Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Masterarbeit

Im Studiengang Physik

vorgelegt von

Martin Huber

geboren in Frankenthal

2019

Behavioral Cloning for Autonomous Navigation of Humanoid Robots with Nonlinear Model Predictive Control

Die Masterarbeit wurde von Martin Huber

ausgeführt am

Institut für Optimierung, Robotik und Biomechanik

unter der Betreuung von

Frau Prof. Katja Mombaur

Department of Physics and Astronomy

University of Heidelberg

Master thesis

in Physics

submitted by

(name and surname)

born in (place of birth)

(year of submission)

(Title)

(of)

(Master thesis)

This Master thesis has been carried out by (Name Surname)

at the

(institute)

under the supervision of

(Frau/Herrn Prof./Priv.-Doz. Name Surname)

Verhaltensklonung zur autonomen Navigation humanoider Roboter mit Nichlinearer Modellprädiktiver Regelung:

In dieser Arbeit erkunden wir die Möglichkeiten der Verhaltensklonung zur autonomen Navigation humanoider Roboter durch bloße Bilder. Hierfür wird eine nichtlineare, Modellprädiktive Regelung, die es ermöglicht, stabile Lauftrajektorien in Echtzeit zu erzeugen, implementiert und evaluiert. Es wird demonstriert, dass minimale Veränderung in der Bildverarbeitung genügen, um vielseitige Bewegungsstrategien in vielfältigen dynamischen und statischen Umgebungen zu erlernen. Diese Einfachheit der Lösung wird als passende Ergänzung zur Meidung von Konvexen Hindernissen identifiziert, welche durch Randbedingungen die Lösungen der nichtlinearen Modellprädiktiven Regelung einschränken. Alle Experimente werden an Heicub, einer Variente des iCub, durchgeführt, welcher speziell für Optimalsteuerung in der Fortbewegung am Istituto Italiano di Tecnologiia in Genove entwickelt wurde. Die Auswertung von Stabilitätskriterien zeigt weiterhin, dass ein menschlicher Kontrolleur, einem künstlichen Agenten gegenüber, nicht überlegen ist. Um die präsentierte Methode schließlich auf tauschende Aufgaben zu erweitern, vereinfachen wir die wechselnden Umgebungen auf ein gut gelöstes Klassifizierungsproblem.

Behavioral Cloning for Autonomous Navigation of Humanoid Robots with Nonlinear Model Predictive Control:

In this work we investigate the capabilities of behavioral cloning for autonomous navigation of humanoid robots from raw image input. Therefore, a nonlinear model predictive control that allows for real time generation of stable walking trajectories is implemented and evaluated. It is demonstrated that minor modifications in the vision pipeline are sufficient for the learning of versatile motion strategies in various dynamic and static environments. This simplicity is identified as a well suited addition to the avoidance of convex obstacles, which are represented by constraints to the solution of the implemented nonlinear model predictive control. All of the experiments are carried out on Heicub, a descendant of the iCub, which was especially designed for optimal control in locomotion at the Istituto Italiano di Tecnologia in Genova. The evaluation of stability criteria further reveals that there is no superiority of a human controller over an artificial agent. Finally, to extend the proposed approach to changing tasks, we boil the variation of environments down to a well solved classification problem.

Contents

1 Introduction

2 State of the Art

3 Background

To generate dynamically balanced walking trajectories for humanoid robots and to let them navigate the environment autonomously, there are several posed challenges that we need to cover. As the logical starting point, in section 3.1 - Humanoid Walking, we want to address the real time generation of walking trajectories for humanoid robots first, and then think of ways to replace the human user by an artificial agent in the control loop (fig. 3.1). The generation of patterns in real time becomes feasible by treating the robot's physics in a simplified way as those of an inverted pendulum (sec. 3.1.1). The zero moment point of the linear inverted pendulum will therefore serve as the balance criteria for the solution of a sequentially quadratic problem (sec. 3.1.2). Resulting positions and orientations for the center of mass and the feet will then be interpolated (sec. 3.1.3) and passed as constraints to the inverse kinematics (sec. 3.1.4) so to transform them into joint angles that can be sent to the humanoid's motor controllers.

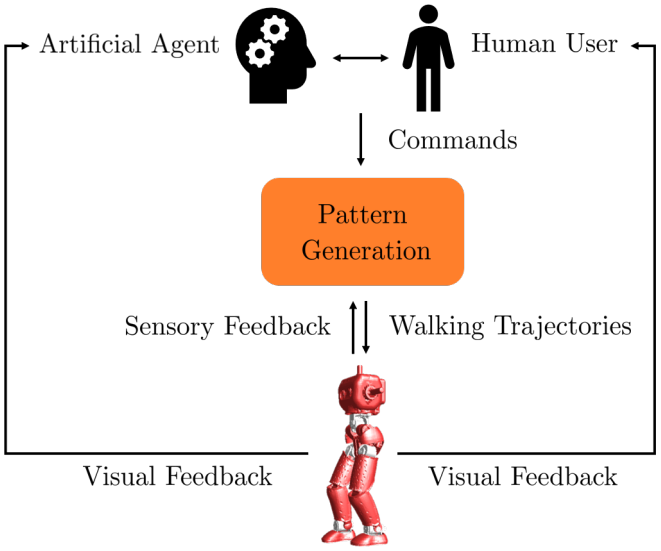


Figure 3.1: Simplified version of the proposed control loop to navigate the robot with either a human user or an artificial agent.

To close the control loop and to steer the robot towards desired goals, whilst avoiding obstacles, requires some sort of high level command that arises from visual feedback. As discussed in section 2 - State of The Art, there are several ways to achieve this, among them human users. Of particular interest to us are novel methods that evolved from the toolbox of machine learning techniques, as they decrease the computational cost into non existence. Let alone this fact enables us to run

them onboard on light weight hardware with low energy usage, which is critical in the domain of humanoid robots. Center to these new methods will be neural nets that we will train on solving the task of autonomous navigation in two different ways. One of which clones the behavior of a human user (sec. 3.2.2), whereas the second presented method (sec. 3.2.3) explores policies and tries to find solutions on its own.

As a side note, within the following chapters there will always be made references to the actual implementation of the presented concepts. This shall enable future readers to bridge the gap between theory and application.

3.1 Humanoid Walking

To get started with and to understand the presented concepts that generate dynamically balanced walking trajectories, we shall have a look at figure 3.1 once more. The pattern generation therein (orange box), consists of four main building blocks: Forward kinematics, nonlinear model predictive control (NMPC), interpolation, and inverse kinematics. The relation between these four building blocks is shown in fig. 3.2. The natural entry point, to this otherwise closed control loop, is given by the

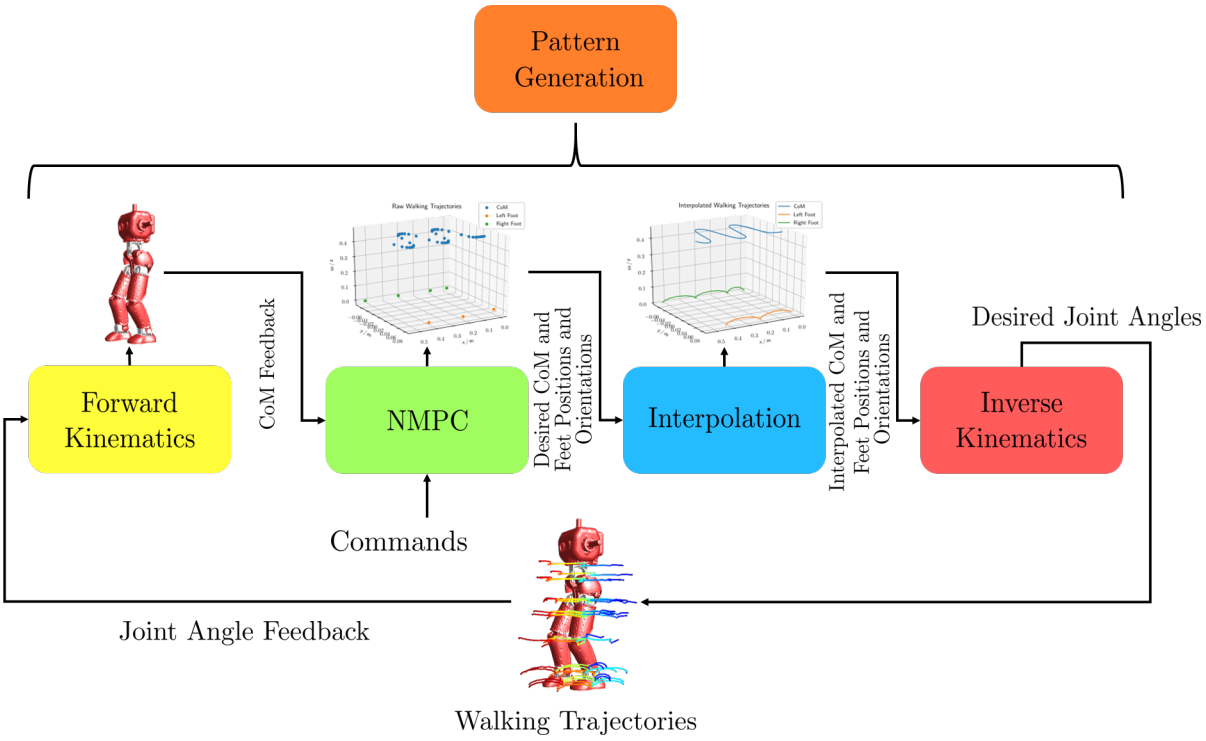


Figure 3.2: Building blocks of the pattern generation. To understand the greater picture, a connection can be drawn to fig. 3.1, where the orange box represents the one shown in this figure.

commands that enter the nonlinear model predictive control. Commands are passed

in the form of a desired velocity $\mathbf{v}_{\text{ref}}$ that the robot's center of mass (CoM) shall satisfy optimally according to a cost function that also takes dynamic balance and a smooth motion into account. The future desired positions and orientations for the CoM and the feet then result from the solution to a sequentially quadratic problem that tries to minimize this cost function. The balance criteria within this problem formulation is based upon the zero moment point (ZMP) around which the whole control framework is built. It is only by simplifying the robot's model that we can solve the optimal control problem in real time. Therefore, we assume the robot to be a linear inverted pendulum, for which we have a well defined analytical relation between the CoM and the ZMP. The minimization of the distance between the analytical expression of the ZMP and the foot placement results in the desired dynamic balance. As shown in fig. 3.2, the desired CoM and the feet positions and orientation, as they are obtained from the NMPC, are sparsely distributed in space. Moreover, there is neither information about how the feet shall move along the z-axis, nor along the x-, and y-axis, but only where they should be placed in the x-y-plane. Therefore, as the subsequent step to the NMPC, we need to add an interpolation. The interpolation interpolates the trajectories of the CoM to obtain a finer sampling time. Additionally, the movement of the feet in the x-, y-, and z-direction, as well as their orientation, is computed by polynomials that we require to satisfy the initial and end conditions of the foot placement. Put together, the nonlinear model predictive control and the interpolation between the resulting subsequent solutions for the positions and orientation of both, the CoM and the feet, describe dynamically balanced trajectories, given that the humanoid robot of interest resembles the physics of an inverted pendulum. Now to bridge the gap between dynamically balanced trajectories in Cartesian space, and a humanoid robot that actually satisfies them with its CoM and its feet, the inverse kinematics problem needs to be addressed. The inverse kinematics, which follow immediately after the interpolation step, take the positions and orientations of the CoM and the feet as constraints and find a composite of joint angles that fulfill them. The continuity of subsequent solutions is therein assured by initializing the inverse kinematics with the previous solution. Resulting joint angles, once passed to the humanoid, then result in walking trajectories, as indicated in fig. 3.2 by the colored lines at the joints of the robot. Due to the inherent mismatch of the robot's physics from that of an inverted pendulum, as well as other effect like friction, there is a chance that the desired joint angles differ from the actually achieved ones. To compensate for the discrepancy, the last building block of the pattern generation is the feedback of the measured CoM to the NMPC. The CoM is computed by reading out the achieved joint angles, so that the forward kinematics can be utilized to determine the positions and orientations of the humanoid's links in space, and therefore the CoM.

