{C}{\operatorname{argmin}} ||C \cdot T - S_c||_F^2,$$

where $T = \left(T_1|T_2|\ldots\right)$ contains all the timings $t_i \in [0,1]$ for each pendulum simulation step with the B-spline definition

$$T_i = \begin{pmatrix} (1-t_i)^2 \\ t_i \cdot (1-t_i) \\ t_i \cdot (1-t_i) \\ t_i^2 \end{pmatrix}.$$

With $c_i = C_m \cdot T_{\mu_i}$, one can sample a base for the footsteps approximating the base pendulum $\mathcal{P}_c$ at any time. The low-dimensionality of C_m is a desirable property as it ignores small perturbations in the pendulum trajectory.

5.3.2 Synchronisation

Both the base pendulum $\mathcal{P}_c$ and the CoM pendulum $\mathcal{P}$ simulate independently (The base pendulum $\mathcal{P}_c$ only transforms the force response, not the states themselves) and can very easily diverge from each other. To prevent this, we move the first state of the base pendulum to the first state of the CoM pendulum each simulation step. This ensures the continuity of the CoM pendulum and prevents divergence. To reduce sway, we use a B-Spline approximation of $\mathcal{P}$ similar to the approach discussed above. Implementation has shown that it is important that only the pendulum location, angle, and height are synchronised. Overriding velocities introduces feedback-loops where the character keeps walking in place.

Chapter 6

Motion Synthesis

6.1 Inverse Kinematics

For simplicity our motion synthesis is implemented using an *Inverse Kinematics* (IK) solver[1]. This is sufficient since we only consider the motion of the legs. The final foot trajectories are already computed by our optimisation algorithm. The IK solver uses *Newton's Method* (similar to the *Jacobian Transpose Method*[Bus04]) for optimisation with objectives outlined below.

End-Effector Objective: End-effector locations are given as ground contacts (e.g. three per foot) which are already conveniently computed by the support polygon optimisation (equation (5.1)). With multiple end-effectors spanning each foot, this solves for both their location and rotation.

CoM Objective: The pendulum trajectory models the center of mass which doesn't trivially correspond to a specific bone from the character armature. Instead, we introduce a CoM objective whose Jacobian matrix describes the relationship between all degrees of freedom (joint torques and root translation) and the center of mass. This objective alone only translates the CoM of the character to a desired location depending on the pose. There is no direct equivalent for the rotation of the CoM, thus we apply it directly to the pelvis (root) bone. Note that a more sophisticated objective approximating the character's orientation from its pose could be used for better looking results while turning.

Smooth Motion Objective: To counteract erratic movement, the smooth motion objective adds an inertia objective to all joint angles; it punishes changes in angular velocity. To prevent foot sliding, the objective weight is set smaller the closer it is to the feet. This is not a problem since the end-effector objec-

[1] There are well established, physics-based approaches such as full-body motion control [ZCY+15] that could be applied as well.

tive already guarantees smooth movement for the feet[2].

Joint Limit Objective: Joint limits are important as to prevent overextending joint (e.g. the knee) and preventing saddle points in the objective function during optimisation.

6.2 Foot Roll

Every limb has an *anchor point* relative to the pendulum's center of mass where it is attached to the pelvis/chest. Considering the closest point on a foot's ground contacts D_i we construct a rotation axis and a pivot. Rotating around this pivot works for both swing and ground phase. The angle depends on the distance between the anchor, the foot trajectory height, and the distance from the pivot to the foot's origin. For simplicity, we neglect foot twist around the up-vector.

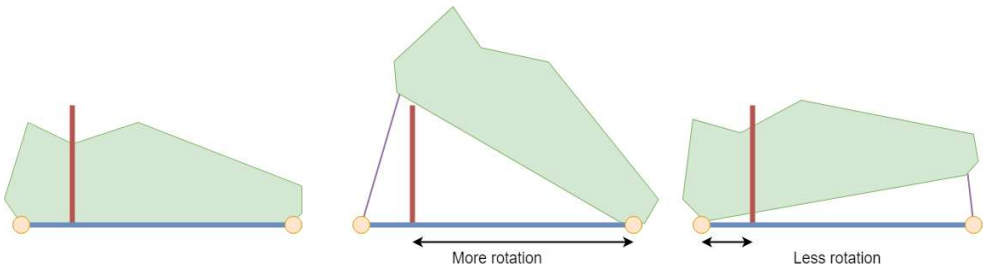

Figure 6.1: The heel being closer to the ankle rotates the foot less than when standing on toes.

