

Aalto University  
School of Science  
Degree Programme in Computer Science and Engineering

Jussi Perämäki

# Interactive Control of a Simulated Biker

Master's Thesis  
Espoo, November 2, 2015

Supervisor: Assistant Professor Perttu Hämäläinen  
Advisors: Antti Mannisto M.Sc. (Tech.)  
Joose Rajamäki M.Sc. (Tech.)

<b>Author:</b>	Jussi Perämäki	
<b>Title:</b>	Interactive Control of a Simulated Biker	
<b>Date:</b>	November 2, 2015	<b>Pages:</b> vii + 44
<b>Major:</b>	Media Technology	<b>Code:</b> T-111
<b>Supervisor:</b>	Assistant Professor Perttu Hämäläinen	
<b>Advisors:</b>	Antti Mannisto M.Sc. (Tech.) Joose Rajamäki M.Sc. (Tech.)	
<p>Procedural animation in video games is becoming more and more common. It brings realism to games where physics engines are heavily used, GTA V is a good current day example of this. Procedural animation can be done with online motion synthesis. However, research on online motion synthesis is rarely applied in video game industry. One reason for this is that online motion synthesis can be very resource demanding.</p> <p>In this thesis we explore if the control of bikes in videogames can be improved with online motion synthesis. This is accomplished by applying a pre-existing algorithm for online motion synthesis to a simulated biker and testing it in various situations we thought would simulate gameplay situations.</p> <p>Based on the results we got from testing we concluded that the algorithm could be suitable for applications in video games. However, the algorithm used can be very resource demanding, which is not acceptable for gaming applications. With correct optimizations to the simulated characters and performance improvements to the algorithm video game industry could benefit from an online motion synthesis algorithm like the one used.</p>		
<b>Keywords:</b>	animation, motion synthesis, video game controls, motorcycle simulation	
<b>Language:</b>	English	

<b>Tekijä:</b>	Jussi Perämäki		
<b>Työn nimi:</b>	Simuloidun moottoripyöräilijän interaktiivinen ohjaus		
<b>Päiväys:</b>	2. marraskuuta 2015	<b>Sivumäärä:</b>	vii + 44
<b>Pääaine:</b>	Mediatekniikka	<b>Koodi:</b>	T-111
<b>Valvoja:</b>	Apulaisprofessori Perttu Hämäläinen		
<b>Ohjaajat:</b>	Diplomi-insinööri Antti Mannisto Diplomi-insinööri Joose Rajamäki		
<p>Proseduraalinen animaatio videopeleissä on yleistymässä. Se lisää realismia peleissä, joissa fysiikkamoottorien käyttö on runsasta — hyvänä nykypäivän esimerkkinä tästä on GTA V -videopeli. Proseduraalinen animaatio pystytään toteuttamaan ajonaikaisella liikesynteesillä. Tuloksia ajonaikaisen liikesynteesin tutkimuksesta ei kuitenkaan kovin yleisesti käytetä peliteollisuudessa. Yksi syy tähän on se, että se on laskennallisesti raskasta.</p> <p>Tässä diplomityössä tutkimme, voiko videopeleissä esiintyvien moottoripyörien ohjausta parantaa ajonaikaisella liikesynteesillä. Tutkimus toteutetaan hyödyntämällä jo olemassaolevaa ajonaikaisen liikesynteesin algoritmia moottoripyörän ohjauksessa ja testaamalla toteutusta erilaisissa tilanteissa, joiden uskomme vastaavan erilaisia peleissä esiintyviä tilanteita.</p> <p>Testituloksien pohjalta saavuin johtopäätökseen, että käyttämäämme algoritmia voisi olla mahdollista käyttää videopeleissä. Ongelmaksi voi silti muodostua algoritmin laskennallinen vaativuus. Optimoimalla simuloituja hahmoja ja parantamalla algoritmin suorituskykyä uskoisimme, että peliteollisuus voisi kuitenkin hyötyä käyttämästämme ajonaikaisen liikesynteesin algoritmista.</p>			
<b>Asiasanat:</b>	animaatio, liikesynteesi, videopelien ohjaus, moottoripyöräsimulaatio		
<b>Kieli:</b>	Englanti		

# Acknowledgements

I wish to thank my thesis supervisor, professor Perttu Hämäläinen for giving me the opportunity to work on this thesis, the original idea for the thesis, and the advice and guidance through the process. I learned a lot during the project and it showed me a whole new area of study.

In addition, I would also like to thank my instructors Joose Rajamäki and Antti Mannisto for guidance and insight. I would also like to thank Antti Ilvessuo for being my contact in the games industry and providing insight into motorcycling games.

Finally I would like to thank my friends, family, and especially Satu for all the support and patience through the thesis.

Espoo, November 2, 2015

Jussi Perämäki

# Abbreviations and Acronyms

API	Application Programming Interface
COM	Center of Mass
C-PBP	Control Particle Belief Propagation
DDP	Differential Dynamic Programming
DOF	Degrees of Freedom
IDE	Integrated Development Environment
IK	Inverse Kinematics
NEAT	Neuroevolution of Augmenting Topologies
ODE	Open Dynamics Engine
PD-controller	Proportional-Derivative Controller
PID-controller	Proportional-Integral-Derivative Controller

# Contents

<b>Abbreviations and Acronyms</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>4</b>
2.1 Bicycle and motorcycle dynamics . . . . .	4
2.2 Teaching computers how to ride a bike . . . . .	6
2.2.1 PID controllers . . . . .	7
2.2.2 Reinforcement learning . . . . .	7
2.3 Online motion synthesis . . . . .	8
2.3.1 Past research on bipedal locomotion synthesis . . . . .	8
2.3.2 Differential Dynamic Programming . . . . .	10
2.3.3 Control Particle Belief Propagation . . . . .	11
<b>3 Environment</b>	<b>13</b>
3.1 Unity . . . . .	13
3.2 Open Dynamics Engine . . . . .	13
3.3 UnityODE . . . . .	14
<b>4 Implementation</b>	<b>15</b>
4.1 Overview . . . . .	15
4.1.1 Player Controls . . . . .	17
4.1.2 Levels . . . . .	18

4.2	Character . . . . .	21
4.3	Controllers . . . . .	23
4.3.1	Optimizer . . . . .	25
4.3.2	Arcade . . . . .	28
<b>5</b>	<b>Evaluation</b>	<b>30</b>
5.1	Testing . . . . .	30
5.1.1	Qualitative analysis . . . . .	30
5.1.2	Automatic testing . . . . .	31
5.2	Results . . . . .	31
5.3	Remarks . . . . .	34
<b>6</b>	<b>Discussion and Conclusions</b>	<b>36</b>
6.1	Implementation . . . . .	36
6.2	Testing and results . . . . .	37
6.3	Conclusions . . . . .	39
6.4	Future work . . . . .	39

# Chapter 1

## Introduction

This thesis investigates the problem of real-time interactive control of a simulated motorcycle ("the bike"). Bicycles and motorcycles are an integral part of most cultures. The bicycle was invented in 1818 and it became popular in the 1860s [20]. Nowadays almost everyone owns a bicycle or can at least ride one. Motorized bicycles or motorcycles were invented around the same time bicycles became popular. Bicycle racing and motorcycle racing have existed from 1860s [32] and 1904 [8] respectively. Driving or racing video games have existed since 1973 when Atari released Space Race. In our opinion the most important aspect of a racing game are the controls of the vehicles used. The controls play a big part in the feel of the game.

In video games players rarely have direct control over the individual parts of the character and only give the character high level instructions like *turn left* or *move forward*. Because of high level instructions, the character movements are often implemented with animation as that is the simplest way to produce them. Good example of high level instructions are racing games. Player controls the acceleration and steering to drive the vehicles. There are also games focused on the precise control of individual body parts. In a game called QWOP the player controls the playable character's thighs and calves [9] to finish a 100 meter dash at Olympic Games. Another example is Octodad, a game where the playable character is an octopus that tries to act like a person and the player has to move its tentacles as legs and arms [45]. In Surgeon Simulator the player controls the right hand fingers, wrist, and arm of a surgeon to perform various operations in different situations.[5]. These games either control a part of the playable character at the same time (Octodad) or only a limited set of character's parts (QWOP, Surgeon Simulator). Controlling a whole character with that level of detail would

need a large amount of controls, which is hard for the player to handle. In this thesis we try to achieve something in between the high level instructions and high level precision, leaving the precision in the hands of the computer and high level controls to the player.

Most animation in video games is premade with keyframes, motion capture or scripting. Premade animation does not allow dynamic adjustment when things do not go as they were planned. For example, a character falls down onto an object but does not change pose. This leads to the character leaning on the object in a non-natural way. One way of avoiding this is ragdolling, where the character goes limp and does not exert any force via its limbs. Some dynamic adjustment can also be done with inverse kinematics (IK) [28] and some engines like Euphoria from NaturalMotion [26] can dynamically synthesize animation for characters reacting to their surroundings. Euphoria is used for example, in GTA IV [25] video game to handle characters being drunk or falling and handling sudden disturbances that happen to the characters.

Online motion synthesis is used to create motion and animation at runtime to, for example, to handle unexpected situations like falls and disturbances to the characters. Some methods of online motion synthesis use pre-computed data or keyframes and some do not, the common thing is that the motion itself is generated at run-time. It is a well researched subject, which we will show in Chapter 2, and its use in the game industry is becoming more common. Euphoria is a good example of the usage of motion synthesis in the game industry.

The goal of this thesis is to apply a pre-existing online optimization algorithm for humanoid character control to a motorbike and its driver. After that, testing will be done to see if controlling the bike with the optimizer-based control method feels more natural or works better than with arcade mechanics. Arcade mechanics in this context are simple balance equations and external forces that are used to steer the bike and keep it upright. Whether *the control of bicycles/motorcycles in video games can be improved with online motion synthesis* is studied in this thesis.

We start by introducing the background bicycle and motorcycle dynamics and showing what has been done to make computers ride simulated bicycles. Then we continue to look into the research of online motion synthesis and focus on the algorithm we use in this thesis in Chapter 2. When the prerequisites of the implementation have been introduced we move on to Chapter 3 where we describe our tools used and continue to describe our implementation, problems solved, evaluation, and testing in Chapters 4 and 5. Finally,

in the last chapter we gather all this together and discuss the thesis as whole, draw our conclusions, and think about future work.

## Chapter 2

# Background

In this chapter we look into the dynamics research on bicycles and motorcycles, see what has been done to teach computers to drive simulated bicycles, glance at online motion synthesis research, and then focus on the online motion synthesis method we will use in this thesis. First part of the goal of this thesis is to simulate a bike so we begin with the bicycle dynamics research. Second part is the online motion synthesis added to the bike. There has not been much research into motion synthesis with vehicles, mainly just on bipedal locomotion. Bipedal locomotion is a well researched subject, as shown later in section 2.3.1, and one of the areas it focuses in is balancing. This aspect makes us deem the algorithms developed for bipedal locomotion also at least partly applicable to our research.

### 2.1 Bicycle and motorcycle dynamics

Extensive reviews of bicycle dynamics studies have been performed before by Hand [14], Sharp [33] and Meijaard et al. [20]. Our focus is not the research on bicycle dynamics, but to find the information that we need to build a working bike, so we do only a shallow look on the research on bicycle dynamics.