As already highlighted in the previous paragraph, special attention has to be given to the zero moment point, since it defines the central concept of the presented pattern generation. We therefore will explain its theoretical foundations, as well as

its analytical relationship to the CoM for simplified physical models, and ways to measure it with force torque sensors in the section that lies ahead - Zero Moment Point.

3.1.1 Zero Moment Point

The key metric in this work, for the generation of a dynamically balanced gait, is the zero moment point. The concept was first introduced by Miomir Vukobratović and Davor Juričić in 1968 [1][2] and first utilized 1984 to generate walking trajectories for the WL-10RD robot [3]. The most intuitive understanding for the ZMP arises by thinking about the realization of the simplest arbitrary possible walking motion for which a humanoid robot will not fall. This motion is achieved by ensuring the feet's whole area, and not only the edge, is in contact with the ground [4], or put in other words, we require the robot not to rotate about its feet edges. This constraint can be met by having a reaction force $\boldsymbol{F}_r$ between the foot and the ground, which compensates for all external moments $\boldsymbol{M}_x$, and $\boldsymbol{M}_y$ around the x-, and y-axis at any time (fig. 3.3). The point $\boldsymbol{r}$, at which the reaction force acts, is physically

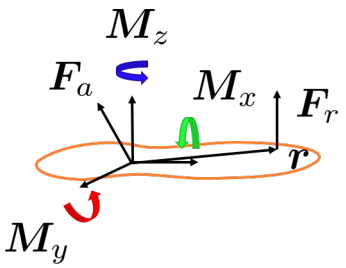


Figure 3.3: Forces acting on the sole.

only meaningful if it lies within the support polygon of the foot. Not only can it not exist outside of the support polygon, since there was no point of interaction between the foot and the ground then, but also was the robot to overturn under these circumstances. Therefore, the ZMP is defined as that point on the ground at which the net moment of the inertial forces has no component along the horizontal axes [5][6]. We now came to appreciate the importance of the support polygon for the definition of the zero moment point. The support polygon is defined as the

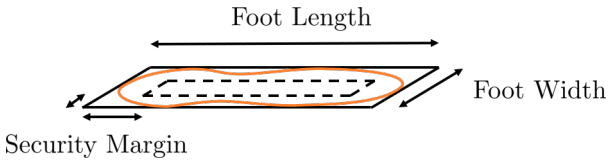


Figure 3.4: Full support polygon, and the resulting support polygon with security margin (dashed lines).

convex hull of all contact points of the feet with the ground, so the minimal number

of points to fully contain all of them. As the most restrictive case for balance, in this work we will only consider the support polygon of one foot at a time. Since the convex hull of a foot is well described by a rectangle, we only rely on the foot width (link), and foot length (link) to fully describe it. Also, to ensure that the zero moment point never comes close to the edges of the feet and therefore to provide balance, we define a security margin to their boarders (link). The respective values are robot specific and can be set in the configurations file by following the provided links.

As already pointed out, within this work, we will use a simplified physical model of the humanoid solve the optimal control problem in real time. We will deal with this approximation in the following paragraph - Zero Moment Point of a Linear Inverted Pendulum.

Zero Moment Point of a Linear Inverted Pendulum

Dynamically balanced walking trajectories can be generated by simplifying the dynamics of humanoid robots to those of a linear inverted pendulum [7]. A rigorous derivation for the analytic relation between the center of mass and the zero moment point of a linear inverted pendulum can be found in [8], but for the sake of simplicity we rather explain the physics in terms of cutting forces, for which a short introduction can be found in the summary of the lecture Robotics 1 (link). The

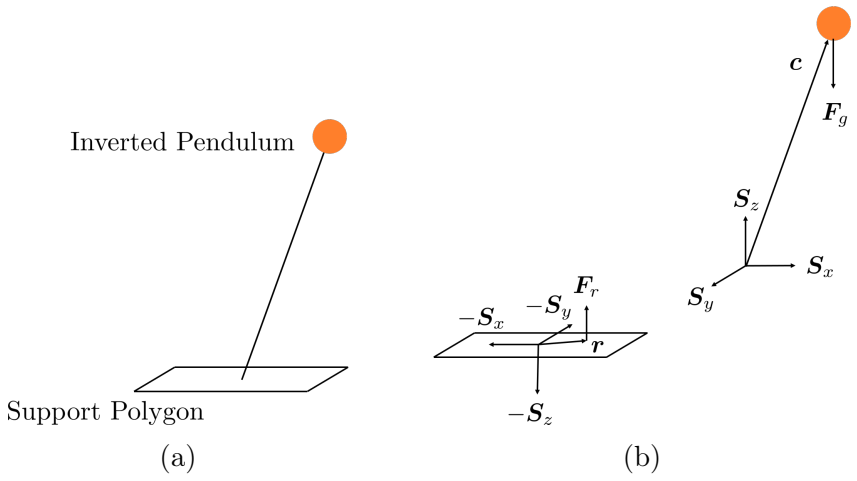


Figure 3.5: Linear inverted pendulum with a support polygon (a), and the corresponding free body diagram with cutting forces $\boldsymbol{S}_{x/y/z}$ (b).

system of interest is shorty depicted in figure 3.5. We assume the support polygon of the shown linear inverted pendulum to have zero mass. By introducing cutting forces $\boldsymbol{S}_{x/y/z}$ for each degree of freedom in which the motion of the linear inverted pendulum is restricted, we obtain the free body diagram (fig. 3.5), for which the

acting forces are

$$m\ddot{\boldsymbol{c}} = \quad \boldsymbol{S} - \boldsymbol{F}_g \tag{3.1}$$
$$\mathbf{0} = -\boldsymbol{S} + \boldsymbol{F}_r \tag{3.2}$$

where $\boldsymbol{S} = \boldsymbol{S}_x + \boldsymbol{S}_y + \boldsymbol{S}_z$. The respective moments, since we do not take any inertias into account, are given by

$$\mathbf{0} = (\mathbf{0} - \boldsymbol{c}) \times \boldsymbol{S} + \boldsymbol{M} \tag{3.3}$$
$$\mathbf{0} = (\boldsymbol{r} - \mathbf{0}) \times \boldsymbol{F}_r - \boldsymbol{M} \tag{3.4}$$

where the transfer of the moment $\boldsymbol{M}$ may for example be induced by friction. If we replace $\boldsymbol{S} = \boldsymbol{F}_r$ from eq. 3.2, equations 3.3 and 3.4 yield

$$\mathbf{0} = (\boldsymbol{r} - \boldsymbol{c}) \times \boldsymbol{S} = \begin{pmatrix} (r_y - c_y)S_z - (r_z - c_z)S_y \\ -(r_x - c_x)S_z + (r_z - c_z)S_x \\ (r_x - c_x)S_y - (r_y - c_y)S_x \end{pmatrix} \tag{3.5}$$

Since our goal is to have a robot that does not fall, we want to achieve that the acceleration along the z-axis becomes zero, hence $\ddot{c}_z = 0$. Given this assumption, we can infer from eq. 3.1 that $S_z = mg$, as well as $S_x = \ddot{c}_x m$, and $S_y = \ddot{c}_y m$. Furthermore, our foot shall not lift off the floor, and therefore we have $r_z = 0$. If we take these assumptions and plug them into the first to rows of eq. 3.5, we find

$$r_x = c_x - c_z \frac{\ddot{c}_x}{g} \tag{3.6}$$
$$r_y = c_y - c_z \frac{\ddot{c}_y}{g} \tag{3.7}$$

Therein, r_x, and r_y are the x-, and y-coordinates of the zero moment point, given the assumption of a linear inverted pendulum. We can see that the position if dependent on the height of the point mass, which is in turn dependent on the robot. The specific values can be set in the configurations file (link).

We have now found a simple analytic expression for the relationship of the zero moment point and the center of mass, which will help us to formulate an optimal control problem that we can solve in real time. This simplification is of course only true to some extend, and we need to find a way to verify its accuracy. The easiest way to do so is to measure the real zero moment point. We will further elaborate on this within the next paragraph - Measurement of the Zero Moment Point, and we will derive a method that only relies on force torque sensors in the ankle.

Measurement of the Zero Moment Point

There are several methods that enable us to measure the position of the zero moment point, among them the utilization of pressure sensitive soles, as outlined in [8].

Furthermore, there exist approximate approaches that involve the knowledge of all acting external forces [9], which can for example be obtained from unconstrained inverse dynamics [10]. Since we can rely on measurements of force torque sensors that are located at the ankles, we will infer the position of the zero moment point from them [8]. If we consider the force torque sensor to be located at position $\boldsymbol{p}_i$ (TODO fig. **??**), then we can obtain the moment about any point $\boldsymbol{p}$ according to eq. 3.8.

$$\boldsymbol{\tau}(\boldsymbol{p}) = (\boldsymbol{p}_i - \boldsymbol{p}) \times \boldsymbol{f}_i + \boldsymbol{\tau}_i \tag{3.8}$$

by definition, the moment about the zero moment point vanishes along the horizontal axes, therefore we can then set $\tau_x = \tau_y = 0$ in eq. 3.8 and then solve for the position to obtain the zero moment point (eq. 3.9 and 3.10).

$$p_x = \frac{[-\tau_{i,y} - (p_{i,z} - p_z) f_{i,x} + p_{i,x} f_{i,z}]}{f_{i,z}} \tag{3.9}$$

$$p_y = \frac{[-\tau_{i,x} - (p_{i,z} - p_z) f_{i,y} + p_{i,y} f_{i,z}]}{f_{i,z}} \tag{3.10}$$

If we further choose our coordinate system to lie along the z-axis of the force torque sensor (fig. ...), we can simplify equations 3.9 and 3.10 to find

$$p_x = \frac{(-\tau_{i,y} - f_{1,x} d)}{f_{1,z}} \tag{3.11}$$

$$p_y = \frac{(\tau_{i,x} - f_{1,y} d)}{f_{1,z}} \tag{3.12}$$

We can use equations 3.11 and 3.12 to determine the position of the zero moment point for the left and the right foot with respect to coordinates frames that are attached to the respective foot. These circumstances change once not only one, but both feet are in contact with the ground. What still holds true, in the case of a dynamically balanced gait, is the fact that the positions which we just obtained from equations 3.11 and 3.12 represent points where the interaction of the robot with the environment can solely be described by a single force along the z-axis. All other forces or torques cancel out. Therefore, to determine the position of the zero moment point for the double support phase, we need to modify equation 3.8 slightly. This yields

$$\boldsymbol{\tau}(\boldsymbol{p}) = \sum_{i \in \{L,R\}} (\boldsymbol{p}_i - \boldsymbol{p}) \times \boldsymbol{f}_i \tag{3.13}$$

where the individual torques are now zero and the only forces $\boldsymbol{f}_i$ that exist between the robot and the environment can be described by the z-component which are measured at the ankles' force torque sensors. Yet again, to obtain the position of

the zero moment point, we have to set the x-, and y-components of the torque in equation 3.13 to zero and find

$$p_x = \frac{\sum_{i\in\{L,R\}} p_{i,x} f_{i,z}}{\sum_{i\in\{L,R\}} f_{i,z}} \tag{3.14}$$

$$p_y = \frac{\sum_{i\in\{L,R\}} p_{i,y} f_{i,z}}{\sum_{i\in\{L,R\}} f_{i,z}} \tag{3.15}$$

These expressions of course only hold true in a shared coordinate system and therefore we need to transform the position of the zero moment point which we obtained from equations 3.11 and 3.12 to the world frame. Finally, we can write down the formulation for the zero moment point which holds equally true for the single and double support phase

$$p_x = \frac{p_{R,x} f_{R,z} + p_{L,x} f_{L,z}}{f_{R,z} + f_{L,z}} \tag{3.16}$$

$$p_y = \frac{p_{R,y} f_{R,z} + p_{L,y} f_{L,z}}{f_{R,z} + f_{L,z}} \tag{3.17}$$

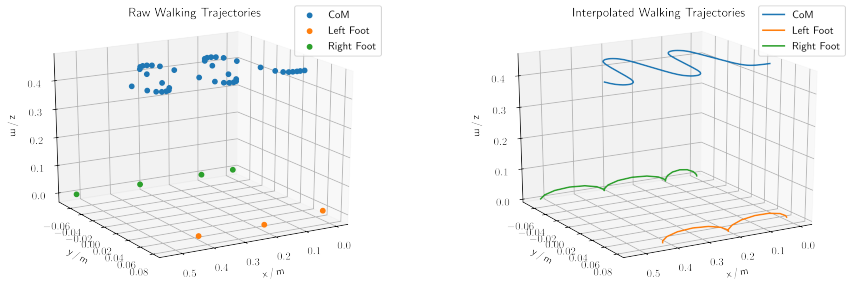
At this point we are now equipped with a general understanding for the zero moment point, as well as with the knowledge of simplified models to compute it analytically, and a method to measure it so that we can evaluate the performance of a potential pattern generator which is based upon the zero moment point. Therefore, in the next chapter - Nonlinear Model Predictive Control, we will try to understand a method that allows us to generate dynamically balanced center of mass and feet trajectories, which satisfy the just introduced concepts optimally, given a weighting factor.

