Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master Thesis

Playing Piano with a Shadow Dexterous Hand

vorgelegt von

Benjamin Scholz

17.09.2019

Universität Hamburg

Fakultät für Mathematik, Informatik und Naturwissenschaften

Fachbereich Informatik

Studiengang: Master of Science Informatik

Matrikelnummer: 6428073

Erstgutachter: Prof. Dr. Jianwei Zhang

Zweitgutachter: Dr. Norman Hendrich

Betreuer: Michael Görner

Abstract

The piano is a versatile instrument. There are many fine details in playing the piano, which is why it takes years of training for humans to master it. The Shadow Dexterous Hand is a tendon-driven robotic hand modeled after the human hand. The scenario of playing the piano offers several interesting aspects in relation to the Shadow Dexterous Hand. The sophisticated hand postures required when playing the piano can be used to compare the performance between the Shadow Dexterous Hand and the human hand. For the task of pressing a key, it is desirable to use a certain force, which results in a certain volume. The unmodeled dynamic response of the hand poses a challenge for this aspect. Another interesting facet of piano playing is the precise timing required when pressing the keys.
In this thesis, a software pipeline is created for the purpose of exploring suitable mechanisms to handle these tasks. Inverse kinematics for multiple targets is used to place fingers above the keys they are supposed to press. Trajectory scaling is used to address the problem of pressing the keys with precise timing. Mixture Density Networks are used for the purpose of generating suitable finger movements to press the keys with a certain force.

Zusammenfassung

Das Klavier ist ein vielseitiges Instrument. Es gibt viele feine Details im Klavierspiel, weshalb es Jahre der Ausbildung braucht, bis der Mensch es beherrscht. Die Shadow Dexterous Hand ist eine sehnengesteuerte Roboterhand, die der menschlichen Hand nachempfunden ist. Das Szenario des Klavierspiels bietet mehrere interessante Aspekte in Bezug auf die Shadow Dexterous Hand. Die komplexen Handhaltungen, die beim Klavierspielen erforderlich sind, können genutzt werden, um die Leistung zwischen der Shadow Dexterous Hand und der menschlichen Hand zu vergleichen. Für die Aufgabe, eine Taste zu drücken, ist es wünschenswert, eine bestimmte Kraft einzusetzen, was zu einem bestimmten Volumen führt. Die unmodellierte dynamische Reaktion der Hand stellt eine Herausforderung für diesen Aspekt dar. Eine weitere interessante Facette des Klavierspiels ist das präzise Timing, das beim Drücken der Tasten benötigt wird. In dieser Arbeit wird eine Software-Pipeline erstellt, um geeignete Mechanismen zur Bewältigung dieser Aufgaben zu untersuchen. Inverse Kinematik für mehrere Ziele wird verwendet, um die Finger über die Tasten zu legen, die sie drücken sollen. Trajektorienskalierung wird verwendet, um das Problem des zeitgenauen Tastendrucks zu lösen. Mixture Density Networks werden verwendet, um geeignete Fingerbewegungen zu erzeugen, um die Tasten mit einer bestimmten Kraft zu drücken.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Playing the piano requires a lot of delicate and precise movements. The fingers need to be coordinated in just the right way. The keys need to be hit at just the right time with a certain force.

The multifaceted nature of this instrument makes it an interesting problem for robotics. Humans need to learn playing the piano for some time to achieve a level of prowess. Some spend their lifetime to refine further their ability to play. In that aspect, it differentiates itself from daily tasks most non-disabled people can carry out.

Additionally, it is a task that is interesting to pursue with the Shadow Dexterous Hand [Shadow Robot Company, 2013]. There is a broad range of tasks that can be solved with just a 2-finger or 3-finger gripper. With these types of grippers, it is possible to push or pick up objects. First of all, the keys of a keyboard are small, so that it is difficult for many grippers to press only one key at a time. Another factor is the speed and the delicacy of pressing the keys, which is better to control when using a robotic hand.

Pressing multiple keys at once furthermore presents an exciting aspect in regards to the Shadow Dexterous Hand. For this purpose, multiple fingers need to be put into an appropriate position to press down the keys. In this context, inverse kinematics with multiple targets becomes of interest.

Another element of interest is the comparison to human performance. We can compare how well a robotic finger can handle pressing the key compared to how humans conduct themselves. The Shadow Dexterous Hand is designed to resemble a human hand. This poses the question of whether it is modeled closely enough to replicate techniques used by humans or if there are any kinematic limitations that restrict the hand from certain movements.

Humans learn how to play with a certain volume with their ears. It requires to hit the keys with a certain speed. For this purpose, we can use the MIDI output from the electronic keyboard instead. It contains which key was pressed with how much velocity.

1.2 Related Work

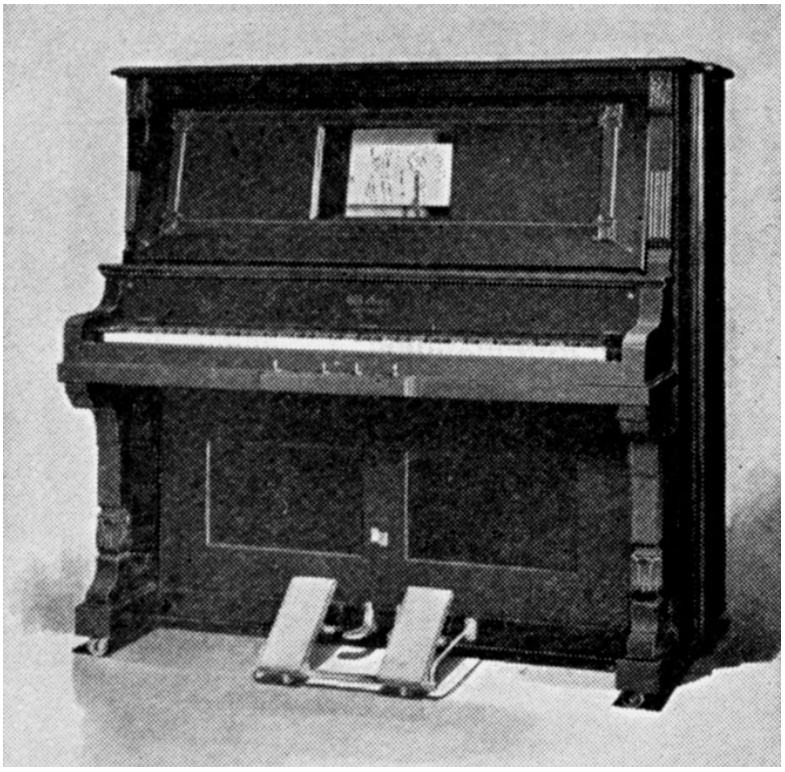

FIGURE 1.1: Example of a player piano. In particular, the Pianola Piano by the Aeolian Company released in 1904. (Retrieved from `http://www.pianola.org/history/history_playerpianos.cfm`, accessed on 10.09.2019)

Automatically playing pianos has a long history. So-called player pianos were started to be manufactured in the 1890s. These types of pianos used pneumatic mechanisms to push the keys. Early versions required pumping pedals by foot to push air into the system. The pieces to be played were encoded on paper rolls by perforations. The perforations would allow the air to get to the specific key that is supposed to be pushed. Figure 1.1 shows the example of the Pianola manufactured by the Aeolian Company. This specific model was so popular that many people refer to all player pianos as Pianolas.

The piano has also been an object of interest for robotics researchers for a long time.

One popular approach has been to design robotic hands specifically for playing the piano. Sugano and Kato [1987] designed a hand with 21 degrees of freedom,

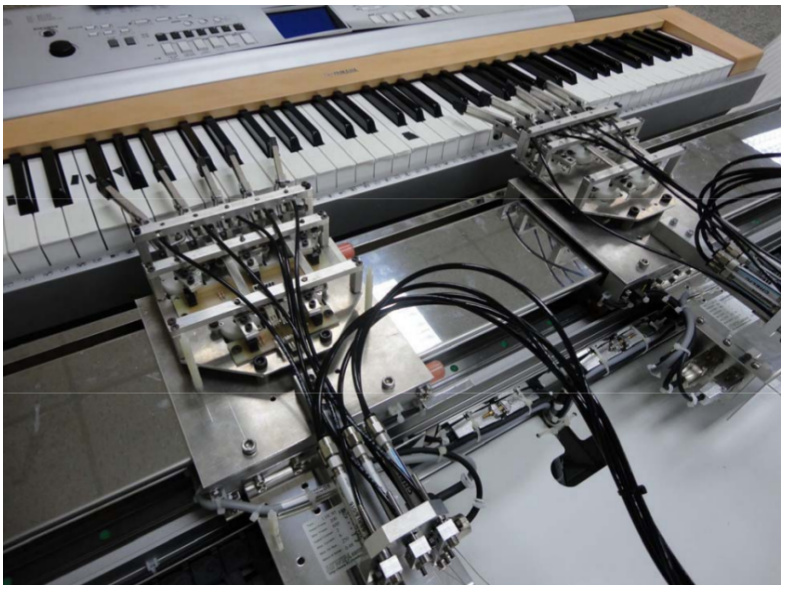

FIGURE 1.2: Robot designed to play the piano. Its movement is restricted to one axis by a motor stage [Lin et al., 2010].

where the fingers are all positioned directly above the keys. The hand is mounted on a 7 degrees of freedom arm on a humanoid robot. Based on the sheet music fed to the system, it is able to decide where to place the wrist at a given time and which fingers to use. The wrist of the hand always moves in a straight line. The joint configurations required to move the wrist to certain positions are stored in a table.

Lin et al. [2010] designed the hand shown in figure 1.2, which has 2 degrees of freedom per finger. In order to position the fingers above the piano, they use a linear motor stage. It restricts the movement of the hand to one axis. The fingertips are moved by a steel string, which is dragged by a pneumatic cylinder. They designed a controller for the hand with a number of predefined positions for the wrist and the orientation of the fingers. There are several subsequent papers. A controller was designed to coordinate the movement of two hands and compute optimal positions for the hands given a music score [Li and Chuang, 2013] [Li and Chi-Yi Lai, 2014] [Li and Li, 2017]. They use a touchscreen and a foldable keyboard as a means to create new music scores as input for their robot piano system [Li and Lai, 2016]. In [Li and Huang, 2015] a force controller was designed for the fingers.

Zhang et al. [2009] built a hand as well, that is restricted in movement to one axis with the use of a rail. For reaching the black keys the hand does not need to be moved forward, instead they move only the finger over the key with the use of an extra joint in each finger. The mechanism is shown in figure 1.3.

Batula and Kim [2010] use a small humanoid platform to play the piano. The two hands of the robot only have one finger each, that had to be modified to be small enough to be able to hit only one key at a time. Due to its small size, the

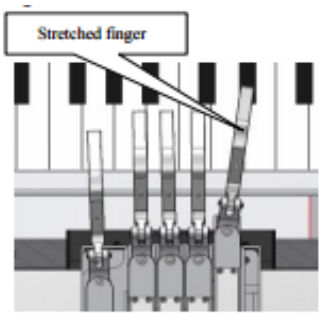

FIGURE 1.3: An additional joint is used to move fingers over the black keys [Zhang et al., 2009].

