Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

Creating Dynamic Stand-Up Motions for Bipedal Robots Using Spline Interpolation

vorgelegt von

Sebastian Stelter

MIN-Fakultät

Fachbereich Informatik

Technische Aspekte Multimodaler Systeme

Studiengang: Informatik

Matrikelnummer: 6944438

Abgabedatum: 25.06.2020

Erstgutachter: M.Sc. Marc Bestmann

Zweitgutachter: Dr. Norman Hendrich

Abstract

In this thesis a new system for generating stand up motions for bipedal robots from the prone and supine position is presented. The system uses quintic splines to generate trajectories for each of the four end-effectors. By applying PID control to the robots inertia measurement unit, the robots centre of pressure is kept inside the support polygon during the motion and the overall stability is increased. Through an IK solver, the spline goals are transformed into motor goals and then applied to the robot. The evaluation of the proposed system shows significant improvements over the conventionally used keyframe approach, both in success rate and execution time.

Zusammenfassung

In dieser Arbeit wird ein System vorgestellt, das Aufstehbewegungen eines humanoiden Roboters aus der Bauch- und Rückenlage erzeugt. Das System verwendet quintische Splines, um trajektorien für die vier Endeffektoren zu generieren. Mithilfe eines PID controllers, der die Werte der inertialen Messeinheit kontrolliert, wird der Druckmittelpunkt innerhalb des Stützpolygons gehalten. Durch inverse Kinematik werden die Splineziele in Motorziele umgewandelt und dann angesteuert. Die Evaluation zeigt, dass das beschriebene System eine höhere Erfolgsrate aufweist und schneller agiert, als die bislang verwendeten Ansätze basierend auf dem Keyframeverfahren.

Contents

List of Figures

List of Listings

List of Acronyms

IMU inertia measurement unit

ROS Robot Operating System

PID controller proportional-integral-derivative controller

DoF degrees of freedom

IK Inverse Kinematics

CoP center of pressure

URDF Unified Robot Description Format

SRDF Semantic Robot Description Format

SEA Series Elastic Actuator

1. Introduction

In the context of RoboCup Humanoid Soccer, teams of humanoid robots face each other in a soccer match. These robots have to act fully autonomous during the game. Therefore it is important that they are capable of getting up after a fall. Since collisions between robots are common and walking on uneven ground is still a problem for the league, robots fall easily during the game, and without a reliable way to get back up, no gameplay can happen.

Currently the stand up motions are mainly controlled by keyframe animations which is a quick and easy way to solve the problem, but also brings a major flaw: The animations are static which means the robot can not adjust for different environmental situations, like uneven turf or being touched by other robots, and will not succeed in every attempt. For example, in an open loop system, the direction of the artificial turf can not be factored in and so the motion may succeed when looking in one direction, but fail when looking in the opposite. According to the RoboCup Humanoid Soccer rules a robot that is not able to get up after 20 seconds is deemed incapable and has to be removed from play, and suffers a penalty [1]. Because of that a reliable and quick way to stand up is a huge advantage over other teams.

Outside of the soccer environment a reliable solution is important as well. Bringing robots into the real world and in contact with humans is difficult and robustness is required to make this work. On the way to getting robots to assist humans we need to find a way to recover from tripping over obstacles and accidentally or intentionally being knocked over.

The main goal of this thesis is to create a closed-loop variant of the typical stand up motion for bipedal humanoid robots. For that purpose the movements of the four end-effectors will be modelled with quintic splines and the motions calculated with an inverse kinematic solver. By doing so sensor input the robot provides can be factored in and used to stabilize the motion. The information provided by an inertia measurement unit (IMU) will be used to calculate the centre of mass and keep it inside the support frame of the robot.

Chapter 2 gives an overview of the environment this system is applied to, as well as the resources needed to implement and evaluate this approach. Chapter 3 presents and discusses other approaches at creating a stand up motion. These approaches range from keyframe animations over reinforcement learning to motion tracking. This chapter also discusses related work about motion planning with splines. In chapter 4 the approach is discussed. The different parts of the package are explained, as well as the parameter tuning process. Chapter 5 evaluates the findings of this paper, both in a simulated environment and the real world. Chapter 6 discusses the results and compares them to other approaches, while chapter 7 gives a conclusion and takes a look at future work.

2. Fundamentals

The following sections give an overview of the context this work is done in, as well as the concepts used. Section 2.1 presents the RoboCup context as the main environment of this thesis. Section 2.2 presents the robot platform this thesis was developed and tested on and section 2.3 gives an overview over ROS, the Robot Operating System used in this thesis. In section 2.4 a quick overview over the MoveIt! framework is given. Section 2.5 explains the calculation of inverse kinematics necessary to calculate the positions of the robots end-effectors. Section 2.6 describes the quintic splines used to model the robots movements and Section 2.7 explains the concept of a proportional-integral-derivative controller (PID controller) used for stabilization.

2.1. RoboCup

The RoboCup is an annual robotics competition held by the RoboCup Foundation [2]. This 1992 founded research initiative set its main goal to beat the reigning human soccer world champion with a team of robots by 2050, following the rules of a standard human soccer match. In doing so, they created an environment to compare robotic approaches under real world circumstances in a multi agent system. The soccer environment is complex enough to cover a variety of research topics: From vision over behaviour to motion, everything plays an important role in playing soccer.

While different leagues exist, this thesis will mainly focus on the humanoid kid size soccer league. In order to keep the competition comparable, a set of rules restrict the robots specifications [1]. The robot has to be humanoid, which means, it needs to have two arms, two legs and a head. It is also only allowed to have sensors that a human has as well, and in places where the human has them. Allowed are for example pressure cells, IMUs and cameras. Sensors like compasses or laser range finders are not allowed. The robots height in the kid size is defined as 40 to 90 cm. Further, a couple of rules define the relations of the robots body parts. The most important rule for the stand up motion describes the size of the feet. Each foot has to fit inside a rectangle of size $\frac{1}{32}(2.2 \cdot H_{COM})^2$, where H_{COM} is the height of the centre of mass. The length of the legs (H_{leg}, compare figure 2.1) has to satisfy $0.35 \cdot H_{top} \leq H_{leg} \leq 0.7 \cdot H_{top}$. The arms have to be at least $H_{top} - H_{leg} - H_{head}$ long, the maximum length is restricted by a rule that says, that the robot must not have a configuration where it exceeds $1.5 \cdot H_{top}$.

The most common arrangement in the league is for the robot to have 20 degrees of freedom (DoF)s, even though the rules do not specify this. With that configuration each arm has 3 DoF, each leg 6 and the head 2. Compared to a human the robot is mainly missing the shoulder yaw and the mobility in the spine. The robot is also missing

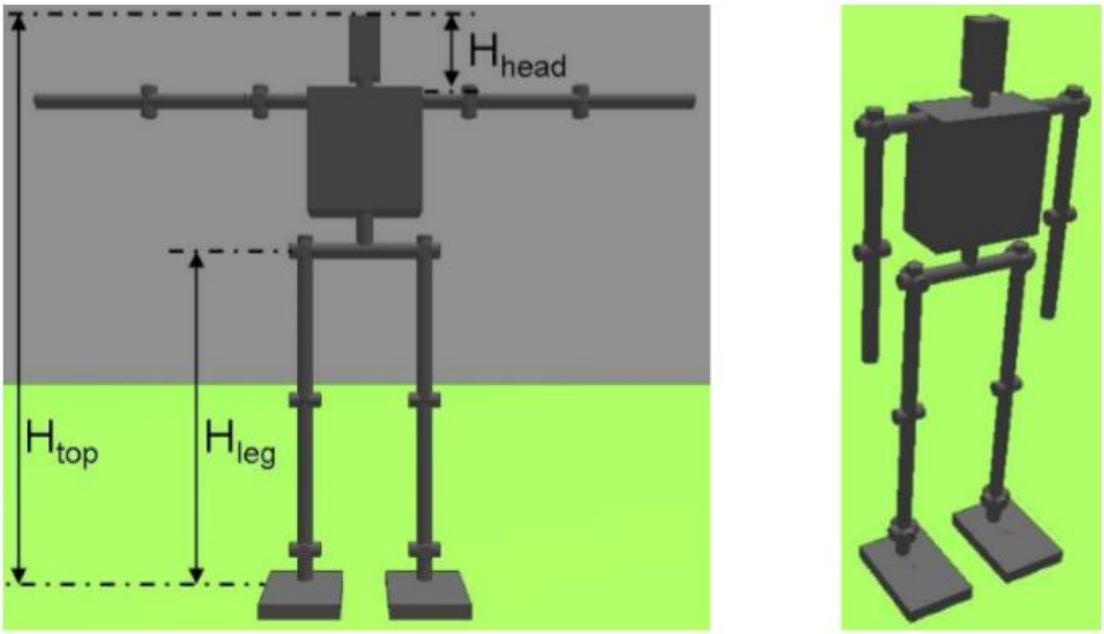


Figure 2.1.: Body plan of a humanoid robot according to the RoboCup humanoid soccer league rules. H_{top} describes the robots height from the top of the head to the footplate, H_{head} describes the height of the head from the top of the head to the top of the torso, and H_{leg} describes the length of the legs from the bottom of the torso to the footplate [1].

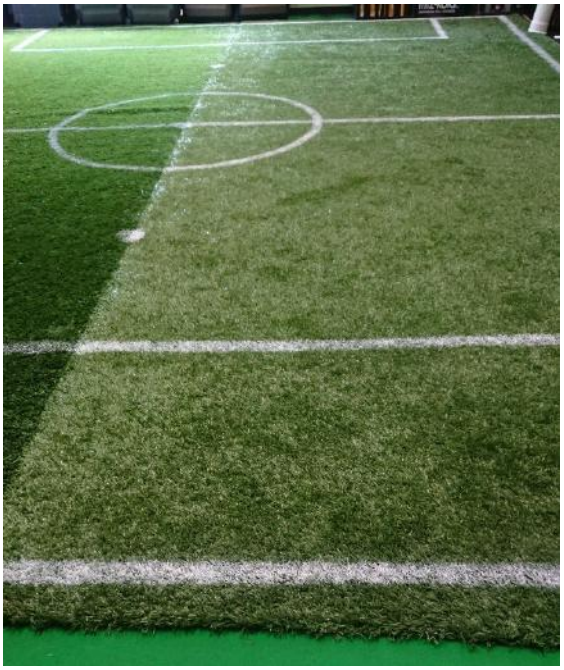


Figure 2.2.: Training soccer field of the Hamburg BitBots. The colour difference between the left and right half does not stem from different kinds of grass, but solely from the different alignment of the blades. This causes a completely different texture depending on the direction. Image courtesy of Judith Hartfill.

freedom in its toe movement, which is especially interesting for replicating human gaits, as it is not possible for these robots to roll their feet.

The rules also state that the robot has to walk on artificial turf approximately 3 cm high. This adds a lot of difficulty to the stand up problem, since the conditions of the turf change drastically with the direction of the grass blades (compare figure 2.2). These unclear and changing conditions may cause the robot to slide a bit in one direction, but not in the other, or cause the robot to stumble as it pushes its footplate against the grain. Additionally the grass is usually not brushed completely into the same direction, but contains a lot of unevenness, which might cause imbalance for the robot. This is especially problematic for the commonly used keyframe approaches, while the approach presented in this thesis is able to dynamically respond to environmental changes like these.

2.2. Robot Platform

For this thesis the Wolfgang robot platform was used (Compare figure 2.3). This humanoid robot is based upon the Nimbro-OP [3] and was further developed by team WF Wolves.[1]. The platform was refined and optimized by team Hamburg BitBots and is currently used in its third version, while a fourth version is in active development.

The robot has 20 DoFs and consists primarily of carbonfibre and aluminium, giving it an approximate weight of 8 kg. According to the definition given by the rules (compare figure 2.1, H_{top}), the robot is 78 cm high and has 38 cm long arms and 41 cm long legs. The robots centre of mass is estimated to be at the robots base link located at the bottom of the torso, centered between the legs. The transformation tree displaying the relationships between the robots links can be found in appendix A.

The codestack runs on an Intel Nuc, a Nvidia Jetson TX2 for image processing and an Odroid XU4. By using the Robot Operating System (ROS), these three machines can be treated as one internally. For sensing its environment, the robot is equipped with a camera in its head, eight load cells in its feet and an IMU in its torso. Internally, the robot can sense temperature, position, velocity and accelleration, as well as torque and voltage of each motor. The robot uses Dynamixel MX-64 and MX106 motors with a stall torque of 7.3 Nm (at 14.8 V, 5.2 A) and 10.0 Nm (at 14.8 V, 6.3 A), respectively [4] [5].

To reduce the strain on single parts, the robot is equipped with a variety of protective gear. 3d printed separators are mounted in the arms and legs to reduce torsion, and elastic bumpers are attached to the front and back of the torso, which reduces shock. On those motors, that experience the largest forces if the robot falls, i.e. both shoulder roll motors and the head pitch motor, compliant elements are mounted, turning the motors into Series Elastic Actuator (SEA)s. These 3d printed components made from both normal filament and a special elastic filament give in to strong forces rather than resisting and possibly damaging the motor. Due to its shape, the SEA returns to its original position after the force is removed. However, these SEAs also mean, that the motors in these joints cannot apply strong forces and motions along these axis can easily be blocked by obstacles. This is something that has to be considered when developing a stand up motion.

[1] https://www.wf-wolves.de/

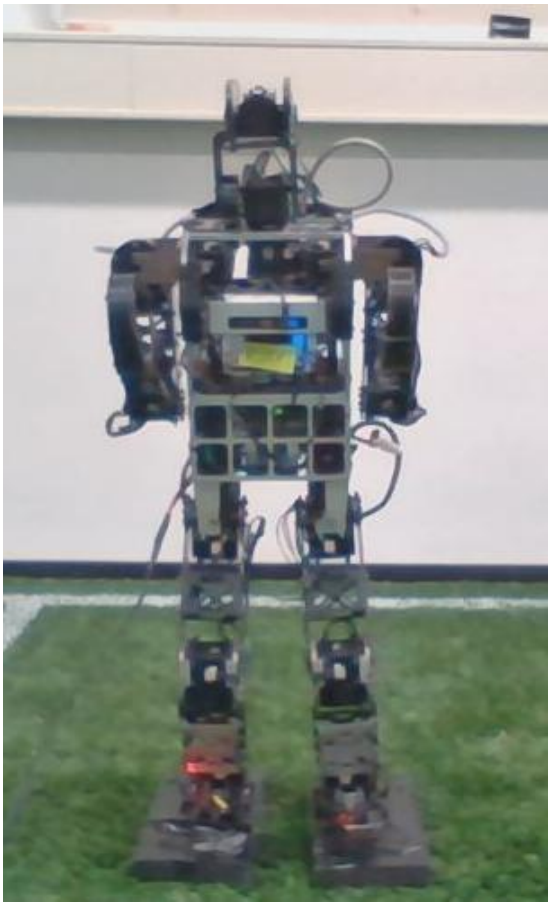


Figure 2.3.: One of the four Wolfgang robots used by the Hamburg BitBots.