The foot trajectory height is given mostly by the weight $\bar{w}$ computed in the chapter Footstep Generation. It is significantly enhanced by the foot roll to be similar to human gait [IMRS09] as illustrated in figure 6.2.

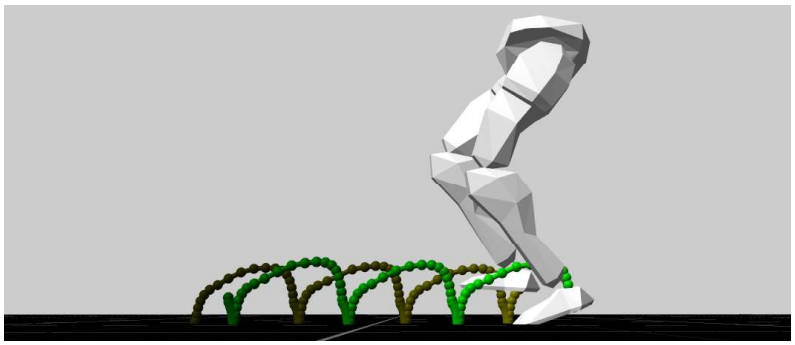

Figure 6.2: Trajectory of the ankle computed both from foot roll and height

[2]The End-Effector Objectives are given by the computed foot trajectories which are smooth.

Chapter 7

Results and Discussion

We demonstrate our system on a selection of robots with different leg configurations. The upper body and other appendages are omitted since we are only interested in leg motion. The robots were created in Blender [Com21] using a custom robot export extension. All joints have multiple rotation axes and angle limits to support a selection of gaits. Every limb has different amounts of end-effectors for the ground contacts, resulting in varying support polygon shapes.

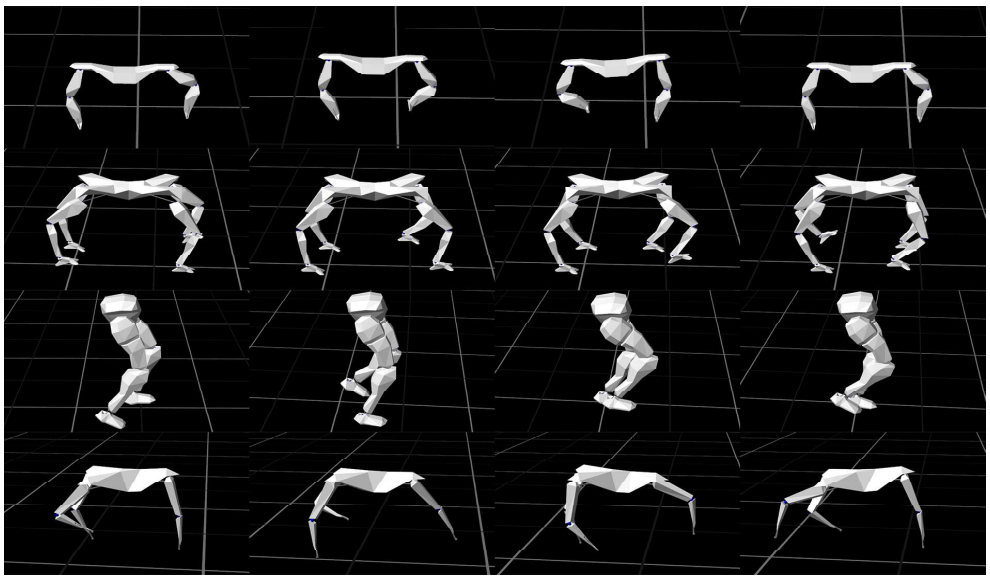

Figure 7.1: Different robot models of varying sizes and leg configurations in the same order: Tablebot, Quadbot, Bipedbot, Crawlbot

In our simulation the user can change input and control parameters such as heading direction and target velocity. Given a valid gait pattern, our system

generates the character motion without any further input. Obstacles and unsafe stepping zones are defined as a first-order differentiable heatmap.