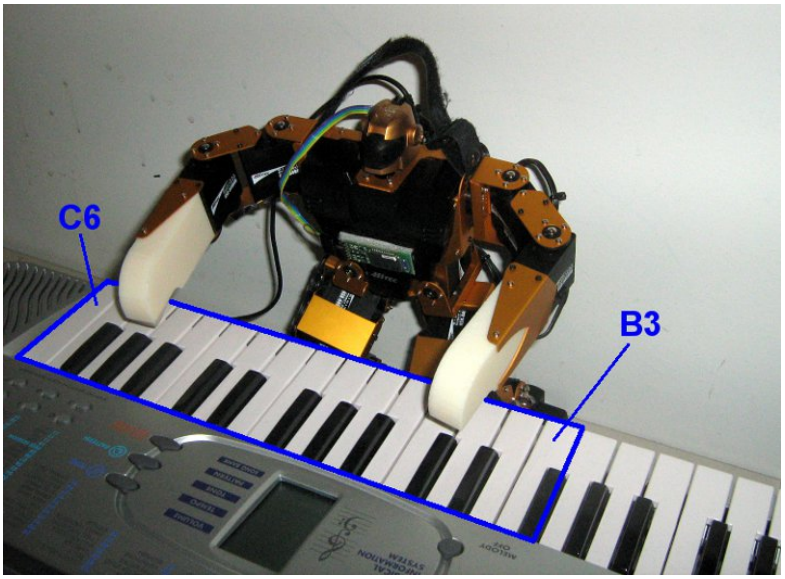

FIGURE 1.4: A small humanoid robot playing piano [Batula and Kim, 2010].

robot can only reach the range of one octave with each hand. Figure 1.4 shows the platform used and the range of reachable keys.

Jaju et al. [2016] use pieces of wood as fingers that are pulled with a string by a motor to press down on the keys. 8 fingers are placed directly over the keys to cover the range of an octave.

Topper and Maloney [2019] designed a hand for the purpose of playing the piano. The goal of their work was to study expressive aspects of a human piano player and try to replicate those with their system. The fingers of the hand can only press down and lift up. Otherwise, they are locked in place and were designed in a way that each of the fingers is settled above a key. Only the arm is used to move the fingers above different sets of keys. Each finger has two joints. The hand consists of 5 fingers in total, but for the sake of simplicity, none of them are designed after a thumb. Figure 1.5 shows the setup used.

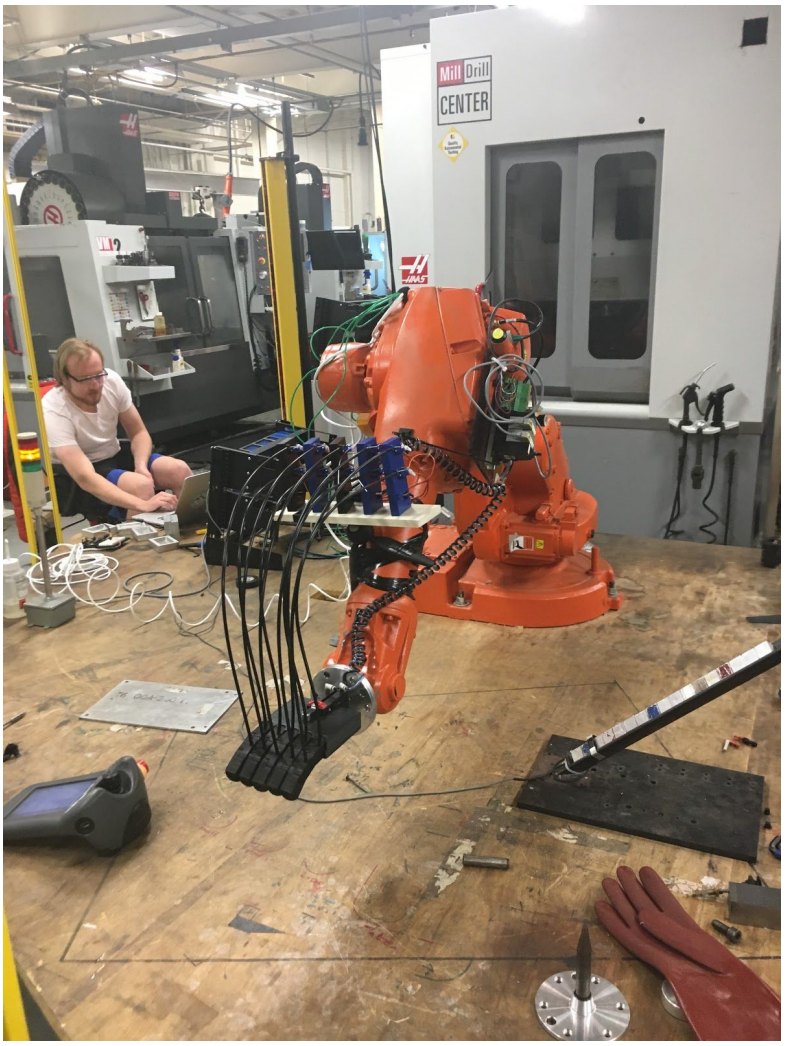

FIGURE 1.5: A hand designed to play piano [Topper and Maloney, 2019].

Hughes et al. [2018] use a passive hand, meaning the hand itself has no motors and thus cannot move by itself. Thus, the entire hand needs to be moved by the UR5 robotic arm to be able to press any keys. Figure 1.6 shows the hand. The work focuses on the impact certain changes in the ligaments of the fingers have on certain aspects of playing the keys of the piano.

Zhang et al. [2011] compare the performance of the Anatomically Correct Testbed (ACT) robotic hand with human performance. The ACT hand was designed to replicate not only the structure but also the biomechanics of a human hand. They let a number pianists at different levels play and compare the results under the aspects of keystroke velocity and the precision of timing for striking and releasing the keys.

There are a couple of papers that revolve around generating or processing data from humans playing the piano. Wee and Mariappan [2017] use a grid of electrodes to track the position of the fingers on a piano. A neural network with the processed signals of the electrodes is used to predict the position of the fingers on a key

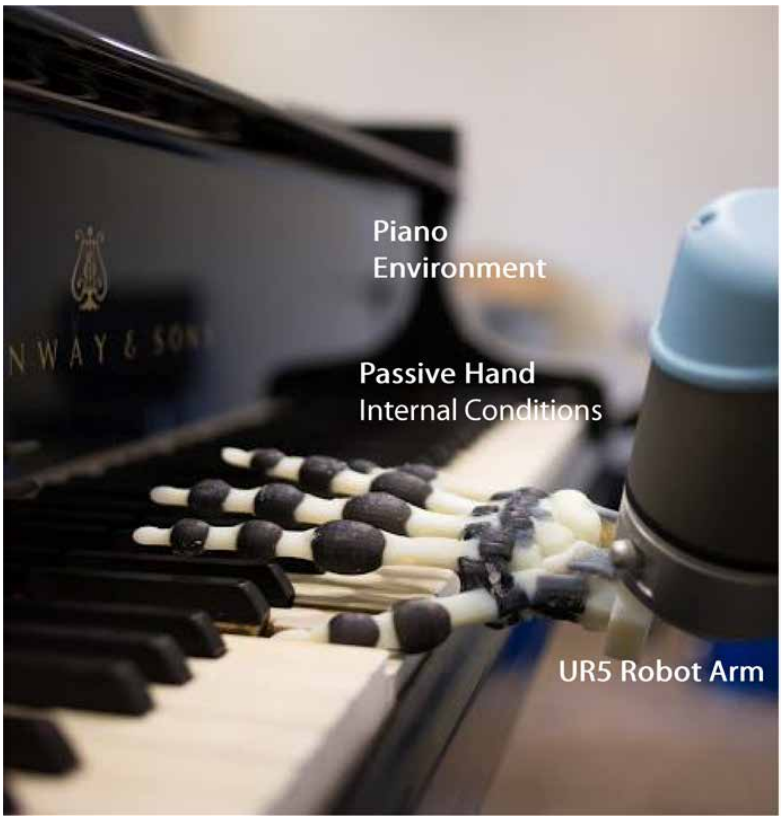

FIGURE 1.6: A passive hand playing piano [Hughes et al., 2018].

as X and Y coordinates. Yamamoto et al. [2010] record the movements of a human playing piano and use this data to generate hand and arm movements in simulation.

Besides playing the piano, there are many more attempts at playing other instruments using robots. Solis et al. [2008] built a humanoid robot with artificial lips, tongue and lung to play the flute. Alford et al. [1999] play the theremin with two robotic arms. Weinberg and Driscoll [2007] designed a robot to play the marimba.

1.3 Outline of the Thesis

Chapter 2 covers the basics. At the start, it goes over aspects of musical notation relevant for this thesis. This is followed by an introduction to the MIDI protocol, which is used throughout this thesis to collect data. The next section presents the kinematics of the Shadow Dexterous Hand. The tools for the purpose of moving the hand to certain targets are presented in the subsequent sections about inverse kinematics, motion planning and quintic spline interpolation. At last mixture density networks are explained, as they become important in the next chapter.

Chapter 3 dives into the implementation of the different components used to play the piano. It starts with the model and localization of the piano. A simple prototype for playing the piano is shown in section 3.4. Afterward, motion generation and timing are discussed with the goal to be able to press keys at specific points in time. Section 3.7 discusses how to predict parameters for pressing keys with a certain force to create a desired MIDI velocity output and how to predict the duration of a press. The following section of the chapter revolves around the kinematics of the hand for the sake of comparison with the abilities of a human hand. The last section gives an overview of the complete software pipeline created for this thesis.

Chapter 4 presents experiments and their results. The first experiment tests how precisely given MIDI velocities can be hit by predicting parameters for pressing the keys. The second experiment compares the aspect of timing when playing keys consecutively with trajectory scaling and without trajectory scaling. A subsequent experiment tests the ability to play quarter, half and full notes with exact timing. This is followed by analyzing the performance in quickly playing two alternating notes, also called a trill. The last section assesses the performance in playing a short piano piece.

Chapter 5 draws conclusions from the work presented in the previous chapters.

Chapter 6 gives an outlook about potential future work.

Chapter 2

Basics

2.1 Musical Notation

Usually, musical notes are used to describe a piece of music in western society. This short introduction is supposed to highlight some important aspects that are later relevant for choices in the implementation.

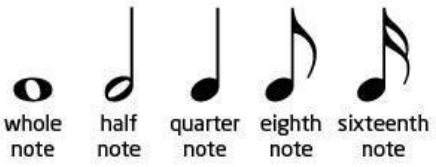

FIGURE 2.1: Example of a whole note to sixteenth note. A half note is played half as long as a whole note, a quarter note a quarter as long as a whole note and so on. (Retrieved from `https://wsfcs.learning.powerschool.com/8142864827/5thgrade/cms_page/view/24925885`, accessed on 17.07.2019)

Figure 2.1 shows the different notations to describe the timing for hitting certain notes. A half note is played half as long as a whole note, a quarter note is played a quarter as long as a whole note and so on. This is also called the value of a note. For implementation, this implicates the possibility to hit keys at a certain point in time.

Another important aspect is piano fingering. Which finger to use to play a certain note in a musical piece often depends on personal preference. Hands differ from person to person, and so it also differs what feels comfortable. Still, there is fingered sheet music, where each note of a piece has a finger assigned to it. The fingers are numbered, as shown in figure 2.2. Additionally, the figure shows a color coding for the fingers, which is used for graphs in this thesis in later chapters.