To steer a bicycle or a motorcycle, the driver must first turn to the opposite direction to push the vehicle to a fall towards the side he wants to turn and then balance the bike by steering to the desired direction. This maneuver is called counter-steering [7] and without it the bicycle would just fall to the opposite direction of the desired turn.

The first study about bicycle dynamics was published 1869 by Rankine

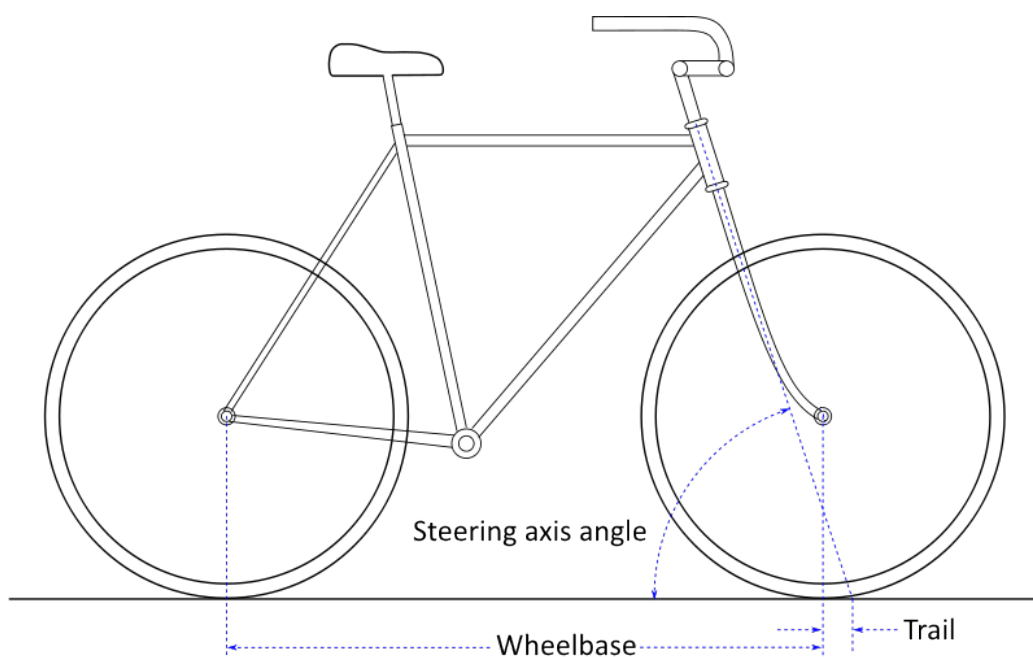


Figure 2.1: Dimensions of an average present day bike [18], featuring similarly sized wheels, bent front fork and a tilted steer axis. Wheelbase is the distance of the wheels' ground contact points from each other.

[30]. A large amount of studies has been performed since, the study by Jones [17] being the most well-known. Nowadays the dynamics of bicycles and motorcycles are well understood.

Bicycle got its current form, shown in Figure 2.1, in 1890s characterized by wheels of equal size, pedals connected to the rear wheel with a chain and a tilted steer axis with a bent front fork. This model is also known as the safety bicycle. The term is now considered obsolete, because the bicycle design became the standard. The bent fork and tilted steer axis produce a trail which is important for the controllability of the bicycle [17]. The trail is the horizontal distance from where the steering axis intersects the ground to where the front wheel touches the ground. Trail is positive if the ground contact point is behind the intersection point and negative in the opposite situation.

In 1970 Jones experimented with the safety bicycle to see if gyroscopic effect of the front wheel is important for the stability of the bicycle [17]. He reported that he could ride the bicycle easily, but it was not self-stable anymore. He also experimented with the trail, increasing and making it negative to see how it affected the control of the bicycle. Larger positive

trail made the bicycle more stable but harder to steer and negative trail made the bicycle have negative self-stability and difficult to steer. From this Jones concluded, that gyroscopic effect was less important than trail for bicycle stability.

Fajans [7] used a simplified version of the mathematical model of bicycle steering equations. He studied steering with bicycles and motorcycles and confirmed that gyroscopical forces play only a limited role in balancing and steering as concluded by Jones previously. He also mentions that counter-steering and hip thrusts are the most common ways of creating a lean to prepare for a turn, but other ways exist also. Rider can leverage uneven surfaces, push one pedal harder than other, or accelerate with the wheel turned.

## 2.2 Teaching computers how to ride a bike

Computers have been used to simulate bicycle dynamics since 1970, when Jones wrote Fortran subroutine called BICYC to solve trigonometrical equations related to bicycles and then simulate different bicycle geometries [17]. To make computers ride simulated bicycles several methods have been used: Simple proportional-derivative controller for the handlebar angle [16], which we introduce later in section 2.2.1, and reinforcement learning [29], introduced in section 2.2.2, combined with policy searches [27] [38] for the joint control values of the rider.

Hodgins et al. studied unicycles [15] and later demonstrated [16] that normal cycling activities, like balance and steering, could be achieved with a simple proportional-derivative control for the handlebar angle. Randløv and Alstrøm [29] used reinforcement learning and shaping to teach a bicycle first to balance itself and then to drive to a goal. Their algorithm needed thousands of simulation trials to reach the target and result was still suboptimal. Their bike's target was 1 kilometer away and it took from 1.7 to 7 kilometers of driving for their bike to get there. Ng and Jordan [27] used the same bicycle simulator and applied policy search to the same problem. Their approach yielded significantly better results than the study from Randløv and Alstrøm [29], with results of from 995 meters to 1070 meters. Tan et al. [38] took on a task to teach bicycle stunts to a computer. They used policy search for Partially Observable Markov Decision Process and Neuroevolution of Augmenting Topologies (NEAT) [36] to teach their controller to perform stunts like wheelie, endo, and bunny hop. Their offline learning process taught

their controller to perform these stunts in minutes to be then performed in an online simulation with or without user interaction. Unfortunately their controller could not perform more than one of these tricks at the same time as each trick was taught to the controller individually.

### 2.2.1 PID controllers

Proportional-integral-derivative (PID) controller [21] is a control loop feedback controller, which continuously calculates an error value as the difference between desired value and measured value. The proportional part accounts for present error  $e$ , integral part for past error, and derivative part predicts the future error. The whole function is shown in Equation 2.1. PID controllers were originally developed for automatic ship steering at the beginning of the twentieth century and presently are used in various control systems. One of the first PID-type controllers was developed by Perry in 1911 [4] and first analysis of PID controllers was done in 1922 by Minorsky [21]. In computer science PID controllers, or just parts of them, like PD controllers, are often used in humanoid joint simulations [44][16]. The three parts of the control each have their own tuning parameters ( $K_P$ ,  $K_i$ ,  $K_d$ ), which are adjusted to avoid oscillation with the controller. Oscillation is mainly caused by an incorrectly tuned integral part, so in P or PD controllers oscillation is less severe.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt} \quad (2.1)$$

### 2.2.2 Reinforcement learning

Reinforcement learning [37] is a form of machine learning and it uses software agents taking actions in an environment to maximize cumulative reward function given to the agents. In reinforcement learning the agent must discover which actions yield the best rewards by trying them. Reinforcement learning agents need also to be able to sense their environment somehow to make decisions on how to proceed. In some cases agents' actions may affect not only the immediate reward, but also all subsequent rewards. One of the challenges of reinforcement learning is the trade-off between trying new actions and using already proven actions. This exploration-exploitation dilemma is that with the usage of only one of these actions, the agent will not reach the goal state. Some policy of choosing when and how often which action has

to be used. The agent must try a variety of actions and progressively favour the combinations of actions that appear to be the best.

One key feature of reinforcement learning is that it explicitly considers the whole problem, unlike other approaches that consider subproblems without thinking how the solutions would fit into the whole problem. The agents in reinforcement learning have explicit goals and they plan their actions to reach the end goals.

## 2.3 Online motion synthesis

In online motion synthesis animation or motions are created at run-time with or without the help of offline learning, state machines or reference motions. Online motion synthesis is a hard problem but it is solvable as many studies have demonstrated. Often the methods for online motion synthesis are based on differential dynamic programming (DDP) [24][43], controllers with inverted pendulum models [6][22], or sampling [11][12][13]. Both DDP and some sampling-based approaches use the Bellman equation [3], which breaks the problem into small parts, which can then be solved in order. With Bellman equation the current optimal control can be calculated from current and future parts, without considering the past actions.

In this thesis we use Control Particle Belief Propagation (C-PBP) by Hämäläinen et al. [12] as our method for online motion synthesis. C-PBP is a sampling based method to generate interactive and physically valid humanoid movements without any offline training or reference data in near real-time. How C-PBP works will be explained later in Chapter 2.3.3.

### 2.3.1 Past research on bipedal locomotion synthesis

Most of the online motion synthesis research focuses on bipedal locomotion and balancing. C-PBP is also originally demonstrated with humanoid character movements. Here we offer a brief overview of some of the past research before C-PBP.

The control of a bipedal locomotion is a challenging problem. The challenge is reinforced by the need of multiple types of gait, stylized motions, reactions to variable terrain, and reactions to external disturbances to the character. In addition to simulated bipeds, the results from motion synthesis research can be applied to robotics.

Morimoto et al. [24] developed a robust control policy design and method for high-dimensional state spaces by using differential dynamic programming with a minimax criterion. They applied their controller to a simulated biped robot, which performed better in walking with unknown disturbances than a controller generated by standard differential dynamic programming. They also applied their controller to a real biped robot to optimize the controller.

SIMBICON by Yin et al. [44] was the first framework to demonstrate a large set of integrated, physically-simulated bipedal skills. It uses a simple finite state machine, with requirements and feedback error learning, to create controllers that can be applied to 2D and 3D bipeds based on motion capture data. They demonstrate their method with controllers walking in all directions, running, skipping and hopping.

Coros et al. [6] presented a control strategy for physically-simulated walking motions. Their method works by integrating tracking, using proportional-derivative control foot placement, using an inverted pendulum model, and adjustments for gravity and velocity errors, using Jacobian transpose control, a method for solving IK problems. With their method character proportions and motion styles can be authored interactively. Editing the character or motion styles realize instantly in a suitable controller. They demonstrated that naive users can use their method to author character proportions, gait parameters, and motion styles.

Mordatch et al. [22] presented a physics based locomotion controller based on online planning with a high level interaction from user specified goals using a low-dimensional model. Their planner used a simplified preview model based on Spring-Load Inverted Pendulum [10] model to optimize the trajectory and then apply changes to the full-body character, that its center of mass (COM) follows instantaneous plan accelerations while maintaining balance. They demonstrated their controller with different types of gaits, including walking, running and jumping. These gaits emerged automatically from their fixed 6 phase preview schedule, which is based on the Spring-Load Inverted Pendulum. Schedule contained 3 phases for each leg: Double-stance with both feet on the ground, single-stance with one foot, and flight. For example walking emerged from alternating double-stance and single-stance and running emerged from alternating single-stance and flight. They showed that effective online motion planning of highly dynamic behaviour can be performed using a low-dimensional model.

Wu et al. [43] described a framework that produces interactive biped locomotion controllers that adapt to uneven terrain at run-time. Their framework consists of two components: a per-footstep end-effector planner and a

per-timestep generalized-force solver. Parameters for the planner are solved in offline optimizations and using the plan the solver uses quadratic programming to obtain joint torques to drive the biped at each timestep of the simulation. They demonstrate their controllers with complex navigation tasks including gradual and sharp turns, walking in uneven terrain, and moving forwards, backwards, and sideways as interactive task goals. Their controllers are also able to handle morphological changes to the character. The synthesis method is fully automatic and needs no manual tuning or captured motion data.