3.1.2 Nonlinear Model Predictive Control

At the heart of nonlinear model predictive control stands sequential quadratic programming. Before we come to the actual problem formulation, we need to understand how sequential quadratic programming can be used to solve nonlinear optimization problems. We will then come to recognize that if we can find a canonical formulation of our problem, it will become possible to apply sequential quadratic programming to it. The next paragraph - Sequential Quadratic Programming, will therefore shortly introduce the reader to the desired method that will be used to solve the nonlinear optimization problem, while the subsequent paragraph - Canonical Formulation of Nonlinear Model Predictive Control, will then explain how to fit humanoid walking into this framework.

3.1.3 Interpolating Trajectories

As already shortly depicted in figure 3.2, we need to interpolate the trajectories that we obtain from the nonlinear model predictive control. This especially holds true for the feet, since the computed results do only consider the pendulums dynamic balance with respect to the x-, and the y-position, but not with respect to a robot that has to lift its feet along the z-axis. Furthermore, the feet's movement shall be executed such they stop moving when they are about to touch the ground. This constraint, and others, will be achieved by interpolating the feet trajectories with polynomials. In order to then match the center of mass trajectory's temporal resolution with the feet trajectories', we will upscale it under the already well known assumption of a linear inverted pendulum. The resulting trajectories are shown in figure 3.6, and will further be explained in the following.



(a) (b)

Figure 3.6: Uninterpolated trajectories (a), as obtained from the nonlinear model predictive control, and interpolated trajectories (b) for the feet and the center of mass.

Interpolating the Feet Trajectories

Any trajectory can in principal be approximated by a polynomial function. For our purposes, we want to approximate positions p as they evolve over time t, and further

obtain the corresponding velocities $\dot{p}$ and accelerations $\ddot{p}$ (equations 3.18 - 3.20).

$$p(t) = \sum_{i=0}^{N} a_i t^i \tag{3.18}$$

$$\dot{p}(t) = \sum_{i=1}^{N} i a_i t^{(i-1)} \tag{3.19}$$

$$\ddot{p}(t) = \sum_{i=2}^{N} i(i-1) a_i t^{(i-2)} \tag{3.20}$$

The coefficients a_i of the polynomials can be chosen such that certain boundary conditions $\boldsymbol{b}$ are satisfied. For the lift-off and the drop-down of the robot's feet, these boundary conditions must satisfy a zero initial velocity $\dot{z}_{\text{init}}$ and a zero end velocity $\dot{z}_{\text{end}}$, as well as a zero initial height z_{init} and a zero end height z_{end}, and a maximum step height $z_{T/2}$ in between, or else they will hit the ground in an unbalanced way. These conditions are listed below, where each height $z(t)$ and each velocity $\dot{z}(t)$ is written in terms of a polynomial, just as in equations 3.18 and 3.19, respectively.

$$z(t=0) = z_{\text{init}} = 0 \tag{3.21}$$

$$\dot{z}(t=0) = \dot{z}_{\text{init}} = 0 \tag{3.22}$$

$$z(t=\frac{T}{2}) = z_{T/2} \tag{3.23}$$

$$z(t=T) = z_{\text{end}} = 0 \tag{3.24}$$

$$\dot{z}(t=T) = \dot{z}_{\text{end}} = 0 \tag{3.25}$$

To satisfy 5 boundary conditions, it is required to have a polynomial of 4th order with 5 coefficients $a_{z,i}$ in total. In matrix formulation we can express equations 3.21 - 3.25 as follows

$$\boldsymbol{M}_z \boldsymbol{a}_z = \boldsymbol{b}_z \tag{3.26}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & \left(\frac{T}{2}\right) & \left(\frac{T}{2}\right)^2 & \left(\frac{T}{2}\right)^3 & \left(\frac{T}{2}\right)^4 \\ 1 & T & T^2 & T^3 & T^4 \\ 0 & 1 & 2T & 3T^2 & 4T^3 \end{pmatrix} \begin{pmatrix} a_{z,0} \\ a_{z,1} \\ a_{z,2} \\ a_{z,3} \\ a_{z,4} \end{pmatrix} = \begin{pmatrix} z_{\text{init}} \\ \dot{z}_{\text{init}} \\ z_{T/2} \\ z_{\text{end}} \\ \dot{z}_{\text{end}} \end{pmatrix}. \tag{3.27}$$

Inversion then yields

$$\boldsymbol{a}_z = \boldsymbol{M}_z^{-1} \boldsymbol{b}_z \tag{3.28}$$

$$\begin{pmatrix} a_{z,0} \\ a_{z,1} \\ a_{z,2} \\ a_{z,3} \\ a_{z,4} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -\frac{11}{T^2} & -\frac{4}{T} & \frac{16}{T^2} & -\frac{5}{T^2} & \frac{1}{T} \\ \frac{18}{T^3} & \frac{5}{T^2} & -\frac{32}{T^3} & \frac{14}{T^3} & -\frac{3}{T^2} \\ -\frac{8}{T^4} & -\frac{2}{T^3} & \frac{16}{T^4} & -\frac{8}{T^4} & \frac{2}{T^3} \end{pmatrix} \begin{pmatrix} z_{\text{init}} \\ \dot{z}_{\text{init}} \\ z_{T/2} \\ z_{\text{end}} \\ \dot{z}_{\text{end}} \end{pmatrix}. \tag{3.29}$$

The obtained coefficients a_i are then used to compute the height of each foot during a single support phase (link). The maximum step height $z_{T/2}$ (link), and the single support time T (link), which is the step time minus the double support time, can be set in the configurations file. For the x-, and the y-positions of the feet, we can define boundary conditions in a similar fashion. In contrary to the computation of the z-position, the x-, and the y-position interpolation of the feet allows for feedback. Therefore, we require additional constraints that satisfy the accelerations as follows

$$x(t=0) = x_{\text{init}} \tag{3.30}$$
$$\dot{x}(t=0) = \dot{x}_{\text{init}} \tag{3.31}$$
$$\ddot{x}(t=0) = \ddot{x}_{\text{init}} \tag{3.32}$$
$$x(t=T) = x_{\text{end}} \tag{3.33}$$
$$\dot{x}(t=T) = \dot{x}_{\text{end}} \tag{3.34}$$
$$\ddot{x}(t=T) = \ddot{x}_{\text{end}}. \tag{3.35}$$

Again, we can rewrite equations 3.30 - 3.35 in matrix formulation

$$\boldsymbol{M}_x \boldsymbol{a}_x = \boldsymbol{b}_x \tag{3.36}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & T & T^2 & T^3 & T^4 & T^5 \\ 0 & 1 & 2T & 3T^2 & 4T^3 & 5T^4 \\ 0 & 0 & 2 & 6T & 12T^2 & 20T^3 \end{pmatrix} \begin{pmatrix} a_{x,0} \\ a_{x,1} \\ a_{x,2} \\ a_{x,3} \\ a_{x,4} \\ a_{x,5} \end{pmatrix} = \begin{pmatrix} x_{\text{init}} \\ \dot{x}_{\text{init}} \\ \ddot{x}_{\text{init}} \\ x_{\text{end}} \\ \dot{x}_{\text{end}} \\ \ddot{x}_{\text{end}} \end{pmatrix}, \tag{3.37}$$

and inversion yields the polynomial's coefficients $a_{x,i}$

$$\boldsymbol{a}_x = \boldsymbol{M}_x^{-1} \boldsymbol{b}_x \tag{3.38}$$

$$\begin{pmatrix} a_{x,0} \\ a_{x,1} \\ a_{x,2} \\ a_{x,3} \\ a_{x,4} \\ a_{x,5} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -\frac{20}{T^3} & -\frac{12}{T^2} & -\frac{3}{T} & \frac{20}{T^3} & -\frac{8}{T^2} & \frac{1}{T} \\ \frac{30}{T^4} & \frac{16}{T^3} & \frac{3}{T^2} & -\frac{30}{T^4} & \frac{14}{T^3} & -\frac{2}{T^2} \\ -\frac{12}{T^5} & -\frac{6}{T^4} & -\frac{1}{T^3} & \frac{12}{T^5} & -\frac{6}{T^4} & \frac{1}{T^3} \end{pmatrix} \begin{pmatrix} x_{\text{init}} \\ \dot{x}_{\text{init}} \\ \ddot{x}_{\text{init}} \\ x_{\text{end}} \\ \dot{x}_{\text{end}} \\ \ddot{x}_{\text{end}} \end{pmatrix}. \tag{3.39}$$

The exact same formalism is used to interpolate the foot's y-position during single support phase (link). In contrast to the interpolation of the feet's positions, the center of mass positions will be extrapolated under the introduced assumption of a linear inverted pendulum. The method will be shortly explained in the following paragraph - Interpolating the Center of Mass Trajectories.

Interpolating the Center of Mass Trajectories

TODO Link linear time stepping scheme from nmpc

3.1.4 Kinematics

Forward Kinematics

Inverse Kinematics

3.2 Machine Learning

Machine learning methods do play a major role for autonomous navigation of robots, and whilst most recent approaches mainly dealt with tree search methods in 3D point-clouds, we aim at utilizing neural networks for solving the task at hand, since it enables us to combine spatial, semantic, and temporal understanding into one approach. Within this chapter, we will therefore explain the required fundamentals on neural networks in section 3.2.1, then cover two possible methods for training a neural network, one of which is supervised 3.2.2, whereas the second method is based on reinforcement learning 3.2.3, and finally explain image processing techniques in section 3.2.4, which allow us to extract depth maps from stereo images, so to help the neural networks understand the seen content. The goal here clearly is to introduce a method that is biologically inspired, in that it works directly in the image domain, which is very similar to how humans observe their environment. Therefore, we will shortly explain the biological similarities to a human brain within the next section - Neural Networks.

3.2.1 Neural Networks

Neural networks are inspired by the brain's structure, and as we will see later in this chapter, their building blocks can be thought of as neurons (figure 3.7). And

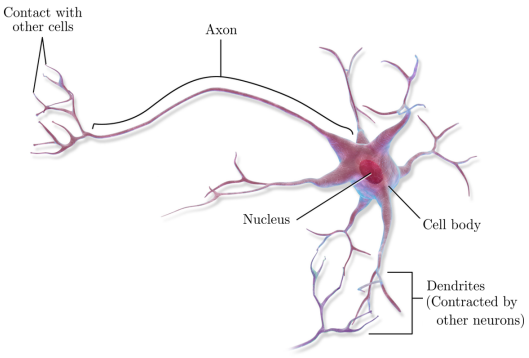


Figure 3.7: Biological neuron, which connects its cell body to dendrites of surrounding neurons via an axon. [11]

although neural networks have gained early attention in research, they have only recently become powerful for their implementation on graphic processing units (GPUs) [12]. This was caused by their mathematical description that is linear and can be

parallalized very well. Running neural nets on GPUs on the other hand consumes a lot of energy, which stays in contrast to biological neurons, which communicate by brief energy efficient spikes [13]. And while there are around 100 billion neurons in the human brain, currently huge neural networks have around a factor of 10000 less [14]. Not only is the number of neurons in a neural net comparably small, but also is their complexity way below that of a biological neuron. To tackle this discrepancy, and to learn complicated tasks, it is therefore required to introduce activation functions for neural networks, such as the rectifying linear unit [15]. This enables neural networks to be used on a variety of problems, but they currently lack in transferring knowledge between different domains, and tend to over-fit certain tasks. There exist methods to deal with this tendency, such as max-pooling [16] or dropout [17] layers, which efficiently just reduce the number of neurons within a neural net. To build a good understanding of neural networks, we will introduce different architectures in the following that will be used throughout this thesis, and we will start with the simplest in the next paragraph - Fully Connected Neural Network.