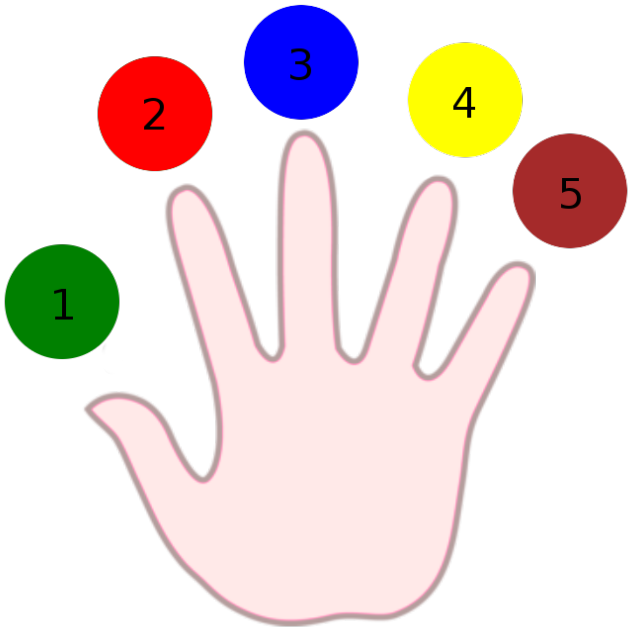

FIGURE 2.2: The numbers assigned to the fingers for fingered sheet music. Additionally, a color code was added, that is later used in graphs. (Based on image retrieved from `https://medium.com/@xiaoyunyang/how-playing-the-piano-taught-me-math-8917f84a4326`, last accessed on 13.09.2019)

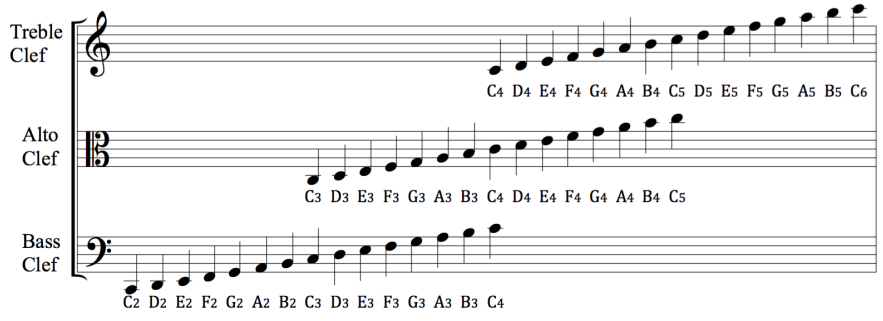

FIGURE 2.3: The different clefs used in musical notation to determine the notes on the staff. (Retrieved from `https://en.wikipedia.org/wiki/Clef`, last accessed on 13.09.2019)

Figure 2.3 shows the different clefs used in western musical notation. They indicate the pitch for the notes on the staff. The pitch of the notes on the staff are ordered from lower to higher.

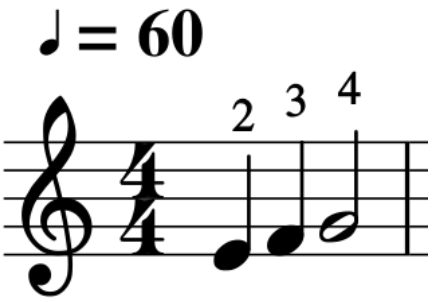

FIGURE 2.4: Short example of musical notation with a clef, time signature, fingering and instruction for beats per minute.

Figure 2.4 adds a time signature on the staff and shows an example for the notation of fingered notes. The lower number of the timing signature signifies, which note value represents one beat. In this example, a quarter note represents one beat. On the staff, the notes are segmented into bars, which are indicated by the horizontal lines on the staff. The upper number of the time signature indicates how many beats form one bar. In this example, 4 beats form a bar. Here a bar could be filled with 4 quarter notes, or as in this concrete example by 2 quarter notes and a half note.

The tempo is determined by the beats per minute (BPM). For example, hitting 1 beat each second would result in 60 BPM. The instruction at the top of figure 2.4 shows the typical notation to indicate with how many BPM a piece is supposed to be played. In this case, it is 60 BPM, with quarter notes as the beat.

The playing of multiple notes at the same time is called a chord. Concerning the piano, this means playing multiple keys at the same time.

The rapid alternation between two adjacent notes is known as a trill. On the piano, this is achieved by playing two keys, that are next to each other, in quick alternation.

Another interesting aspect is the distinction between staccato and legato. Staccato describes playing the notes in a detached style where the keys are only pressed down for a short duration. Legato, on the other hand, describes a smoother style of playing, where the keys are pressed down for a longer amount of time, and the transition to the next key occurs more fluently. Figure 2.7 explains the difference in a diagram. The transition between the first and second note is played legato, with a smooth transition. Between the second and third note, there is a noticeable gap, which indicates the staccato style. Figures 2.5 and 2.6 show examples for such notes. A dot indicates that the note should be played staccato, while the slur above notes indicates that they should be played legato style.

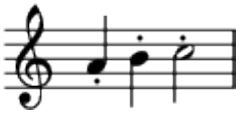

FIGURE 2.5: Example of staccato notes. They indicate that the notes are supposed to be played in a detached manner, where the keys are held down only for a short duration. (Retrieved from `https://en.wikipedia.org/wiki/Legato`, accessed on 17.07.2019)

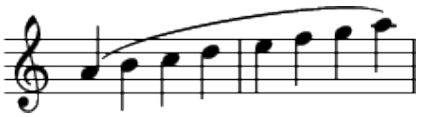

FIGURE 2.6: Example of legato notes. They indicate that the notes are supposed to played in a smooth manner, where the keys are held down longer and the transition from on key to the next occurs in a fluent manner. (Retrieved from `https://en.wikipedia.org/wiki/Legato`, accessed on 17.07.2019)

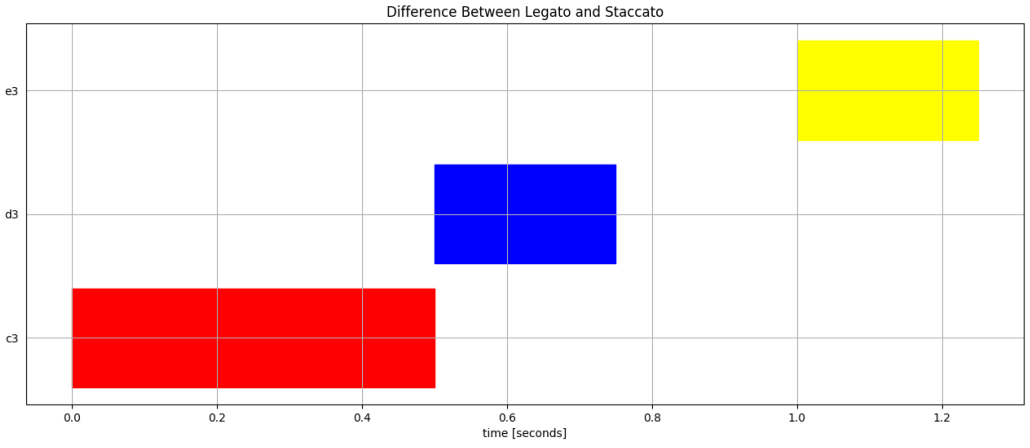

FIGURE 2.7: The bars represent how long keys get pressed. They start at the push and end at the release. The transition between C3 and D3 is played in the legato style, with no pause between both notes. Between D3 and E3 there is a gap, which means they were played staccato.

2.2 MIDI

Most modern keyboards and e-pianos offer a MIDI [MIDI Manufacturers Association, 2014] output. MIDI is an acronym for Musical Instrument Digital Interface. Usually, the data is fed to synthesizers to trigger the sounds.

A MIDI message is made up of one status byte followed by a number of data bytes. The most significant bit of the status byte is always set to 1, while it is set to 0 for the data bytes.

There are system messages, carrying for example data for time synchronization intended for all receivers in the system, and channel messages, which carry data for a specific channel such as turning notes on and off. The channel messages carry the kind of data that is interesting in the context of this thesis.

message type	status byte	1. data byte	2. data byte
Note off	$8x_{16}$	key number	velocity
Note on	$9x_{16}$	key number	velocity

TABLE 2.1: Example message for note on and note off messages. The status byte determines the message type. Here the x stands for the channel number. The first data byte signifies which key the message belongs to. The second data byte represents the velocity with which the key was pressed or released.

Table 2.1 shows the necessary bytes used to represent note on and note off messages. The status byte incorporates information about the type of message and which channel it should be sent to. The first data byte represents the key the data belongs to and the second data byte stores the velocity of the press and release, respectively.

The numbers representing the keys are mapped to certain frequencies, meaning rather than 0_{10} representing the first key, it represents a certain frequency. To convert from the number to the frequency the following formula can be used:

$$f = 440 * 2^{(n-69)/12} \tag{2.1}$$

This corresponds to 24_{10} representing the first C on the keyboard and the following numbers sequentially match the further keys on the keyboard.

The keyboard used in this thesis does not support release velocities. In that case, either the velocity of the "note off" message gets ignored, or a "note on" message is used with the velocity set to 0. The keyboard used in this thesis uses the latter solution.

As an example, the command "note on" is sent for channel 1 for the first C key on the keyboard with a velocity of 67_{10}. The resulting hexadecimal values would be 91_{16}, 18_{16}, 43_{16}. As explained the x in the status byte is replaced by the channel

number. The first C on the keyboard matches with the number 24_{10}, which is 18_{16}. The velocity is directly converted to its hexadecimal value, which is 43_{16}.

Since the most significant bit of the data bytes is always set to 0, the range for both key numbers and the velocities is 0 to 127. The produced velocity of a press is determined by how firm and fast a key is pressed. One common method to measure the velocity is to put two different buttons underneath each of the keys. The time difference between triggering buttons then determines the velocity.

This data is used by synthesizers to choose which sounds to produce. For example, the key number determines which note was played, and the velocity determines the sound level or amplitude of the produced signal.

In this thesis, the data is used in later chapters to learn certain aspects of how the keys get pressed.

2.3 Shadow Dexterous Hand

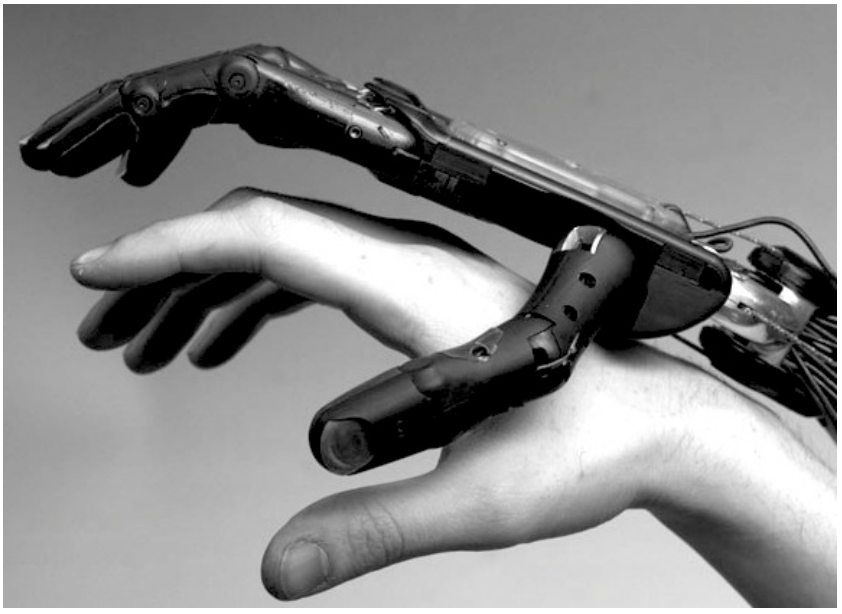

FIGURE 2.8: Comparison of the Shadow Dexterous Hand and a human hand. (Retrieved from `https://www.shadowrobot.com/card-flick/`, accessed on 14.09.2019)

The hand used throughout the thesis is the Shadow Dexterous Hand [Shadow Robot Company, 2013]. The Shadow Dexterous Hand is a multi-finger robot hand, controlled by motor-driven tendons. The dimensions of the hand were modeled after the dimensions of an actual human hand. Figure 2.8 shows a comparison.