Ha et al. [11] demonstrated a sampling based method to generate agile and natural landing motions in real-time without any motion capture data or pre-scripted motion sequences. Their algorithm works in three stages: impact, rolling and getting-up. With an arbitrary initial position and velocity in the air, their method could devise a landing strategy and an optimal sequence of actions to achieve the desired landing velocity and angle of attack. Their method works with different body shapes, starting conditions, and environments. Real-time user generated perturbations are also supported. Their method avoids applying large stress on character's joints, which translates to the resulting movements to be safe for real humanoids.

Hämäläinen et al. [12] presented a Model-Predictive Control system for online synthesis of interactive and physically valid character motion. They demonstrate their algorithm in near real-time with a complex 3D human character who can balance in a given pose, dodge user generated projectiles, and improvise a strategy to get up if forced to lose balance. Their method works online, without any handcrafted state machines or reference motion data sets, which previously were required for motions as complex as they demonstrated.

Mordatch et al. [23] combined machine learning and trajectory optimization in a way that the optimization results sped up the machine learning process. The speeding up resulted from the trajectory optimization acting as a teacher to the machine learner. The combined optimizer did not only provide the trajectories to the machine learner but also feedback gains as training data to the learning network. They demonstrated their controller with rolling, reaching, swimming, and walking tasks.

### 2.3.2 Differential Dynamic Programming

DDP is an optimal control algorithm of the trajectory optimization class introduced by Mayne in 1966 [19]. DDP works iteratively by performing a

backward pass on the nominal trajectory to generate a new control sequence, and then a forward pass to compute and evaluate a new nominal trajectory. This is iterated until backward and forward passes converge. The main limitation of DDP is that it is based on a Taylor approximation of the dynamics near the current nominal trajectory, which means that it is not ideal for handling contact discontinuities and multimodality. Because DDP is differential, it can get stuck into local optima.

### 2.3.3 Control Particle Belief Propagation

C-PBP is a general-purpose Model-Predictive Control algorithm which combines multimodal, gradient-free sampling and a Markov Random Field factorization to effectively perform simultaneous path finding and smoothing in high-dimensional spaces [13]. Hämäläinen et al. demonstrated C-PBP with a humanoid that balances itself on a flat surface, balances on a ball, juggles a ball, recovers from small and extreme disturbances, reaches toward user defined targets, and performs fully steerable locomotion in an environment with obstacles. Their demonstration was built in Unity [41] using Open Dynamics Engine [35] and ran at 20-30 frames per second on a reasonably powerful desktop computer.

C-PBP works in three phases. First it samples control vectors with random walkers and uses dynamics simulation to obtain the corresponding next states for  $N$  trajectories. Then it smooths the optimal control obtained from the previous phase by recursive backwards local refinement. Lastly, it deploys the obtained control to the system and transmits information of the hypothesized time evolution to the next frame or iteration. During each simulation step, if the algorithm finds too many high cost trajectories, a resampling step is performed. In the resampling step high cost trajectories are cut at that timestep and resampling distributes the computing resources to forking the low cost trajectories.

In general, motion can be formulated as spacetime constrained optimization problem [42]. In C-PBP the animated character is guided with a cost function, minimizing the cost. It is used, for example, to keep the ball to be juggled on the character's head or at least on some body part. The cost function in this case gives higher cost when the ball is lower than head of the character and the cost is evaluated to zero if the head's position is lower than the position of the ball. All of the scenarios Hämäläinen et al. demonstrated C-PBP with, were done with individual cost functions and actions shown could not be combined in the demonstration. But combining the cost

functions or crafting specific ones, these actions could be combined, either dynamically at run-time or statically. Because C-PBP does not use any pre-computed data, machine learning, state machines or reference motion data sets, cost functions can be rapidly designed and tested.

Hämäläinen et al. believe that that online motion synthesis methods like C-BPB provide new opportunities for animation research and development of interactive experiences such as games. The latter is one of the reasons why we chose to use C-PBP in this thesis. Another benefit of using C-PBP is its adjustability. For example the user can easily adjust prediction horizon length, simulation timestep, cost function, resampling threshold, and prior values used inside Unity.

## Chapter 3

# Environment

In this chapter we present the tools used to implement the project. The tools used were already in use in the research group this thesis was done in, so the decision to use them came naturally to us.

### 3.1 Unity

Unity [41] is a cross-platform game engine with a build-in IDE by Unity Technologies. Its development started in 2001 and its first version was released 2005 and has become very popular among game developers. While being a game engine, it is also used for non-game purposes like research and animation. Unity version used in this project was 5.0.2f1 32bit Personal Edition.

### 3.2 Open Dynamics Engine

Open Dynamics Engine (ODE) [35] is an open source, high performance library for simulating rigid body dynamics. Version used in this project was 0.12. We used ODE because it is open source and because of its fast Coulomb friction model implementation and capability for preserving angular momenta. ODE also provides means to model the physics of multiple trajectories simultaneously, which the physics engine of Unity can not do as it can run only one physics world at a time. Also, the implementation of C-PBP used was built using ODE [12].

### 3.3 UnityODE

UnityODE is a wrapper for ODE's C Application Programming Interface (API), which provides access to ODE in C# from Unity. It is written in C++ and most of the C# wrapping is generated automatically using SWIG. UnityOde was written by Perttu Hämäläinen and Sebastian Eriksson for online motion synthesis studies using Sequential Monte Carlo sampling [12] and C-PBP [13].

## Chapter 4

# Implementation

In this chapter we describe our implementation and how we solved the problems we encountered. We describe the simulation model of the playable character and the different controllers used to control the character and we explain our control setup and different levels used to test the implementation.

### 4.1 Overview

We implemented a game where the player can ride a motorcycle. C-PBP optimizer was used to find approximately optimal control values based on player input. Our intention was to make the bike balance itself but not in all situations, so that the player would have some challenge driving the bike. In situations like colliding forcefully with objects and making sharp, almost 90 degree, turns the bike should not stay in balance.

We built a biker with a bike in Unity using ODE. The bike and biker formed a system of articulated rigid bodies. Biker's hands were connected to the handlebar and feet to the part of bike's body that acted as the foot rest of real motorcycles. As C-PBP requires one to simulate the physics forward with different sampled controls, physics simulation complexity determines how many samples one may draw, and therefore, how good controls the method is able to find. We reduced the dimensions of the character to include just one leg, one arm, and one bone for the torso. This way the computation was less demanding and we could draw more samples to get better results from the optimizer.

As the focus of this thesis is to test C-PBP applied to a bike and biker system, we needed to devise how player input was used to control the bike.

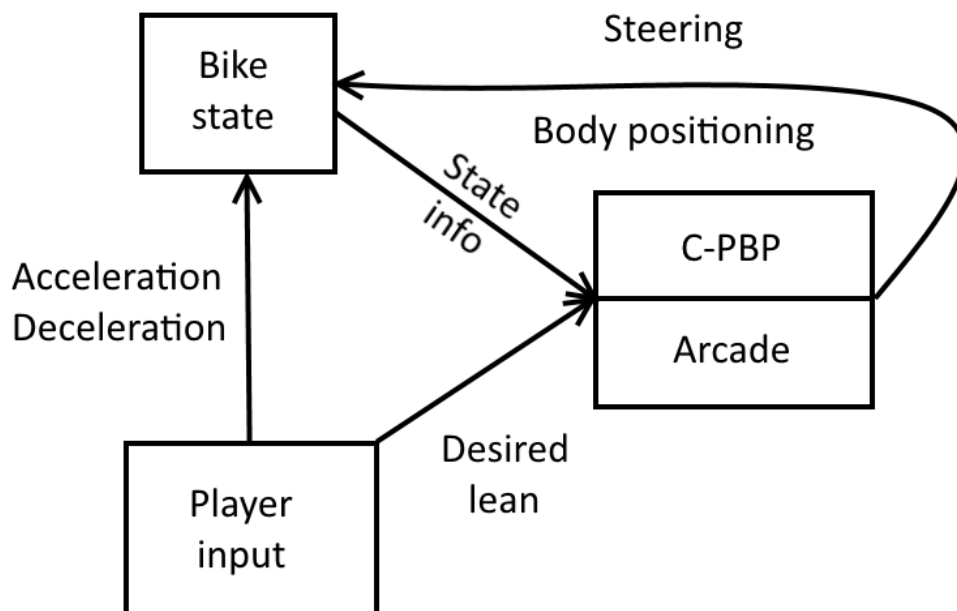


Figure 4.1: Simplified diagram of the control scheme.

Direct control over the steering of the bike does not work because of counter-steering, which is needed to maintain balance while riding a bicycle or a motorcycle. According to our initial testing, counter-steering precisely enough is not possible using consumer game controllers. That is because the movements in counter-steering are so small and quick that the controllers struggle to provide enough control to the players. In addition to this, most people do not even realize that counter-steering exist, even if they can ride bicycles. Therefore to steer, the player controls the desired lean angle and the optimizer handles the positioning of the biker and steering to match that angle, as can be seen in diagram shown in Figure 4.1.

To evaluate the C-PBP bike implementation we created a so called arcade controller as a baseline for comparing the control methods. It does not use the C-PBP optimizer to calculate the desired values for the character and bike joints. Instead, it works by calculating the front fork steering angle from the desired lean angle to keep the bike in balance. In other words it is a simple PD controller for the steering angle. In addition to the optimizer-

only controller and arcade controller, we have a combined controller which uses the arcade PD controller calculations as a warm start trajectory for the optimizer, explained in Chapter 4.3. To test the different controllers we implemented three levels that focused on different aspects of driving the bike.

### 4.1.1 Player Controls

To make the controls feel as responsive as possible, we let the user directly control acceleration and braking (the angular velocity of the back wheel ODE motor), and also control the desired lean, which is then realized through the arcade or C-PBP control system. The challenge for the player is to handle the bike delicately enough for the desired leaning to be possible given the throttle and the environment. To be precise, the player does not control the lean directly, but via adjusting the turning radius. The lean is then calculated from the turning radius and speed using equations specified in Section 4.3 and shown to the player.

An Xbox 360 controller, layout shown in Figure 4.2, is used for controlling the game. To adjust the desired lean of the bike player uses left stick of the controller. Right trigger is used for acceleration and left trigger for braking. The position of the camera could be rotated around the bike by turning the right stick left or right. Switching between different control methods happens with the start button and back button pauses the game and opens the pause menu. The B button is used to reset the bike's position and rotation back to current spawn point and the A button is used to lift the bike back upright on the spot. Left and right bumpers are used to switch between multiple spawn points, if those were implemented to the level. For automatic testing, the X button is used to turn automatic acceleration on. When automatic acceleration is turned on the bike drives forwards with maximum acceleration.

Acceleration, braking, steering, and camera controls were chosen as those are usually the default controls in games. The reset button was influenced by the Trials [31] series and other buttons were added to buttons that were free at the time as those controls were not as important for driving.