2.3. ROS

ROS, the robot operating system, is a framework that acts as a structured communication layer on top of the actual operating system [6]. In ROS software components are modeled as *nodes* that communicate via *messages*. This grants a lot of flexibility, as software components can be exchanged easily without altering the rest of the system, as long as the input and output stays of the same type. Messages are handled with *topics*. Any node can subscribe to a topic or publish to it. There is no limit to how many nodes can be subscribed to or are publishing on a topic at a time. The connection between publishers and subscribers is handled by a common *roscore* node. While messages only offer one way communication, *services* can be used for two way exchanges. A service consists of two parts: If a node offers a service, a client can send a request message to that service. The client will then wait for the response. ROS also provides several tools for visualization and debugging, such as RViz for 3D visualization [7].

2.4. MoveIt!

MoveIt! is a motion planning and mobile manipulation framework for ROS [8]. It can cover various motion based robot tasks, such as motion planning, direct and inverse kinematics, collision checking and trajectory processing. While MoveIt! has many more capabilities, for this thesis however, only the RobotModel and the RobotState classes were used. These classes are the core classes in terms of kinematics.

The RobotModel class contains a description of the relationship between all the robots joints and links. This information is gathered from the robots Unified Robot Description Format (URDF) and Semantic Robot Description Format (SRDF) files. The robot model is constructed as a tree. Starting from a base link, each link holds a reference to one parent joint and zero to n children joints. Each joint holds reference to one parent link and at least one child link. This ensures that a leaf is always a link. The joints and links can be separated into different planning groups that can be influenced without affecting the rest of the model. That is particularly useful if the robots arms should be moved separately from its legs. The RobotModel class also manages the joints properties, such as joint limits.

The RobotState class manages the robots state at a snapshot in time. It contains various information about the joint states, such as position, velocity and acceleration. It has methods to change these values, either by setting them directly or by interacting with the Inverse Kinematics (IK) solver. By calling the `setFromIK` method, the joint values are calculated for the end-effector of a move group. If this calculation succeeds before either a set timeout or a number of attempts, the joint values are set to the result of the function. The RobotState class can also calculate Cartesian paths and Jacobians.

2.5. Inverse Kinematics

Inverse kinematics describe the process of calculating joint positions based on the end-effectors pose. Depending on the DoFs of the kinematic chain and the joint limits of each joint, the inverse kinematic may have several solutions and is thus more difficult to calculate than the forward kinematics.

There are various approaches to solve inverse kinematics. While in some cases it is possible to solve the inverse kinematics analytically, which is fundamentally faster, for more complex robots a numerical solution usually is preferred. The most common numerical approach to an inverse kinematics problem is the Jacobian inverse technique (section 2.5.1), but heuristic methods like gradient descend (section 2.5.2) can also be used. This thesis uses the BioIK solver (section 2.5.3), which combines evolutionary optimization with particle swarm optimization [9].

2.5.1. Inverse Jacobian Method

The inverse Jacobian method iteratively minimizes the offset of the endeffector compared to the goal position. Each iteration begins with calculating the Jacobian matrix, which contains all first order partial derivatives of the position function, i.e. the forward kinematics function:

$$J(j) = (\frac{\delta p_i}{\delta j_k}(j)) = \begin{bmatrix} \frac{\delta p_1}{\delta j_1}(j) & \frac{\delta p_1}{\delta j_2}(j) & \cdots & \frac{\delta p_1}{\delta j_k}(j) \\ \frac{\delta p_2}{\delta j_1}(j) & \frac{\delta p_2}{\delta j_2}(j) & \cdots & \frac{\delta p_2}{\delta j_k}(j) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta p_i}{\delta j_1}(j) & \frac{\delta p_i}{\delta j_2}(j) & \cdots & \frac{\delta p_i}{\delta j_k}(j) \end{bmatrix} \tag{2.1}$$

where J is the Jacobian matrix, p is the end effector pose, j is the joint value, i is the goal pose dimension and k is the joint variable index.

With the Jacobian matrix the relationship between joint movement and end-effector movement can be calculated:

$$\Delta p = J(j) \cdot \Delta j \tag{2.2}$$

Since the goal is to move the end-effector towards the goal position, Δp should be the difference between the current end-effector position and the goal position:

$$g - p = J(j) \cdot \Delta j \tag{2.3}$$

The joint offset Δj defines the difference to the joints current position that is needed to get the end-effector closer to the goal position. To get the joint offset, the equation is reversed. As the Jacobian matrix usually is not square and therefore cannot be inverted, in these cases the pseudo inverse is used instead:

$$\Delta j = J(j)^{-1} \cdot (g - p) \tag{2.4}$$

The joints are then moved by the calculated offset:

$$j_{n+1} = j_n + J(j_n)^{-1} \cdot (g - f(j_n)) \tag{2.5}$$

This process is repeated until either a accuracy threshold is met or a timeout is reached [9].

2.5.2. Gradient Descent

Another common optimization method is the gradient descent. Gradient descent works by projecting the problem onto a plane. Starting at a random point on the plane, the slope is calculated and a step in that direction is taken. The process is repeated, until every possible step has a positive gradient. The learning rate influences the step size. A larger learning rate gets to the solution faster, but risks overshoot. A smaller learning rate mitigates overshoot, but takes significantly longer to reach a solution and is more prone to reaching local minima. The gradient descend method is generally easier to implement, compared to the other models, but is more prone to finding local minima and therefore diverging from the optimal solution.

2.5.3. BioIK

The BioIK is a memetic inverse kinematics solver for MoveIt!. Memetic algorithms combines evolutionary or population based global search with local refinement procedures [10]. An efficient search needs to effectively trade off between exploration and exploitation of the search space. While most evolutionary algorithms quickly explore the search space, they have deficiencies in exploitation. By combining them with independent local search techniques, these deficiencies can be overcome, leading to a faster and more accurate search algorithm.

The BioIk uses a combination of "evolutionary optimization, particle swarm optimization, and gradient based methods" [9, p. III]. Evolutionary algorithms take inspiration from biological evolution: The problem is modelled as a fitness function. A population of possible solutions is rated against this function, whilst solutions with higher fitness have a higher chance of reproduction. In each generation the solutions are altered with crossover, i.e. combining the genomes of two parent solutions, or mutation, i.e. randomly changing some genomes. Then the fitness of the children is evaluated and the children with the highest fitness are selected for the next generation [11].

The particle swarm optimization approach works by modelling possible solutions as particles. During each iteration each particle moves in a random direction. If that movement improves the result, it is kept, else it is reverted. Each particle also accumulates momentum: Every time a random movement increases the particles fitness, that movement is also added to its momentum. Therefore particles that continuously move in the right direction make larger steps and reach the optimum faster.

For the gradient based optimization several approaches have been implemented. These approaches include gradient descent as described in section 2.5.2 and algorithms from the CppNumericalSolvers library. [2]

[2] https://github.com/PatWie/CppNumericalSolvers

2.6. Quintic Splines

In order to create our get up animation we need a way to model motion. In the most simple case a motion is described by two time points, t_0 and t_1, as well as constraints for the position, velocity and acceleration at each of the points. The mathematical problem now is to find a continuous function between t_0 and t_1 that satisfies all constraints. This can be done by using polynomials:

$$q(t) = c_0 + c_1 t + c_2 t^2 + \cdots + c_n t^n, t \in [t_0, t_1] \tag{2.6}$$

Since the stand up problem consists of more than two time points, multiple polynomials are combined to a spline. This greatly simplifies the problem by reducing the amount of constraints per polynomial, therefore reducing the necessary degree of the polynomial and thus avoiding Runge's phenomenon for higher degree polynomials, which describes, that higher order polynomials cause oscillation at the edges of intervals [12].

The robots motion should be continuous in position, velocity and acceleration to avoid potential damage to either the robots components or its environment. This means three constraints per time point, or six constraints per polynomial. Therefore the splines used have to be at least quintic. Figure 2.4 displays the problems of using lower degree splines: In a cubic spline the position and velocity are continuous, but the acceleration is not. This could result in an abrupt change of velocity which could then cause large forces applied to the robot. In comparison a quintic spline is continuous in its acceleration and therefore smooth in its velocity. It is notable, that the splines jerk is left unconstrained in quintic splines. This could be avoided by using septic splines instead, which can solve up to eight constraints, but the increase in smoothness does not justify the additional complexity and therefore additional computing time caused by that change.

The polynomial can be described as

$$q(t) = c_0 + c_1(t - t_0) + c_2(t - t_0)^2 + c_3(t - t_0)^3 + c_4(t - t_0)^4 + c_5(t - t_5)^5 \tag{2.7}$$

with

$$\begin{cases} c_0 = q_0 \\ c_1 = v_0 \\ c_2 = \frac{1}{2} a_0 \\ c_3 = \frac{1}{2T^3}[20h - (8v_1 + 12v_0)T - (3a_0 - a_1)T^2] \\ c_4 = \frac{1}{2T^4}[-30h + (14v_1 + 16v_0)T + (3a_0 - 2a_1)T^2] \\ c_5 = \frac{1}{2T^5}[12h - 6(v_1 + v_0)T + (a_1 - a_0)T^2] \end{cases} \tag{2.8}$$

where q_0 and q_1 are the positions, v_0 and v_1 the velocities and a_0 and a_1 the accelerations at the t_0 and t_1, $h = q_1 - q_0$ is the displacement and $T = t_1 - t_0$ is the duration [13].

As a spline has multiple points, generally the velocities for the middle points are not defined. This missing information can be determined heuristically.

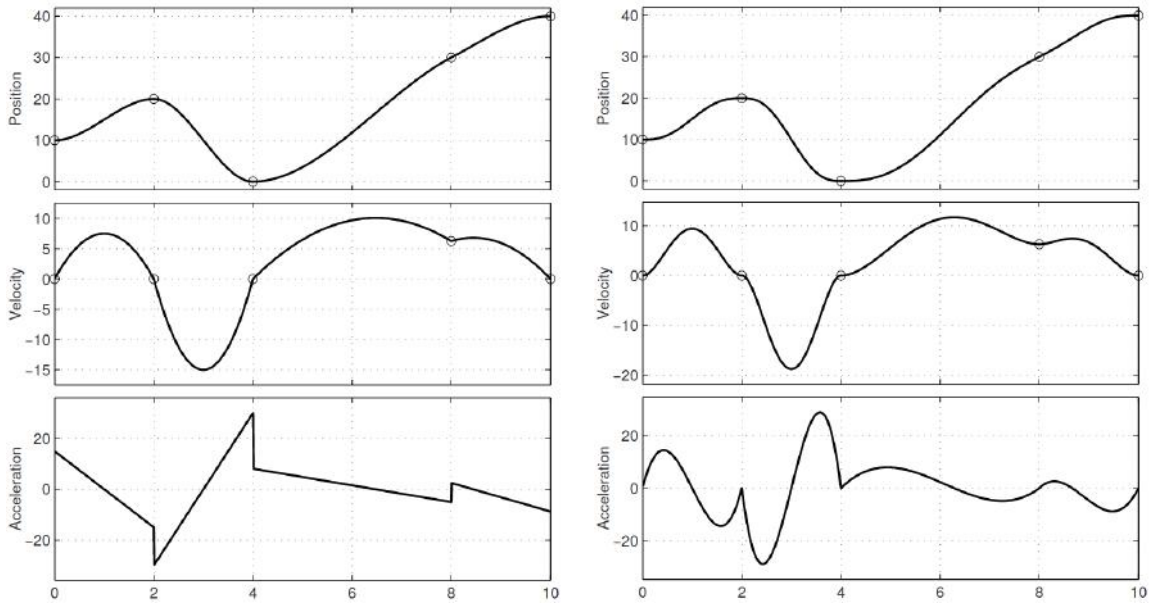


Figure 2.4.: Splines (top) with their respective first (middle) and second (bottom) derivative. (a) shows a cubic spline, which is continuous in its position and velocity, but not in acceleration, and thus not smooth in its velocity. (b) displays a quintic spline, which is smooth in both position and velocity, and continuous up to its second derivative [13].

2.7. PID Controller

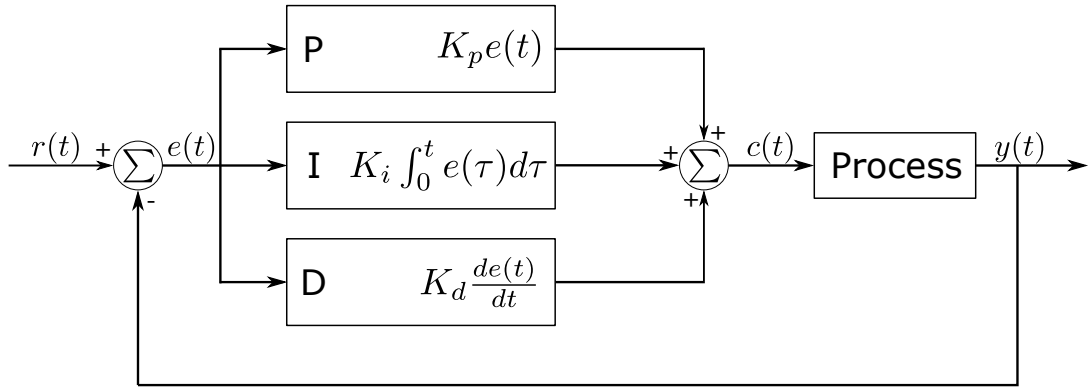


Figure 2.5.: Schematic of a pid controller. r(t) is the desired process value, y(t) is the measured process value. e(t) is the error between these two and c(t) is the corrected error.