Figure 2.9 includes kinematic diagrams, as well as measurements of the Shadow Dexterous Hand. Each finger has 4 degrees of freedom (DOF), except for the little

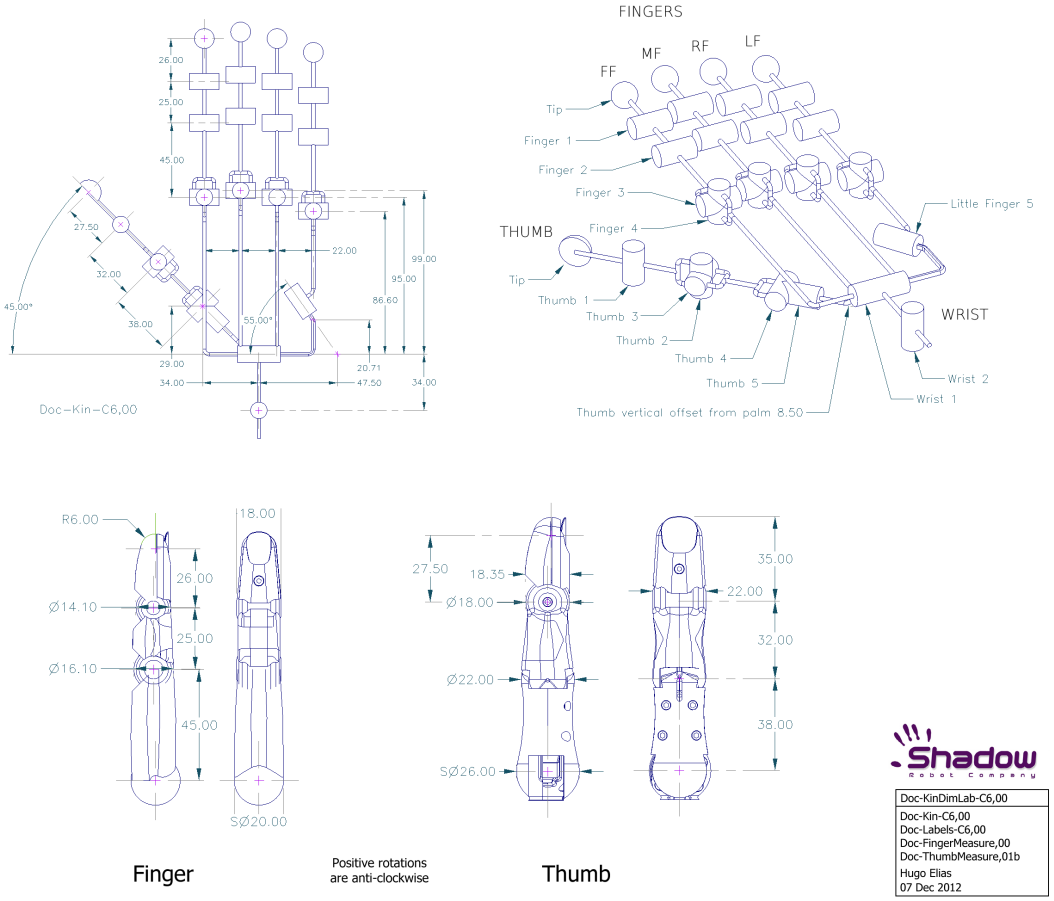

FIGURE 2.9: Kinematic diagrams and measurements of the Shadow Dexterous Hand [Shadow Robot Company, 2013].

finger, which has an additional 5th degree of freedom placed at the palm of the hand. The thumb has 5 DOF. The hand has 24 DOF in total. The placements of the joints were also chosen, to resemble the kinematics of a human hand as closely as possible. Figure 2.10 contains a table with the joint limits of the hand.

2.4 Inverse Kinematics

Both joint space and Cartesian space play an important role in robotics. The joint space is defined by the parameters of the joints of a robot. Whenever a goal for the robot to move to is defined, it is usually done with Cartesian coordinates. The Cartesian coordinates are often defined as the target position of the end-effector. Motion planning in Cartesian space is usually avoided. The required criteria for smooth and connected motions are much easier to achieve when planning in joint space. For this reason, inverse kinematics is an important tool to consider in

Joint(s)	Degrees		Radians		Notes
	Min	Max	Min	Max	
FF1, MF1, RF1, LF1	0	90	0	1.571	Coupled
FF2, MF2, RF2, LF2	0	90	0	1.571	
FF3, MF3, RF3, LF3	0	90	0	1.571	
FF4, MF4, RF4, LF4	-20	20	-0.349	0.349	
LF5	0	45	0	0.785	
TH1	0	90	0	1.571	
TH2	-25	25	-0.436	0.436	
TH3	-12	12	-0.209	0.209	
TH4	0	70	0	1.222	
TH5	-55	55	-0.960	0.960	
WR1	-45	30	-0.785	0.524	
WR2	-28	8	-0.489	0.140	

FIGURE 2.10: A table of the joint limits of the Shadow Dexterous Hand. [Shadow Robot Company, 2013].

relation to the motion planning for the arm and hand. With inverse kinematics, it is possible to convert Cartesian coordinates into joint configurations.

A library for inverse kinematics commonly used with the Robot Operating System (ROS) [Quigley et al., 2009] is TRAC-IK [Beeson and Ames, 2015]. Their IK solver is based on an inverse Jacobian method. Here a configuration q_{seed} is used as start configuration. Forward kinematics is used to compute the Cartesian pose of the configuration and to compute the Cartesian error p_{err} between the pose of the current configuration and the target pose. The Jacobian J defines the partial derivatives in Cartesian space with respect to the joint space. Consequently, the inverse Jacobian J^{-1} represents the partial derivative in the joint space with respect to the Cartesian space. The following equation is then used iteratively to move the seed configuration q_{seed} closer to the target pose [Beeson and Ames, 2015]:

$$q_{next} = q_{prev} + J^{-1} p_{err} \tag{2.2}$$

The solution is returned after some stopping criteria are fulfilled. Due to the iterative nature of the algorithm, it is prone to local minima. Beeson and Ames [2015] mitigate this shortcoming by detecting any minima and using a new seed afterward. For most robots, the solve rate of TRAC-IK is above 99%.

bio_ik [Ruppel, 2017] [Starke et al., 2017] is based on evolutionary algorithms. The genes represent the space of joint configurations. The algorithm requires fitness functions to function. The general definition of the function is as follows [Starke et al., 2017]:

$$\Phi = \Omega(x) = \sqrt{\frac{1}{k} \sum_{i=1}^{k} w_i \mathcal{L}_i^2(X_i, Y_i)} \tag{2.3}$$

Here w_i is a weight and $\mathcal{L}_i^2(X_i, Y_i)$ are a number of defined loss functions. As an example, these can be defined as distances between orientations, positions are poses. Alternatively, bio_ik also offers the possibility to use classic optimization algorithms in favor of the evolutionary ones, for example, gradient descent. In contrast to TRAC-IK, bio_ik is able to compute solutions for robots with multiple end-effectors. For this reason, it is a suitable IK solver for the Shadow Dexterous Hand used in this thesis.

The resulting solutions are joint configurations. The arm still needs to be moved from its initial configuration to the target configuration. This can be done with motion planning.

2.5 Motion Planning

Motion planning is used to create a trajectory to move a robot from an initial joint configuration to a target joint configuration.

The biggest challenge in motion planning is obstacle avoidance. Without any obstacles, it would be possible to interpolate from one joint configuration to the goal configuration. With obstacles, it is necessary to check for any collisions and find trajectories that avoid them.

The obstacles are defined in Cartesian space, while the motion planning itself is happening in joint space. In consequence forward kinematics needs to be used to determine the Cartesian coordinates for a joint configuration in order to check for any collisions.

Naive approaches to motion planning would consider the entire workspace to find a motion plan, similar to methods like the A* search algorithm [Hart et al., 1968]. First of all these kinds of algorithms only work for discretized spaces and only work well for rather small spaces, like for example in a 2D space. However, joint configurations often consist of 5 and more continuous values, and it becomes computationally infeasible to consider the entire space.

For this reason, modern methods for motion planning usually rely on some form of randomization. The randomized potential field method [Barraquand and Latombe, 1991] was among the first methods to popularize such an approach. Later algorithms based on the rapidly exploring random tree (RRT) algorithm [LaValle, 1998] became a popular method for motion planning. RRT-Connect [Kuffner and LaValle, 2000] is an example of a variant of this algorithm.

RRT-Connect enhances the original algorithm by building two trees with the goal to connect them. One tree is built from the start configuration, and one tree is built from the target configuration. The trees are built with the following steps:

1. Draw a random sample q from the search space

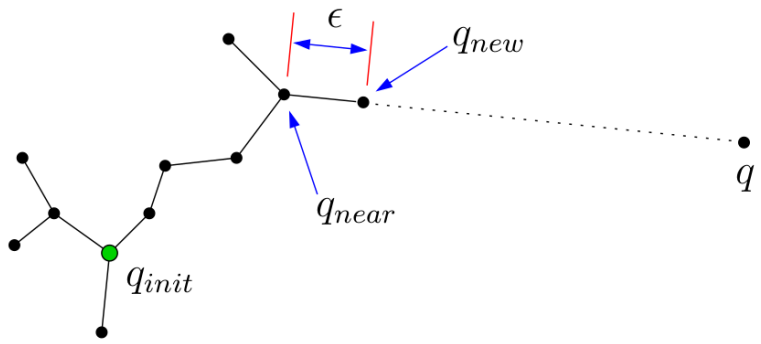

FIGURE 2.11: Extending a tree in RRT [Kuffner and LaValle, 2000].

2. Search for the nearest node q_{near} in the tree
3. Create a new node q_{new} by going a small step ϵ from q_{near} to q, if no collision was found

This operation to extend the trees is used in the full RRT-Connect algorithm as follows:

1. Extend a tree
2. Extend the other tree using the newly created node q_{new} as target
3. Repeat step 2 until a collision was found or the trees are connected. If a collision was found go back to step 1

RRT-Connect is an efficient method to find motion plans due to its probabilistic nature. Only a subset of the search space has to be checked until both trees are connected and a motion plan is found from the start point to the target point.

The result of motion planning is a trajectory. A trajectory consists of a number of waypoints. Each waypoint should at least carry information about the time at which the waypoint should be reached as well as the position of each joint. Additionally, the waypoints can also carry information about the velocity and acceleration of each joint.