We also implemented a way for the player to tweak different variables of the bike in-game. Selection of simulation complexity (C-PBP trajectory amount) and the level used are possible at the start of the game. Other variables like switching trajectory visualization on and off, showing the physical or 3D model of the character, and adjusting motor forces and maximum speed could be accessed from the pause menu. In the pause menu player could move between menu items with vertical movement of the left stick and

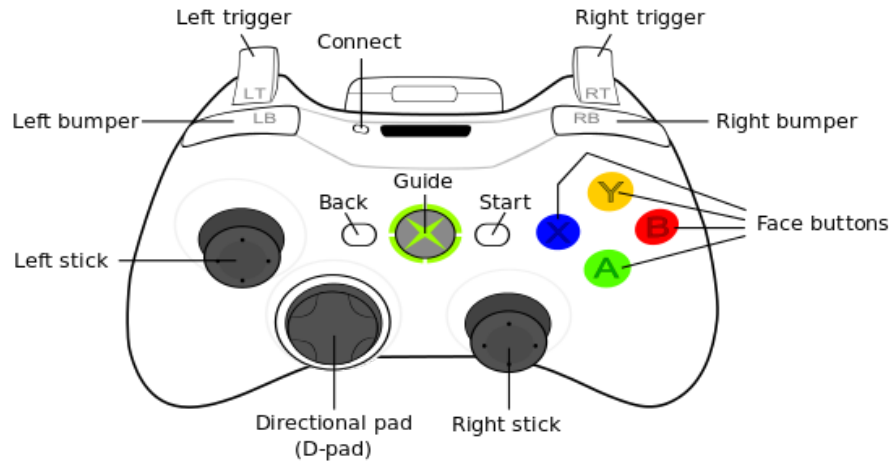


Figure 4.2: Xbox 360 controller layout [1].

adjust the items with horizontal movement of the left stick.

### 4.1.2 Levels

Three playable levels were made for the game. One for automatic testing, shown in Figure 4.3, and two for driving around to test various aspects of driving the bike. The automatic testing level featured an environment of randomly laid out bumps on the ground, two walls, and a goal area. Player's goal is to drive through the terrain to the goal area at the end without falling over or getting stuck to the walls. The first one of the manual levels, shown in Figure 4.4, is a timed track that player is supposed to complete as fast as possible, so the focus of this level is on speed. It featured long stretches of level ground, ramps, hills, and a drop. The aim of the first level is used to test how well the bike could be controlled in high velocity situations, like a jump from a ramp as shown in Figure 4.6. The other one of the manual levels is a skate park-like level, shown in Figure 4.5, in which fine control played bigger part than speed. It features small ramps, platforms, a small hill, and low obstacles to get over. This level is used for testing low velocity precise movements and tricks like wheelie and lifting the bike onto a platform, as can be seen in Figure 4.7. The two manual levels had multiple spawn points, which were scattered around the levels and the player could move the reset position freely between the spawn points.

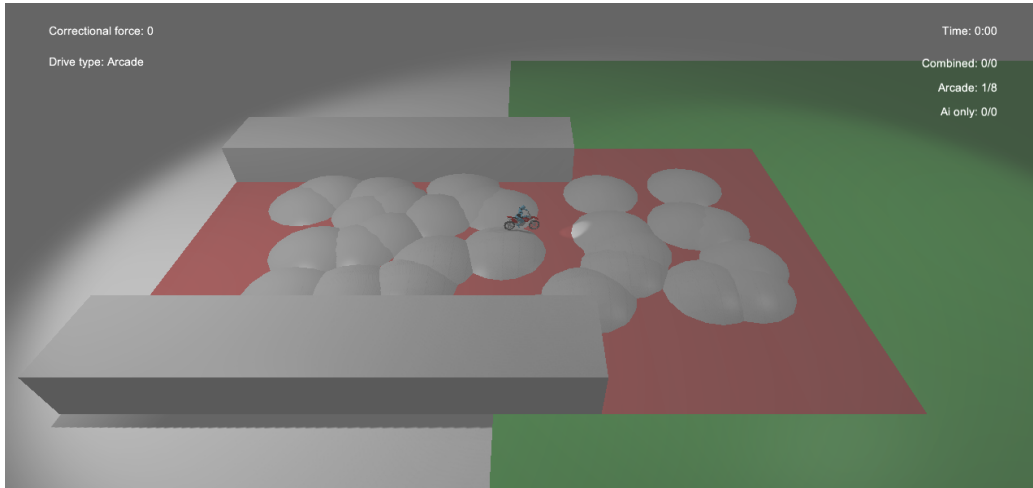


Figure 4.3: Test level. Green area is the goal area and red area is the area the character needs to pass through in the appointed time limit. The character starts from the left.

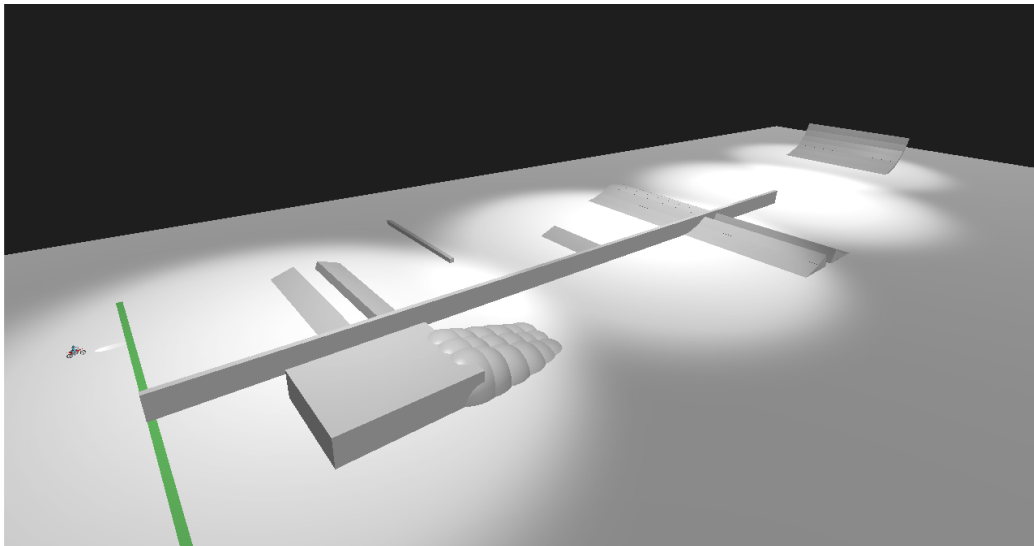


Figure 4.4: Track level. Green line is the starting/goal line and player is supposed to drive around the track counterclockwise. The character is shown at the starting point.

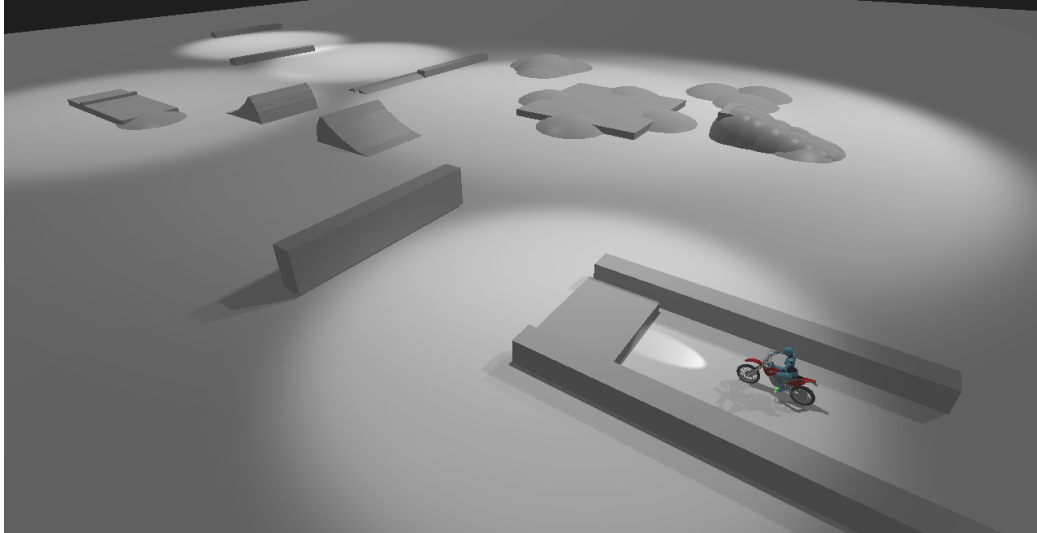


Figure 4.5: Skate park level. Focused on fine control over the bike.

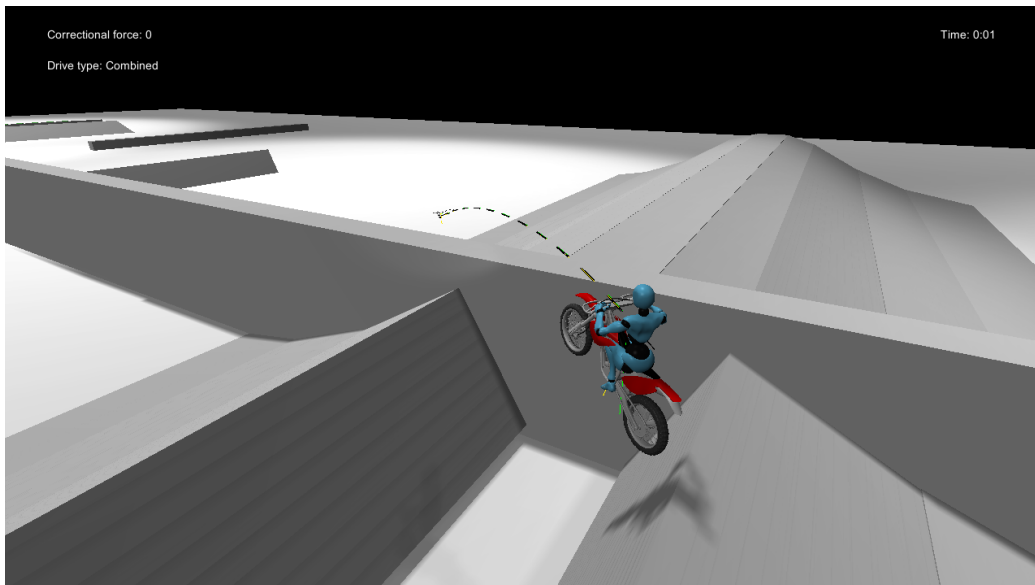


Figure 4.6: Highlight from the track level. Player performing a risky jump, trajectories show that it will land on top of the wall.