Fully Connected Neural Network

The biologically inspired perceptron model [18] lay the foundation for neural networks and it got extended pretty soon to the multi-layer perceptron model [19], which is known today as the fully connected neural network. Due to its similarity to the brain's structure, it is often depicted as in figure 3.8, where each orange circle represents a neuron that is connected to its surroundings via simple weights w_{ij}, as neurons inside the brain are by synapses. Mathematically speaking, the feed

Figure 3.8: Fully connected neural network with three inputs and four outputs. Each orange circle represents what is often referred to as neuron, while the black lines indicate the connections between each neuron.

forward process can be described as a simple matrix multiplication with all weights $\boldsymbol{W}$, where the input $\boldsymbol{x}$ gets converted to the output $\boldsymbol{y}$ via

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{3.40}$$

Therein, the bias $\boldsymbol{b}$ can be understood as a shift of isolines that are introduced by hyperplanes. These hyperplanes are learned and expressed by the layer's weights w_{ij} (figure 3.9). The output $\boldsymbol{y}$, is then further passed through an activation function f, and therefore can be compared to the action potential inside a neuron, as it

determines the amount by which the next neuron gets excited. This activation function can be anything from a simple step function for classification to a linear function for regression. In practice there are some activation functions that have shown to be of particular use and we will introduce them later in this chapter.

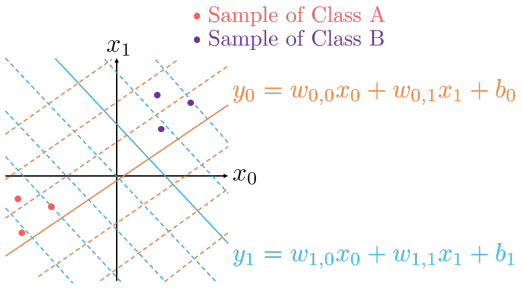


Figure 3.9: Simple interpretation of a fully connected neural network with one layer that takes $\boldsymbol{x} = (x_0\ x_1)^T$ as input. The dotted lines are isolines to the hyperplane, which showcase the effect of the bias $\boldsymbol{b}$.

Convolutional Neural Network

The concept of convolutional neural networks was first inspired by biological structures inside the visual cortex of the human brain. It was introduced as neocognitron [20], and soon after termed convolutional neural network for its mathematical properties, in that it equals convolutions. Figure 3.10 shows how an input $\boldsymbol{x}$ is fed forward through the network architecture across two layers. The operation is again

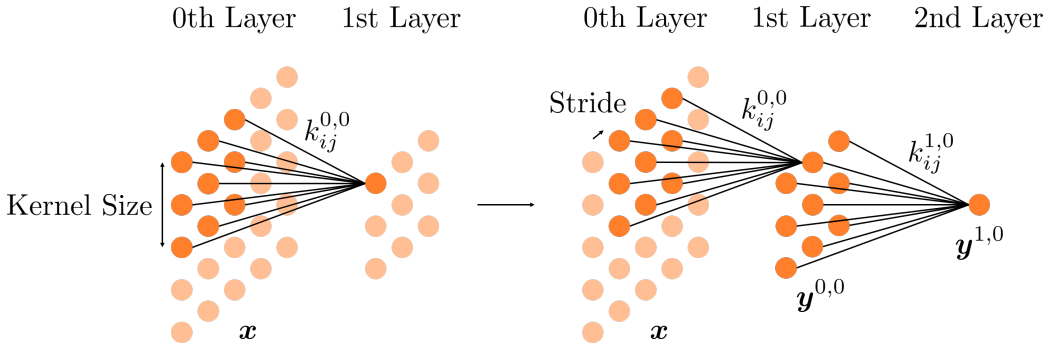


Figure 3.10: Convolutional neural network with a total of three layers of which one is the input layer. For visualization, the kernel size is set to be three, and the stride is set to be one.

visualized by utilizing orange circles, which may be referred to as neurons. These neurons are connected by weights $k_{ij}^{l,n}$, which together constitute the kernel $\boldsymbol{k}^{l,n}$ of each convolution. Therein, l stands for the current layer, and n indexes the kernel within a layer, as there may in principle be many different kernels for a single layer.

Mathematically speaking, we can formulate the process as follows

$$\boldsymbol{y}^{0,0} = f(\boldsymbol{x} * \boldsymbol{k}^{0,0}) \tag{3.41}$$
$$\boldsymbol{y}^{1,0} = f(\boldsymbol{y}^{0,0} * \boldsymbol{k}^{1,0}) \tag{3.42}$$

One thing to notice is that the deeper we go, meaning the more layers we have, the more of the initial input contributes to the current activation. This can be seen in figure 3.10, where each neuron within the first layer only sees a kernel size sized snipped of the original input $\boldsymbol{x}$, whereas a neuron within the second layer already sees all of it. This intuitive understanding of convolutional neural networks is backed by visualizations of the highest activity neuron's gradient with respect to the input, where the gradient itself equals the transposed convolution [21]. Figure 3.11 shows transposed convolutions for a classification network that was trained on ImageNet [22]. It can be seen that while superficial layers learn to understand edges, deeper layers grab more complex spatial correlations within images, such as car wheels within the third layer. Therefore, convolutional neural networks are particularly

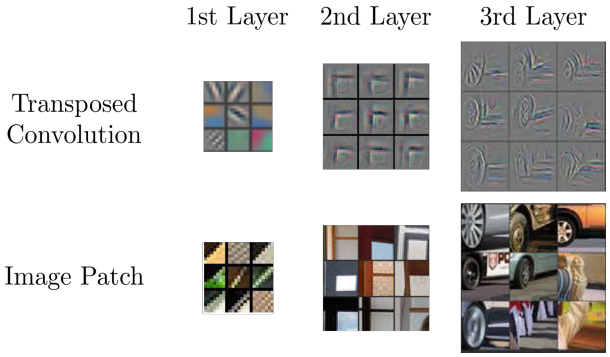


Figure 3.11: Analysis of neurons with highest activation, corresponding to a subset of ImageNet. For the first layer, the kernel itself is shown, and image patches at the kernel's scale. For the second and third layer, a single neuron with highest activation is upscaled by transposed convolutions to the first layer's feature map. Images taken from [23].

well suited for understanding spatial correlations, and while recent advancements also propose the promising use of convoltional neural networks for time series analysis [24], a simpler approach are long short-term memory units, which will be explained in the next section - Long Short-Term Memory.

Long Short-Term Memory

Long short-term memory units were introduced to overcome the vanishing and exploding gradient problem for recurrent neural networks in time series analysis [25]. That is the gradient tends to diverge exponentially over the course of the backward pass towards earlier inputs, resulting in intractable updates for the weights of the neural network. This issue got solved by adding a constant recurrent self-connection

that allows for constant error propagation through the network, which got further refined by replacing the constant self-connection with a forget gate [26]. The inner workings of a long short-term memory unit is shown in figure 3.12. The underlying

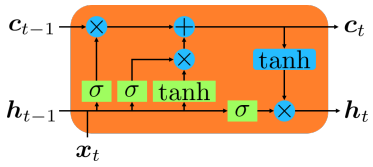


Figure 3.12: Long short-term memory unit with cell states $\boldsymbol{c}_i$, hidden states $\boldsymbol{h}_i$, input $\boldsymbol{x}_t$, and activation functions $\sigma/\tanh$, as well as addition $+$ and multiplication $\times$ operators.

mathematical operations can therein be expressed as follows

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{ii}\boldsymbol{x}_t + \boldsymbol{b}_{ii} + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hi}) \tag{3.43}$$
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{if}\boldsymbol{x}_t + \boldsymbol{b}_{if} + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hf}) \tag{3.44}$$
$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}_{ig}\boldsymbol{x}_t + \boldsymbol{b}_{ig} + \boldsymbol{W}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hg}) \tag{3.45}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{io}\boldsymbol{x}_t + \boldsymbol{b}_{io} + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{ho}) \tag{3.46}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \cdot c_{t-1} + \boldsymbol{i}_t \cdot \boldsymbol{g}_t \tag{3.47}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \cdot \tanh(\boldsymbol{c}_t), \tag{3.48}$$

where $\cdot$ is an element-wise multiplication. The sigmoid function σ, which ranges from 0 to 1, assures that the input gate $\boldsymbol{i}_t$, the forget gate $\boldsymbol{f}_t$, and the output gate $\boldsymbol{o}_t$ let only pass values of interest into, and out of the cell. Multiple long short-term memory units can then be linked for time series analysis as shown in figure 3.13.

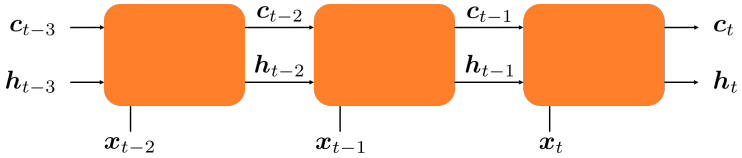


Figure 3.13: Chain of long short-term memory units for temporal understanding of the input sequence $\boldsymbol{x}_i$.

Backpropagation

The currently most popular way to train a neural network is backpropagation, which got first introduced in [27]. It has no biological equivalent but poses an effective way to optimize a huge amount of parameters. Newer methods use evolutionary algorithms and treat network parameters as population that develops over time [28], but we wont consider them further. The reason for why backpropagation works so well to optimize neural networks, is the simplicity of the mathematical operations that make them up. Not only do the activation functions have an analytical derivative,

but further can we just apply the chain rule multiple times on the loss function, so to find the gradient for every network parameter. Suppose we have the loss L, then the derivative with respect to the weights $\boldsymbol{W}_l$ of layer l, is just given as

$$\frac{\partial L}{\partial \boldsymbol{W}_l} = \boldsymbol{\delta}_l \boldsymbol{x}_{l-1}^T, \tag{3.49}$$

where for the last layer N, and all previous layers l, we have

$$\boldsymbol{\delta}_N = \frac{\partial L}{\partial \boldsymbol{x}_N} \cdot f_N'(\boldsymbol{W}_N \boldsymbol{x}_{N-1}) \tag{3.50}$$

$$\boldsymbol{\delta}_l = \boldsymbol{W}_{l+1}^T \boldsymbol{\delta}_{t+1} \cdot f_l'(\boldsymbol{W}_l \boldsymbol{x}_{l-1}), \tag{3.51}$$

with f' being the derivative of the corresponding activation function. The update for the next time-step $t+1$ is then performed according to an optimizer specific learning rate $\boldsymbol{\alpha}_{\boldsymbol{W}_l^t}$ as follows

$$\boldsymbol{W}_l^{t+1} = \boldsymbol{W}_l^t - \boldsymbol{\alpha}_{\boldsymbol{W}_l^t} \cdot \frac{\partial L}{\partial \boldsymbol{W}^l}, \tag{3.52}$$

where $\cdot$ is an element-wise multiplication.