2.6 Quintic Spline Interpolation

In some circumstances, it is required to create new waypoints in between two existing waypoints. Interpolation is often used in that case. Spline interpolation [Bartels et al., 1995] is popular for this task, due to its ability to produce smooth curvatures. Quintic splines, in particular, are used to ensure a smooth curvature for positions, velocities and accelerations. Spline interpolation encompasses a number of different methods. Here the focus is Hermite spline interpolation.

The goal is an interpolation between two waypoints with given position, velocity and acceleration. As a consequence the function c(t) should fulfill the following criteria, with p being the position, v being the velocity and a being the acceleration [L. Finn, 2004]:

$$\begin{aligned} c(0) =& p_0,\ c'(0) =& v_0,\ c''(0) =& a_0 \\ c(1) =& p_1,\ c'(1) =& v_1,\ c''(1) =& a_1 \end{aligned} \tag{2.4}$$

That means the start of the spline should match the start waypoint and the end of the spline should match the end waypoint. For this purpose a quintic function is well suited [L. Finn, 2004]:

$$c(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5 \tag{2.5}$$

The coefficients b_1 to b_5 need to be determined. From the criteria laid out above follows [L. Finn, 2004]:

$$\begin{aligned} p_0 =& b_0 \\ v_0 =& b_1 \\ a_0 =& 2b_2 \\ p_1 =& b_0 + b_1 + b_2 + b_3 + b_4 + b_5 \\ v_1 =& b_1 + 2b_2 + 3b_3 + 4b_4 + 5b_5 \\ a_1 =& 2b_2 + 6b_3 + 12b_4 + 20b_5 \end{aligned} \tag{2.6}$$

It is possible to use matrix inversion to solve for the required variables b_0 to b_5. For this purpose the equations can be restated with 3 matrices:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} p_0 \\ v_0 \\ a_0 \\ p_1 \\ v_1 \\ a_1 \end{bmatrix} \tag{2.7}$$

The equation can be rewritten as:

$$AX = B \tag{2.8}$$

The goal is to solve for X. This can be done by multiplying A^{-1} to both sides of the equation:

$$\begin{aligned} A^{-1}AX =& A^{-1}B \\ X =& A^{-1}B \end{aligned} \tag{2.9}$$

This results in the following solution:

$$\begin{aligned} b_0 =& p_0 \\ b_1 =& v_0 \\ b_2 =& 0.5a_0 \\ b_3 =& -10p_0 - 6v_0 - 1.5a_0 + 10p_1 - 4v_1 + 0.5a_1 \\ b_4 =& 15p_0 + 8v_0 + 1.5a_0 - 15p_1 + 7v_1 - a_1 \\ b_5 =& -6p_0 - 3v_0 - 0.5a_0 + 6p_1 - 3v_1 + 0.5a_1 \end{aligned} \tag{2.10}$$

The quintic function 2.5 is now well defined and upholds all criteria required.

2.7 Mixture Density Networks (MDN)

The goal of Bayesian supervised machine learning methods is essentially to model the following distribution:

$$p(t|x) \tag{2.11}$$

This is the conditional density of t, given some input vector x. For regression methods, it is often assumed that the shape of this distribution is Gaussian. Unfortunately, in many cases, this assumption does not hold to be true.

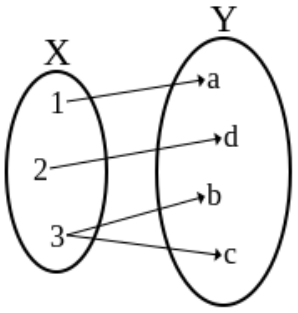

FIGURE 2.12: Example of a multivalued function. A value x from domain X can be associated with more than one value y from domain Y. (Retrieved from `https://en.wikipedia.org/wiki/Multivalued_function#/media/File:Multivalued_function.svg`, accessed on 19.07.2019)

Inverse problems are one particular class, where this assumption often does not hold true and leads to models that strongly diverge from the real underlying distribution. The previously discussed inverse kinematics is an example of an inverse problem. For the forward problem, there is only one solution for each set of input values. For the inverse problem, there are usually multiple solutions for many sets of inputs. This kind of function is also called multivalued, and it is exactly the kind of problem where the assumption of the underlying function being Gaussian can not hold true. Figure 2.12 shows an example of a multivalued function in a diagram.

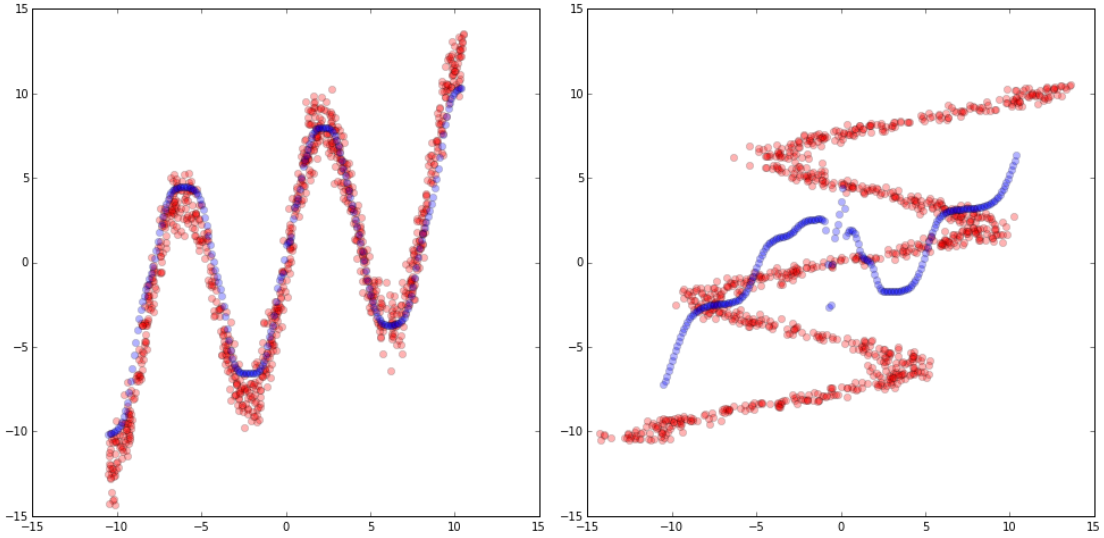

FIGURE 2.13: The blue points are outputs from a neural network trained on the red training data points. Here the X-axis represents the input variables x and the Y-axis represents the output variables t. On the left is the forward problem. On the right is the inverse problem. With more than one possible output for each input, the model is not able to model the underlying function for the inverse problem [Ha, 2015].

Neural network regression is one particular method that fails in modeling inverse functions. When such a network is trained with back-propagation, it is only able to predict the mean value for a given input. When there are multiple solutions for a given input, then the mean value of these solutions is meaningless. This leads to problems whenever the underlying function is multivalued [Bishop, 1994]. Figure 2.13 shows an example of a failed approximation of an inverse problem.

Bishop [1994] propose their Mixture Density Networks (MDNs) as a solution for the problem of regression for multivalued functions. It is able to provide a more complete representation of the underlying distribution.

MDNs are based on mixture models, which are defined by the following equation [Bishop, 2006]:

$$p(t|x) = \sum_{k=1}^{K} \pi_k(x) N(t|\mu_k(x), \sigma_k^2(x)) \tag{2.12}$$

The model consists of k different components. Here, the components were chosen to be Gaussian, as they are a good fit for continuous functions. The mixture coefficient π sums to 1 and describes the likelihood that the output vector t was generated from component k given an input x. Note that π depends on input vector x.

These are the resulting parameters of the mixture model:

$$\begin{aligned} \text{Mixing Coefficients: } & \pi_{\mathrm{k}}(\mathrm{x}) \\ \text{Means: } & \mu_{\mathrm{k}}(\mathrm{x}) \\ \text{Variances: } & \sigma^2_{\mathrm{k}}(\mathrm{x}) \end{aligned} \tag{2.13}$$

In the MDN model, these parameters are learned with a neural network. The components of the mixture model are denoted by K and the components of t by L. In consequence there are K activations $\alpha^{\pi}_{\mathrm{k}}$ that determine the mixing coefficients $\pi_{\mathrm{k}}(\mathrm{x})$, K outputs that determine the kernel width $\sigma_{\mathrm{k}}(\mathrm{x})$ and $\mathrm{L} \times \mathrm{K}$ outputs $\alpha^{\mu}_{\mathrm{kj}}$ that determine the means $\mu_{kj}(x)$. In total this results in $(\mathrm{L}+2) \times \mathrm{K}$ outputs [Bishop, 2006].

There are certain conditions the parameters need to satisfy. The mixing coefficients need to sum to 1 and be positive:

$$\sum_{\mathrm{k}=1}^{\mathrm{K}} \pi_{\mathrm{k}}(\mathrm{x}) = 1 \qquad 0 \leq \pi_{\mathrm{k}}(\mathrm{x}) \leq 1 \tag{2.14}$$

In the network this can be achieved by using softmax outputs:

$$\pi_{\mathrm{k}}(\mathrm{x}) = \frac{\exp(\alpha^{\pi}_{\mathrm{k}})}{\sum_{\mathrm{l}=1}^{\mathrm{K}} \exp(\alpha^{\pi}_{\mathrm{l}})} \tag{2.15}$$

For the variances the following condition has to be met:

$$\sigma^2_{\mathrm{k}}(\mathrm{x}) > 0 \tag{2.16}$$

This can be solved by using the exponentials of the activations:

$$\sigma_{\mathrm{k}}(\mathrm{x}) = \exp(\alpha^{\sigma}_{\mathrm{k}}) \tag{2.17}$$

The means can directly be represented by the activations:

$$\mu_{\mathrm{kj}}(\mathrm{x}) = \alpha^{\mu}_{\mathrm{kj}} \tag{2.18}$$

These parameters are contained in the vector of weights w of the network. The error function is defined by the negative logarithm of the likelihood:

$$\mathrm{E}(\mathrm{w}) = -\sum_{\mathrm{n}=1}^{\mathrm{N}} \ln \left\{ \sum_{\mathrm{k}=1}^{\mathrm{k}} \pi_{\mathrm{k}}(\mathrm{x_n}, \mathrm{w}) \mathcal{N}\big(\mathrm{t_N} | \mu_{\mathrm{k}}(\mathrm{x_n}, \mathrm{w}), \sigma^2_{\mathrm{k}}(\mathrm{x_n}, \mathrm{w})\big) \right\} \tag{2.19}$$

With the derivative of the error function E(w) it is possible to use back-propagation to minimize the function. Because the error function is a summation of the input data points N it is possible to find the derivatives for the single data points and then sum them up [Bishop, 2006].

With the assumption that the mixing coefficients $\pi_k(x)$ are prior probabilities dependent on x the posterior probabilities are given by:

$$\gamma k(t|x) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^{K} \pi_l \mathcal{N}_{nl}}, \text{ with } \mathcal{N}_{nk} = \mathcal{N}(t_n|\mu_k(x_n), \sigma_k^2(x_n)) \tag{2.20}$$

The resulting derivatives with respect to the activations are:

$$\begin{aligned} \frac{\partial E_n}{\partial \alpha_k^\pi} =& \pi_k - \gamma k \\ \frac{\partial E_n}{\partial \alpha_{kl}^\mu} =& \gamma k \left\{ \frac{\mu_{kl} - t_l}{\sigma_k^2} \right\} \\ \frac{\partial E_n}{\partial \alpha_k^\sigma} =& -\gamma k \left\{ \frac{||t - \mu_k||^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right\} \end{aligned} \tag{2.21}$$

Once the network has been trained it is possible to predict a fitting mixture model given an input x. From the resulting parameters it is then possible to sample values.