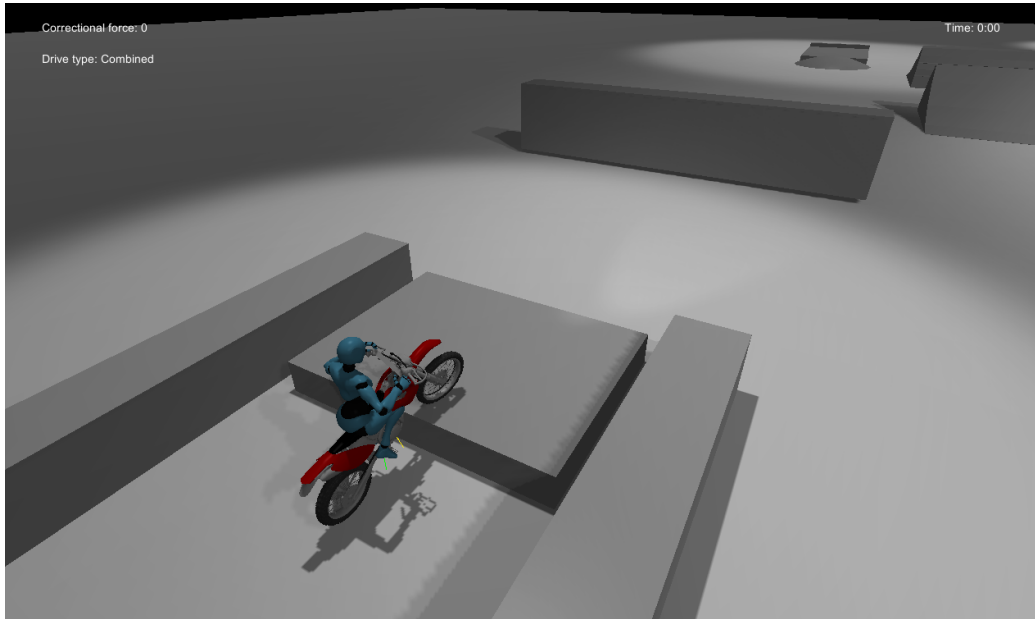


Figure 4.7: Highlight from the skate park level. Player rising on top of a platform. To achieve this player has to accelerate rapidly to perform a wheelie and then balance it until the platform.

## 4.2 Character

The playable character in the project is a biker with a motorbike. The character uses a simplified humanoid model, which consists of upper and lower arm, torso, and upper and lower leg. Because humanoids are left-right symmetric we could simplify the model to lower the complexity of the simulation. The symmetricity allows us simplify the simulation without losing much precision. Because the character is essentially on a vertical plane, we made it so that the character's body parts could not collide with each other. 3-Degrees of Freedom (DOF) ball joints were used at wrist, shoulder, waist, and ankle while 1-DOF hinge joints were used at elbow and knee. These joints had human-like limits. The bike itself has three 1-DOF hinge joints at back wheel axis, front fork/handlebar axis, and front wheel axis. Optimizer used the front fork axis, back wheel axis, and all the character joints except the ankle, in total of 13 joint parameters and 5 tuning parameters for different classes of joints. Tuning parameters were used to limit the forces applied on the joints. To simplify the simulation, the bike has no springs or suspension implemented. Accelerating and braking work by directly adjusting the target angular velocity of the back wheel.

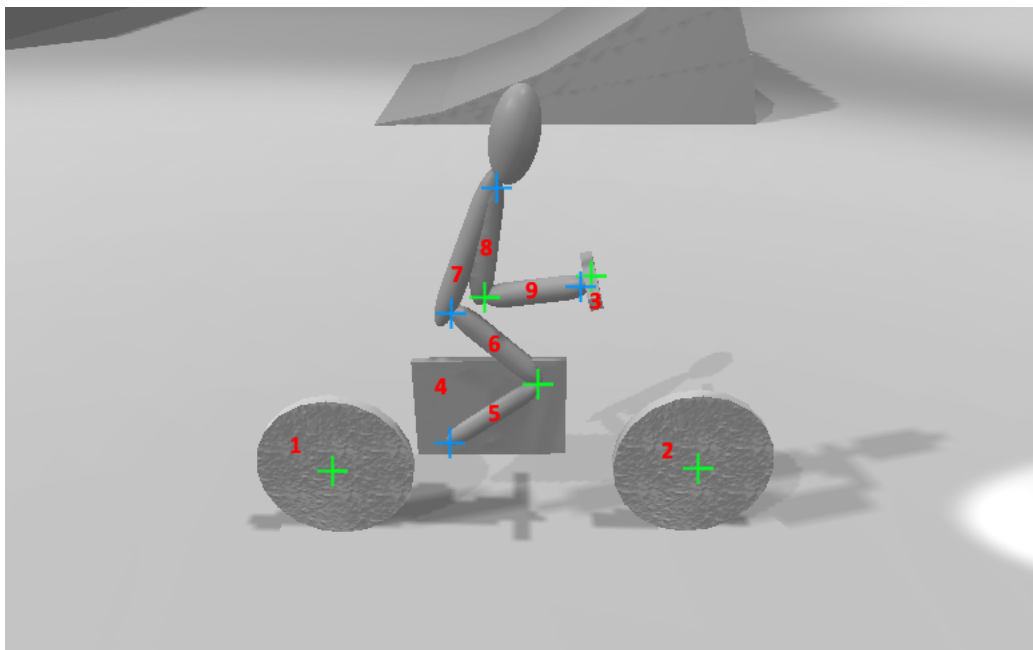


Figure 4.8: Physical model of the character. Bike consists of wheels (1,2), handle bar (3), bike's body (4). Biker consists of leg (5,6), torso (7), and arm (8,9). Green plus-signs are hinge joints and blue signs are ball joints.

A detailed 3D model of a motorcycle was provided by a Finnish game company to be applied on top of the physical model of the bike. A humanoid robot model was applied on top of the physical one-armed and one-legged character model and then animated through IK provided by Unity [40]. Full character is shown in Figure 4.8. In Figure 4.9 we can see the 3D model of the character with the physical model hidden.

The character tries to keep a preset pose while maintaining balance with the bike. To keep balance the COM of the character has to move around, which translates to natural looking motions with the IK character model. Control of the character is executed through target angular velocity parameters of its ODE joints motors.



Figure 4.9: Visual model of the character consisting of the bike 3D model and biker 3D model, which were used to hide the physical representation of the playable character.

### 4.3 Controllers

To control the character, its joints' target angular velocities are tuned via the optimizer and the arcade controller. There is also a third controller, which is a combination of the arcade and the optimizer controllers. The optimizer uses two trajectories for warm starting the algorithm in every frame, the first one is the best trajectory from the last frame and the second one is a C-PBP's attempt of refining the trajectory with local linear models of the last frame's trajectories. In the combined method we replace the second warm start with the arcade control.

In-game the player can have the trajectories visualized to help to see where he might end up in the immediate future. The visualized trajectories are divided in three classes, example shown in Figure 4.10. The green one is the previous best trajectory, the one that the character is most likely to follow, the yellow one is either the local refinement done by C-PBP or the arcade trajectory. The black ones are the other trajectories that were generated by C-PBP for this frame.

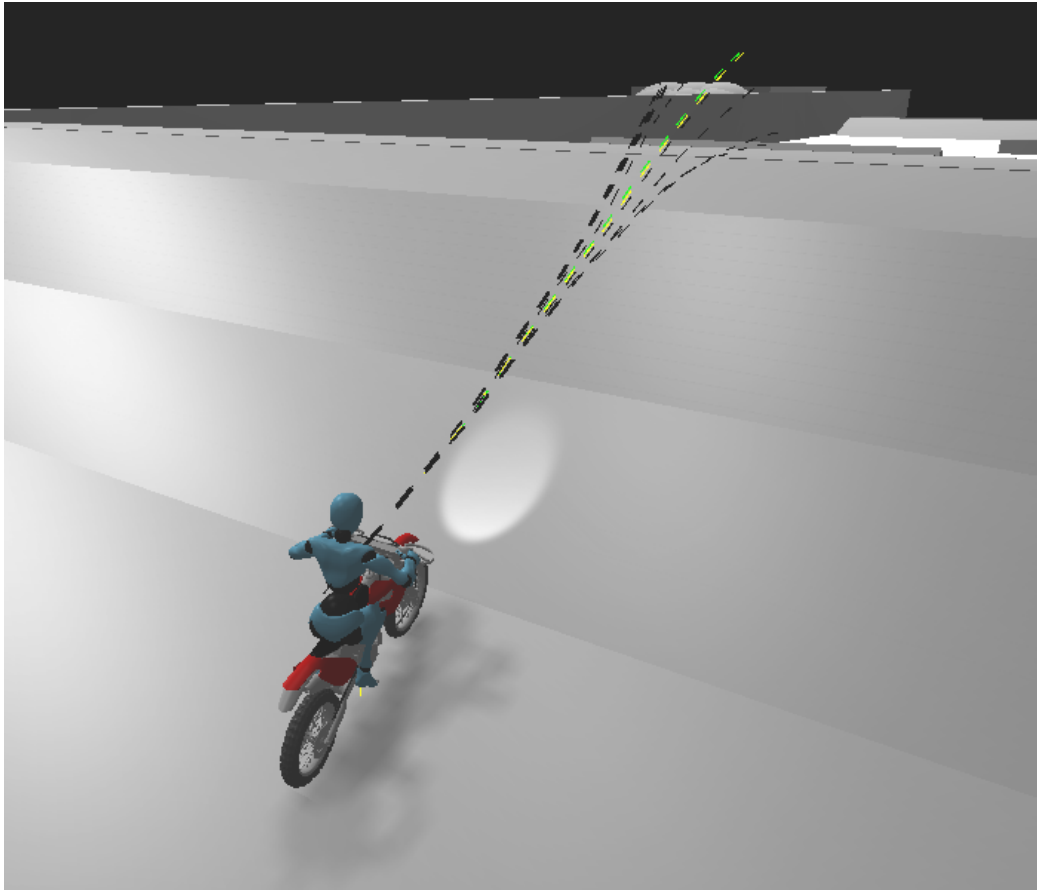


Figure 4.10: Close up of the calculated trajectories. The green trajectory is the last frame's best trajectory, the yellow is the local refinement done by C-PBP or the arcade trajectory, and the black trajectories are other trajectories calculated by C-PBP for current frame.

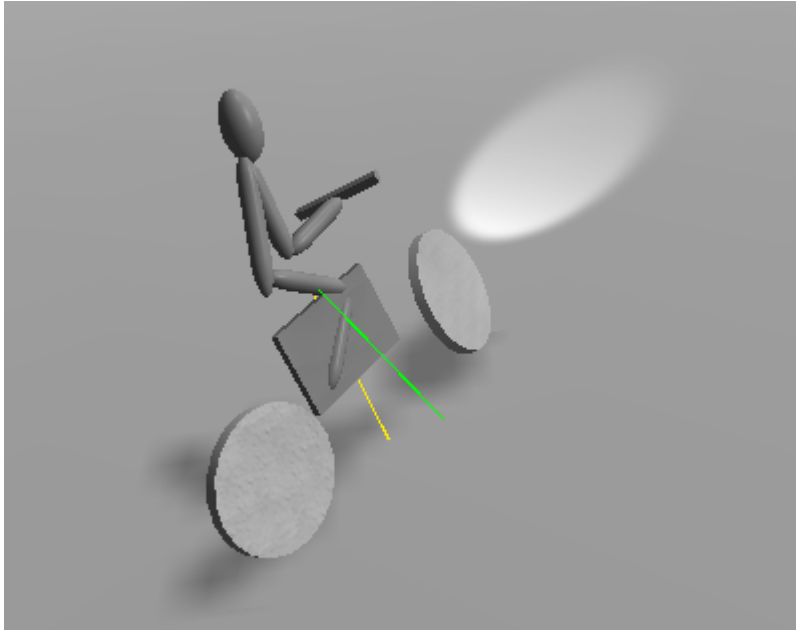


Figure 4.11: Desired (green) and current (yellow) lean vectors, used to calculate the cost. Current lean angle is the angle between global down vector and current local down vector of the bike. Desired lean angle is used to rotate the global down vector around bike’s forward axis to get the desired lean vector.