A PID controller (proportional-integral-derivative controller) is a feedback control mechanism. It takes sensor input and calculates the error between this measured value and the desired setpoint. From this error a correction is determined and applied to a control function. PID controller controllers consist of three separate controllers (Compare figure 2.5). The proportional controller controls the present error. It can be adjusted by manipulating the proportional gain K_p. The error depends inversely on the gain, meaning a higher gain causes a lower error. However, a higher P gain also causes the system to overshoot and might make the system unstable. The P controller can be displayed as

$$P = K_p \cdot error(t) \tag{2.9}$$

where t is the current time.

The integral controller controls the error accumulated over the past. It corrects the sum of instantaneous errors that should have been corrected previously. A higher I gain also increases the overshoot, worsens the transient response and makes the system unstable. The I controller can be formulated as

$$I = K_i \int_0^t error(t')dt' \tag{2.10}$$

The significance of the integral controller can be observed when comparing a P controller with an PI controller (Compare figure 2.6). While the P controller gradually reduces the amount of new errors made, it does not correct for the amount of errors that already exist. The P controllers trajectory thus follows the reference trajectory parallelly, but never reaches it. A P controlled motion will always be to some degree erroneous. By adding an I controller, the accumulated error is accounted for and corrected as well. The PI trajectory follows the reference trajectory way more closely.

However, the PI controller also tends to overshoot the target, adding oscillation to the process. In this example, this is most likely due to the K_i gain being too high. A well tuned I controller can generally perform better than just a P controller. [14]

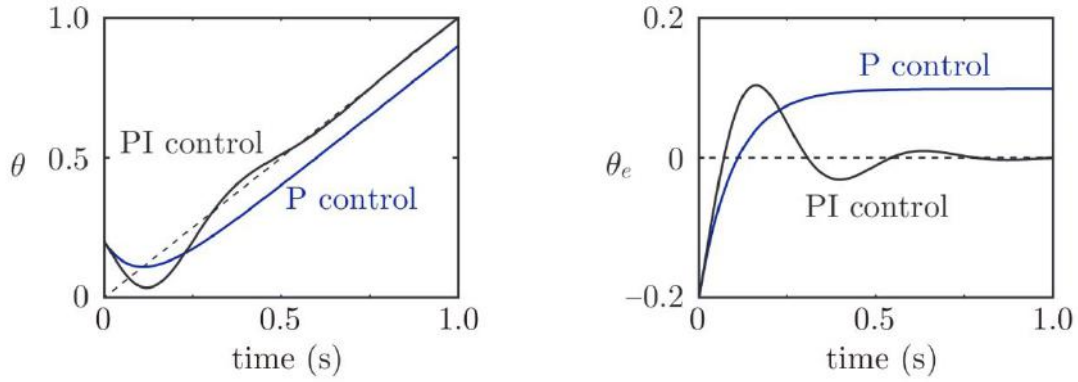


Figure 2.6.: The motion of P-controlled and PI-controlled joints, with initial position error, tracking a reference trajectory (dashed) where $\theta_d(t)$ is constant. (Left) The responses $\theta(t)$. (Right) The error responses $\theta_e(t) = \theta_d(t) - \theta(t)$. K_p is equal for both controllers [14].

The derivative controller estimates the future trend of the error based on the current rate of change. The D controller reduces the rate of change of the PID controller controller and by doing so reduces the overshoot and improves the transient response. It can be formulated as

$$D = K_d \cdot \frac{d \cdot error(t)}{dt} \tag{2.11}$$

The controllers are then added to form the overall control function:

$$u(t) = K_p \cdot error(t) + K_i \int_0^t error(t')dt' + K_d \cdot \frac{d \cdot error(t)}{dt} \tag{2.12}$$

[15]

3. Related Work

This chapter gives an overview of work by other researchers on the topic of stand up motions and spline-based motion. Section 3.1 describes how the keyframe method has been used to generate stand up motions. Section 3.2 displays the use of neural networks to learn the motions. A motion tracking based approach is highlighted in section 3.3. Sections 3.4 and 3.5 show how spline interpolated motions and pid stabilization are used in a different context.

3.1. Keyframe Animation based Approaches

Using keyframe animations to model stand up motions seems to be the most popular approach to the problem in the RoboCup context and is also the approach currently used by the Hamburg Bit-Bots. In his bachelor thesis Timon Giese developed a tool to efficiently develop keyframe animations in real time [16]. Via a command line interface motor positions can be set and saved as keyframes. If a robot is connected to the software, the motion can be viewed in real time on a real life robot. The software also has functionalities to either play single frames or complete animations, duplicate frames, change the order or partially play animations. The software since has been further updated and now uses a graphical user interface and has additional features such as mirroring frames. A similar tool has been created by researchers of the Tama Research Institute, Sony Corporation and Boston Dynamics [17]. However, their tool supports both joint space and euclidean space manipulation. Even though the animation can be created by using inverse kinematics, the result is still stored as joint values for each keyframe. Their tool comes with a 3d simulator and allows manipulation and testing of the motion directly in the program before sending it to a robot. The motion creating system also has a walk pattern generator to assist in creating feasible foot motions. Despite its varying features, the motion creating system is only compatible with the SDR-4X robot platform[1] and mainly focuses on efficiently creating dance routines.

Researchers from the University of Freiburg describe their stand up animation in detail as a four phase plan [18]. When getting up from the back, the first phase is sitting up and moving the arms behind the back. In the second phase the robot gets into a bridge-like position. In the third phase the robot swings its upper body forward to get into a stable position. In the last phase the robot straightens its legs and gets up completely. The motion for getting up from the front starts with lifting up the trunk by pushing up with the arms. During the second phase the robot gets its center of

[1] https://www.sony.net/SonyInfo/News/Press_Archive/200203/02-0319E/

pressure (CoP) as close as possible to the support polygon. The robot then straightens its arms completely and pushes itself onto its feet. The fourth phase is the same as the fourth phase of the back animation. The paper claims that these animations work in 100% of the cases, but since the study assumes a flat carpet as ground, this is no longer applicable to modern RoboCup competitions.

Hirohisa Hirukawa et. al. propose a solution using ten different contact states between which the robot can transition statically [19]. Only the transition between kneeling with knees and toes as support points and squatting with just the sole as support point cannot be made without entering a volatile state. Therefore this state is split into three phases and applies feedback control to mitigate overshoot. In the first phase, the robots hip pitch motors are moved back to generate inertia. After the force is generated, the interpolation between the current state and the desired state is executed as usual, but feedback control is applied. The third phase uses feedback control to keep the robot in the desired state.

Recently team Starkit presented an approach that catches the falling robot before it touches the ground [20]. This active falling process cuts the teams falling related inactivity time by 30% to the front and 65% when falling back.

3.2. Reinforcement Learning based Approaches

Various researchers have created stand up motions with neuronal networks using reinforcement learning. Jun Morimoto and Kenji Doya constructed a hierarchical architecture that uses a Gaussian Softmax Basis Function Network [21] [22]. The hierarchical architecture splits the problem into two parts. The upper level selects the desired posture for each sub-goal. The lower level then takes the joint values and the goal posture as input and calculates the desired joint angles. The reward function is different for each sub-goal. The model trained for only 300 iterations, yet succeeded in five out of 13 cases. However, the model used in this paper was greatly simplified and consisted of only three links and two joints. Whether this approach is scalable to a full size humanoid robot remains unclear.

Researchers from the Beijing Institute of Technology use a neural network that considers friction and ground force besides the support polygon in the reward function [23]. The robot is simplified into a mathematical dynamic model. The ground force and the friction are calculated from the total external force applied to the robot. The valid stable region in the support polygon has been reduced to just a part of the polygon, because as the zero movement point gets closer to the support regions edge, the system becomes less stable. As this behaviour is undesirable, solutions with the zero movement point on the edge of the support polygon are rewarded less. The network has been tested on a BHR6p robot weighting 60kg and measuring 1.7 metres in height.

3.3. Motion Tracking based Approach

Michael Mistry et. al. solved the problem of getting a humanoid robot from a sitting position to a standing position by tracking a humans movement and applying the result to the robot [24]. They captured a human getting up from a chair with an optical motion capture system and applied the recordings, together with the recorded CoP to a 155 DoF skeletal model. The values are then mapped onto the robot model and solved with an IK solver. The centre of pressure is used as a hard constraint for the solver, while the marker trajectories are used as soft constraints. While their approach was successful for some of the tracked motions, the paper states, that constraining the zero movement point and applying inverse dynamics could greatly improve the robots performance.

3.4. Spline Interpolation

Interpolating splines to smoothen motion is not a new concept and has been used in several robotics related tasks, mainly with industrial robots. In 1972 Richard Paul first described the use of polynomial splines to control the movement of a robotic arm [25]. Since then the method has been applied to various different contexts. Rahee Walambe et. al. used spline interpolation to generate optimal trajectories for a robotic car [26]. Cubic splines were used to interpolate between target points in a two dimensional world to generate paths that the car can follow smoothly, even though it is not able to move in every direction due to its wheels.

Hao Dong et. al. used splines to model the gait of a four legged robot. They split the walk pattern generator into three parts: A gait pattern selector on the high level to select the current phase of each leg, a stance designer generating the landing position for the legs, and a locus designer that generates the path each leg takes. The locus is modelled with cubic splines since the legs are supposed to follow a smooth path. The model is then optimized using reinforcement learning. The top speed reached with this approach was 47 centimetres per second, which at that time was faster than any other walking pattern. According to the paper this was mainly due to modelling the locus with splines.

Zhe Tang et. al. used cubic spline interpolation to smoothen the transition between feet during the walk cycle of a bipedal humanoid robot [27]. The problem this work focuses on is the sudden change of velocities when transitioning between single support phases with a swinging leg and double support phases. These abrupt changes cause a huge instability in the robot. To resolve this problem, cubic splines are used, which ensure continuity in the first and second derivative and thus smoothness in the first derivative, i.e. the robots velocity. A similar approach with quintic splines has been made by team Rhoban [28] and later adopted by the Hamburg Bit-Bots [29].

3.5. PID Stabilization

PID controllers have been used for various tasks, but in robotics its main purpose is stabilizing motion [30]. Nguyen Gia Minh Thao et. al. applied PID control to a two wheeled robot for self balancing. Their backstepping PID controller consists of three control loops to balance the robot. A backstepping controller keeps the robot at equilibrium, a PD controller controls the position of the robot and a PI controller monitors the direction of the motion. This combination allows the robot to move to a given destination while still maintaining its balance, even when external disturbance is applied.

Safa Bouhajar et. al. created a predictive PID controller to balance the motion of a walking humanoid robot [31]. By combining a generalized predictive controller with a classic PID controller, they managed to balance a simplified model of a humanoid robot. Due to the PID controller, the model was real time applicable, however the GPC component used a lot of computing power and therefore required a fast CPU. PID control has also been used to balance drones and unmanned helicopters when carrying a payload [32]. The additional weight, especially if not loaded centrally, can cause possibly fatal disturbances that can be balanced out by a PID controller.

4. Approach

The work of this thesis is based upon the DynUp package created by Timon Engelke for the Hamburg Bit-Bots[1]. It heavily modifies this basis, adds pid control, visualization and support for multiple splines, models the motion and changes the use of the IK interface. The final result of the thesis is available as a ROS package and consists of four parts (Compare figure 4.1): The `dynup_node` creates a rosnode and handles communication between the components as well as communication with other nodes like the IMU or the robot model. The `dynup_engine` models the splines for each limb and returns the tf2 Transform for each spline for the current timestamp. This information is then passed to the `dynup_stabilizer`, which applies two PID controller. The result is then passed on to the `dynup_ik` which makes the IK requests.

Two structs for internal communication between the separate parts were implemented, as well as a visualizer for debugging purposes, that displays the planned splines, as well as the orientations in each spline point. Also dynamic reconfigure support was added.

[1] `https://github.com/bit-bots/bitbots_motion/tree/master/bitbots_dynup`

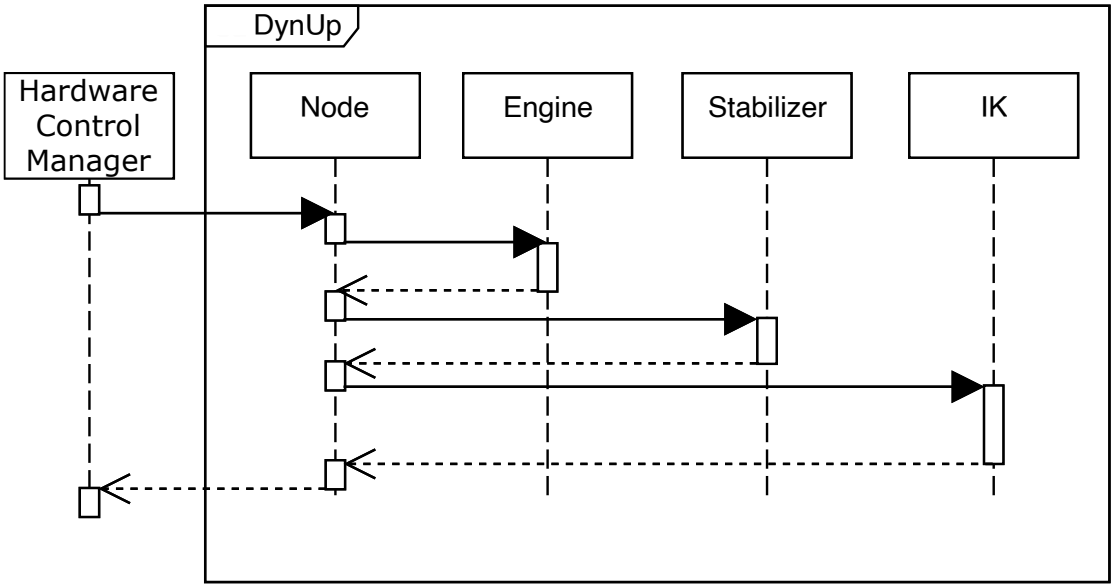


Figure 4.1.: Display of the information flow through the program. The request to the node is only made once, after that the program loops until the engine confirms the process to be completed. The visualizer is not displayed, since it is not necessary for the program to succeed.