3.2.2 Behavioral Cloning

Behavioral cloning in itself is not always related to machine learning, but poses one possible way of training a neural net in a supervised manner. The presented concept is easy to understand and got inspired by [29], where it was used for self-driving cars, and since having a car drive along the road is easier to achieve than having a robot walk around an environment, we will deal with the additional details later to focus on the main points for now. The proposed method utilizes the control loop, which was already introduced in figure 3.1. In order to then replace the human user by an artificial agent, we have a human user perform a desired behavior, and copy it. The required extended control loop is shown in figure 3.14. It simply takes the velocity commands from the human user, and stores it alongside RGBD images with a corresponding timestamp to some storage, where the RGBD images are obtained from stereo RGB images by an image processing step that is explained in section 3.2.4. The timestamp allows to correlate seen images to desired velocities afterwards, which in turn enables an artificial agent to train on the stored data. For our purposes, the artificial agent is a neural network. An appropriately chosen network architecture will then enable us to learn the taught behavior and ultimately lets us replace the human user. This procedure relies on prior knowledge to achieve certain tasks, namely the stored data. It is therefore extremely import to assure that the sampled data, from which we want to learn a task, does not introduce any unwanted bias, that is, we need to take care of the distribution from which we sample in the first place. In principle, it is possible to learn any arbitrary behavior with this technique, but this requires not only good data, but also a vast amount

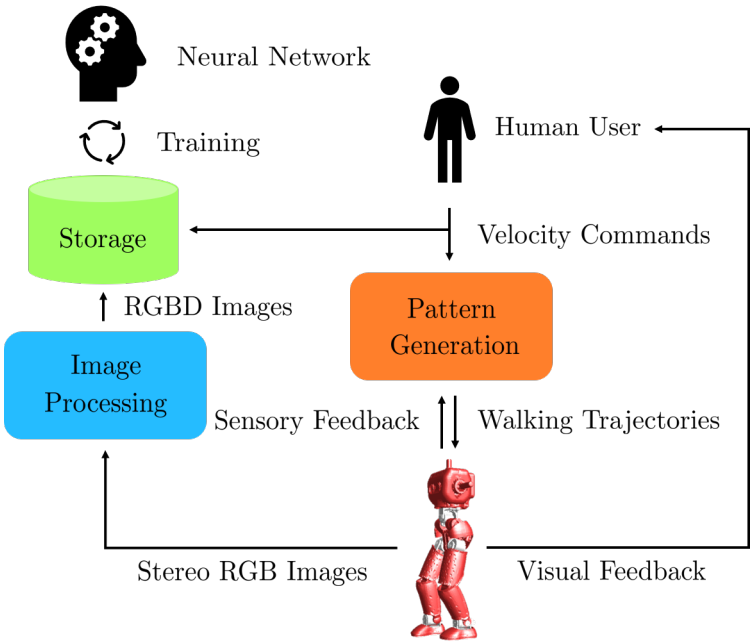


Figure 3.14: Pipeline for behavioral cloning. The neural network is trained on stored RGBD images, and corresponding velocity commands that are correlated by a timestamp

of it. There are other algorithms that explore the state space on their own, and for which we could for example use the taught behavior as prior as well. These algorithms belong to the class of reinforcement learning methods, and we will have a look at a particular one in the next section.

3.2.3 Reinforcement Learning

The goal in reinforcement learning is not only to learn actions a_t at time-step t, given a state s_t, like it is in behavioral cloning, but further to explore actions and states. This is usually performed as shown in figure 3.15, where an agent interacts with an environment to receive a reward r_t, and also changes the state as cause of its action. Therein, the actions a_t are sampled from a policy $a_t \sim \pi_\theta(a_t|s_t)$ that depends on parameters θ, which for our case are simply the weights of a neural network. The difficulty in optimizing the policy π_θ, is to have an agent to discard immediate rewards over future expected rewards. For discrete action spaces, this got well solved by deep Q-learning [30]. Different approaches for continuous action spaces like trust region policy optimization [31] are rather complicated. The, to this date, most elegant way of solving continuous control problems in a reinforcement learning setup, is proximal policy optimization [32], and we will elaborate on it in the following. Gradient policy methods, such as proximal policy optimization, try to update the policy π_θ, such that the expected total future reward $\mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$ is maximized. For the incremental update, it is therefore required to find the gradient

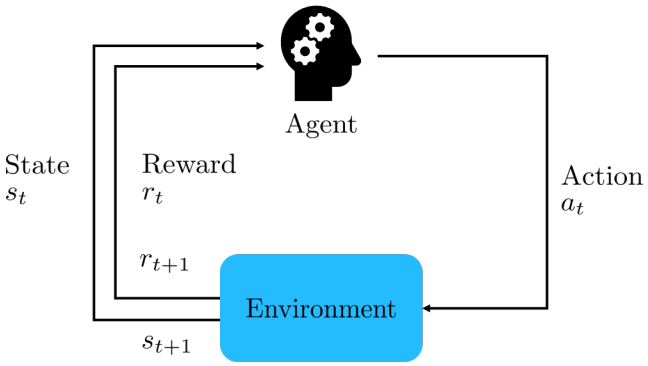


Figure 3.15: Reinforcement learning setup. As the agent interacts with the environment, the state of the environment changes.

of this expression

$$\nabla_\theta \mathbb{E}_{a_t \sim \pi_\theta} \left[\sum_{t=0}^{\infty} r_t \right] = \mathbb{E}_{a_t \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \psi_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right], \tag{3.53}$$

where ψ_t can take many forms. Since the gradient is just an estimate, as it is computed from samples being taken from the reinforcement learning environment, it usually suffers from variance and bias. As shown in [33], we can trade-off variance for bias and the other way around, by replacing ψ_t for the general advantage estimate $\hat{A}_t^{\text{GAE}}$ as follows

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V, \tag{3.54}$$

where $\delta_{t,l}^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference [34], and V the value function, which is given by a critic network. The critic's goal then is to have the gradient estimate steer the acting policy network towards actions that maximize the reward. A huge problem therein is that the policy may diverge and that policies may be discarded on the basis of a gradient estimate with a high variance. This is prohibited in proximal policy optimization by clipping the general advantage estimate, and therefore the gradient, under the following objective

$$L^{\text{CLIP}} = \min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t), \tag{3.55}$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio of the old and the new policy. The loss is shown in figure 3.16, and it assures for a positive advantage estimate that the gradient does not diverge towards actions that are way more likely under the new policy, than they have been for the old policy. Also, for a negative advantage estimate, it assures that the gradient points aways from these policies. The total objective $L_t^{\text{CLIP+VF+S}}$ of proximal policy optimization is further extended by an

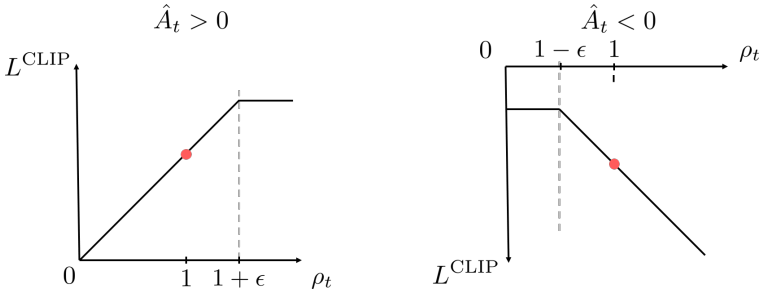


Figure 3.16: Policy gradient clipping in proximal policy optimization for positive and negative advantage estimates $\hat{A}_t$.

entropy term S that results in exploration, and the critic's loss L^{VF}, such that it can steer the gradient (equation 3.56).

$$L_t^{\text{CLIP+VF+S}} = \mathbb{E}\left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\right] \tag{3.56}$$

The critic's loss therein is the squared-error of the value function estimate and the explored values $(V_\theta(s_t) - V_t^{\text{target}})^2$. TODO add algorithm?

3.2.4 Image Processing

In the previous sections we have learned about two different approaches to train neural nets on solving certain tasks. Although we came to understand that the complexity of the task to be solved correlates strongly with the amount of data at hand, there exist domains from which it is undeniably easier to do so. To equip a neural net with some sort of prior knowledge by switching the domain may therefore not only be highly desirable but sometimes also needed if the amount or quality of data is not sufficient. One domain which is of special interest when it comes to interacting in a three dimensional environment is a domain that represents depth information. If there are any, it may sometimes be possible to extract this kind of prior knowledge from a depth camera. As for this work, we need to rely on stereo cameras and powerful algorithms that allow us to compute depth images in real time. The algorithm that helps us to do so, in terms of the extraction of weighted least squares disparity maps [35], will be presented in the following paragraph - Depth Map Extraction.

Depth Map Extraction

As already pointed out, the depth map is generated from stereo camera images by a technique called stereo block matching [36]. This method works best for edge filtered images, as will become clear soon. To obtain edge filtered images $\boldsymbol{E}$, the stereo RGB images are first converted into grayscale $\boldsymbol{G}$, which are then convolved with the Sobel kernel $\boldsymbol{S}_x$ along the horizontal axis [37] (equation 3.57, figure 3.17).

$$\boldsymbol{E} = \boldsymbol{S}_x * \boldsymbol{G} \tag{3.57}$$

When having a look at the Sobel kernel $\boldsymbol{S}_x$ (equation 3.58), it immediately becomes

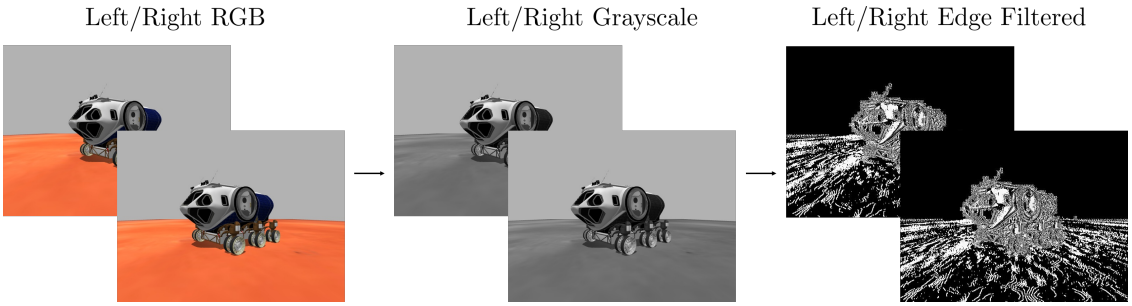


Figure 3.17: Image preprocessing to obtain edge filtered images. The images were taken within the simulation environment Gazebo (link), and show a space exploration vehicle, for which, with the friendly support of NASA, we generated a Gazebo version (link).

clear that it approximates the derivative of an image along the horizontal axis. Therefore, at locations of steep change, or simply put, edges, the convolution of the grayscale images with the Sobel kernel results in high values, and thus in the typical appearance of an edge filtered image.

$$\boldsymbol{S}_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \tag{3.58}$$

To understand the block matching algorithm, we first need to figure out the transformation that images undergo for a change in perspective, which is caused by the two different positions of the cameras within the stereo camera pair. For an ideal setup, we have two identical cameras, and they are neither rotated relatively to each other, nor is there any other translation, but a shift along the x-axis (figure. 3.18). This may of course not always be true, and there are methods to correct for uncertainties, which we will present in the following paragraph, but omit for simplicity right now. The principle goal, for the inference of depth information from two images, is to find points in the right image that correspond to points in the left image. By triangulation, the displacement or disparity of a point in the right image, relative to its corresponding point in the left image, can then be used to extract the depth. The farther a point $\boldsymbol{X}$ lies away from the cameras, the smaller its displacement will be. In figure 3.18, we can see that a point $\boldsymbol{X}$, which is seen by the left camera, could in principle lie anywhere on the epipolar line at $\boldsymbol{x}'$, as seen from the right camera, if there is no depth information available. It results that, to find correspondences, one only has to search along the epipolar line. Also, since points in the right image that correspond to points in the left image, will always be displaced to the left, one only has to search in this direction. The procedure is shown in figure 3.19. Instead of looking for single pixel correspondences, it is advised to search for whole block correspondences, since it reduces the noise drastically. Blocks of a defined block size

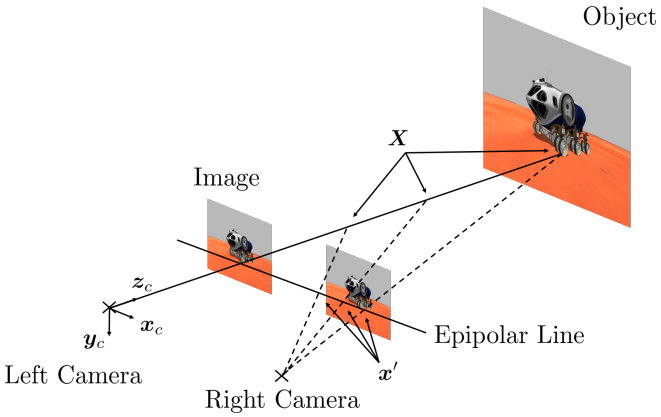


Figure 3.18: The stereo setup with a left and a right camera.