### 4.3.1 Optimizer

The optimizer calculates possible trajectories for the bike to take. A trajectory consists of simulation timesteps, which all have their own control values for the bike’s joints. For each frame the C-PBP optimizer calculates a set of trajectories and evaluates the cost of each trajectory. A cost of a trajectory is the sum of the costs of its steps. From this the optimizer chooses the lowest cost trajectory and applies the control values from it to the next frame of the simulation.

Control over the character from C-PBP is based on a cost function, which evaluates the current state of the simulation and compares it to the user input. The cost function evaluates the squared difference between actual and desired lean angles, shown in Figure 4.11. For the cost function to evaluate the difference it will first calculate the desired lean angle from user input. Circular motion equations, shown in Figure 4.12, are used to calculate the desired lean angle  $\theta$  from bike’s speed  $v$ , turn radius  $r$ , and gravity  $g$ .

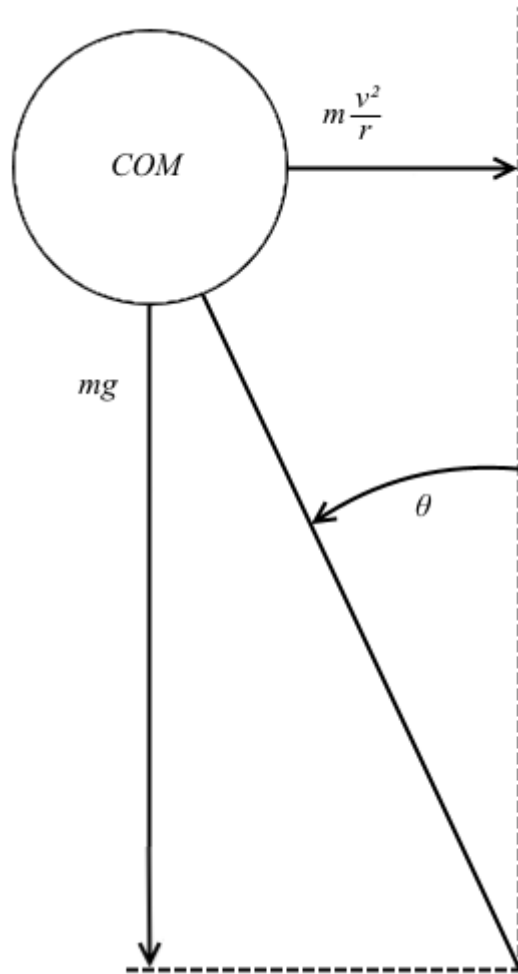


Figure 4.12: The forces affecting the bike in a turn.  $COM$  is the center of mass for the bike,  $mg$  is the gravitational force and  $mv^2r^{-1}$  is the inertial force.  $\theta$  is the lean angle.

From the laws of circular motion we get the inertial force,

$$F_c = \frac{mv^2}{r} \quad (4.1)$$

which we then combine with the gravitational force,

$$F_g = mg \quad (4.2)$$

to form the equation for the lean angle. As seen in Figure 4.12 the geometry implies that the inertial and gravitational are related as a fraction.

$$\tan(\theta) = \frac{F_c}{F_g} = \frac{\frac{mv^2}{r}}{mg} = \frac{v^2}{gr} \quad (4.3)$$

In Equation 4.1 for the inertial force  $F_c$  from laws of circular motion,  $m$  is mass,  $v$  is speed, and  $r$  is the turn radius. It is used together with Equation 4.2 for the gravitational force  $F_g$ , where  $m$  is mass and  $g$  is gravity, to form Equation 4.3. In Equation 4.3  $\theta$  is the desired lean angle. Equation 4.3 comes from applying inertial and gravitational forces to COM of the bike and the resulting force points towards the support point as seen in Figure 4.12. The angle between resulting vector and up vector is the lean angle. From Equation 4.3 we can solve the desired lean angle as shown in Equation 4.4.

$$\theta = \arctan\left(\frac{v^2}{rg}\right) \quad (4.4)$$

To calculate the difference between desired lean angle and current lean angle, we must rotate the global down vector  $\mathbf{u}_d$  around the bike's forward axis  $\mathbf{b}_f$  by  $\theta$  degrees. To rotate the vector we must first construct the rotation matrix shown in Equation 4.5 from the desired lean angle and forward vector. After rotating the down vector in Equation 4.6, the resulting vector is the desired lean vector  $\mathbf{b}_d$ .

$$\mathbf{R} = \cos \theta \mathbf{I} + \sin \theta [\mathbf{b}_f]_{\times} + (1 - \cos \theta) \mathbf{b}_f \otimes \mathbf{b}_f \quad (4.5)$$

$$\mathbf{b}_d = \mathbf{R} \mathbf{u}_d \quad (4.6)$$

In Equations 4.5 and 4.6 we form a rotation matrix from a direction vector and an angle using Rodrigues' rotation formula and then rotate the

downwards vector with it to get the desired lean vector. In Equation 4.5,  $\mathbf{R}$  is the rotation matrix,  $\mathbf{b}_f$  is the forward vector of the bike,  $[\mathbf{b}_f]_\times$  is the cross product matrix of  $\mathbf{b}_f$ ,  $\otimes$  is the tensor product, and  $\mathbf{I}$  is the identity matrix. In Equation 4.6  $\mathbf{b}_d$  is the desired lean vector of the bike and  $\mathbf{u}_d$  is the global down vector.

Now we can calculate the cost  $c$  from the desired lean vector and the bike's down vector, which is the vector from the COM of the bike,  $\mathbf{x}_{\text{com}}$ , to the ground support point of the bike,  $\mathbf{x}_s$ . In Equation 4.7 the cost is calculate by dividing the squared angle between these two vectors by the square of  $s$ , which is the standard deviation value for leaning.

$$c = \frac{(\angle(\mathbf{b}_d, \widehat{\mathbf{x}_s - \mathbf{x}_{\text{com}}}))^2}{s^2} \quad (4.7)$$

In addition to steering cost, we use a pose cost that minimizes deviation from the initial pose. This is implemented as a pose prior similar to one implemented by Hämäläinen et al. [13]. For steering angular velocity the prior is set as the current angular velocity so that steering would not deviate too much from user input. Priors are used as the basis around which the optimizer samples values.

### 4.3.2 Arcade

Arcade controller is a PD controller that calculates the current lean angle and predicted lean angle based on the current and old lean angles using backward Euler method. We used a PD controller instead of just a P controller to reduce oscillation. PD controller worked well enough for us not to have to implement a PID controller.

Steering with the arcade controller works differently in two different situations. Firstly, when the front wheel has ground contact the bike is steerable, it uses one equation to calculate the steering. Secondly, when the bike does not have ground contact with its front wheel, it needs to match the steering angle to the bike's velocity to keep balance when ground contact happens.

If front wheel is in contact with the ground the controller tries to adjust the steering to minimize the error between the predicted and desired lean angles and the difference between current and previous errors.

$$\omega = \left(\alpha + \frac{\alpha - \alpha_o}{\delta t} p - \alpha_d\right) K_p + \frac{e - e_o}{\delta t} K_d \quad (4.8)$$

In Equation 4.8  $\omega$  is the steering angular velocity,  $\alpha$  is the current lean angle,  $\alpha_o$  is the previous lean angle,  $\alpha_d$  is the desired lean angle,  $p$  is the prediction horizon,  $K_p$  is the proportional coefficient of the PD controller,  $e$  is the current error,  $e_o$  is the previous error,  $K_d$  is the derivative coefficient and  $\delta t$  is the timestep. The coefficients were tuned with manual iterations until the bike did not oscillate after turns. The values we ended up for the coefficients were  $K_p = 10$  and  $K_d = 0.2$ .

The PD control assumes that turning the front wheel will have an effect on the leaning angle. This is not the case during jumps and other maneuvers where the front wheel is not in contact with the ground. Further, if the front wheel hits the ground in a non-neutral angle at the end of a jump, the bike behaves erratically. Hence, when there's no ground contact, we adjust the steering angle towards the neutral angle that one can solve from the bike balancing Equation 4.10 [39] given the current forward velocity  $v$  and lean angle  $\theta$ .

$$r = \frac{v^2}{\tan \theta g} \quad (4.9)$$

$$r = \frac{w \cos \theta}{\sigma \cos \phi} \quad (4.10)$$

Using the circular motion Equation 4.4 from previous section and bicycle dynamics Equation 4.10 for turning radius  $r$ , we can calculate the neutral steering angle  $\sigma$ . From Equation 4.4 we can solve for  $r$  to get Equation 4.9, which we can plug in to Equation 4.10 to get the Equation 4.11.

$$\frac{v^2}{\tan \theta g} = \frac{w \cos \theta}{\sigma \cos \phi} \quad (4.11)$$

$$\sigma = \frac{w \cos \theta \tan \theta g}{\cos \phi v^2} \quad (4.12)$$

In Equations 4.10, 4.11 and 4.12  $w$  is the wheelbase of the bike, which is shown in Figure 2.1,  $\sigma$  is the neutral steering angle,  $\phi$  the steering axis or the caster angle. The Equation 4.12 is a result of solving for  $\sigma$  in Equation 4.11.

## Chapter 5

# Evaluation

In this chapter we describe our testing methods and results of the tests and go through some remarks of the implementation process.

### 5.1 Testing

The evaluating we carried out consisted of two parts, qualitative analysis with the help of a game development company and automatic testing for quantitative data.

#### 5.1.1 Qualitative analysis

Two qualitative analysis sessions were arranged in co-operation with a Finnish game company focusing on motorcycle games. The first session focused on coming up with the features the bike should have. The second session then analyzed the features decided upon in the first session. Due to the implementation of the bike, it is to be controlled like a real bicycle, with no sudden motions, to maintain balance. Approaching the controls of the bike like it would be a usual video game bike, leads to the bike falling over very quickly.

The analysis showed us that the control setup is not as intuitive as usually in video games. Often in video games player's input translates directly into in-game actions one to one. In our case the steering of the bike lags behind a small amount due to the optimizer. Analysis of the feel of the control of the character revealed it to be natural and realistic. Difference between the control methods was not apparent, but the bike tends to fall over more easily with the arcade controller. The preferred control method

of our analysis group was the combined method, especially in low trajectory amount situations. Consensus was that it was the most stable and the easiest to control.

### 5.1.2 Automatic testing

To get some quantitative data, we set the bike to automatically drive forward through a testing track with all three different control methods. As there are no standardized testing metrics for the kind of testing we wanted to execute, we devised the following environment to test the control methods.

To test the different control methods we created an environment with a randomized layout of a fixed number of bumps in the terrain, shown in Figure 4.3. The bumps' layout was randomized each time the bike started a new test attempt on the track. The track was 25 meters wide and 40 meters long. For the first 25 meters it had walls on the sides making it 15 meters wide and after the walls the track opened up. The object of the test was for the bike to pass the track in a certain amount of seconds, we used 25 for this test. If the bike did not reach the goal area in time, the try was counted as a fail. 25 seconds was chosen because it allowed the bike to pass the test area and gave some buffer in case the bike had trouble driving straight. First we ran the tests with 20 second time window, as it would have been enough for the bike to pass the testing area at any of the used target speeds. But it ended up to be too little time for the bikes that collided with the walls and then recovered and drove towards the goal area.

For the test the bike accelerated to a set maximum speed and then drove without any steering input from the player, control methods only tried to keep it upright. We used speeds of 10, 15, 20, and 25 kilometers per hour. For the optimizer using methods we used 5, 10, and 15 trajectories in C-PBP. Each combination of settings was ran through the test for 500 tries. Test results can be seen in Figure 5.1. The chance of falling is reported with a 95% confidence interval in a Bernoulli distribution.