4.1. Node

The dynup node handles communication between the programs components and with other packages. On initialization the node invokes an ActionServer, initializes the other program components and loads the robot model. It then gathers some unchanging distance parameters from the model, creates necessary publishers and subscribers and then starts the server.

The node then lies idle until a dynup request is made. Two callback methods ensure that the current joint states and the IMUs orientation keep getting updated when they change. A third callback method reacts to changes to the parameters in dynamic reconfigure.

When a dynup request is made, the `executeCb` method is triggered (compare figure 4.1). At first all components reset method is called, in case there has been a dynup request before. Then, the current poses of the limbs are gathered and formed into a request object (compare listing 4.1). This works by creating an empty, zero initialized stamped pose with each end-effectors frame as its frame id, and then transforming these poses into the desired reference frame. If debug is activated, the visualizer is called now to display the debug splines. In the next step the main loop is called, which returns the motor goals for every time step.

```
1  struct DynupRequest {
2    bool front;
3    geometry_msgs::Pose l_foot_pose;
4    geometry_msgs::Pose r_foot_pose;
5    geometry_msgs::Pose l_hand_pose;
6    geometry_msgs::Pose r_hand_pose;
7   };
```

Listing 4.1: The DynupRequest struct used to send a request to the engine. It contains one pose message for each endeffector and a boolean that indicates whether the robot has fallen to the front.

Each iteration of the loop starts with getting the updated goals from the engine. The information gathered from this is propagated through the components using a response object (compare listing 4.2).

```
1  struct DynupResponse {
2    tf2::Transform l_foot_goal_pose;
3    tf2::Transform r_foot_goal_pose;
4    tf2::Transform l_hand_goal_pose;
5    tf2::Transform r_hand_goal_pose;
6  };
```

Listing 4.2: The DynupResponse struct used to send the goal positions back from the engine at each timestep. It contains a transform for each endeffector.

After setting the stabilizers changing parameters, the stabilizing is applied. The result is passed to the ik, which then returns joint goals. The joint goals, as well as some progress feedback are published and it then is checked whether the dynup process is completed. Before starting the next iteration of the loop, `ros::spinOnce()` is called to ensure vital ROS processes can be run, before going into a sleep to keep up with the correct engine rate.

4.2. Engine

The dynup engine manages the splines. At start-up four splines are initialized, using the `bitbots_splines`[2] package. These quintic splines take three position and rotation arguments, as well as a time argument for every spline point. When a stand up request is made, each splines first point is set to the current position of the end-effector in order to keep continuity and not start with an abrupt and potentially dangerous movement. Since it is unclear how the robot has fallen and whether there are any obstacles in the way, the robot might not be able to reach a predetermined starting position and trying to reach this position in an abrupt motion might damage the robot.

When the request is made, the hardware control manager also passes the direction in which the robot has fallen, which is now used to determine which side we have fallen

[2] https://github.com/bit-bots/bitbots_motion/tree/master/bitbots_splines

onto and which motion has to be modelled with the splines. Depending on that decision the spline points are set (compare listing 4.3). The motion used follows the current keyframe approach used by the Hamburg Bit-Bots [16]. If the robot has fallen to the front and is lying in prone position, it has to push itself back onto its feet (Compare figure 4.2). To do so, in a first step the robot takes its arms above the head as far as possible (phase c). An additional spline point sidewards of the robot ensures that the robot does not try to reach that goal by moving through the ground (phase b). Simultaneously the legs are pulled closer to the body and rotated, so that the foot plates completely touch the floor (phase d and phase e). By moving the arms from above the head in front of the robot (phase f), and then down into the initial position (phase g), the robots torso is pushed up enough to get into a squad. This motion is supported by moving the feet back into the previous squat position. At this point the stabilizing is enabled to compensate for the overshoot that develops from the pushing. Since this robot model cannot stand in its zero position, an offset of 0.18 radians is added to the torsos pitch orientation, which is closer to the walkready position proposed in the `wolfgang_robot`[3] package. After a short pause to further mitigate the force, the robot rises by moving the feet away from the torso, up to a distance defined by a parameter dynamically reconfigurable during runtime. The duration of each spline point can easily be changed with dynamic reconfigure as well.

If the robot instead has fallen to the back and is lying in a supine position, a similar approach is taken from the other side (Compare figure 4.3): Instead of moving the arms above the head, it keeps them next to the robot, as far down as possible (phase a). While pulling the legs towards its torso, the robot pushes up onto its wrists (phase b). By rotating the arms around the shoulder pitch, the torso is further lifted, which allows the legs to be pulled under the torso (phase c). The arms are then extended to push the robot into an upright position, whilst simultaneously bringing the legs back into the squat position (phase d). From that point onward the motion continues in the same way as the stand up motion from the front.

The paths of the hand splines, as well as the right foot spline are defined relative to the robots base link located in the lower torso, as this is the format the IK solver expects. Since the arms positions are easier described relative to the robots shoulder link, a parameter is introduced that gathers the distance between the base link and the shoulders from the robot model. The same technique is used to create a parameter that holds the maximum arm length for those spline points that require the robot to stretch as far as possible.

The left foot spline, in contrast to the other splines is not described relative to the base link, but relative to the other foot instead. This is because we want to keep our feet parallel during the whole movement to reduce the degrees of freedom and therefore implicitly increase stability. However, this means that before making the IK requests, this spline has to be transformed. This is easily done by multiplying the left foots pose with the right foots pose.

The engine also provides an update method to get the should-be position of the end-

[3] `https://github.com/bit-bots/wolfgang_robot`

effector at a certain time as a tf2 Transform, which is later used by the ik to calculate the motor goals. The method uses the getTfTransform method on each of the splines to return the correct transform for the current time and then creates an DynupResponse message that contains these transforms.

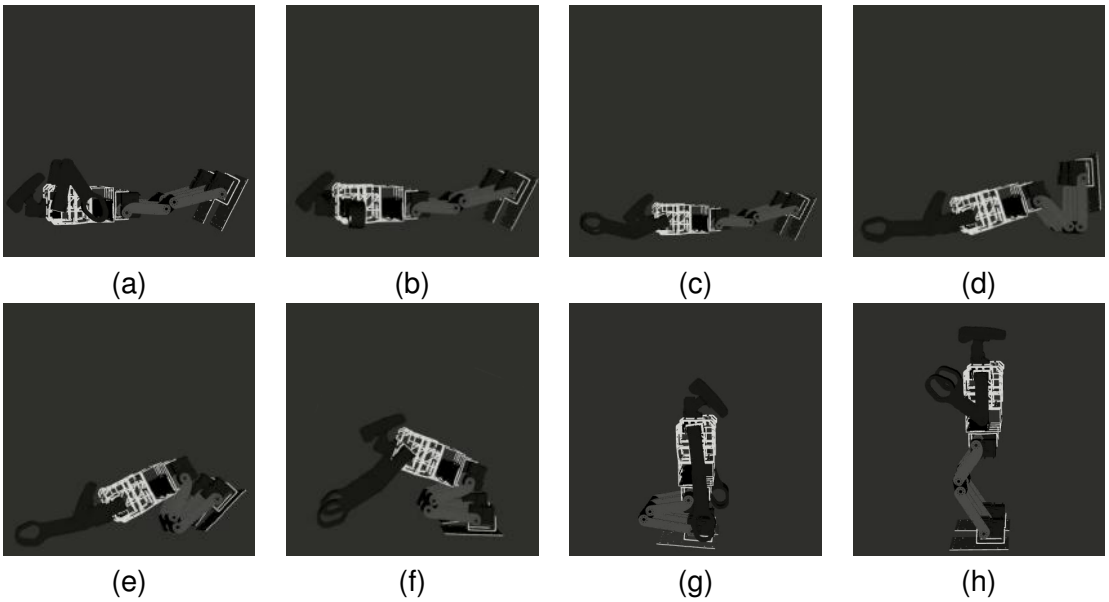


Figure 4.2.: Snapshots from the BitBots keyframe animation for getting up from a prone position, recorded in RViz and rotated to a realistic orientation, as RViz does not simulate physics.

```
1  r_hand_spline_.x()->addPoint(time_start, r_hand_pose.position.x);
2  r_hand_spline_.y()->addPoint(time_start, r_hand_pose.position.y);
3  r_hand_spline_.z()->addPoint(time_start, r_hand_pose.position.z);
4  tf2::convert(r_hand_pose.orientation, q);
5  tf2::Matrix3x3(q).getRPY(r, p, y);
6  r_hand_spline_.roll()->addPoint(time_start, r);
7  r_hand_spline_.pitch()->addPoint(time_start, p);
8  r_hand_spline_.yaw()->addPoint(time_start, y);
```

Listing 4.3: An example definition for a spline point for the right hand. `r_hand_pose` contains the current position of the right hand endeffector.

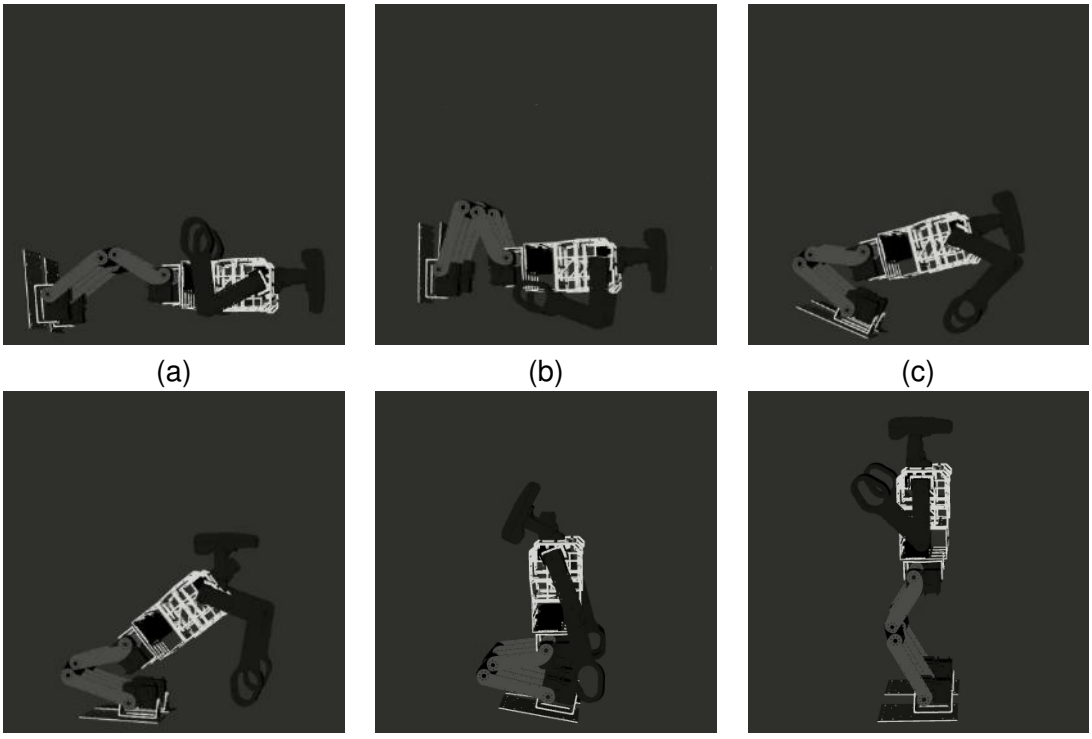


Figure 4.3.: Snapshots from the BitBots keyframe animation for getting up from a supine position, recorded in RViz and rotated to a realistic orientation, as RViz does not simulate physics.

4.3. Stabilizer

The initial approach for the stabilizing part was to use the DynamicBalancingGoal provided by the `bitbots_splines` package. These goals use the robot model to calculate where the centre of pressure would be based on the current joint positions and then adds corrections until the CoP is stabilized over a point defined by a parameter. Unfortunately a different IK interface which does not support the DynamicBalancingGoal had to be used due to reasons explained in section 4.4.

Because of this reason and since the reality gap meant the DynamicBalancingGoal was not precise enough, a PID controller was considered instead. With the `control_tools` ROS package the controllers were easily implemented and could now stabilize the robot more accurately as they are using real time sensor data instead of a model.

When the stabilize method is called, the stabilizer first checks whether stabilizing is enabled and required at the current time, since we don't want to try and stabilize the first part of the motion where it is virtually impossible for the robot to fall over since it is still lying on the floor. If these conditions are met, the stabilizer first transforms the right foot goal to a trunk goal, since we want to balance the trunks CoP over the support frame and therefore need a way to access the trunks pose. After that the PID controller requests are made, using the robots IMU to correct the error between the goal angle and the current angle of the robot according to the PID controller parameters (compare listing 4.4). This process is done both for the robots hip pitch and roll. Lastly the trunk goals rotation is set to the new values. The yaw stays unaltered since it is not beneficial to our stabilizing.

With the corrected error applied we transform the trunk goal back into the right foot goal to bring it into the required format for the IK solver. Regardless of whether we apply stabilizing, we also transform the left foot goal, which is currently relative to the right foot, to be relative to the base link like the other goals, for the same reason.

```
1  double goal_pitch, goal_roll, goal_yaw;
2  tf2::Matrix3x3(trunk_goal.getRotation()).getRPY(goal_roll,
       goal_pitch, goal_yaw);
3  double corrected_pitch = pid_trunk_pitch_.computeCommand(
       goal_pitch - cop_.x, dt);
4  double corrected_roll = pid_trunk_roll_.computeCommand(goal_roll -
        cop_.y, dt);
5  tf2::Quaternion corrected_orientation;
6  corrected_orientation.setRPY(goal_roll + corrected_roll,
       goal_pitch + corrected_pitch, goal_yaw);
7  trunk_goal.setRotation(corrected_orientation);
```

Listing 4.4: Stabilizing the robot with two pid controllers. Lines 1 and 2 turn the current goal rotation into a matrix. In lines 3 and 4 the error between the goal orientation and the measured orientation is calculated. Lines 5 to 7 turn the results into a quarternion and overwrite the rotations of the goal.

The stabilizer also contains a reset and various setter functions.