Figure 2.14 shows that the same distribution that could not be approximated by neural network regression, as seen in figure 2.13, can be approximated with the use of an MDN.

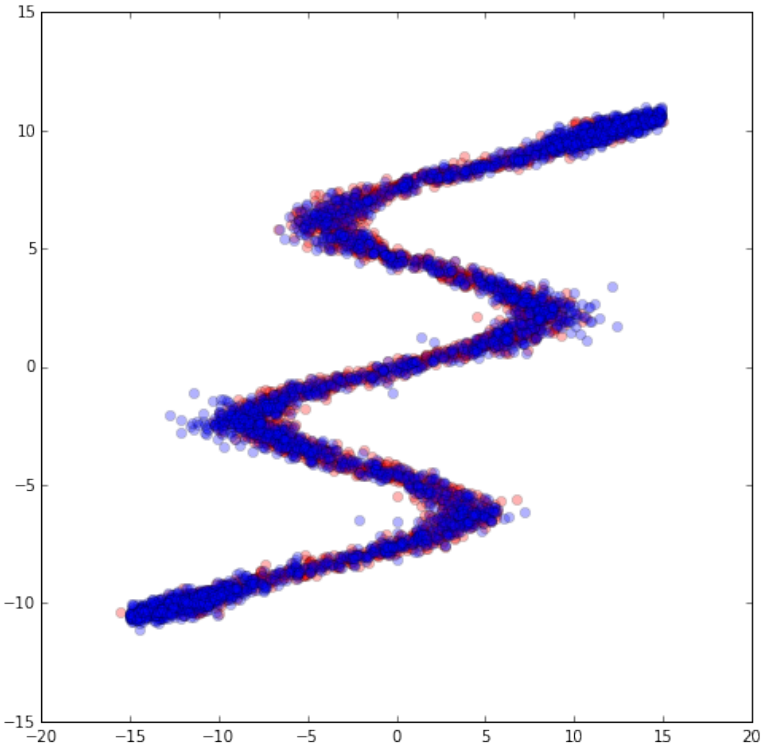

FIGURE 2.14: The blue points are outputs from a mixture density network trained on the red training data points. Here the X-axis represents the input variables x, and the Y-axis represents the output variables t. The points were generated by drawing samples along the X-axis. The probability of drawing from a certain Gaussian was based on its weight assigned by the MDN. This is an example of an inverse problem, that was successfully approximated with an MDN [Ha, 2015].

Chapter 3

Implementation

3.1 The Robot Platform

The platform the implementation is demonstrated on is a PR2 [Willow Garage, 2012] with a Shadow Dexterous Hand [Shadow Robot Company, 2013] mounted on its right arm. The setup can be seen in figure 3.1. The PR2 is a large mobile platform. For this thesis the mobility is not of interest. The left arm is ignored as well and only the Shadow Dexterous Hand is used to play piano.

The Shadow Dexterous Hand resembles the human hand. It has 4 fingers and a thumb. Each finger includes 4 joints, with the little finger adding an extra joint at the hand palm. Tactile sensors are mounted at the tip of the fingers. The sensors are not used in this thesis, but it should be noted that they block the joints beneath the fingertips so that they cannot be moved. The thumb encompasses 5 different joints.

ROS [Quigley et al., 2009] is used as a software framework for the robot.

3.2 Modeling the Piano

To interact with the piano, a model of it is necessary. Figure 3.2 shows the real keyboard and figure 3.3 shows the model. A variety of simple meshes make up the whole model. The largest element is the casing. It was simplified by replacing any slanted surfaces with even ones and leaving out any control elements. It is only needed for collision avoidance. The most important part is the keys. The black keys all use the same mesh. Each white key in an octave has slightly different measurements, which are reflected in the model as well.

The Unified Robot Description Format (URDF, `https://wiki.ros.org/urdf`, accessed on 11.07.2019) was used to create a single model made up of the single meshes from the keys and the casing. This format necessitates the usage of joints

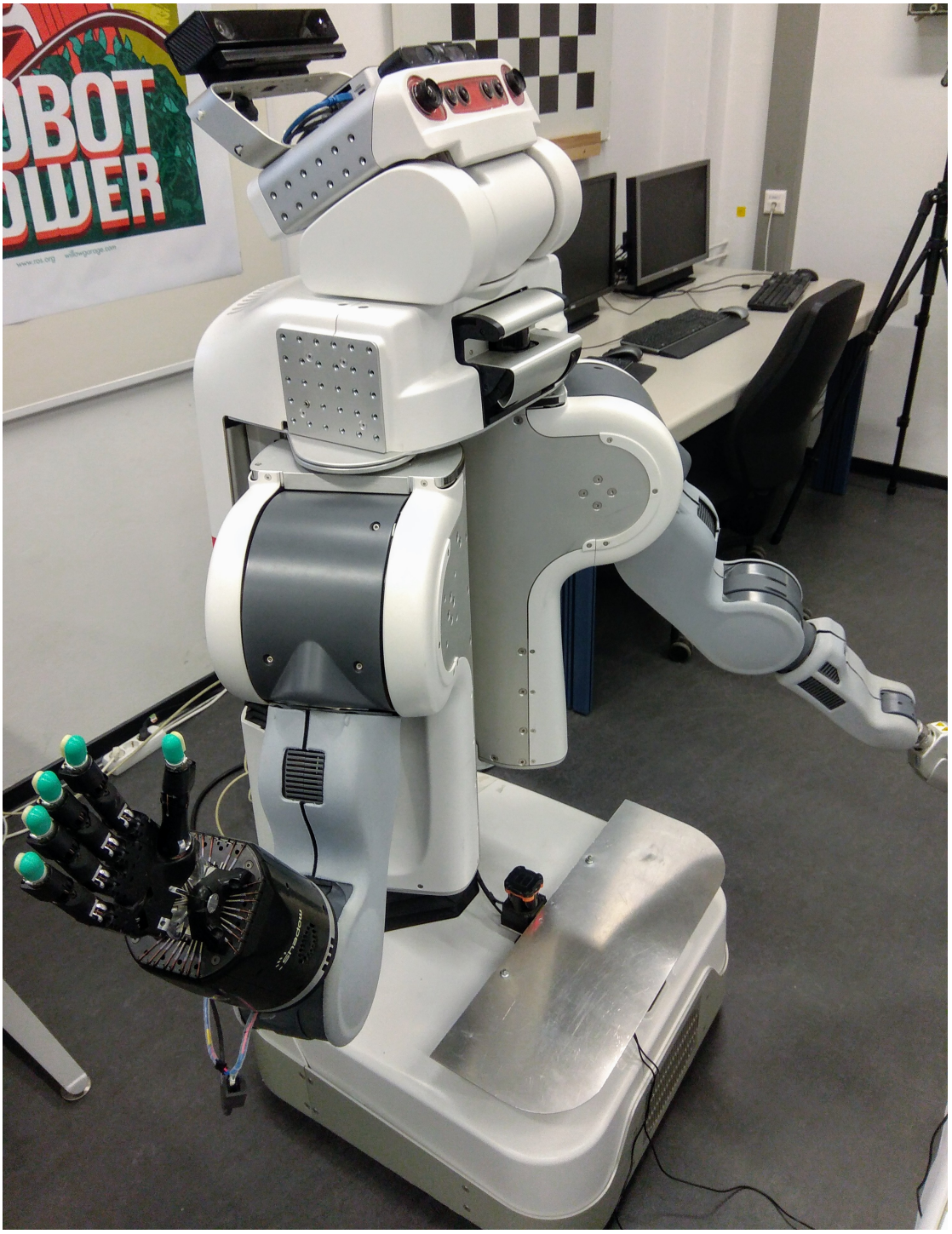

FIGURE 3.1: The PR2 with a Shadow Dexterous Hand mounted on its right arm.

FIGURE 3.2: The keyboard used throughout this thesis, a Roland PC-200 MK II

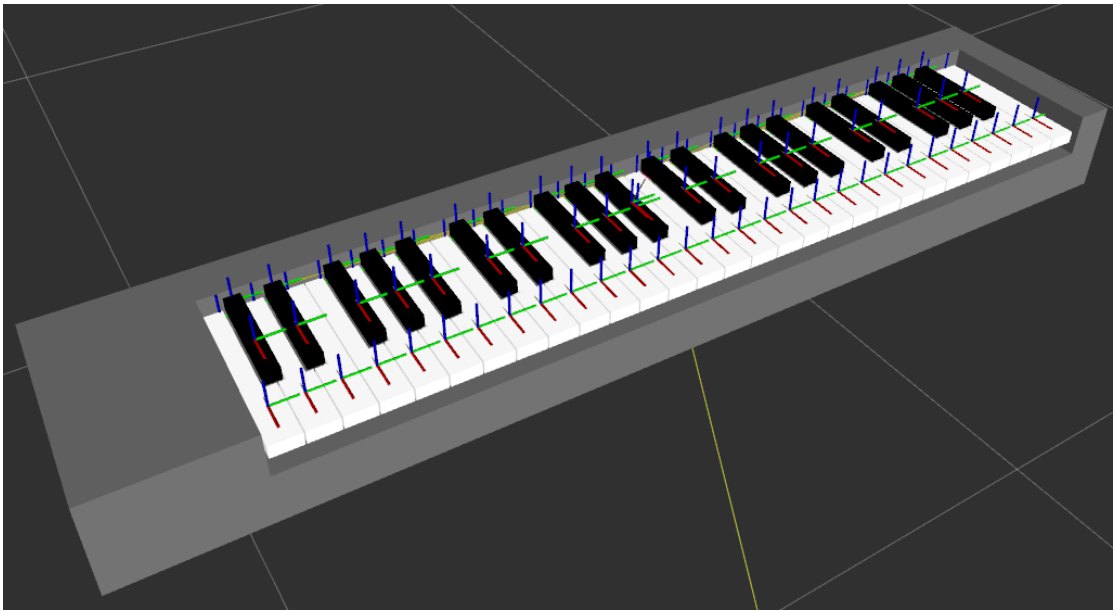

FIGURE 3.3: The model of the keyboard. The casing of the keyboard was only roughly replicated, because it is not essential for playing.

and links to relate the different parts to one another and are later used in the implementation to locate the position of the keys. There is a main joint for the piano in general, that is positioned in the back of the casing. All other joints of the model are connected to this one. There is a frame positioned on the surface of each key so that it is possible to create poses in relation to the key that is supposed to be pressed. Figure 3.4 shows the frames placed on the keys.