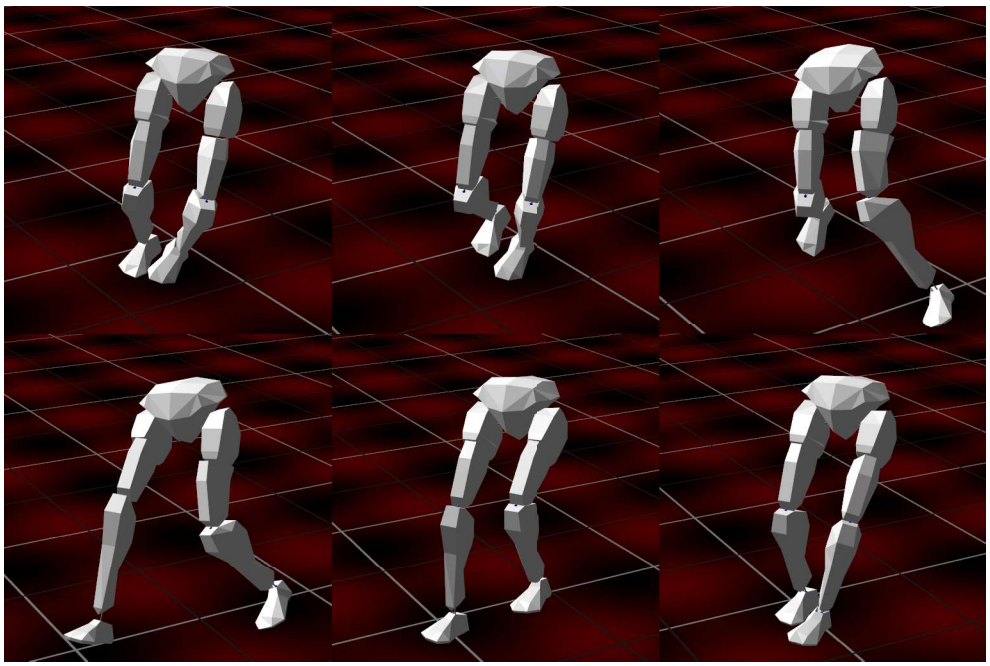

Figure 7.2: Step generation tries to avoid high values in the heatmap (red). Our other foot objectives prevent foot overlapping.

7.1 Support Polygon

For evaluating our support polygon optimisation, we plot the footsteps, convex combination, and the current CoP on top of the support polygon at any simulation step. Since we use an approximation of the support polygon, we represent it as a heatmap; each pixel represents a point in space, and we compute the heat value by optimising the convex combination of the current support polygon towards that pixel by measuring the distance. In figure 7.3, we see that our approximation creates ”fuzzy” edges but also sufficiently approximates the actual support polygon.

As mentioned in the chapter Optimisation, we optimise our system so that the convex combination of the support polygon and the *Center of Pressure* (CoP) are as close to each other as possible while respecting the dynamics of the pendulum. It is expected that this requirement cannot always be fulfilled especially for our bipedal robots and would be impossible for monopedal robots. As illustrated in figure 7.4, the CoP trajectory is pulled towards the fast moving support polygon creating ”pinches” in the trajectory. Due

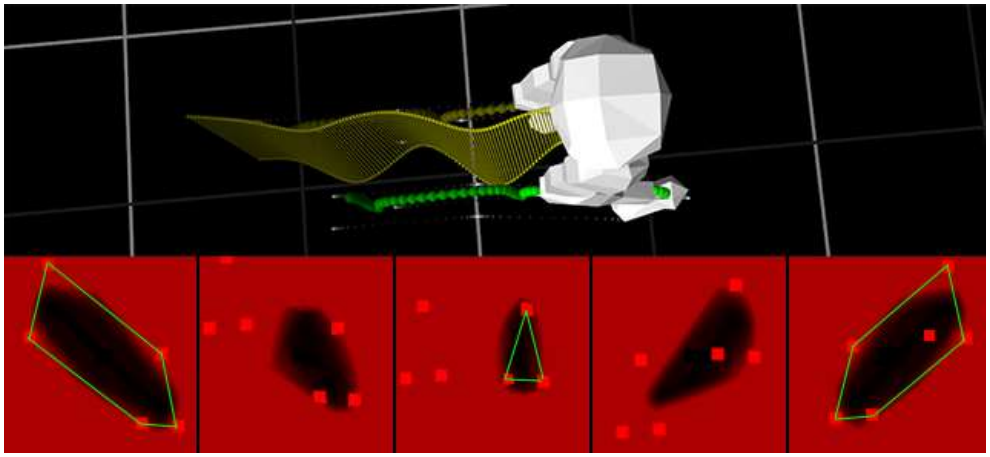

Figure 7.3: Demonstration of the support polygon approximation on our bidepal robot. Red dots are contact points, green lines illustrate the actual convex hull. The contact point weights are smoothly interpolated between different gait poses according to the step sampling from *Footstep Generation*. Notice the contact point in the last frame that is not part of the convex hull, yet our approximation holds.

to also respecting pendulum dynamics, the robot "loses balance" and the CoP temporarily leaves the support polygon. To fulfill the support polygon requirement, the robot would have to move slower, i.e. move significantly slower than the target velocity as shown in figure 7.5.