4.4. IK

Initially the IK calls were made with Eigen vectors. While that approach worked, the requests were processed slowly, which resulted in a frame rate of approximately 30 Hz. Since our motors update their position 200 times per second, the proposed motion needs to reach at least 200 Hz or else some motor goals get lost and as a result the movement looks choppy, makes the motion less stable and causes problems when correcting the error during stabilizing. When analysing the runtime with the swri profiler[4] it became obvious, that most of the time was spent on the four IK requests. Therefore a different interface that uses geometry messages was used instead, which is significantly faster and reaches an average of well above 200 Hz. This decision also caused the changes mentioned in section 4.3.

The robots arm has only three degrees of freedom. This means that not all poses in the range of the arm can be reached precisely. This lead to the decision to also allow approximate solutions. While this approach might also return unwanted results, it is the only way to get the arms to reliably reach their approximate goal position at all. By setting the timeout on the IK function sufficiently high, dangerously wrong movements can be mostly eliminated. By choosing to allow approximate solutions, the KDL kinematics plugin could not be used either, which compared to the BioIK kinematics plugin would have probably been even faster, since it does not allow such an option. This was no problem, however, since the computation was already fast enough as is.

The IKs calculate method starts with setting the kinematic query options to allow approximate solutions. After that the tf2 transforms used for internal communication are converted into geometry messages accepted by the IK. After these preparations the four IK requests are made, while before each request the link transforms need to be updated. If all requests returned a valid solution, all results are combined into a single JointGoals object, which is then returned to the node, from where it is published.

4.5. Visualizer

The visualizer is a simple tool to publish the splines as markers that can be displayed in RViz. It sets up a publisher and is then invoked whenever it is called from the node. It inherits from the abstract visualizer class from the `bitbots_splines` package. The markers are generated by gathering the spline points from each spline, creating line strips for them and then publishing them as a marker array. While this approach does not display the interpolated spline and thus is not very precise, it helps understanding the general concept and assists in debugging potential errors. The visualizer also displays the orientation of the axes in each spline point to assist in debugging orientation errors. The functionality of the visualizer can be seen in figure 4.4.

[4] `https://github.com/swri-robotics/swri_profiler`

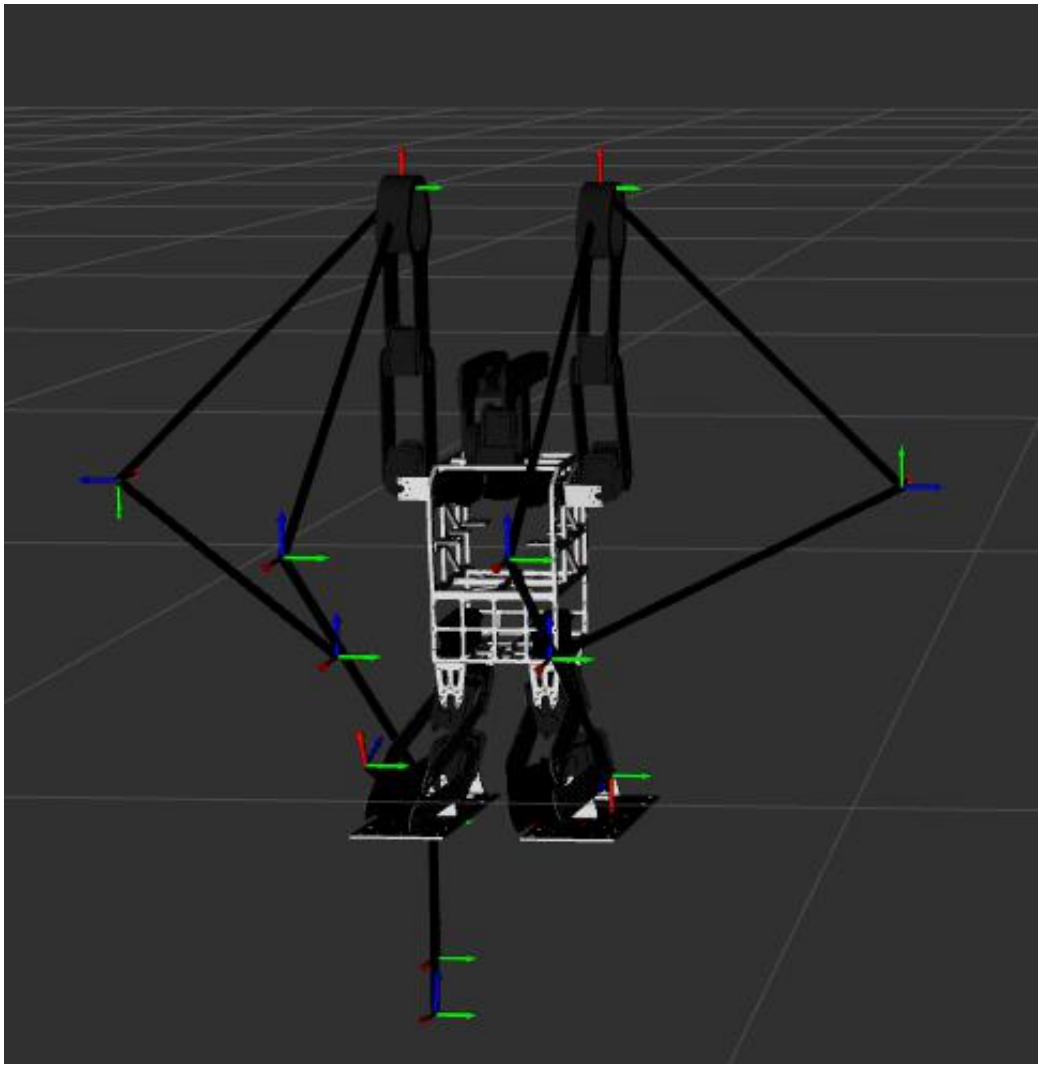


Figure 4.4.: Debug visualizer displaying splines of the front stand up motion. Splines are displayed in black, coordinate systems are shown at each spline point indicating the orientation.

4.6. Parameter and PID tuning

The parameters for the two PID controllers were determined using the Ziegler-Nichols method [33] [34]. For this procedure first, all gains are set to zero. Starting with the trunk pitch controller, the p gain, K_p, is gradually increased until a point is reached where the system starts to oscillate infinitely. This point for the wolfgang robot platform is marked as $K_{pcrit} = 0.70$. Then the length one oscillation period is measured as $T_{crit} = 0.5$. From these two parameters the three gains can be calculated using the following equations:

$$\begin{cases} K_p = 0.6 \cdot K_{pcrit} = 0.6 \cdot 0.70 = 0.42 \\ K_i = 1.2 \cdot \frac{K_{pcrit}}{T_{crit}} = 1.2 \cdot \frac{0.7}{0.5} = 1.68 \\ K_d = \frac{3 \cdot K_{pcrit} \cdot T_{crit}}{40} = \frac{3 \cdot 0.70 \cdot 0.5}{40} = 0.02625 \end{cases} \tag{4.1}$$

The same process was then repeated for the trunk roll controller with $K_{pcrit} = 1.27$ and $T_{crit} = 0.3125$, which results in $K_p \approx 0.76$, $K_i \approx 4.8768$ and $K_d \approx 0.0298$. During testing this controller was deemed too aggressive, especially in regards to the accumulated error, which is why the i gain of the trunk roll controller was reduced to $K_i = 2.0$. A blind test with an unbiased third party confirms that this causes a better overall performance.

The parameters corresponding to the robots goal state were taken from the already existing walkready pose and needed no further tuning. The walkready pose has a trunk pitch of 0.18 radians forward. The trunk height is 38 centimetres with a foot distance of 20 centimetres, centre to centre. The minimum possible leg length was measured to be 21 centimetres. The timings of each step were tuned by first setting them to any value that is sufficciently high as that the robot does not fail and then one by one decreasing the time until the motion fails, then keeping the last stable timing. A full list of parameters can be found in appendix B. All parameters can be fine tuned easily using dynamic reconfigure.

5. Evaluation

This chapter evaluates the performance of the approach proposed by this thesis compared to several different approaches used by the Hamburg BitBots. Section 5.1 describes the experiments taken in a simulated environment, whilst section 5.2 describes the transfer of those findings to the real world, as well as further experiments on the real robot.

5.1. Simulation

In order to reduce wear on the robots hardware, most of the experiments were taken in a simulated environment. Even though the used simulation tries to represent the real world as close as possible, a reality gap still exists and a success in this environment does not necessarily entail a similar result in the real world. However, the information gathered still serves as a general proof of concept and showcases a use of the program as it would be used in the RoboCup 3d simulation league[1]. The simulation has been modified and optimized for the wolfgang platform in a previous thesis, which helps keeping the reality gap as small as possible [35].

5.1.1. Setup

The experiment is set up in the Gazebo simulator[2]. For each attempt the robot is placed on a simulated playing field, lying either face up or face down. The robot is not placed flat onto the field, but instead standing in its walkready position and then rotated along the y axis past its tipping point, so that the falling detection detects the robot first as falling and then fallen to either side and makes the dynup request. The stand up routine could be invoked manually after placing the robot, but it was chosen not to do so, in order to simulate a more realistic starting position, since in the real world the robot would have fallen as well.

The simulation takes the texture of the turf into account, but it does not consider the direction of the blades. Therefore it is not necessary to change the direction the robot is facing during each attempt. However, since the robot tends to slide more on the simulated turf, compared to reality, it is important that the robot is far enough away from the edge of the field, so that it does not leave by accident.

A Gazebo model plugin was created to constantly apply forces to the robot, to add noise to the simulation, which otherwise would always return the same result. In every

[1] `https://ssim.robocup.org/3d-simulation/`
[2] `http://gazebosim.org/`

tick the plugin generates a random force between zero and 100 newton in a random direction on the x-y-plane. The force is applied to the centre of mass of the robots base link, i.e. the torso. This force is large enough to create randomness in the results, but not large enough to majorly alter the course of the attempts or even topple over the robot in a normally stable position. Forces along the z axis are not applied, since an unfortunate succession of forces could send the robot flying and forces in that direction, other than the gravitational pull are highly unlikely to happen in reality. Even though this setup does not perfectly resemble reality, it gives a common ground to compare the different approaches sturdiness. The source code of the plugin can be found in appendix C.

The experiment compares four different stand up routines: The dynup system proposed in this work, the dynup system, but with the stabilizer disabled, a combined system that uses a keyframe based approach to reach the squad position and the dynup approach to get from squat to standing[3], and lastly a purely keyframe based approach[4]. For each candidate ten attempts per direction are taken. Each candidate is rated by the number of successes, as well as the average, maximum and minimum time per attempt. The time is measured from the moment the request is made to the moment the robot reaches its walkready position. The time for failed attempts is discarded. Since the simulator can not properly replicate real time, only simulated time is considered for this setup. For each failed attempt the cause of failure is noted, e.g. fallen to the front, fallen to the back or overload error.

The case of a robot intitially fallen to its side is not considered, as the typical solution to that problem is to roll over to either front or back and start to get up from there.

5.1.2. Result

The experiment shows that the proposed dynup approach has the highest success rate of the four candidates with 18 of 20 successful attempts, followed by the disabled stabilizing attempts with 16 successes, the keyframe approach with 14 successes and lastly, the hybrid approach with 12 successful attempts (compare figure 5.1). It is noteworthy, that both the hybrid, and the keyframe approach performed better when only considering the attempts from the back. Both failed attempts from the dynup system were due to not being able to directly roll over into the squat position. This most likely happened, as the robot has its hands way closer to its feet in the dynup approach (compare figure 5.2). Therefore the robot has no stable fallback position in between the two poses. Changing the distance would probably prevent these failures. It should also be noted, that in both cases the robot was capable to recover in a second attempt and reach the walkready pose in less than 20 seconds as required by the rules.

Three of the four failures of the no stabilizing attempts happened since the robot was not able to balance out the forces generated from pushing up onto its feet, once from back to front while still in the squat, and twice from front to front after getting up. These

[3] https://github.com/bit-bots/bitbots_motion/tree/ef41ac
[4] https://github.com/bit-bots/wolfgang_robot/tree/65888ee

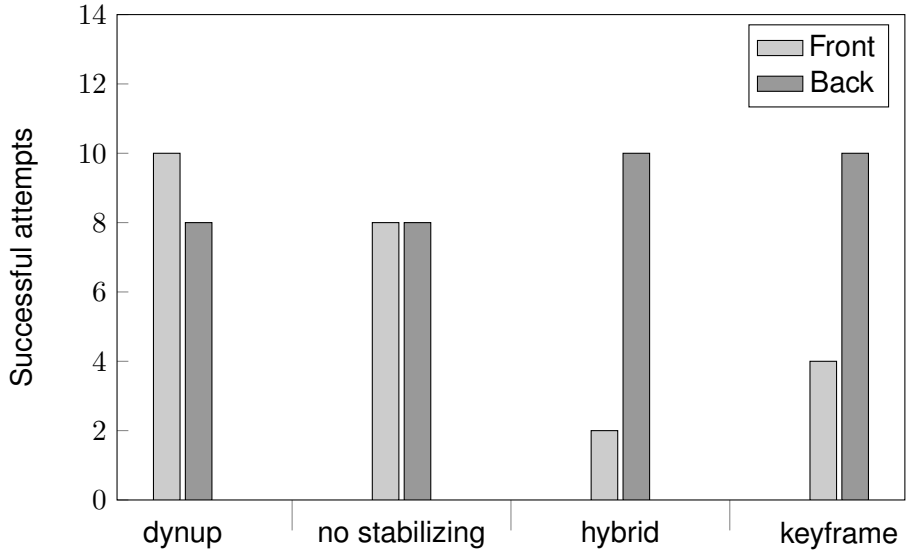


Figure 5.1.: Number of successful attempts per approach, out of ten trials per direction. x axis lists the different approaches, y axis displays number of successful attempts.

failures could have been compensated with the PID controller, if it was enabled. The fourth failure happened due to the same reasons described in the previous paragraph.