N are taken from the left image, and then the sum of absolute differences SAD is computed for every displacement d in the right image, ranging from zero to number of disparities D (equation 3.59, figure 3.19).

$$\mathrm{SAD}(d) = \sum_{x,y=0}^{N} |\boldsymbol{E}_{\mathrm{left}}(x,y) - \boldsymbol{E}_{\mathrm{right}}(x-d,y)| \tag{3.59}$$

The disparity d that minimizes the sum of absolute differences SAD is taken to serve as the best correspondence and is therefore used in the disparity map. Here we can already see that due to the uniqueness of the edge filtered the images $\boldsymbol{E}$, it is easier to find correspondences there, rather than in the grayscale or RGB images. To

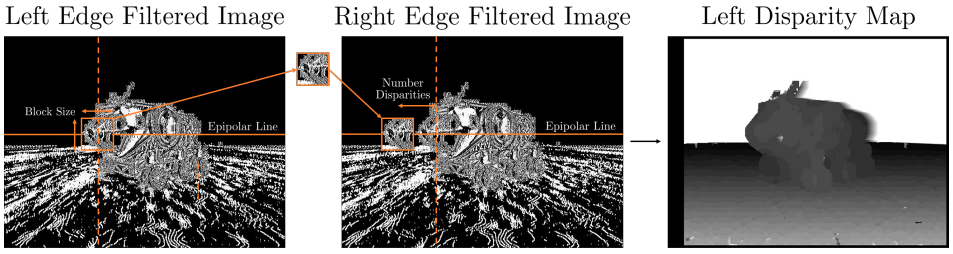


Figure 3.19: Generation of the left disparity map by the block matching algorithm.

further refine the disparity map, and especially to assure good results in textureless regions, we apply a weighted least squares filtering, which is based on the confidence of depth measures. The confidence of depth measures is obtained from the variance within the disparity map $\boldsymbol{D}$ (equation 3.60, figure 3.20).

$$\mathrm{Var}(\boldsymbol{D}) = \mathbb{E}\left[\boldsymbol{D}^2\right] - \mathbb{E}\left[\boldsymbol{D}\right]^2 \tag{3.60}$$

Therein, the expectation value for $\boldsymbol{D}$ is computed by a convolution with the kernel $\boldsymbol{K}$ from the following equation

$$\boldsymbol{K} = \alpha \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix} \tag{3.61}$$

$$\mathbb{E}\left[\boldsymbol{D}\right] = \boldsymbol{K} * \boldsymbol{D}, \tag{3.62}$$

where $\alpha = \frac{1}{\text{width} \cdot \text{height}}$ is the normalization factor. The expectation value of the disparity map squared $\mathbb{E}\left[\boldsymbol{D}^2\right]$ is computed in the same way, except for that all elements are squared prior to summing them up. Given the variance, we can introduce a concept which is named confidence map. The confidence map $\text{Con}(\boldsymbol{D})$ is a measure for the certainty of the computed disparity, and is defined to be linearly dependent on the variance as follows

$$\text{Con}(\boldsymbol{D}) = \max\left(1 - r\text{Var}(\boldsymbol{D}), 0\right), \tag{3.63}$$

where r is a roll-off factor that defines the change of confidence with growing variance. The resulting disparity confidence is shown in figure 3.20, and is used to outweigh outlying disparity values from the final weighted least squares disparity map. Prior to that, we further introduce an additional measure for the prevention

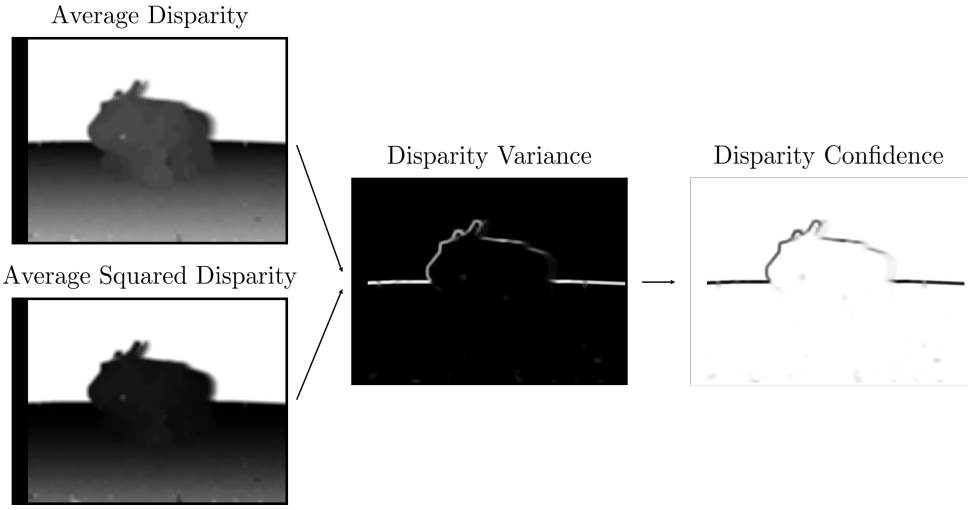


Figure 3.20: Generation of the confidence map from the variance within the disparity map.

of accidentally assigned correspondences in the initial block matching algorithm by using a left right consistency check [38]. Therefore, the block matching algorithm is used on the right image, and we search for correspondences in the left image. In contrast to the computation of the left disparity map $\boldsymbol{D}_{\text{left}}$, for the right disparity map $\boldsymbol{D}_{\text{right}}$, we only need to check for correspondences along the epipolar line in the

positive displacement direction. The left right consistency $\boldsymbol{L}$ is then obtained by

$$\boldsymbol{L}(x,y) = \begin{cases} \min\left[\mathrm{Con}(\boldsymbol{D}_{\text{left}})(x,y), \mathrm{Con}(\boldsymbol{D}_{\text{right}})(x+d_{\text{left}},y)\right] & \text{for } \Delta d < t \\ 0 & \text{else} \end{cases}, \tag{3.64}$$

where d_{left} is the disparity of $\boldsymbol{D}_{\text{left}}$ at position (x,y), and therefore represents the index shift which results from the block matching algorithm. Furthermore, if $\Delta d = \boldsymbol{D}_{\text{left}}(x,y) + \boldsymbol{D}_{\text{right}}(x+d_{\text{left}},y)$ is smaller than a threshold t, then the left right consistency $\boldsymbol{L}$ is taken to be the lower bound approximation of the left and right confidences. Otherwise, the consistency is taken to be false, and hence zero (figure 3.21). As already pointed out, the left right consistency, which is nothing but a confidence measure, usually reveals uncertainties in textureless regions. The weighted least squares filtering that we are about to present uses this fact to its advantage. In a first step, a consistency weighted disparity map $\boldsymbol{C}$ is computed via equation 3.65 (figure 3.21).

$$\boldsymbol{C} = \boldsymbol{L} \cdot \boldsymbol{D}_{\text{left}}, \tag{3.65}$$

where $\cdot$ is an element-wise multiplication. Further, the weighted least squares filter is based on the idea of a bilateral filter [39], and it will try to minimize an energy function $J(\boldsymbol{U})$, which takes the original grayscale image as guidance to compute a weight $w_{p,q}$ for neighboring pixels p, and q as follows

$$w_{x,y,i,j}(g) = exp(-|g_{x,y} - g_{i,j}|/\sigma). \tag{3.66}$$

Depending on the range parameter σ, this weight will be high for similar neighboring pixels of the grayscale image g, and therefore lead to huge costs in the following energy function $J(\boldsymbol{U})$ that we try to minimize

$$J(\boldsymbol{U}) = \sum_{x,y} \left[(u_{x,y} - c_{x,y})^2 + \lambda \sum_{(i,j)\in\mathcal{N}(x,y)} w_{x,y,i,j}(g)(u_{x,y} - u_{i,j})^2 \right], \tag{3.67}$$

where $c_{x,y}$ are single pixels of the consistency weighted disparity map. The formulation of this energy function results in a solution $\boldsymbol{U}$ that encourages the propagation of disparity values from high- to low-confidence regions (figure 3.21). Additionally, the weight w, together with the smoothing parameter λ, ensure to have similar disparity values in regions with similar texture. The final disparity map $\boldsymbol{D}_{\text{final}}$ is then obtained by normalizing the resulting image $\boldsymbol{U}$ with

$$\boldsymbol{D}_{\text{final}} = \frac{\boldsymbol{U}}{\mathrm{WLS}(\boldsymbol{L})}, \tag{3.68}$$

where $\mathrm{WLS}(\boldsymbol{U})$ is the weighted least squares filtered version of the left right consistency $\boldsymbol{L}$.

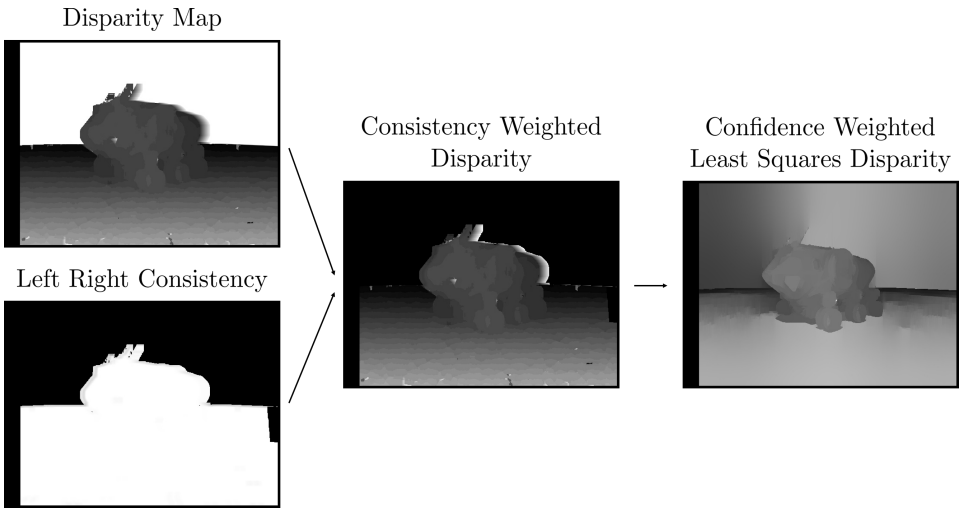


Figure 3.21: Generation of the confidence weighted least squares disparity from the disparity map, and the left right consistency.

As already mentioned for figure 3.18, the assumption of a simply translated stereo camera pair is almost never correct. In addition, there exist camera intrinsics that deform the observed image, and so the epipolar lines. Therefore, as a requirement for the algorithm to work properly, it is important to calibrate the robot's cameras. The next chapter - Mono and Stereo Camera Calibration, will explain in detail how this is done.

Mono and Stereo Camera Calibration

To correct images, as we observe them with a camera, it is required to have a mathematical description of it. A simple one for a camera is the pinhole model, which is shown in figure 3.22. For a pinhole camera model, the image plane lies behind the coordinate frame of the camera, and is turned the other way around, but it is easier to describe the image in a virtual plane, which is located at a distance f along the z_c-axis, where f is the focal length. According to the intercept theorem, a point $\boldsymbol{X}_c = (X, Y, Z)^T$ is then simply projected to the image plane by the camera matrix $\boldsymbol{K}$ with

$$\boldsymbol{x}_c = \boldsymbol{K}\boldsymbol{X}_c = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \boldsymbol{X}_c. \tag{3.69}$$

Therein, $\boldsymbol{K}$ contains the intrinsic camera parameters, such as the focal lengths and the principal point. For a true pinhole camera $f_x = f_y = f$, but due to errors, usually two different values are chosen. The principal point lies at the position where a light ray connects perpendicularly to the image plane after passing the pinhole, and therefore just defines an offset. For a real setup, it is also required to put a lens at the pinhole's position, which adds some distortion to the image.