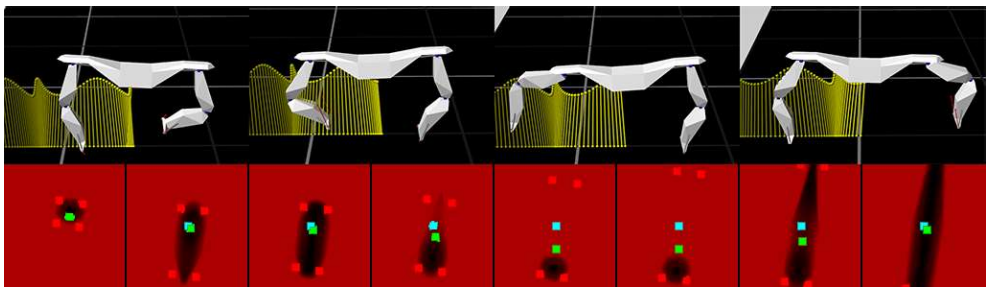

Figure 7.4: The CoP trajectory (cyan) does not always overlap with its convex combination counterpart (green) due to being constrained by pendulum dynamics.

7.2 Discussion

Using the *Inverted Pendulum on a Cart* model for our simplified dynamics was indeed a simple and robust choice. Despite the added dynamics, it comes with inherent problems that would need to be addressed at least with additional objective functions. Rotating the character around the pendulum axis using a dampened spring model does not take body physics and current step locations into account; the character will overextend their limbs if their

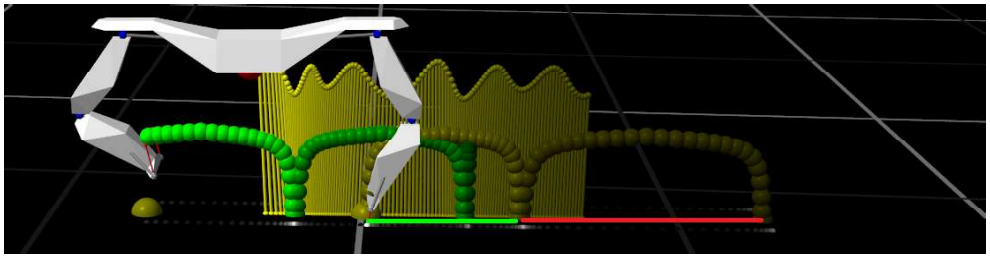

Figure 7.5: The support polygon objective requires the robot to go slower. If we increase the footstep weight, the optimisation is much more inclined to move the footstep locations. As a result, the starting trajectory length (red) is much longer than optimised (green).

heading direction is changed too quickly. Additionally, as the character is modelled as a point mass, they can appear hovering (floating) on stronger movement changes. This project presents a proof of concept of using our extended IPC model and support polygon optimisation, but it could be greatly enhanced using additional dynamics (such as rigid-body simulation and full-body control). The continuous nature of this system makes it very suitable for adaptation to another dynamics model; it allows direct alteration of its initial state parameters without invalidating previous optimisation results.

Chapter 8

Appendix

8.1 Appendix IPC

Following further definitions for the models used in the IPC Dynamics chapter.