## 5.2 Results

Qualitative analysis gave us some basis on which it was easier to build levels. Game development professionals from a Finnish game company had opinions about what to include in the levels and about what we should be able to do with the bike. Their opinion was that to really test the bike we should

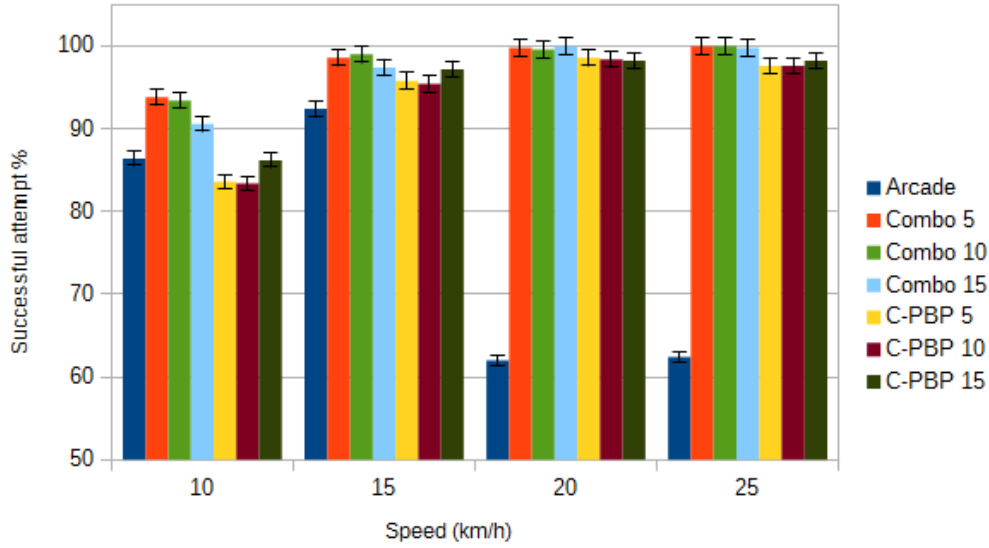


Figure 5.1: Automatic testing results. Combo is the combined method and C-PBP is the optimizer-only method. Different numbers after Combo and C-PBP are the trajectory amounts used. Black bars are the calculated error values.

have different kind of ramps to test the bike’s performance in high speed situations and to have the possibility to tweak bike’s variables during in-game. The implemented pause menu enabled player to tweak some of the bike’s variables. They also mentioned about the camera positioning and angle in relation to the player to be unusual. The camera angle shown in Figure 4.10, which was unusual.

In our tests 15 trajectories could be run on our laptop with Intel Core i7-2760QM @ 2.40GHz processor around 11 frames per second. A high amount of trajectories would be needed to rely just on the optimizer. Stable enough driving was achieved with 5 trajectories, combined control method and with physics timestep of 32 milliseconds in real-time and with a framerate of about 31. In this case most of the balancing was done with the arcade control part and optimizer mainly corrected deviations from the arcade control. Used prediction horizon for the optimizer was 1.2 seconds, which lead to each trajectory to have 37 steps.

The test results show that almost all combinations of optimizer-based methods and trajectories performed similarly well. Only the low speed (10 km/h) test had major differences between combined and optimizer-only methods. The arcade method was the worst performer of our test, perfor-

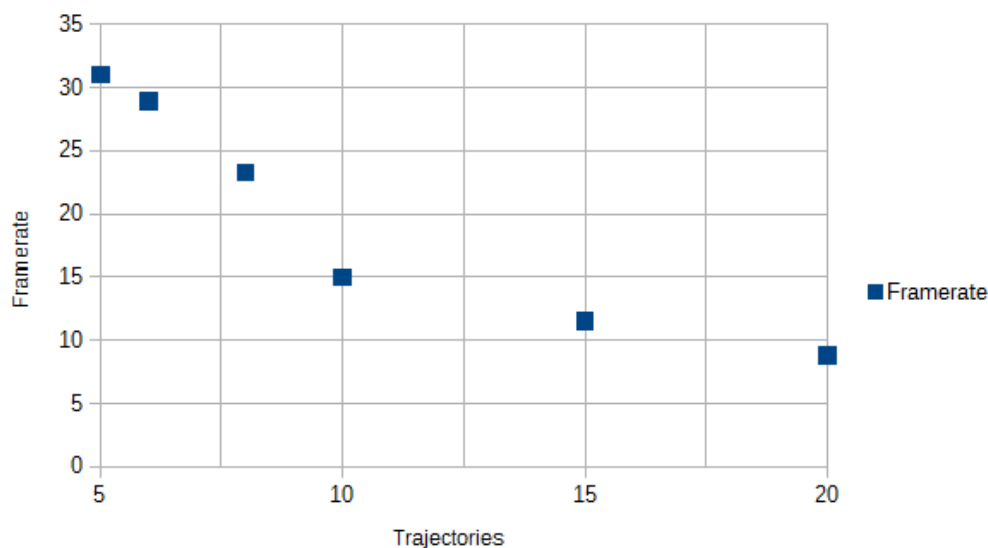


Figure 5.2: Game framerates with different trajectory amounts. Measured trajectory amounts were 5, 6, 8, 10, 15, and 20 trajectories. Framerate limit was 31, which is a result from having the 32 millisecond physics timestep

mance decreasing rapidly when speed reaches 20 kilometers per hour and lagging behind the top performer in 10 and 15 km/h tests. Overall the 5 trajectory combined control had the best relative quality compared to the used processing power. On our testing laptop only the 5 trajectory combined method achieved robust enough driving in real-time. With 10 trajectories the game ran at 15 frames per second, in which the framerate drop was more noticeable than the performance increase of the bike. As seen in Figure 5.2 the game's performance measured in frames per second dropped rapidly when trajectory amount was raised. Framerate was limited to 31 frames per second, which resulted from the physics timestep being 32 milliseconds. Because of how C-PBP works the framerate in-game does not have major fluctuations.

Testing with slightly higher trajectory amounts than 5, to see where the framerate begins to drop, showed that framerate starts dropping rapidly after 8 trajectories. With 6 and 8 trajectories the frame rate drops were only small, but still lowered the framerate under 30 frames per second, which is considered the lowest usable framerate in video games. Below that the responsiveness of the game decreases noticeably.

### 5.3 Remarks

During the implementation of the bike we experimented with different ways to implement different aspects of the simulation. We tried a couple of different approaches of how to implement steering, how to control the joints and we experimented with legs as a different method to balance the bike when it is not moving.

To decide how to steer the bike with C-PBP we experimented with a few steering implementations. First one was a directional control, where the player controlled the desired movement direction of the bike in global coordinates. In this setup the camera followed the character but always stayed in the same rotation in global context. It lacked finer control and did not feel as realistic as we wanted. Then we tried to control the lean and speed of the bike with the optimizer, with the camera positioned behind the character and looking towards it. While this control method felt more natural than the first control method, it was slightly unpredictable at times. Unpredictability was a result of speed being optimized, which lead to the bike automatically avoiding obstacles to maintain the desired speed. Lastly we removed speed as a parameter to be optimized and only used desired lean and gave the player direct control over acceleration of the bike. The desired lean is not absolute, but relative to the speed of the bike, so the player cannot overlean. Still even with the overlean prevention the player can easily make the bike fall over.

For joint control, we ended up controlling target angular velocities of the joints' ODE components, but we also tried to control the joints by setting them to desired positions. It was implemented by calculating the angular velocity needed to reach the positions in the next frame. That method was found to be not as effective and stable as controlling only the angular velocities.

When the character stops with the bike it should stay upright. Real world bicycle and motorcycle drivers use their legs to keep balance while not moving. To mimic this balancing behaviour we tested the character with a second leg. Quick testing with the added leg showed promise, as it has been shown that C-PBP can drive characters to use environment, like walls and floor, for support when needed [13]. Eventually we dropped the leg feature as it was not essential to the goal of the thesis. Still with a better balance cost function it would have been a nice addition to the biker, but it would have taken too much time to implement properly and was in the end considered beyond scope of this thesis. Our control methods could balance

the character for a while when it was not moving, even without the leg, especially the optimizer based methods with a high trajectory count.

## Chapter 6

# Discussion and Conclusions

Here we discuss if our implementation worked as intended and whether online motion synthesis method like C-PBP is feasible to use in the video game industry.

### 6.1 Implementation

We implemented a simplified biker with a motorbike with Unity and ODE and applied an optimizer using an online motion synthesis algorithm to the simulation as explained in Chapters 4.2 and 4.3. The online motion synthesis algorithm used was Control Particle Belief Propagation (C-PBP) by Hämäläinen et al. [13]. Usage of C-PBP is very resource heavy. Our example was running 11 frames per second in a setting of 15 trajectories with 1.2 second prediction horizon, which is not a desirable performance in video games where framerates are usually more than 30, preferably 60, frames per second. To achieve a better framerate we would have to simplify the character model even further, which is a reasonable choice. Faking things in video games is normal and players most likely do not care how features are implemented if they look like they work. So we could simplify our hidden model down to 2 bones for the driver while keeping the current, already simple, bike model. Current character has 5 bones and 3 bike parts with a total of 18 parameters to optimize, as described in Chapter 4.2. With simpler model we could use bones with variable length connected to the handle bar, foot rest, and each other. That would lead to 10-DOF character model, with 4-DOF for the two bones, 1-DOF for handlebar angle and motor, and 2 joint force tuning parameters, one for the motor and one for the rest. Also one method to gain

more processing power could be to use GPU for optimizer calculations, as the current implementation of C-PBP uses only CPU.

Our implementation lacked sound effects. We noticed that sometimes it is hard to realize what the bike is doing as we lacked the information of the desired speed or our interpretation of it was wrong. Sound effects, especially motor sounds, would be a helpful addition for the player to get a feel of how fast the bike should be moving or accelerating.

The arcade control method we used as an approximation of a generic video game control method was only a physically calculated bike. Usually in video games these "arcade" controllers use external forces to keep the bikes upright, which our control method did not use. Also in usual video game controllers there is no counter-steering implemented, but the actions from the player are delivered directly to the bike. Our implementation was to be used together with the optimizer so we had to implement the arcade controller as a physical controller for the steering angle and with no external forces. We had a support for these external forces to keep the bike upright, but they were not used in automatic testing, so that we could evaluate the controller better compared to the optimizer-based control methods.

## 6.2 Testing and results

To test our three controller methods we implemented different levels for different testing aspects: An automatic randomized test level that gave us quantitative testing data to compare the different control methods statistically and two non-automatic levels where players could drive around a track or play area gave us qualitative data. From the quantitative data we attained from the automatic testing in Chapter 5.1.2, we can see that the only statistically significant difference is that arcade control is worse than the other two control methods in our test setup when using speeds of 15 km/h and faster. In 10 km/h scenario the combined method worked best with all trajectory amounts. The two optimizer-based methods worked both approximately equally well compared to each other in the three fastest scenarios. The combined method being seemingly slightly better than the optimizer-only method, but not with a significant margin. From the qualitative analysis, we found out that the controls of the bike were somewhat enjoyable and felt realistic. During testing we also raised the trajectory amount to a high number and we could not tip the bike over, which is encouraging, but framerate in this configuration was really low.