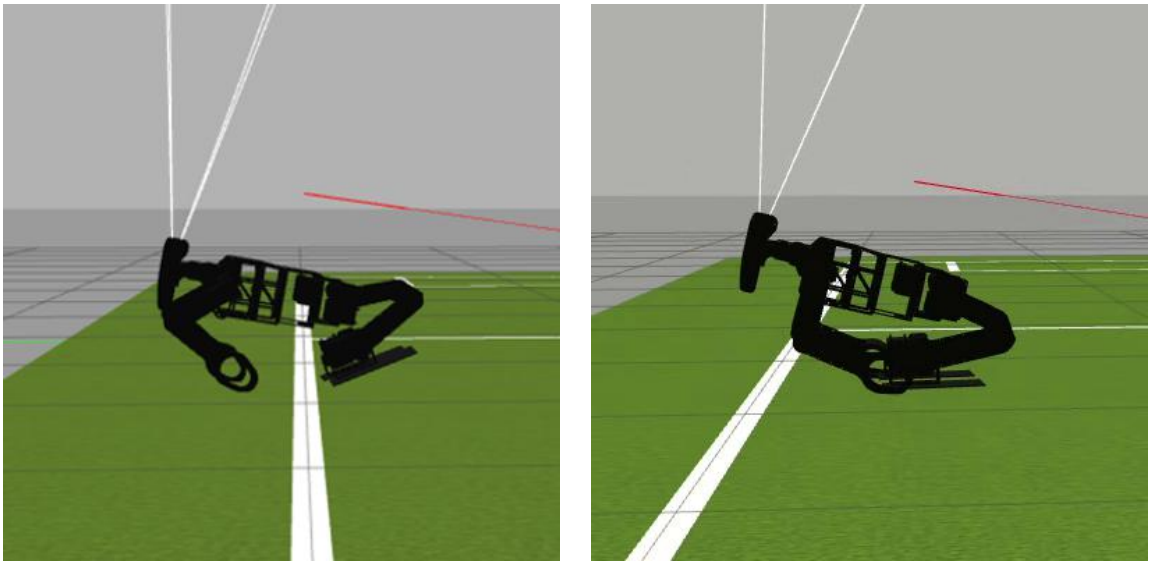
The hybrid approach had the most problems with getting up from the front: Seven times the robot fell, because it didn't manage to push itself onto its feet strong enough, the eighth time it pushed too strongly and fell backwards while raising its body. Whether this approach ever worked reliably in the simulation or whether a bug caused this huge amount of failures is unclear. It is likely that a better result could be achieved with several hours of manual tuning, but it would likely still not reach the results proposed by this thesis, as the balancing is still problematic. This especially shows true since the approach succeeds when no disturbance is applied.

Lastly, the purely keyframe based approach failed 6 times in total when trying to get up from the front, four times as it couldn't push up into the squat, once because it fell over to the back while getting up and once because it fell to the side. The attempt where it fell to the side should be regarded as an outlier, since it happened in a highly stable position, before even reaching the squat and high forces were necessary to tip the robot over out of that position. It is highly unlikely that this situation would occur again, neither simulated, nor in the real world.

As can be seen in figure 5.3, both dynup based attempts perform significantly faster than the keyframe based attempts. On average the dynup system performed 3.2 seconds faster than the keyframe based approach when getting up from the front, and 2.8 seconds faster from the back. Similar results apply to the hybrid approach with an improvement of 3.1 seconds from the front and 2.7 seconds from the back. Given the

fact that the keyframe based approach only takes 9.3 seconds from the front and 8.0 seconds from the back, that is an improvement by 33.4 to 34.3 %. In reality this means that, whereas the keyframe approach could barely do two attempts before the robot was removed for being incapable, the dynup approach can complete three attempts in the same time and still has some time to spare.

The systems performance is relatively lightweight with an average CPU usage of 2.5% when idle, and 10.91% when performing, on an Intel NUC6i5SYK, which is the computer used in the wolfgang platform and also commonly used in other robot platforms, and an average memory usage of 237.6 Mb.



(a) (b)

Figure 5.2.: Comparison between the arm positions of the backwards stand up motion. (a) shows the keyframe approach, while (b) shows the dynup approach. During the transfer to the real world, the dynup system was modified to more closely resemble the motion shown in (a).

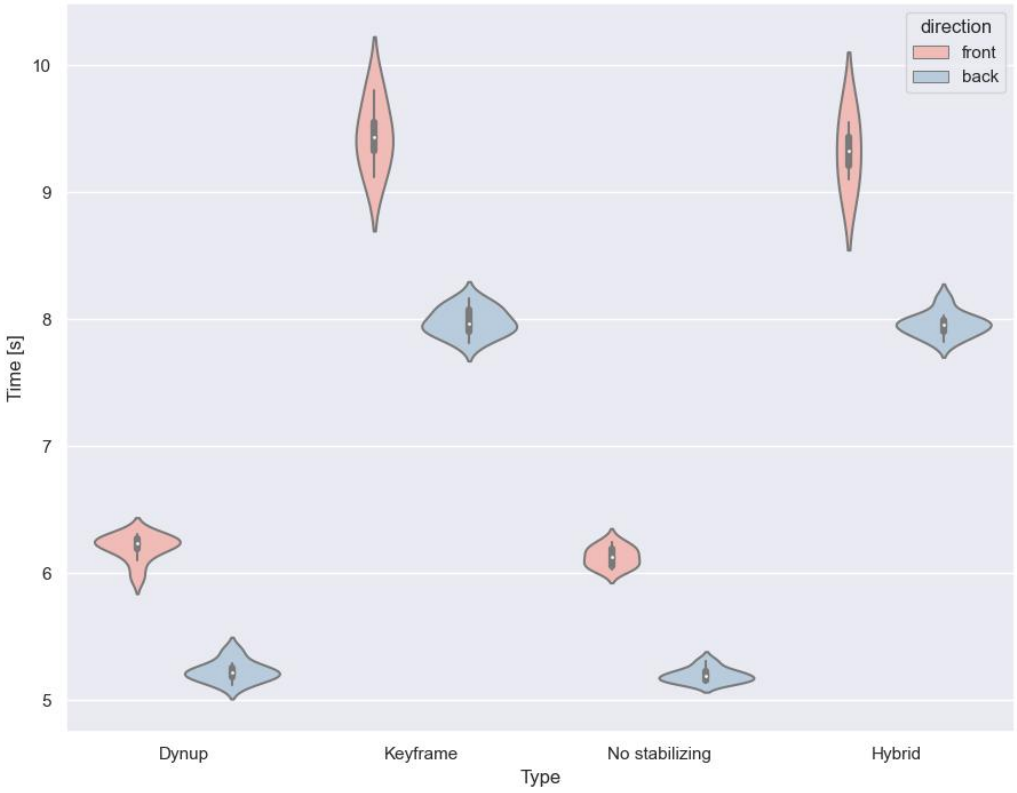


Figure 5.3.: Average, minimum and maximum time needed to complete the stand up attempt from registering as fallen to reaching walkready. x axis displays the different approaches, y axis displays time per attempt in seconds. For each approach the red line displays the attempts from prone position, the blue line displays attempts from supine position.

5.2. Transfer to real world

After the experiment was conducted, several changes were applied to make the system feasible to use on a real robot. As mentioned in the previous section, the distance between the arms and the legs was increased to give it a more stable intermediate position as it caused problems in some situations. The coordination between the arms and the legs was also restructured, as it caused problems in some situations. The front motion needed not to be altered and could be transferred without any issues.

Testing on the robot shows that the motion works in multiple consecutive attempts without error. Figure 5.2 shows a successful attempt at getting up from the front with the proposed system. After the robot has fallen (phase a), it enters its falling position (phase b). Usually, this would happen while the robot is falling to protect the arms from the high forces, however, since the falling detection currently is disabled, this was invoked manually, after the robot was placed. Next, the robot moves its arms to the side and then to the front (phase c and d). Figure phase e and f show the robot pushing itself up into a squat position. As soon as it reaches the squat (phase g), the stabilizing is enabled to compensate for the applied forces and safely enable the robot to rise up into its walkready position (phase h and i).

For getting up from the back, an successful attempt is shown in figure 5.2. After being placed on its back, the robot pushes its arms into the ground, rolling itself forward onto its feet (phase b to f). In contrast to the animation used for the simulated experiment, the distance between arms and legs has been increased to allow a safe resting point in phase e . From this position, the robot forcefully pushes itself onto its feet (phase f), at which point the stabilizing is enabled. From here on the robot gets up to its walkready pose, similar to getting up from the front.

To prove the efficacy of the stabilizing, the readings of the IMU was recorded (compare figure 5.6). Figure 5.6.a shows an stand up attempt from the front with the dynup system. It can be seen how the pitch returns to around -10 degrees in the beginning of the recording, to provide enough momentum to get onto the robots feet. The excess momentum is quickly mitigated by the PID controller, clearly visible through the oscillations recorded in the graph. After a moment to stabilize, the robot starts to rise up, returning from its leaned back position to the walkready orientation. The overshoot of that motion is also corrected through the stabilizer, which results in a second set of oscillations. Especially during the rise period a disturbance in the robots roll becomes visible, mitigated through pid control and also clearly visible in the graph.

In comparison, figure 5.6.b shows the attempt with the solely keyframe based approach, without any stabilizing. The abrupt changes of the IMU's roll between zero and ± 180 whenever the robot is close to its prone or supine position stem from gimbal lock of the IMU and can be ignored here, as well as in all other figures. In this figure however, it can be seen, that upon pushing to the squat position, the robots pitch has a way higher impact compared to the dynup approach. With the keyframe animation the amplitude of the peak compared to the goal is way higher and in this case after swinging back and forth once, ultimately leads to the robot falling over to the back. In

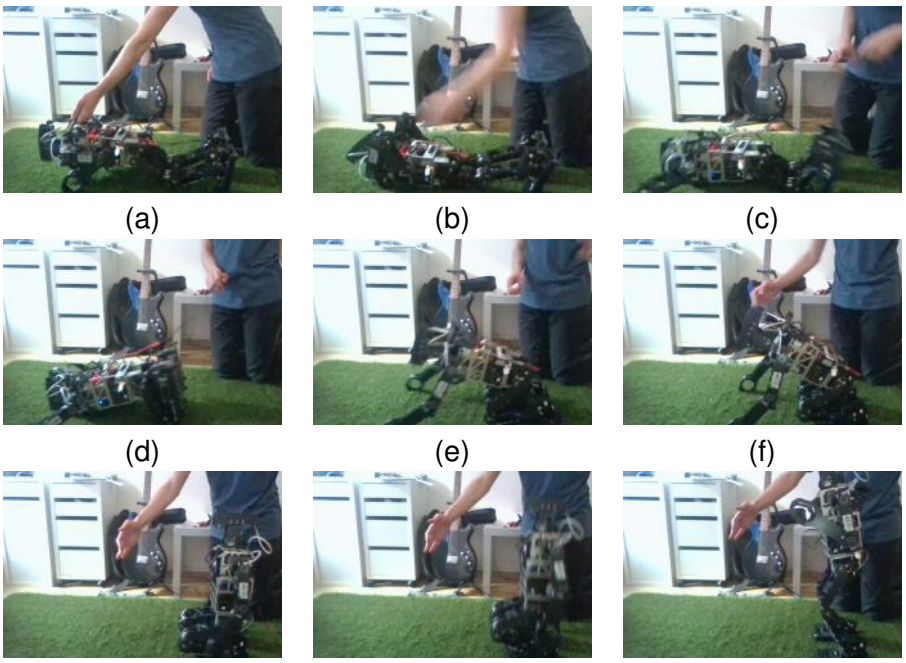


Figure 5.4.: Snapshots from an dynup attempt at standing up from a supine position.

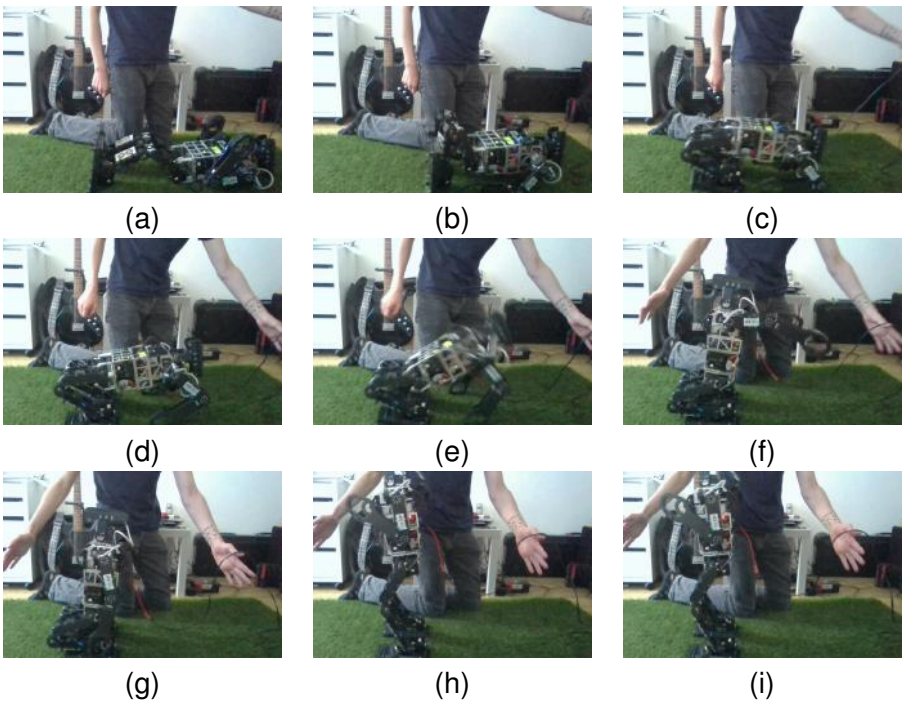


Figure 5.5.: Snapshots from an dynup attempt at standing up from a prone position.

this case the robot has been manually reset after falling, which explains the sudden seemingly normal IMU values.

Figures 5.6.c and 5.6.d show the respective attempts from the back. The big abrupt roll changes in the beginning of 5.6.c stem from the unstabilized movements prior to the squat position. When pushing up into the squat, the pitch increases in a smooth line and is balanced out once it reaches its goal. Compared to that, the keyframe approach in 5.6.d shows a jagged line during the push to the squat, which leads to instability, and ultimately failure of the attempt.