In theory, this model could be used in simulation, for example in Gazebo [Koenig and Howard, 2004]. At the end of each key, where the axis of rotation would lie when pressing them down, there is already a joint placed. Moving these joints could be used to simulate moving the keys. Gazebo is able to simulate contacts with the keys and the forces acting upon the keys. It is also possible to add controllers to joints that could be used to simulate a force that keeps the keys upright

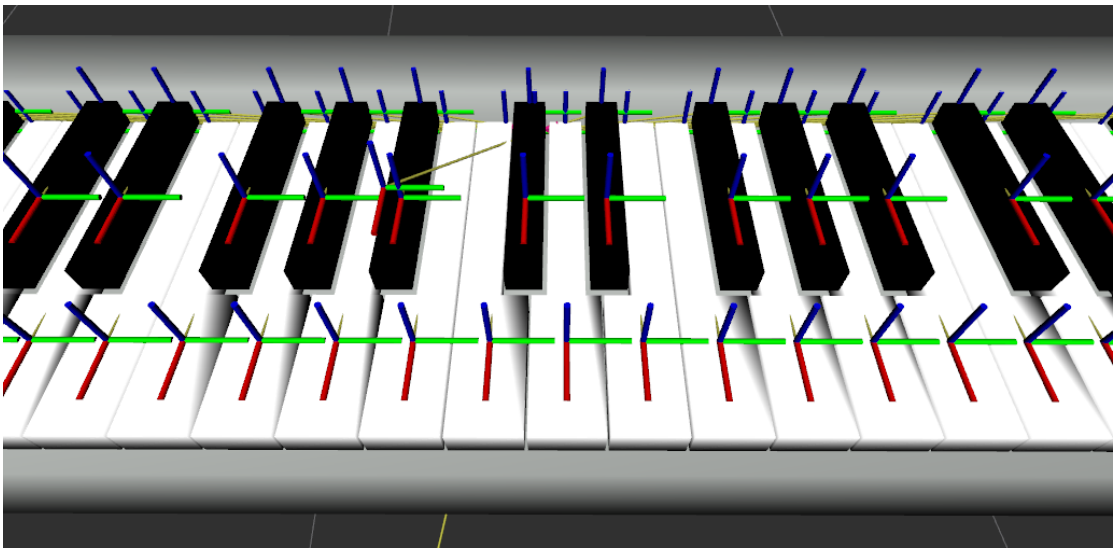

FIGURE 3.4: Each key has a link on its surface, so that poses can be evaluated in relation to the keys.

unless they get pushed down. This could, for example, open up the possibility to use simulation for faster and easier data collection.

3.3 Localizing the Piano

For the robot to press any keys, the position of the piano needs to be known in relation to the robot. This can be achieved by localizing the model of the previous section.

There are numerous approaches to solve this problem. One obvious choice would be a visual localization. The regular and distinct pattern of the keys lends itself for this kind of solution. However, it would require fine-tuning the kinematic chain from the camera to the fingers. This can be avoided by choosing a physical approach instead.

The pose of each finger is known in relation to the base by using the robot's kinematic chain. The ROS package tf (`http://wiki.ros.org/tf`, accessed on 11.07.2019) can be used to compute the finger's positions. It is possible to exploit that information for localization. The PR2 can be switched into mannequin mode (http://wiki.ros.org/pr2_mannequin_mode, accessed on 02.06.2019), which allows the user to move the arms around directly. In this mode, the finger can be placed to press down a key. Figure 3.5 shows an image of the process. Since the pose of the finger is known, and the MIDI signal carries the information, which key was pressed, the position of the said key can be approximately inferred. To determine the orientation of the whole piano, the position of a second key needs to be known. With these two positions known, trigonometry can be used to fully localize the whole piano with the simple assumption that it stands upright.

FIGURE 3.5: The process to localize the keyboard. The arm gets moved to put the finger above the key and press it down.

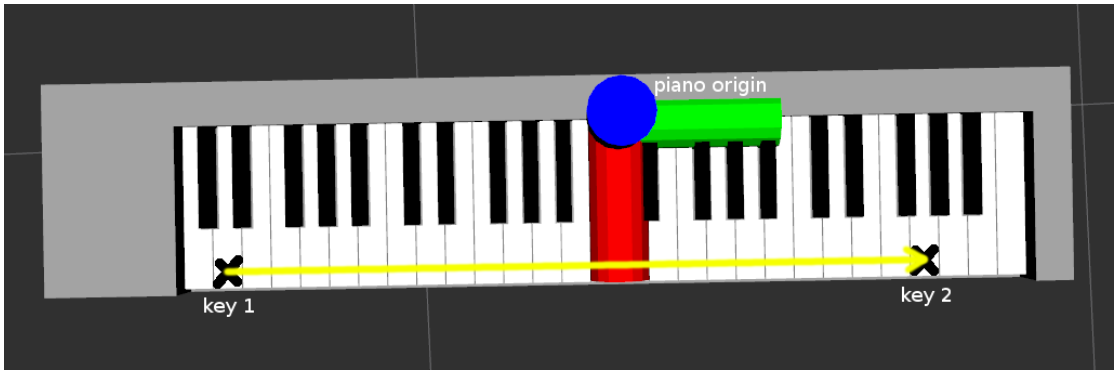

FIGURE 3.6: The position of key 1 and key 2 is roughly known. The orientation can be inferred by creating a vector of the difference of these two positions. The yellow arrow represents the direction of this vector. The Y-axis of the origin of the piano points in the same direction as this vector. Knowing the direction of the Y-axis of the frame it is possible to infer the direction of the missing two axes, assuming the keyboard is placed upright.

First, we want to compute the position of the origin of the keyboard, which is shown in figure 3.6. The position of two keys is known. In the model, the distance from these keys to the origin is known as well. Using the distances from the keys to the origin in the model as well as the distance between the recorded positions of the keys, it is possible to form a simple triangle.

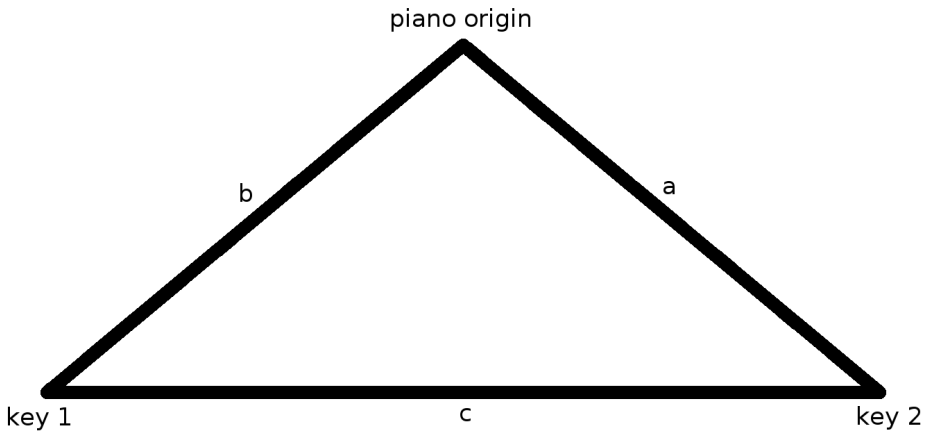

FIGURE 3.7: The triangle that can be created from the two known positions of the keys and the origin of the piano. a, b and c represent the length of the sides of the triangle.

Figure 3.7 shows the resulting triangle. The length a, b and c were determined and can now be used to compute the position of the origin in relation to the position of the first key.

$$\begin{aligned} \mathrm{C_x} &= \frac{\mathrm{b}^2 + \mathrm{c}^2 - \mathrm{a}^2}{2 * \mathrm{c}} \\ \mathrm{C_y} &= \sqrt{\mathrm{b}^2 - \mathrm{x}^2_{\mathrm{origin}}} \end{aligned} \tag{3.1}$$

Because equation 3.1 only uses the lengths of the triangles sides it disregards the actual coordinates and the real orientation of the triangle. For this reason the result needs to be rotated and then added to the position of $key1$:

$$\begin{aligned} \mathrm{keydiff} &= \mathrm{key2} - \mathrm{key1} \\ \theta &= -\operatorname{atan2}(\mathrm{keydiff_x}, -\mathrm{keydiff_y}) \\ \mathrm{piano\_origin} &= \mathrm{key1} + \begin{pmatrix} \cos(\theta) & -sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} * \mathrm{C} \end{aligned} \tag{3.2}$$

C gets rotated by the angle θ so that it is in the correct position in relation to $key1$. Adding $key1$ results in the final position of the origin of the piano.

To determine the full pose of the piano, the orientation still needs to be computed. Figure 3.6 shows the frame of the origin of the piano we want to determine the orientation of. The position of two keys is known. The difference between these two keys results in the yellow vector shown in figure 3.6. This vector points in the same direction as the Y-axis of the frame of the origin, which is represented by the green axis. If we assume the keyboard is placed upright in the real world, we can just assume the Z-axis to be $(0\ 0\ 1)$, so that the Z-axis points straight up. The direction of the X-axis can be calculated by rotating Y-axis by -90 degrees around the Z-axis.

With a vector for the direction of each of the axes, it is possible to build a rotation matrix, which describes the full orientation of the frame of the origin.

3.4 The First Prototype

With the localized model of the keyboard, it is now possible to specify finger poses that make it possible to press the keys. For the first prototype, it was opted to start as simple as possible. For that reason, only the index finger was used, as shown in figure 3.8.

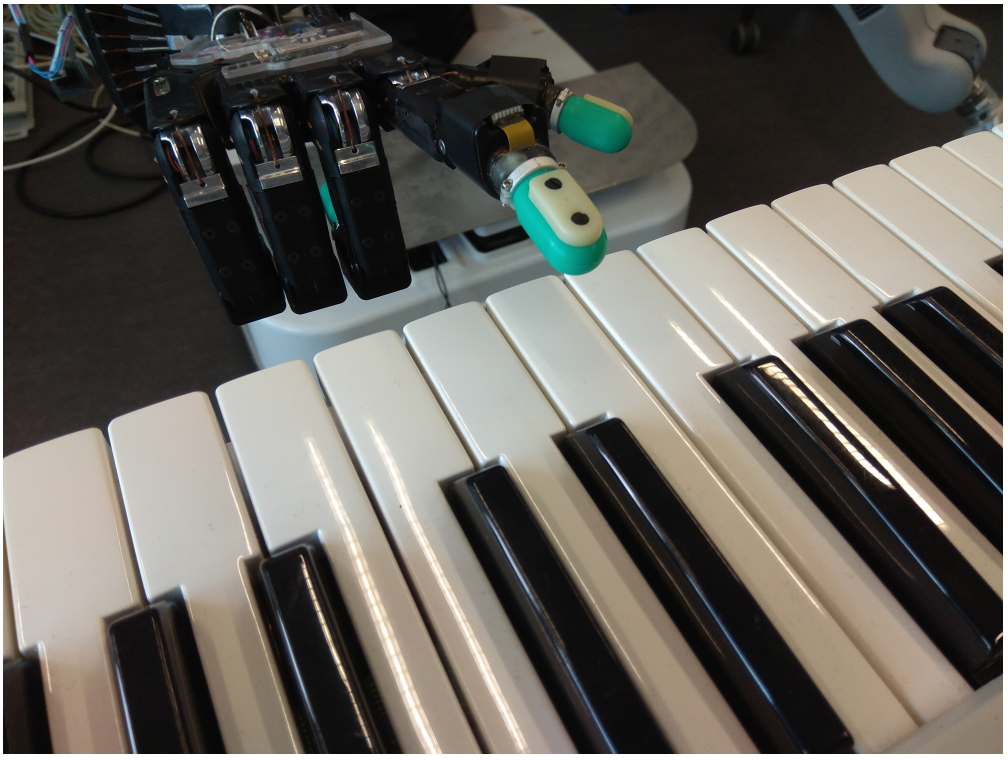

FIGURE 3.8: Only the index finger was used in the first prototype.