With the qualitative analysis from Chapter 5.1.1 taken into account, of our three control methods, optimizer-only, combined, and arcade, the best results gave the combined method. It gave the best success percentages in the automatic testing, it was our preferred method, and it had the best success rate compared to used processing power. The combined method needed fewer trajectories to be stable. We achieved stable driving and 31 frames per second with only five trajectories for the optimizer using the combined controller. Five trajectories was not enough for the optimizer-only controller, steering left or right made the bike fall easily. The stability of the arcade controller did not depend on the amount of trajectories as it calculated only one action per frame and only used information from current and previous frame to make decisions.

Some of the failures in automatic testing can be explained by ODE friction implementation, explained in the ODE manual [34]. Wheels can sometimes get stuck to environment geometry in unfavourable collision conditions. This was most apparent in arcade control and low velocity testing. In general bikes seemed to be less stable in low speeds (10 km/h), as seen in Figure 5.1. Arcade control did not seem to work well in higher speeds than 15 km/h either.

Even though our test setup for the different control methods was adequate for our purposes, other ways to implement the automatic testing setup can be devised. For example, instead for testing success rates in a limited time, one could check if the bike has fallen over inside the testing area as it does not have the ability to lift itself back up. Also different level layouts are possible, like a circular level, where the character starts in the middle and tries to pass difficult uneven terrain until it reaches goal area. In this case the walls would not hinder the bike's movement and test would focus more on the balancing of the bike than just clearing the course.

The game industry professionals mentioned the camera angle used in the game. The positioning of the camera is an interesting problem in these kind of games where you need to have a good view both of the environment and the character. Positioning the camera straight behind the character would not let the player see the character or the bike and looking at the character from the side limits the view of the environment. Our decision to put the camera behind the character and slightly to the side avoided the pitfalls in our opinion. It showed enough both of the character and of the environment to make decision how to proceed in the levels.

Combined and optimizer-only methods were expected to work better with more trajectories, but test result from 10 km/h test showed inconsistencies.

As seen in Figure 5.1, combined control with 15 trajectories performed worse than with 5 or 10 trajectories in 10 km/h test, not much worse but the difference is significant enough.

An interesting result was that the arcade control did not work well on high speed. Another interesting result was that both optimizer-based methods performed worse on lower than higher speeds. On higher speeds the 5 trajectory combined method seems to be able to fix the mistakes the arcade controller makes. Failures of the optimizer-based methods on low speeds might happen because of the sampling noise causes bigger deviations, especially with low trajectory amount. High speed arcade failures could be explained with bikes becoming slightly unstable when they pass the speed of 6.0 m/s (21.6 km/h) as mentioned by Basu et al. [2]. This could be one of the reasons as our simulation is not a perfect simulation of the real world.

### 6.3 Conclusions

The goal was to test if introducing an online motion synthesis algorithm to controlling a bike could lead to better or more natural steering of the bike compared to usual control setups in video gaming. We found out that the used motion synthesis algorithm was too resource heavy and not usable in video games as it is, even though our qualitative analysis revealed that the control was nice and felt natural when not considering the performance issues. Even with the framerate issues our simplified bike implementation worked well. With better performance it could add depth to steering or make the feel of controlling the bike more natural in video games.

In the end the result was satisfactory. With C-PBP the control over the bike was improved compared to the arcade controller, which we used as an approximate representation of a generic video game controller. Similar results with a better framerate could be possible to achieve with an even more simplified physical character model and a faster optimization algorithm. Training a machine learning controller based on the optimizer results is another way to achieve similar results, as Mordatch et al. demonstrated [23].

### 6.4 Future work

As future work we could add position control to the character on top of the bike. For example the right stick could be used to move the character to gain

better control of wheelies, lean forwards to limit wheelies and lean backwards on top of the bike to make them easier to start. In air this extra control could be used to adjust the bike's lean under the biker to match incoming slopes. For example player could jump from a ramp to a wall and turn the bike to match the wall's angle. With our current implementation we would have to rely on C-PBP to hopefully rotate the bike to match the wall's angle.

Also better modelling of the bike with suspension would help balancing the bike. Also, as mentioned earlier, further optimizing the character model or using machine learning could lead to better performance. A more detailed simulation with sounds and better visuals could also help the player to get a better feel of the simulation state.

More work is being done with the C-PBP to make it more efficient and to produce better quality motion by Hämäläinen et al. in Aalto University.

# Bibliography

- [1] ALPHATHON. Diagram of an Xbox 360 controller with button labels, March 2010. [https://commons.wikimedia.org/wiki/File:360\\_controller.svg#/media/File:360\\_controller.svg](https://commons.wikimedia.org/wiki/File:360_controller.svg#/media/File:360_controller.svg) Accessed 16.9.2015.
- [2] BASU-MANDAL, P., CHATTERJEE, A., AND PAPADOPOULOS, J. M. Hands-free circular motions of a benchmark bicycle. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* (2007), vol. 463, The Royal Society, pp. 1983–2003.
- [3] BELLMAN, R. E. *Dynamic Programming*. Dover Publications Inc, 2003.
- [4] BENNETT, S. Nicolas Minorsky and the Automatic Steering of Ships. *Control Systems Magazine* (November 1984), 10–15.
- [5] BOSSA STUDIOS LTD. Surgeon simulator 2013, 2013. <http://www.surgeonsim.com/surgeon-simulator-2013/> Accessed 5.10.2015.
- [6] COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. Generalized biped walking control. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 130.
- [7] FAJANS, J. Steering in bicycles and motorcycles. *American Journal of Physics* 68, 7 (2000), 654–659.
- [8] FIM, FÉDÉRATION INTERNATIONALE DE MOTOCYCLISME. About the FIM, 2010. <http://www.fim-live.com/en/fim/the-federation/about-the-fim/> Accessed 19.10.2015.
- [9] FODDY, B. QWOP, 2008. <https://www.foddy.net/Athletics.html> Accessed 5.10.2015.
- [10] FULL, R. J., AND KODITSCHKEK, D. E. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of Experimental Biology* 202, 23 (1999), 3325–3332.

- [11] HA, S., YE, Y., AND LIU, C. K. Falling and landing motion control for character animation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 155.
- [12] HÄMÄLÄINEN, P., ERIKSSON, S., TANSKANEN, E., KYRKI, V., AND LEHTINEN, J. Online motion synthesis using sequential monte carlo. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 51.
- [13] HÄMÄLÄINEN, P., RAJAMÄKI, J., AND LIU, C. K. Online control of simulated humanoids using particle belief propagation. In *Proc. SIGGRAPH '15* (New York, NY, USA, 2015), ACM.
- [14] HAND, R. S. *Comparisons and stability analysis of linearized equations of motion for a basic bicycle model*. Cornell University, May, 1988.
- [15] HODGINS, J. K., SWEENEY, P. K., AND LAWRENCE, D. G. Generating natural-looking motion for computer animation. In *Proceedings of the conference on Graphics interface'92* (1992), Morgan Kaufmann Publishers Inc., pp. 265–272.
- [16] HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. Animating human athletics. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 71–78.
- [17] JONES, D. E. The stability of the bicycle. *Physics today* 23, 4 (1970), 34–40.
- [18] KEITHONEHEARTH. Dimensions of a typical bike, October 2009. [https://en.wikipedia.org/wiki/Bicycle\\_and\\_motorcycle\\_dynamics#/media/File:Bicycle\\_dimensions.svg](https://en.wikipedia.org/wiki/Bicycle_and_motorcycle_dynamics#/media/File:Bicycle_dimensions.svg) Accessed 26.10.2015.
- [19] MAYNE, D. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control* 3, 1 (1966), 85–95.
- [20] MEIJAARD, J. P., PAPADOPOULOS, J. M., RUINA, A., AND SCHWAB, A. L. Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* (2007), vol. 463, The Royal Society, pp. 1955–1982.
- [21] MINORSKY, N. Directional stability of automatically steered bodies. *Journal of ASNE* 42, 2 (1922), 280–309.

- [22] MORDATCH, I., DE LASA, M., AND HERTZMANN, A. Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 71.
- [23] MORDATCH, I., AND TODOROV, E. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)* (2014).
- [24] MORIMOTO, J., ZEGLIN, G., AND ATKESON, C. G. Minimax differential dynamic programming: Application to a biped walking robot. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* (2003), vol. 2, IEEE, pp. 1927–1932.
- [25] NATURALMOTION. Case Study: GTA IV. <http://www.naturalmotion.com/gta-iv/1598/> Accessed 23.9.2015.
- [26] NATURALMOTION. Morpheme with Euphoria. <http://www.naturalmotion.com/middleware/euphoria/> Accessed 23.9.2015.
- [27] NG, A. Y., AND JORDAN, M. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 2000), UAI'00, Morgan Kaufmann Publishers Inc., pp. 406–415.
- [28] PAUL, R. P., SHIMANO, B., AND MAYER, G. E. Kinematic control equations for simple manipulators. *IEEE Transactions on Systems, Man, and Cybernetics* 11 (1981), 449–445.
- [29] RANDLØV, J., AND ALSTRØM, P. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML* (1998), vol. 98, pp. 463–471.
- [30] RANKINE, W. J. M. On the dynamical principles of the motion of velocipedes. *The Engineer* 28, 79 (1869), 129.
- [31] REDLYNX, A UBISOFT STUDIO. Trials Evolution, 2012. <http://www.redlynx.com/trials-evolution> Accessed 30.10.2015.
- [32] RITCHIE, A. The origins of bicycle racing in england: Technology, entertainment, sponsorship, and advertising in the early history of the sport. *Journal of Sport History* 26, 3 (1999), 489–520.
- [33] SHARP, R. The lateral dynamics of motorcycles and bicycles. *Vehicle System Dynamics* 14, 4-6 (1985), 265–283.

- [34] SMITH, R. Open dynamics engine user guide. [http://ode.org/ode-latest-userguide.html#sec\\_12\\_13\\_0](http://ode.org/ode-latest-userguide.html#sec_12_13_0) Accessed 12.10.2015.
- [35] SMITH, R. ODE, 2001-2005. <http://ode.org/> Accessed 8.8.2015.
- [36] STANLEY, K. O., AND MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.
- [37] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.
- [38] TAN, J., GU, Y., LIU, C. K., AND TURK, G. Learning bicycle stunts. *ACM Trans. Graph.* 33, 4 (2014), 50:1–50:12.
- [39] TANAKA, Y., AND MURAKAMI, T. Self sustaining bicycle robot with steering controller. In *Advanced Motion Control, 2004. AMC '04. The 8th IEEE International Workshop on* (March 2004), pp. 193–197.
- [40] UNITY TECHNOLOGIES. Inverse Kinematics in Unity. <url-http://docs.unity3d.com/Manual/InverseKinematics.html> Accessed 21.9.2015.
- [41] UNITY TECHNOLOGIES. Unity, 2005. <http://unity3d.com> Accessed 8.8.2015.
- [42] WITKIN, A., AND KASS, M. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 159–168.
- [43] WU, J.-C., AND POPOVIĆ, Z. Terrain-adaptive bipedal locomotion control. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 72.
- [44] YIN, K., LOKEN, K., AND VAN DE PANNE, M. Simbicon: Simple biped locomotion control. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 105.
- [45] YOUNG HORSES, INC. Octodad: Dadliest Catch, 2014. <http://octodadgame.com> Accessed 5.10.2015.