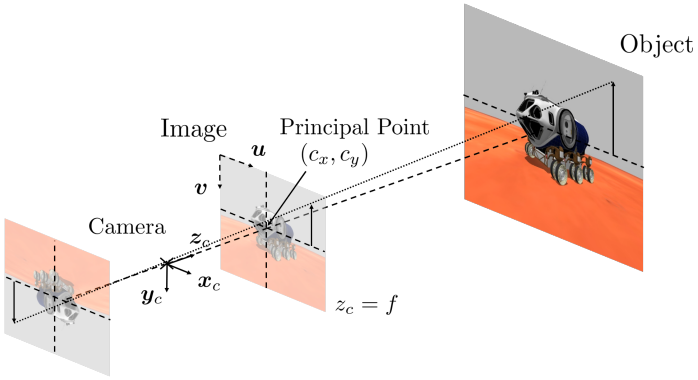


Figure 3.22: Pinhole camera model.

According to [40], we model radial and tangential distortion by

$$x_{c,u} = x_{c,d}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2x_{c,d}^2) + 2p_2 x_{c,d} y_{c,d} \tag{3.70}$$

$$y_{c,u} = x_{c,d}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_{c,d} y_{c,d} + p_2(r^2 + 2y_{c,d}^2), \tag{3.71}$$

where

$$\begin{aligned}
(x_{c,d}, y_{c,d}) &= \text{distorted image points within the camera frame } c, \\
&\quad \text{as projected onto the image plane} \\
(x_{c,u}, y_{c,u}) &= \text{undistorted image points within the camera frame } c, \\
&\quad \text{as projected by an ideal pinhole camera} \\
k_n &= \text{n}^{\text{th}} \text{ radial distortion coefficient} \\
p_n &= \text{n}^{\text{th}} \text{ tangential distortion coefficient} \\
r &= \sqrt{x_{c,d}^2 + y_{c,d}^2}.
\end{aligned}$$

Together, the focal lengths, the principal point, and the distortion coefficients make up the unknowns within our camera model. Goal of the mono camera calibration is now to find these coefficients from images of a well known calibration pattern. Therefore, images of the calibration pattern are taken from different perspectives (figure 3.23). For the mathematical description, the calibration pattern is taken to be at a fixed position and orientation, while it is assumed that the camera was moved. The position of each corner can then be described by the square's size a as follows

$$\boldsymbol{x}_{nm} = \begin{pmatrix} wa & ha & 0 & 1 \end{pmatrix}^T, \tag{3.72}$$

where we now switched to homogeneous coordinates, and $w \in [0, W], h \in [0, H]$ are whole numbers, corresponding to the width and the height of the pattern. It is then required to find the rotation $\boldsymbol{R}$ and translation $\boldsymbol{t}$, which transforms the object points to the image plane. They are estimated by solving a perspective-n-point problem

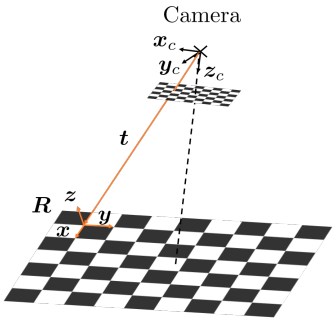
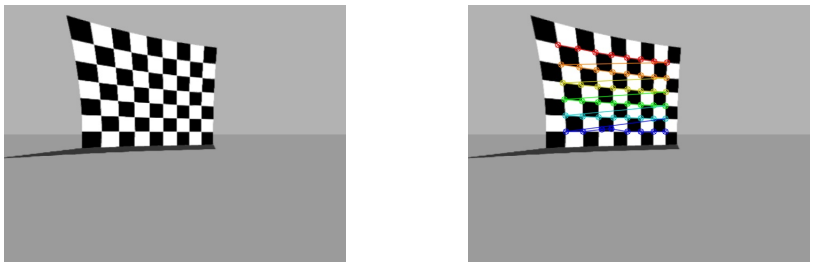


Figure 3.23: Calibration pattern as observed from the camera's coordinate system C. Within the object's coordinate system, all chessboard corners lie at a zero z-position.

[41]. Therefore, as shown in figure 3.24 (b), a corner detecting algorithm finds the corners $\boldsymbol{x}_{c,wh}$ within the image plane. Under the assumption of known intrinsic camera parameters, $\boldsymbol{x}_{c,wh}$ are then being undistorted according to equations 3.70 and 3.71. Each $\boldsymbol{x}_{nm}$ can then be transformed to the camera's frame, and further be



(a) (b)

Figure 3.24: Distorted calibration pattern (a), and the image points as found by the algorithm (b).

projected onto the image plane via

$$\boldsymbol{x}_{c,wh} = \boldsymbol{K} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \end{pmatrix} \boldsymbol{x}_{wh}, \tag{3.73}$$

where $\boldsymbol{R}$ describes the rotation, and $\boldsymbol{t}$ the translation of the camera frame to the object frame. Then, equations 3.70 and 3.71 are applied to obtain the undistorted image points from $\boldsymbol{x}_{c,wh}$. The undistorted image points $\boldsymbol{x}_{c,wh,u}$ are then being reprojected by inverting the rotation and translation via

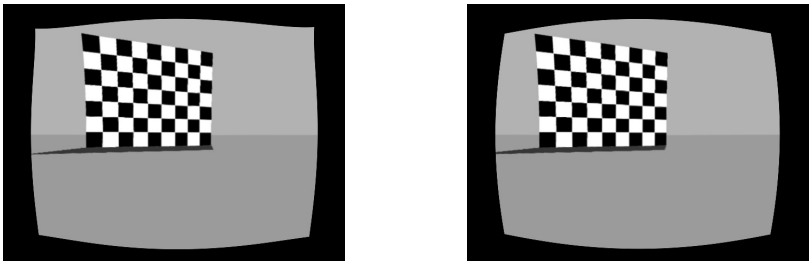
$$\boldsymbol{x}_{wh,u} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \end{pmatrix}^{-1} \boldsymbol{x}_{c,wh,u}, \tag{3.74}$$

from which we compute the re-projection error $\Delta x = ||\boldsymbol{x}_{wh,u} - \boldsymbol{x}_{wh}||_2$. To find the intrinsic parameters, a Levenberg-Marquardt algorithm then optimizes them in an iterative scheme to minimize the re-projection error until convergence [42]. The stereo camera calibration can then be performed by applying the mono camera calibration

to each camera separately, from which the camera intrinsics are obtained. Given the camera intrinsics of both cameras, it is again possible to solve a perspective-n-point problem, which yields the positions and orientations of both cameras with respect to the observed object. This enables us to compute the fundamental matrix $\boldsymbol{F}$, which transforms points from the left camera's view $\boldsymbol{x}_{c_{\text{left}}}$, to points $\boldsymbol{x}_{c_{\text{right}}}$, as seen by the right camera via

$$\boldsymbol{x}_{c_{\text{right}}}^{T} \boldsymbol{F} \boldsymbol{x}_{c_{\text{left}}} = \boldsymbol{0} \tag{3.75}$$

The mapping enables us to rectify the left and the right image [43], which means that we can compute homography transforms that align epipolar lines within the images (figure 3.25). These homography transforms map the images, as we observe



(a) (b)

Figure 3.25: Undistorted calibration pattern, as observed by the left camera (a), and by the right camera (b). For comparison, see the distorted calibration pattern in figure 3.24, and note how the horizons within the images align.

them with the cameras, to a shared virtual plane. Therefore, we reached our initial goal, since it enables us to apply the stereo block matching algorithm, which got introduced earlier, to the transformed images.

4 Methods

4.1 Software

4.2 Implementation

5 Experiments

5.1 Camera Calibration

As described in section 3.2.4, in order for the stereo block matching algorithm to work properly (equation 3.59), it is required to calibrate the cameras. We therefore chose to use a chess-board calibration pattern, see figure 5.1. The used calibration pattern has width of $W = 8$, and a height of $H = 6$, where each square has a size of $a = 22.5\,\text{mm}$ (equation 3.72). We took a total of $N = 60$ images of the calibration pattern for varying orientations and translations with respect to the camera, which results in a total of $W \times H \times N = 2880$ points for the calibration. As the resulting

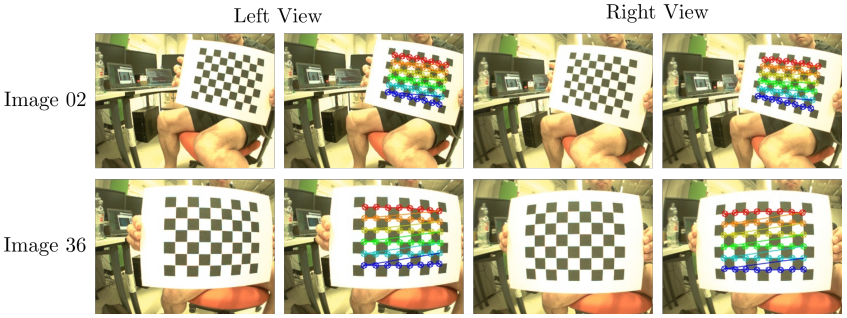


Figure 5.1: Exemplary left and right camera views of the calibration pattern as acquired during the calibration process. The colorful points indicate the detected corners in the image plane. Refer to figure 3.24 for the theory.

mean squared re-projection error $\Delta\bar{x} = 1/(WHN)\sum_0^{WHN}\Delta x$ (equation 3.74), we obtained $\Delta\bar{x}_l = 0.26\,\text{pixel}$, and $\Delta\bar{x}_r = 0.25\,\text{pixel}$, for the left, and the right camera, respectively. According to equations 3.69, 3.70, and 3.71, we therefore determined the camera intrinsic parameters as listed in table 5.1. TODO name rectification transform in background Then given the calibration of each single camera, we computed the rectification transforms $\boldsymbol{R}$, and the projection matrices $\boldsymbol{P}$ in the rectified coordinate system for each camera 5.2. Exemplary rectified images, which rely on the matrices of table 5.2, are shown in figure 5.2. Since there is a slight rotation of the calibration pattern, it is not obvious that the images got rectified well. Therefore, the same images are shown in figure 5.3, but slightly rotated such that the calibration pattern aligns horizontally. The blue line therein indicates that in contrast to the original image, straight lines now appear straight across both images, which is crucial for the block matching algorithm in the next section - Depth Map Parameter Tuning.

Intrinsic Parameter	Left Camera	Right Camera
f_x	$2.36 \cdot 10^2$	$2.32 \cdot 10^2$
f_y	$2.37 \cdot 10^2$	$2.32 \cdot 10^2$
c_x	$1.63 \cdot 10^2$	$1.86 \cdot 10^2$
c_y	$1.11 \cdot 10^2$	$1.30 \cdot 10^2$
k_1	$-4.54 \cdot 10^{-1}$	$-4.58 \cdot 10^{-1}$
k_2	$2.90 \cdot 10^{-1}$	$3.18 \cdot 10^{-1}$
k_3	$-1.21 \cdot 10^{-1}$	$-1.48 \cdot 10^{-1}$
p_1	$-2.73 \cdot 10^{-3}$	$3.02 \cdot 10^{-4}$
p_2	$2.16 \cdot 10^{-4}$	$7.63 \cdot 10^{-4}$

Table 5.1: Intrinsic parameters of single cameras. These parameters can be found as YAML file on GitHub (link).