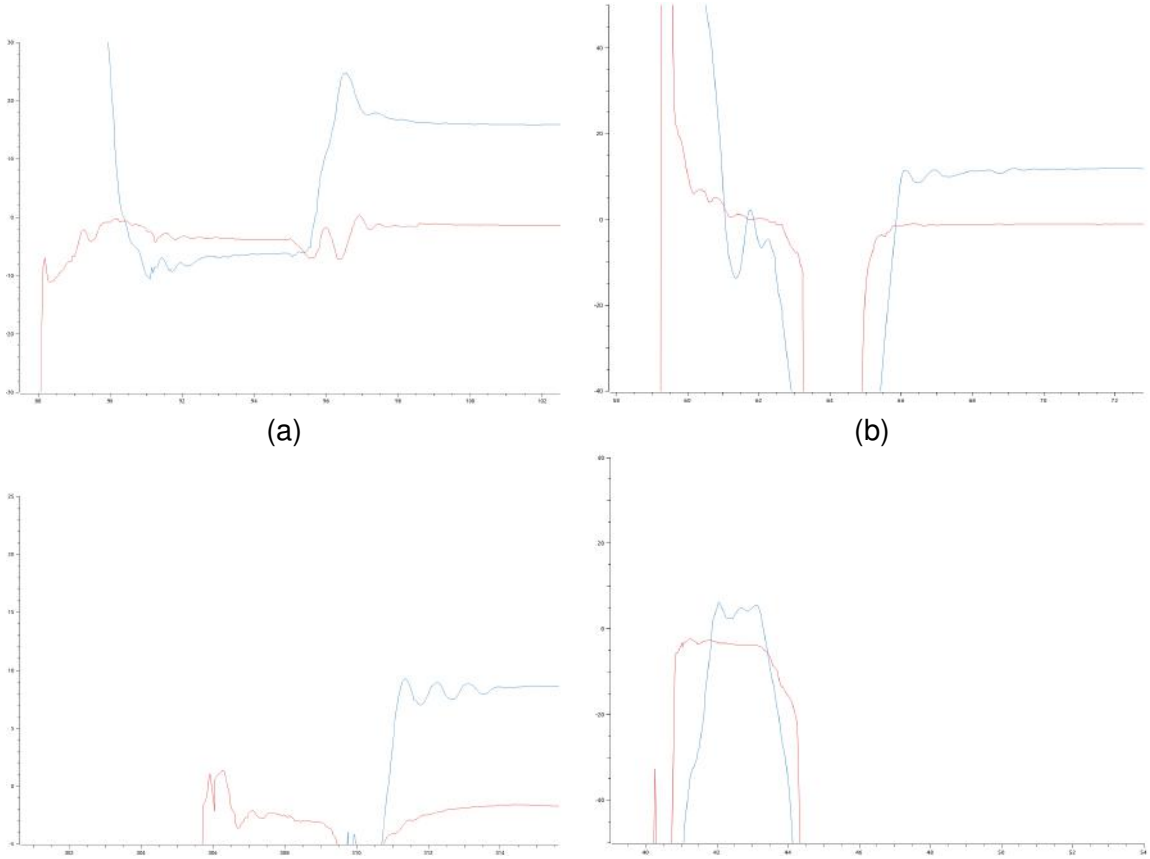
In an additional run the efforts on each motor were measured to compare whether one approach strains the motors more. The dynup approach and the keyframe approach were compared due to their differences in the previous experiments. For both approaches the average, minimum and maximum effort, as provided by the `ros_control` package were measured. `ros_control` calculates the effort based upon the motors current. This approach is rather noisy and measurements might be severely wrong, so the results have to be treated accordingly. To help put things into perspective, the efforts of the robot standing upright were also measured as a baseline. For each approach and each direction, the average over three attempts was calculated to minimize the impact of outliers.

Upon investigating the results, no major advantage of either approach can be seen. The average efforts are all relatively low, as the motions feature long stretches with close to no movement and thus no huge efforts on the motors. Therefore it is more interesting to have a look at the effort spikes and compare these large forces, even if they only last for a short amount of time.

Even though the head motors are mostly irrelevant to the motion, a significantly larger effort can be registered on the HeadTilt motors in the keyframe animation. This is due to the fact, that the keyframe animation controls the head to move it out of the way, as to not damage the camera. The dynamic motion does not control the head, as this task will be taken by the head behaviour in the future. Since the motors never move, the efforts are close to the baseline.

For the front motion, the dynup approach is putting a significantly higher strain on the legs, while the arms are being used more in the keyframe approach. Especially the elbows experience a larger strain with a peak almost twice as high as in the dynup approach. The largest effort is measured in the right shoulder pitch motor with -12.05 Nm while using the keyframe approach. When calculating the average over the absolute extreme values, the dynup approach performs slightly better overall with an average of 4.15 Nm versus 5.26 Nm for the keyframe approach. For this calculation the head motors were disregarded due to the reasons explained above.

For the back motion a similar picture can be seen. Again, the dynamic approach uses the legs more, while the keyframe approach uses the arms. However, in this case when building the average, the keyframe approach gives a better performance with an average of 3.28 Nm versus 4.41 Nm for the dynamic approach. A full list of the measurements can be found in appendix D.



(a) (b)

(c) (d)

Figure 5.6.: Imu readings from different stand up attempts. Blue represents the IMU's pitch, red represents the roll. x axis displays time since recording start in seconds, y displays the measured angle. (a) and (b) show attempts from the front, (c) and (d) show attempts from the back. (a) and (c) have been taken with the Dynup approach, while (b) and (d) have been taken with the keyframe approach.

6. Discussion

The previous chapter shows, that the proposed system works reliably in the simulation and a transfer to the real world was conducted successfully. It performs faster than currently used systems and does so while still maintaining stability.

While this approach increases the amount of successful attempts, it does not have any fail-safes in case the robot falls during an attempt. The largest issue discovered during development was the fact, that the robot continues its attempt, even if it fell during the attempt, potentially damaging the robot in the process. This is especially problematic, since the PID controller keeps trying to correct an error, that can no longer be fixed, which creates fast and large movements in unwanted directions, mainly into the floor or into other robot parts. This could be fixed by externally shutting the process down, when a fall is detected, i.e. by letting the fall checker abort the dynup action and replace it with a falling action. However, this behaviour is not implemented yet.

Another issue, that holds up for virtually all current stand up approaches is the huge performance difference depending on the overall state of the robot. Even though the proposed system tries to compensate for minor complications, larger scale issues like broken gears, connection losses or motor overloads will almost certainly result in a failure. This holds also true for the motor performance loss due to increasing temperature. If the motors are being used for a long period of time, they tend to heat up, which causes the motors to not be able to produce their full power and therefore not reaching all goals correctly. This is especially noticeable in the elbow and knee motors, which after a number of attempts give in. The elbow motors only pose a problem when getting up from the back, since they don't have to exert a huge amount of force otherwise, but the knee motors are being used in any attempt and sometimes fail to reach their goals. When this happens, the robot usually doesn't have enough force to raise up straight, even though the stabilizing tries to mitigate the error, but instead catapults itself backwards. There is no real possibility to fix this issue at the current point, however it is planned to replace the knee motors with more powerful ones, and add torsion springs to the knees in the future, to resolve these issues. Issues also ensue from voltage drops when running the robot on battery power for prolonged periods of time. Similarly to the temperature induced performance loss, the batteries voltage drops over time and thus the motors lose power.

Currently stabilizing is not applied to the whole motion, but only to the last part from the point on where it reaches the squat position. This means, that it does not try to correct any destabilizing movement made prior to this point. That is, because it would be difficult to generate a trajectory for the optimal centre of pressure, which would mean a huge amount of work for close to no benefit. The motion itself is quite stable up to the point where the robot leaps onto its feet and small disturbances cannot knock the

robot over at that point. Only a large amount of force could cause a failure in this part of the motion, like another robot falling onto our robot. In these cases PID stabilizing wouldn't be able to compensate for the error either. At the moment the system also only takes in data from the robots IMU. Even though there is additional data available from eight pressure sensors mounted to the foot plates, this data is not considered, since parsing this information is difficult and the sensors are not available in the simulation and unreliable on the real robot at the current point of time. When this situation changes, combining these sensors may be considered. Nevertheless the system works reliably even without considering the pressure sensors. The only apparent drawback is the fact that false sensor values cannot be detected.

Besides the issues listed above, the system works reliably on our robot platform. The transfer to a new environment, which is especially interesting in the RoboCup context, where competitions are held in different countries with different kinds of artificial turf, can be easily accomplished by tweaking a few parameters with dynamic reconfigure. This is a huge improvement compared to the current keyframe approach, where a difference in terrain meant completely rebuilding the animation for this exact situation by modifying each motors position in each frame. In a competition environment, where only limited time is available to get all components running, assigning two persons and one of the robots to this task for several hours is a huge setback and results in having to neglect other important tasks. Due to the automatic stabilizing, there is even the possibility that the new system adapts to new turf without the necessity of any manual tuning at all.

The system is also very versatile and can be easily transferred to any humanoid with an IMU. This can be accomplished by changing the names of the end-effector frames to the ones provided in the code (or vice versa). The parameters describing the robots physique need to be set to the new values and the PID controllers might need to be retuned. These changes should suffice to get the system running on any other robot platform. As the dynup system describes the motion in a Cartesian space rather than a joint space, the motor names, numbers or degrees of freedom of each kinematic chain should not matter to the ik solver. A low degree of freedom however could mean, that no good solutions can be found, as none exist. The system will still return an approximate solution, which might work, but is not guaranteed to.

Compared to the other approaches tested, the dynup approach performs significantly faster than any other approach. In the RoboCup environment speed is generally more important than safety, for multiple reasons. Firstly, the allowed time to get up is restricted to 20 seconds. As the time is not reset between attempts, a faster average performance means, that the robot gets more attempts to get up, even if it fails more often. Secondly, the robot is incapable whilst lying on the ground. This means it can generally neither influence nor observe play. During any amount of time spent on the floor, gameplay can change significantly and a recovered robot needs more time to orient itself. As lying on the ground is generally an undesirable position, any effort in reducing the time spent that way is an advantage.

7. Conclusion and Future Work

In section 7.1 the thesis is concluded. The findings of this work are summarized and put in context with the current state of research. Section 7.2 gives an outlook on possible future work.

7.1. Conclusion

In this thesis a stand up routine is created using quintic spline interpolation in a Cartesian space instead of the commonly used keyframe approach. This routine performs faster and more reliably than other systems currently in use. It can easily be adapted to other environments, or other robots.

Even though the proposed system is not flawless and improvements can be made, it is a huge advancement from the current state of the art and highlights a new possible solution to a problem that has not been approached in the recent years.

By turning the problem into a closed loop system it is now possible to react to environmental influences and account for the difference between reality and calculations. The feedback based approach can correct the difference between the set goals and the actually reached position, which can diverge due to various factors, such as external factors, e.g. wind, uneven floor or disturbances by other agents, or internal factors, such as defective or overloaded motors, that do not quite reach the goal. Further, it was proven through experiments, that the stabilizing process has a significant impact on keeping the centre of mass inside the support frame and therefore outbalancing strong forces induced during the procedure. Additionally it was concluded that the amount of strain applied to the motors is similar to existing approaches and thus is no reason to choose those approaches over the proposed one, or vice versa.

The code produced for this thesis has been made publicly available on GitHub under the MIT license[1]. Improvement and reuse are both allowed and encouraged. By keeping the project open source, further research is aided.

Overall the results discussed in this thesis show, that a closed loop stand up system with quintic spline interpolation and PID control is possible and desirable compared to other alternatives.

[1] `https://github.com/bit-bots/bitbots_motion/`

7.2. Future Work

Even though the dynup model performs good in its current state, various changes could be explored.

The most interesting modification would be the exchange of the PID controller, from controlling based on the IMU measurements to controlling based on the foot pressure sensors. Doing so would grant more accurate information about the centre of pressure in context of the support polygon and might allow an even better error correction. However, parsing the data of the eight pressure sensors to a single PID controllable value is difficult and the sensors are not working reliably at the moment. A combination of both sensor types would be an interesting way to both increase accuracy and create redundancy by being able to calculate a result, even if one of the two sensors does not return information, or even do damage control if one sensor returns false information. The effectiveness of these changes still needs to be evaluated.

Another step that should still be taken is applying automated parameter tuning to the project. Currently, an tree structured parzen estimator for parameter optimization based on Optuna[2] is being developed by the Hamburg Bit-Bots. This system takes input from dynamic reconfigure and automatically tries to find the best possible parameter set. Comparing the optimized to the calculated parameters could give further information on the parameter tuning process and potentially increase the performance of the dynup system further.

During the creation of this thesis a new and more detailed URDF model for the robot was proposed. Testing the system on the new model, as well as running the system on our robot as soon as the hardware changes discussed in section 6 are applied are both necessary steps to be able to run the dynup system in the next competition. The added torsion springs in the knees are intended to lower the effort applied by these motors further, possibly creating a more distinct advantage for the knee heavy dynup method over the keyframe approach in regards of motor wear. However, these changes to the kinematic chain might bring unforeseen dynamic forces, that could cause problems.

Lastly, as mentioned in section 6, modelling the centre of pressure as a trajectory throughout the entire motion could, even though it does not initially seem so, still improve the dynup approaches performance. This should at least be considered in further development.

[2] `https://optuna.org`

Bibliography

[1] RoboCup Humanoid Technical Committee. *RoboCup Soccer Humanoid League Laws of the Game 2018/2019*. 2019.

[2] Eiichi Osawa et al. “RoboCup: the robot world cup initiative”. In: *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-1996), Kyoto, Japan*. 1996, pp. 9–13.

[3] Max Schwarz et al. “NimbRo-OP humanoid teensize open platform”. In: *In Proceedings of 7th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Osaka*. Citeseer. 2012.

[4] Dynamixel. *MX-64T/R/AT/AR*.

[5] Dynamixel. *MX-106T/R*.

[6] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[7] Hyeong Ryeol Kam et al. “RViz: a toolkit for real domain data visualization”. In: *Telecommunication Systems* 60.2 (2015), pp. 337–345.

[8] Ioan A. Sucan and Sachin Chitta. *MoveIt!*

[9] P Ruppel. “Performance optimization and implementation of evolutionary inverse kinematics in ROS”. PhD thesis. MA thesis. Universität Hamburg, 2017. URL: https://tams. informatik. uni . . ., 2017.

[10] Carla Elena González Uzcátegui. “A memetic approach to the inverse kinematics problem for robotic applications”. PhD thesis. Universidad Carlos III de Madrid, 2014.

[11] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[12] Carl Runge. “Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten”. In: *Zeitschrift für Mathematik und Physik* 46.224-243 (1901), p. 20.