8.1.1 Pendulum P

The Inverted Pendulum on a Cart model P describes a system where a point-mass on an inverted pendulum is fixed to a movable cart. In two dimensions, i.e. fixed to a plane, it has two degrees of freedom – the position p and pendulum angle θ. Given small θ the pendulum dynamics can be approximated by a linear system [CSC06]. Given the pendulum length L, inertia I_p, mass m and gravitational constant g we define the pendulum dynamics as such[1]:

$$\dot{s_x} = P_A \cdot s_x + P_B \cdot u_x$$

$$\begin{pmatrix} \dot{p_x} \\ \ddot{p_x} \\ \dot{\theta_x} \\ \ddot{\theta_x} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mgL^2}{I_p} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{mgL}{I_p} & 0 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ \dot{p_x} \\ \theta_x \\ \dot{\theta_x} \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} + \frac{L^2}{I_p} \\ 0 \\ \frac{h}{I_p} \end{pmatrix} \cdot u_x$$

Equivalently for s_y, the state vector $s_y = \begin{pmatrix} p_y & \dot{p_y} & \theta_y & \dot{\theta_y} \end{pmatrix}$ describes the beforementioned degrees of freedom and u_x the force applied to the cart. For small angles θ_x or θ_y (which is already a requirement for the linearisation) a three dimensional system can be approximated by simulating two of these systems for each axis (x and y) independently.

[1] for simplicity we assume no friction and zero cart mass, for a more detailed model see reference

8.1.2 Rotation R

The rotation around the pendulum axis is trivially defined by a damping coefficient c and inertia I_r with the following dynamics model:

$$\dot{s_r} = R_A \cdot s_r + R_B \cdot u_r$$

$$\begin{pmatrix} \dot{\varphi} \\ \ddot{\varphi} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -c \end{pmatrix} \cdot \begin{pmatrix} \varphi \\ \dot{\varphi} \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{I_r} \end{pmatrix} \cdot u_r$$

The state vector $s_r = \begin{pmatrix} \varphi & \dot{\varphi} \end{pmatrix}$ describes the angle and angular velocity around the pendulum axis and u_r the applied torque. Together with the pendulum states x and y from the previous subsection we can now define a two dimensional euclidean transformation $T_F(\varphi, p_x, p_y)$ that describes the position and rotation of the pendulum in floor space (i.e. projected onto the z-plane).

8.1.3 Height H

The height of the pendulum is defined independently of its length L to not introduce a non-linear element to Px and Py. This approximation produces satisfactory results for values close to L. We define a non-damped spring model that has a constant element for gravity as follows:

$$\dot{s_h} = H_A \cdot s_h + H_B \cdot u_h$$

$$\begin{pmatrix} \dot{z} \\ \ddot{z} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & g \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} z \\ \dot{z} \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} \\ 0 \end{pmatrix} \cdot u_h$$

After LQR optimisation the resulting force response reminds of critical damping. Together with the two dynamic models from the previous subsections we can compute the location of the *Center of Mass* in 3D space.

Chapter 9

Acknowledgement

I want to thank my supervisor Stelian Coros for his continued support of my thesis work. His direct involvement and encouragement was greatly appreciated and made working on this project fun and engaging even through a worldwide pandemic.

Bibliography

[Bus04] Samuel Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Transactions in Robotics and Automation*, 17, 05 2004.

[Clo97] J. R. Cloutier. State-dependent riccati equation techniques: an overview. In *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, volume 2, pages 932–936 vol.2, 1997.

[Com21] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2021.

[CSC06] R. Chanchareon, V. Sangveraphunsiri, and S. Chantranuwathana. Tracking control of an inverted pendulum using computed feedback linearization technique. In *2006 IEEE Conference on Robotics, Automation and Mechatronics*, pages 1–6, 2006.

[IMRS09] Fumiya Iida, Yohei Minekawa, Juergen Rummel, and Andre Seyfarth. Toward a human-like biped robot with compliant legs. *Robotics and Autonomous Systems*, 57:139–144, 02 2009.

[KH10] Taesoo Kwon and Jessica Hodgins. Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, page 129–138, Goslar, DEU, 2010. Eurographics Association.

[KH17] Taesoo Kwon and Jessica K. Hodgins. Momentum-mapped inverted pendulum models for controlling dynamic human motions. *ACM Trans. Graph.*, 36(4), January 2017.

[KLVDP20] Taesoo Kwon, Yoonsang Lee, and Michiel Van De Panne. Fast and flexible multilegged locomotion using learned centroidal dynamics. *ACM Trans. Graph.*, 39(4), July 2020.

[MdLH10] Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.*, 29(4), July 2010.

[ZCY+15] Daniel Zimmermann, Stelian Coros, Yuting Ye, Robert W. Sumner, and Markus Gross. Hierarchical planning and control for complex motor tasks. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 73–81, New York, NY, USA, 2015. Association for Computing Machinery.

[Zim13] Daniel Zimmermann. Complex locomotion tasks for physically-simulated virtual humans. 2013.