Camera	Extrinsic Parameter	
Left	$\boldsymbol{R}$	$\begin{pmatrix} 9.93 \cdot 10^{-1} & -2.65 \cdot 10^{-3} & 1.14 \cdot 10^{-1} \\ 5.41 \cdot 10^{-1} & 1.00 \cdot 10^{0} & -2.39 \cdot 10^{-2} \\ -1.14 \cdot 10^{-1} & 2.43 \cdot 10^{-2} & 9.93 \cdot 10^{-1} \end{pmatrix}$
	$\boldsymbol{P}$	$\begin{pmatrix} 2.34 \cdot 10^2 & 0.00 & 1.88 \cdot 10^2 & 0.00 \\ 0.00 & 2.34 \cdot 10^2 & 4.87 \cdot 10^1 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \end{pmatrix}$
Right	$\boldsymbol{R}$	$\begin{pmatrix} 9.95 \cdot 10^{-1} & -2.30 \cdot 10^{-2} & 9.93 \cdot 10^{-2} \\ 2.07 \cdot 10^{-2} & 1.00 \cdot 10^{0} & 2.38 \cdot 10^{-2} \\ -9.98 \cdot 10^{-2} & -2.16 \cdot 10^{-2} & 9.95 \cdot 10^{-1} \end{pmatrix}$
	$\boldsymbol{P}$	$\begin{pmatrix} 2.34 \cdot 10^2 & 0.00 & 1.88 \cdot 10^2 & -1.60 \cdot 10^1 \\ 0.00 & 2.34 \cdot 10^2 & 4.88 \cdot 10^1 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \end{pmatrix}$

Table 5.2: Rectification transform $\boldsymbol{R}$, and projection matrices $\boldsymbol{P}$, for the left, and the right camera, respectively. These parameters can be found as YAML file on GitHub (link).

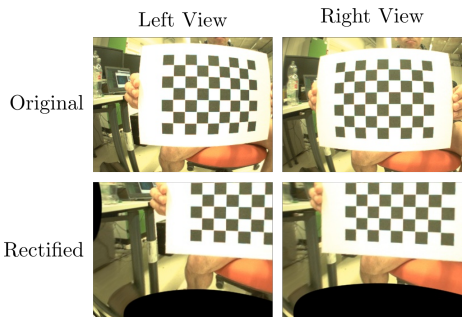


Figure 5.2: Rectified and original view of the stereo camera. Due to a slight rotation of the calibration pattern, it does not immediately become clear that the image got rectified correctly, see figure 5.3. Refer to figure 3.25 for the theory.

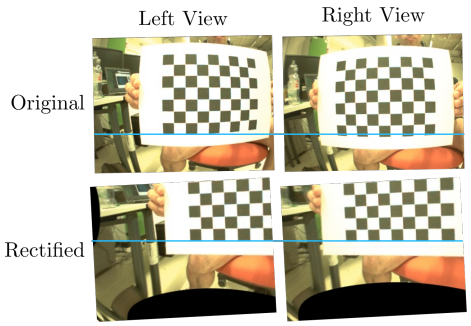


Figure 5.3: Same images as shown in figure 5.2, but rotated such that the blue line indicates that straight lines are now indeed straight. Please note that the blue lines do not correspond to the epipolar lines, but to rotated versions of them.

5.2 Depth Map Parameter Tuning

Within this section, we shortly explore the effects of all tunable parameters on the depth map generation. Therefore, we utilize a simple experimental setup, which is shown in figure 5.4. Within the setup, Heicub points its stereo camera towards three chairs that are located at a distance of 1 m towards each other, and towards the cameras, so to cover close, medium, and far distances. The consecutive chairs are slightly shifted, in order to enable the simultaneous observation of all of them. The rectified view of the environment is shown in figure 5.5.

5.3 Autonomous Walking

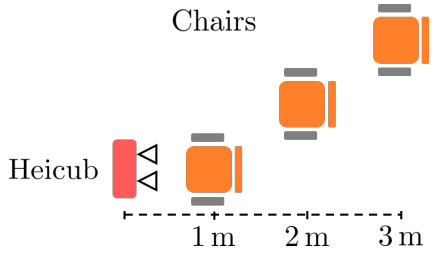


Figure 5.4: Setup for the parameter tuning of the depth map extraction. Heicub's cameras are indicated by the two triangles, of which you can find the view in figure 5.5

(a) (b)

Figure 5.5: Heicub's perspective of the scene for the setup in figure 5.4. (a) shows the left camera's view, while (b) shows the right camera's view.

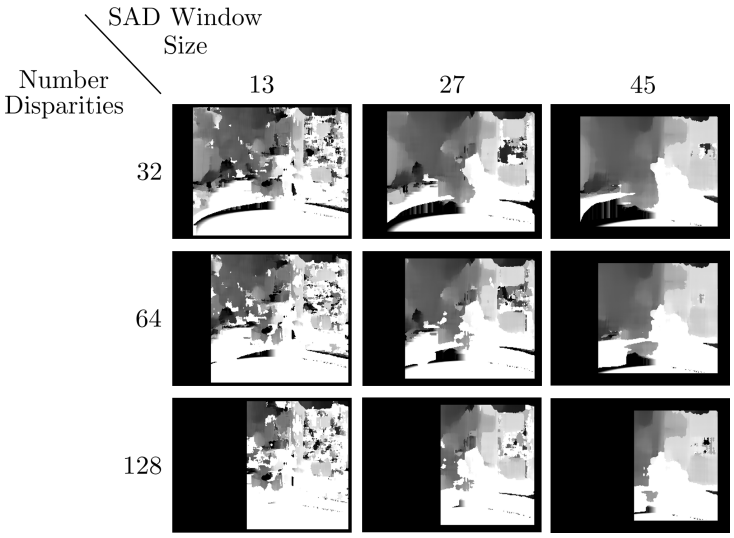


Figure 5.6: Left disparity map for changing SAD window sizes and number of disparities. Please refer to figure 3.19 and equation 3.59 for the theory.

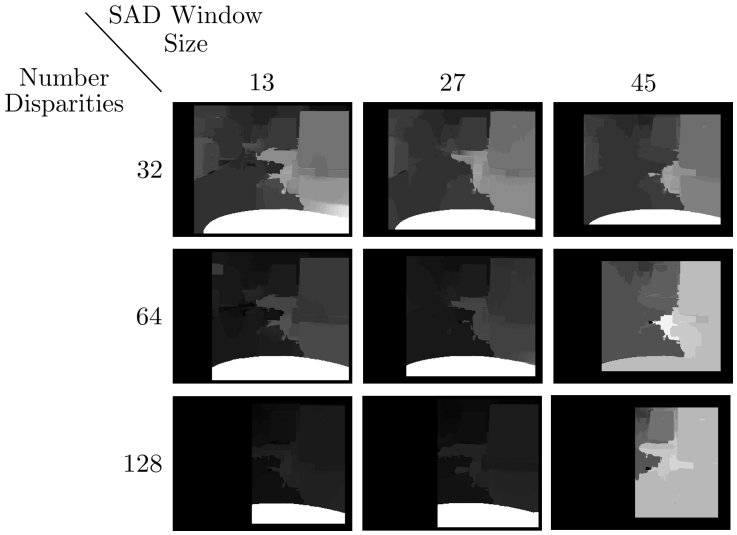


Figure 5.7: Confidence weighted least squares disparity for changing SAD windows sizes and number of disparities. Please refer to figure 3.21 and equation 3.68 for the theory.

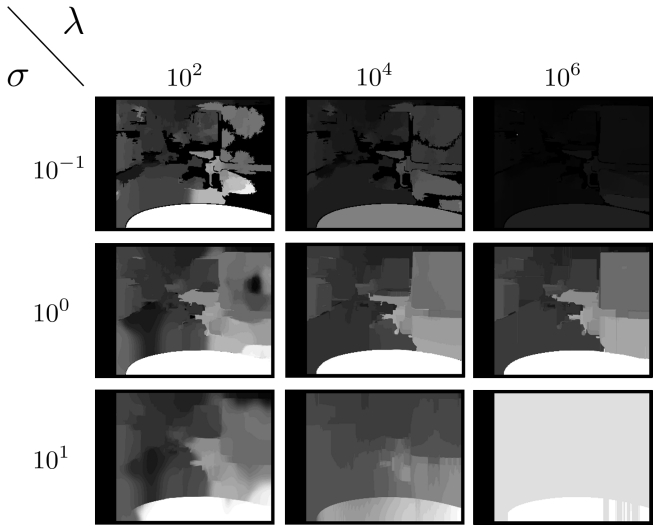
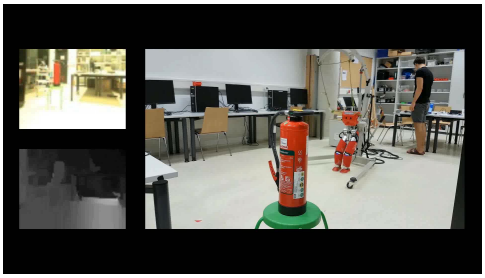


Figure 5.8: σ3.66 λ3.67

(a) (b)

Figure 5.9: Left disparity map, and confidence weighted least squares disparity map without rectification.



6 Conclusion

Part I

Appendix

A Heicub Startup Procedure

B Software Installation

C Lists

C.1 List of Figures

C.2 List of Tables

D Bibliography

[1] Miomir Vukobratovic and Davor Juricic. Contribution to the synthesis of biped gait. *IFAC Proceedings Volumes*, 2:469–478, 1968.

[2] Miomir Vukobratovic and Davor Juricic. Contribution to the synthesis of biped gait. *IEEE Transactions on Biomedical Engineering*, pages 1–6, 1969.

[3] Jin-ichi Yamaguchi, Atsuo Takanishi, and Ichiro Kato. Development of a biped walking robot compensating for three-axis moment by trunk motion. *Journal of the Robotics Society of Japan*, 11(4):581–586, 1993.

[4] Miomir Vukobratovi and Branislav Borovac. Zero-moment point—thirty five years of its life. *International journal of humanoid robotics*, 1(01):157–173, 2004.

[5] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. The development of Honda humanoid robot. *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, 2:1321–1326, 1998.

[6] Anirvan Dasgupta and Yoshihiko Nakamura. Making feasible walking motion of humanoid robots from human motion capture data. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, 2:1044–1049, 1999.

[7] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, 2:1620–1626, 2003.

[8] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to humanoid robotics*. Springer, 2014.

[9] Qiang Huang, Kazuhito Yokoi, Shuuji Kajita, Kenji Kaneko, Hirohiko Arai, Noriho Koyachi, and Kazuo Tanie. Stability compensation of a mo-bile manipulator by manipulator motion: Feasibility and planning. *IEEE Transactions on robotics and automation*, 17(3):280–289, 2001.

[10] Raphael Michel. *Dynamic filter for walking motion corrections*. Bachelor's thesis, Heidelberg University, 2017.

[11] Mikael Haggstrom and Others. Medical gallery of Mikael Haggstrom 2014. *WikiJournal of Medicine*, 1(2), 2014.

[12] Kyoung-Su Oh and Keechul Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[13] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Juyang Weng, Narendra Ahuja, and Thomas S Huang. Cresceptron: a self-organizing neural network which grows adaptively. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 576–581, 1992.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[18] SS Viglione. 4 Applications of pattern recognition technology. In *Mathematics in Science and Engineering*, pages 115–162. 1970.

[19] A G. Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971.

[20] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[21] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[23] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *IET*, 1999.

[27] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

[28] David J Montana and Lawrence Davis. Training Feedforward Neural Networks Using Genetic Algorithms. In *IJCAI*, pages 762–767, 1989.

[29] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, and Others. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[33] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[34] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. 1998.

[35] Dongbo Min, Sunghwan Choi, Jiangbo Lu, Bumsub Ham, Kwanghoon Sohn, and Minh N. Do. Fast global image smoothing based on weighted least squares. *IEEE Transactions on Image Processing*, 23(12):5638–5653, 2014.

[36] Rostam Affendi Hamzah, Rosman Abd Rahim, and Zarina Mohd Noh. Sum of absolute differences algorithm in stereo correspondence problem for stereo matching in computer vision application. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 1, pages 652–657, 2010.

[37] Irwin Sobel. An Isotropic 3x3 Image Gradient Operator. *Presentation at Stanford A.I. Project 1968*, 2014.

[38] Geoffrey Egnal, Max Mintz, and Richard P. Wildes. A stereo confidence metric using single view imagery with comparison to five alternative approaches. *Image and vision computing*, 22(12):943–957, 2004.

[39] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Iccv*, volume 98, page 2, 1998.

[40] C Brown Duane. Close-range camera calibration. *Photogramm. Eng*, 37(8):855–866, 1971.

[41] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[42] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

[43] Charles Loop and Zhengyou Zhang. Computing rectifying homographies for stereo vision. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 1, pages 125–131, 1999.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den (Datum) .....