[13] Luigi Biagiotti and Claudio Melchiorri. *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008.

[14] Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

[15] Rames C Panda. *Introduction to PID controllers: theory, tuning and application to frontier areas*. BoD–Books on Demand, 2012.

[16] Timon Giese. “Entwicklung einer Benutzerschnittstelle zur Echtzeitbearbeitung von Roboteranimationen”. PhD thesis. Universität Hamburg, Fachbereich Informatik, 2016.

[17] Yoshihiro Kuroki et al. “Motion creating system for a small biped entertainment robot”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 2. IEEE. 2003, pp. 1394–1399.

[18] Jörg Stückler, Johannes Schwenk and Sven Behnke. “Getting Back on Two Feet: Reliable Standing-up Routines for a Humanoid Robot.” In: *IAS*. 2006, pp. 676–685.

[19] Hirohisa Hirukawa et al. “The human-size humanoid robot that can walk, lie down and get up”. In: *The International Journal of Robotics Research* 24.9 (2005), pp. 755–769.

[20] I.Khokhlov N.Koperskii V.Litvinenko I.Osokin I.Riakin R.Savlaev S.Semendiaev P.Senichkin A.Babaev E.Davydenko. “Starkit - Extended Abstract”. In: 2020.

[21] Jun Morimoto and Kenji Doya. “Reinforcement learning of dynamic motor sequence: Learning to stand up”. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*. Vol. 3. IEEE. 1998, pp. 1721–1726.

[22] Jun Morimoto and Kenji Doya. “Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning”. In: *Robotics and Autonomous Systems* 36.1 (2001), pp. 37–51.

[23] Libo Meng et al. “Gait Transition Between Standing and Falling Down for a Humanoid Robot”. In: *IFToMM World Congress on Mechanism and Machine Science*. Springer. 2019, pp. 2501–2509.

[24] Michael Mistry et al. “Sit-to-stand task on a humanoid robot from human demonstration”. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2010, pp. 218–223.

[25] Richard Paul. *Modelling, trajectory calculation and servoing of a computer controlled arm*. Tech. rep. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1972.

[26] Rahee Walambe et al. “Optimal trajectory generation for car-type mobile robot using spline interpolation”. In: *IFAC-PapersOnLine* 49.1 (2016), pp. 601–606.

[27] Zhe Tang, Changjiu Zhou and Zenqi Sun. “Trajectory planning for smooth transition of a biped robot”. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*. Vol. 2. IEEE. 2003, pp. 2455–2460.

[28] H. Gimbert L. Gondry L. Hofer O. Ly S. N’Guyen G. Passault A. Pirrone Q. Rouxel J. Allali R. Fabre. “Rhoban Football Club – Team Description Paper Humanoid Kid-Size League, Robocup 2018 Montreal”. In: *RoboCup Symposium*. 2018.

[29] Marc Bestmann et al. “Hamburg bit-bots and wf wolves team description for RoboCup 2019 humanoid KidSize”. In: *RoboCup Symposium*. 2019.

[30] Nguyen Gia Minh Thao, Duong Hoai Nghia and Nguyen Huu Phuc. “A PID backstepping controller for two-wheeled self-balancing robot”. In: *International Forum on Strategic Technology 2010*. IEEE. 2010, pp. 76–81.

[31] Safa Bouhajar et al. “Trajectory generation using predictive PID control for stable walking humanoid robot”. In: *Procedia Computer Science* 73 (2015), pp. 86–93.

[32] Paul EI Pounds, Daniel R Bersak and Aaron M Dollar. “Stability of small-scale UAV helicopters and quadrotors with added payload mass under PID control”. In: *Autonomous Robots* 33.1-2 (2012), pp. 129–142.

[33] Anthony S McCormack and Keith R Godfrey. “Rule-based autotuning based on frequency domain identification”. In: *IEEE transactions on control systems technology* 6.1 (1998), pp. 43–61.

[34] John G Ziegler, Nathaniel B Nichols et al. “Optimum settings for automatic controllers”. In: *trans. ASME* 64.11 (1942).

[35] Tanja Flemming. “Evaluating and Minimizing the Reality Gapin the Domain of RoboCup Humanoid Soccer”. In: *Bachelor Thesis at the University of Hamburg* (2019).

[36] Jasper Güldenstein. “Comparison of measurement systems for kinematic calibration of a humanoid robot”. In: *Bachelor Thesis at the University of Hamburg* (2019).

Appendices

A. Wolfgang Transformation Tree

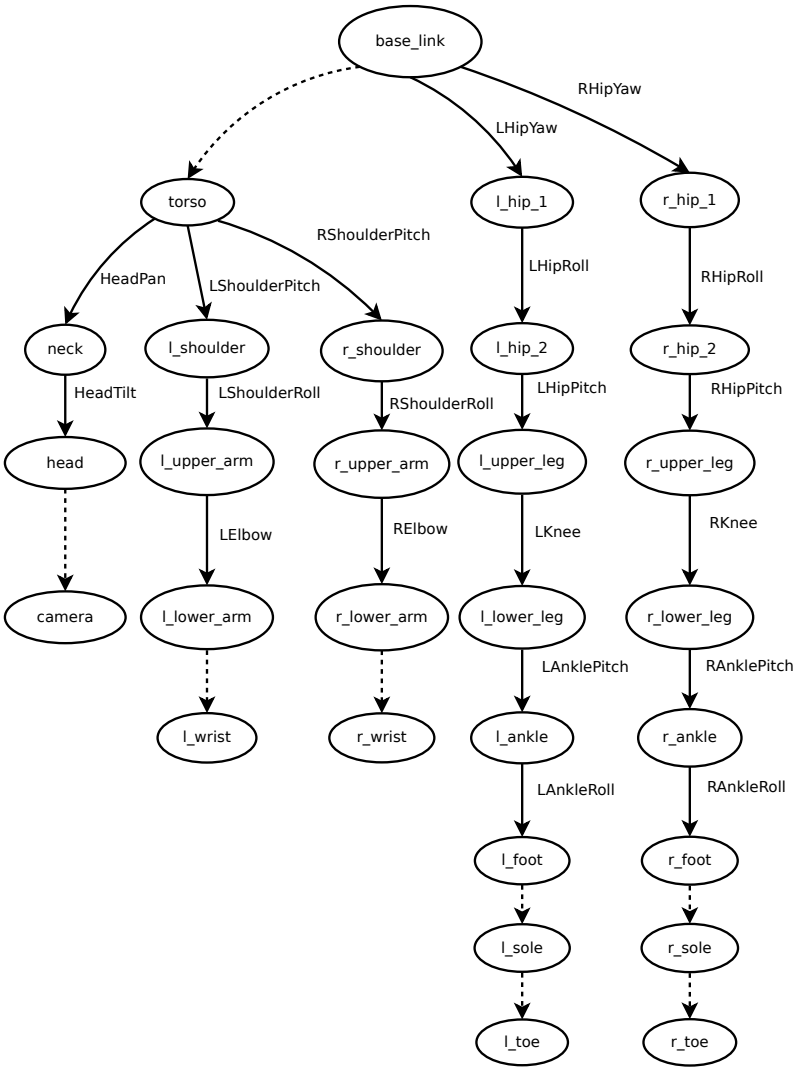


Figure A.1.: Kinematic chain of the robot platform. Each node represents a link. Each solid edge is a revolute joint, each dashed edge is a fixed joint, adapted from [36].

B. Parameters Used for the Dynup System

Parameter	Description	Value
engine_rate	How often the engine updates motor goals	200
foot_distance	How far apart the feet should be from each other [m]	0.2
trunk_height	Height of the trunk at the end [m]	0.38
trunk_pitch	Pitch of the trunk at the end [rad]	-0.18
leg_min_length	Minimum leg length [m]	0.21
stabilizing	Whether automatic stabilizing is used	True
time_legs_close	Back: Time to pull the legs to the body [s]	0.5
time_hands_down	Back: Time to put the wrists flat on the ground [s]	1.0
time_hands_back	Back: Time to push the hands back entirely [s]	1.0
time_squat_push	Back: Time to prepare pushing into the squat position [s]	1.0
time_full_squat	Back: Time to complete the push and completely stand in a squat [s]	0.2
time_hands_side	Front: Time to move the hands to the side when fallen to the front [s]	0.1
time_foot_close	Front: Time to pull the legs to the body [s]	0.5
time_hands_front	Front: Time to move the hands to the front [s]	0.1
time_foot_ground	Front: Time to put the feet on the ground [s]	0.5
time_torso_45	Front: Time to push the torso up to a 45 degree angle [s]	0.2
time_to_squat	Front: Time to push the robot into squat position [s]	1.5
wait_in_squat	Time to wait in squat position [s]	2.0
rise_time	Time to rise to walkready [s]	0.5
display_debug	Whether debug should be published	True
spline_smoothness	How many points the debug splines should have	100

Table B.1.: List of parameters used in the experiment for the dynup system, grouped by their dynamic reconfigure groups. Units described in brackets in the description, if applicable.

C. Gazebo Plugin for Randomized Forces

```
 1  #include <functional>
 2  #include <gazebo/gazebo.hh>
 3  #include <gazebo/physics/physics.hh>
 4  #include <gazebo/common/common.hh>
 5  #include <ignition/math/Vector3.hh>
 6  #include <ros/ros.h>
 7
 8  namespace gazebo {
 9    class ModelPush : public ModelPlugin {
10      public: void Load(physics::ModelPtr _parent, sdf::ElementPtr /*_sdf*/)
            {
11        // Store the pointer to the model
12        this->model = _parent;
13
14        // Listen to the update event. This event is broadcast every
15        // simulation iteration.
16        this->updateConnection = event::Events::ConnectWorldUpdateBegin(
17          std::bind(&ModelPush::OnUpdate, this));
18      }
19      // Called by the world update start event
20      public: void OnUpdate() {
21        // Apply a small linear velocity to the model.
22        double factor = 100.0;
23        double x = (((double) rand() / (RAND_MAX)) * 2 - 1) * factor;
24        double y = (((double) rand() / (RAND_MAX)) * 2 - 1) * factor;
25
26        this->model->GetLink("base_link")->AddForce(ignition::math::Vector3d
              (x, y, 0.0));
27      }
28
29      // Pointer to the model
30      private: physics::ModelPtr model;
31
32      // Pointer to the update event connection
33      private: event::ConnectionPtr updateConnection;
34    };
35
36    // Register this plugin with the simulator
37    GZ_REGISTER_MODEL_PLUGIN(ModelPush)
38  }
```

Listing C.1: Source code for the gazebo plugin that applied randomized forces to the robots base link in the x-y-plane.

D. Results of Effort Measurement

motor	baseline	front						back					
		Dynup			Keyframe			Dynup			Keyframe		
	avg	max	min	avg	max	min	avg	max	min	avg	max	min	avg
HeadPan	0.03	0.09	-0.39	0.00	0.13	-0.20	0.01	0.09	-0.04	0.02	0.11	-0.05	0.01
HeadTilt	-0.60	0.00	-0.45	-0.36	2.49	-2.39	0.38	0.00	-0.49	-0.36	3.10	-4.31	0.12
LAnklePitch	1.02	5.27	-5.01	-1.19	-4.11	-6.30	0.20	7.14	-5.69	-0.65	3.60	-3.69	0.10
LAnkleRoll	0.02	2.21	-0.71	0.07	0.87	-0.23	0.18	0.71	-0.79	-0.04	3.24	-0.25	0.37
LElbow	0.46	3.41	-1.15	-0.17	7.40	-6.69	-0.10	3.82	-10.63	-1.63	4.49	-8.00	-0.74
LHipPitch	-0.15	7.14	-5.01	0.16	8.48	-4.25	0.89	8.87	-3.14	1.68	9.39	-2.90	0.58
LHipRoll	-0.10	1.50	-0.89	0.07	1.53	-1.20	0.09	3.80	-1.57	-0.38	1.75	-0.09	0.38
LHipYaw	0.00	0.11	-4.00	-0.92	2.62	-2.49	0.98	1.47	-2.93	-0.14	0.24	-0.42	0.03
LKnee	0.54	10.81	-6.93	5.12	9.09	-3.74	1.85	10.79	-7.77	4.93	8.30	-3.10	2.62
LShoulderPitch	0.05	9.41	-0.73	1.41	8.99	-6.63	0.97	4.03	-5.09	0.39	9.94	-5.78	0.94
LShoulderRoll	0.00	6.00	-0.35	0.15	6.77	-8.17	0.12	1.33	-0.69	0.10	0.29	-2.04	-0.17
RAnklePitch	-0.80	6.63	-3.55	2.76	7.34	-2.05	-0.14	6.51	-1.97	1.44	3.77	-3.20	0.10
RAnkleRoll	-0.10	1.05	-2.12	-0.34	0.23	-0.89	-0.12	1.26	-0.91	-0.06	0.33	-0.50	-0.17
RElbow	-0.13	4.32	-4.28	0.60	8.03	-9.79	0.77	10.52	-4.55	1.63	8.22	-4.51	0.45
RHipPitch	0.18	4.67	-6.07	0.25	5.30	-8.82	-0.86	2.67	-9.86	-1.43	1.49	-6.56	-0.80
RHipRoll	0.04	2.69	-0.81	0.05	1.10	-0.66	-0.14	7.16	-1.13	0.17	0.21	-0.91	-0.19
RHipYaw	0.17	3.75	-0.04	0.60	2.03	-3.87	-0.25	2.52	-0.87	0.40	0.15	0.0	0.07
RKnee	-1.09	7.40	-10.95	-4.81	4.45	-9.70	-1.28	6.98	-12.08	-3.59	4.08	-7.72	-1.67
RShoulderPitch	-0.05	0.97	-9.47	-1.51	6.84	-12.05	-1.88	4.40	-3.74	-0.41	5.43	-9.45	-0.94
RShoulderRoll	-0.02	0.44	-9.56	-0.21	7.22	-9.12	0.00	0.57	-0.81	-0.08	0.97	-0.43	0.05

Table D.1.: Average, maximum and minimum effort measurements from different stand up procedures. Baseline represents the average effort applied by each motor in the walkready position without any additional forces applied. All units in Nm, rounded to two significant figures.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 25.06.2020 Sebastian Stelter

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 25.06.2020 Sebastian Stelter