In order to press the keys, the task was split into different phases. In the first phase, the index finger is placed above the key. In the next phase, the finger is lowered in order to press the key. In the last phase, the finger is lifted again, and the finger can be placed above the next key afterward.

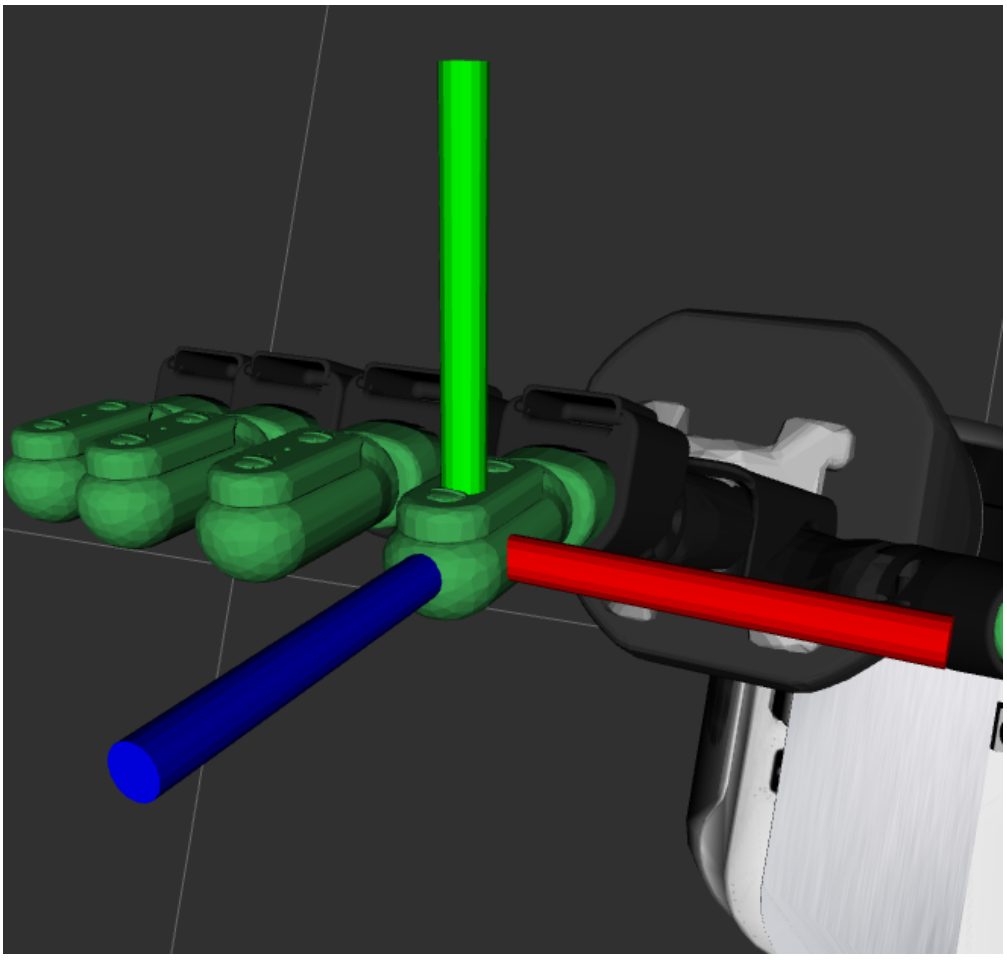

FIGURE 3.9: Frame of the index finger used to specify targets in bio_ik.

Inverse kinematics is used to achieve this, in particular, bio_ik [Ruppel, 2017]. Given a target pose for a chosen frame of the robot, bio_ik calculates the necessary joint angles to reach this pose. Figure 3.9 shows the specific frame used for the index finger.

In general, the hand starts in a specific configuration, with the index finger extended and the other fingers are retracted.

In the first phase the goal was chosen as follows:

```
PoseStamped goal;
goal.header.frame_id = key_frame;
goal.pose.orientation.x = 0.5;
goal.pose.orientation.y = -0.5;
goal.pose.orientation.z = -0.5;
goal.pose.orientation.w = 0.5;
goal.pose.position.x = 0.03;
goal.pose.position.y = 0.0;
goal.pose.position.z = 0.03;
```

The *frame_id* represents the frame in which the pose is defined, here a frame on the surface of the particular key. An example of such a frame can be seen in figure 3.4. The orientation is represented by a quaternion and was chosen so that the finger is placed parallel to the key. The position was chosen to be 3 centimeters above the key and shifted by 3 centimeters along the X-axis.

For the phase of pressing the key, only the finger itself is moved while the rest of the arm and hand is fixed in place. Here only the position is used and no orientation. A specific goal type was created, that accepts solutions in a certain range around the given position on the X-axis. Figure 3.4 shows that the X-axis runs along the longer side of the key. Hence, the range was chosen so that the solution still results in a successful press.

In order to lift the finger in the last phase, the same goal type as in the previous phase gets used, but now with the goal being 3 centimeters above the key.

The same method can be applied using the other fingers as well, except for the thumb. In that case, the hand starts with all fingers extended, and the goals need to be defined for the frame of the particular finger that is supposed to press the key. When done sequentially, it is then possible to press one key after the other while varying which finger presses the key.

After each bio_ik solution, a trajectory is used to move the joints of the arm and hand to the given angles. This means the robot moves and then stops waiting for the inverse kinematics solutions and trajectory planner to finish, resulting in detached motions. Ideally, for piano playing, we want the arm and finger movements to be as quick and smooth as possible.

3.5 Smooth and Predictable Motions

3.5.1 Point-to-Point Trajectory Planner

The goal of a trajectory planner is to connect a start and goal configuration with a path in between. RRT-Connect [Kuffner and LaValle, 2000] is a commonly used trajectory planner. It is very well suited to find solutions, even in cluttered environments. In order to plan the trajectory, it uses randomly generated configurations which get linked to one another. Each newly generated configuration gets collision checked. Then it is connected to the closest existing configuration created

in a previous iteration. The path between both the connected configurations is once again checked for collisions and only allowed if none are found.

The factor of randomness inherent to the algorithm can result in unpredictable and sometimes suboptimal behavior for the purpose of playing the piano. For our purposes, we want a planner that is able to move from the start to the end configuration as quickly as possible in a predictable manner.

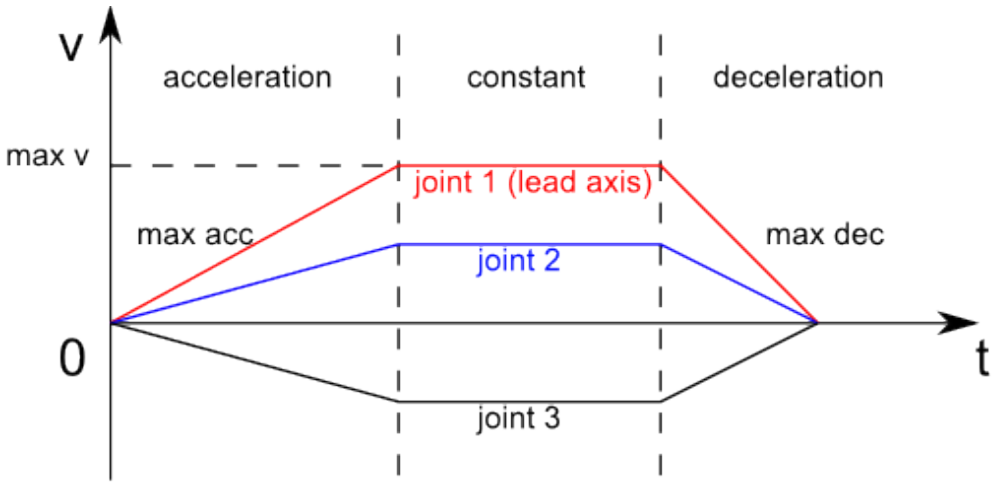

FIGURE 3.10: The PTP planner is divided into the phases of acceleration, constant velocity and deceleration. Each of the joints movements get synchronized according to the slowest joint.

The point-to-point (PTP) planner from the pilz_industrial_motion package (`https://github.com/PilzDE/pilz_industrial_motion/`, accessed on 12.07.2019) fulfills such criteria. It connects the start and end configuration directly instead of spawning random configurations in-between. The advantage here is the predictability as well as the guaranty that the fastest way is chosen. The apparent drawback, on the other hand, is the inability to deal with any obstacles in the way between the configurations and in that case, the arm will not move. In our case, the hand is above the keyboard and moved from key to key, without any obstacles in-between.

Figure 3.10 shows the three different phases of the generated trajectories. Each of the joints has a configured maximum velocity and maximum acceleration. In the first phase, the maximum acceleration is used to reach the maximum velocity for the leading joint, meaning the joint that takes longest amount of time to reach the goal. Afterward, the velocity is held constant until the third phase approaches when the movement of the leading joint is stopped with the maximum deceleration. All other joints get synchronized so that their phases start and end at the same time as the phases of the leading joint.

3.5.2 Trajectory Blending

Another issue is the pauses to plan the trajectories and the stopping of the robot between each of the trajectories. It would be desirable to be able to plan all trajectories beforehand and execute them afterward with smooth motions to connecting the trajectories. One possible solution to this problem is trajectory blending.

In trajectory blending the trajectories are not merely connected, but instead blended into each other, so that the movement does not stop. The pilz_industrial_motion package (`https://github.com/PilzDE/pilz_industrial_motion/`, accessed 12.07.2019) was used for this purpose as well.

To achieve the blending effect, a connecting trajectory has to be created to move from one trajectory to the other. So given trajectory x_1 and trajectory x_2 we want to create a connecting trajectory x.

The connecting trajectory is created by the following equation [Lloyd and Hayward, 1993]:

$$x(s) = x_1(s) + \alpha(s)(x_2(s) - x_1(s)) \tag{3.3}$$

Here s is a normalized time parameter which is 0 at the start of the blending process and 1 at the end of the blending process. $\alpha(s)$ is a polynomial function chosen, so that boundary conditions are met [Lloyd and Hayward, 1993]:

$$\alpha(s) = 6s^5 - 15s^4 + 10s^3 \tag{3.4}$$

It ensures a smooth transition between the two trajectories.

At $s = 0$ only trajectory x_1 is used. With progressing time $\alpha(s)$ gives the term $x_2(s) - x_1(s)$ more relevance and so slowly trajectory x_2 contributes more and more to create trajectory x. At the end with $s = 1$ it follows that $x(s) = x_2(s)$.

In order to choose when to start the transition from one trajectory to the next, a blending radius is used, as shown in figure 3.11. The radius is placed at the point where the two trajectories meet, and the blending is started where the first trajectory enters the radius and ends when the second trajectory leaves the radius.

The blending can be used to connect all trajectories and especially to generate smooth motions between lifting the finger, moving to the next key to be pressed and pressing the next key.

The result of using the PTP planner as well as blending trajectories are fast and predictable movements that do not stop for any additional planning in-between. Another important factor for playing the piano is the timing. Playing a piece of music necessitates hitting the key at a certain point in time. With the methods presented so far, this is not possible. A practical solution to this problem is trajectory scaling.

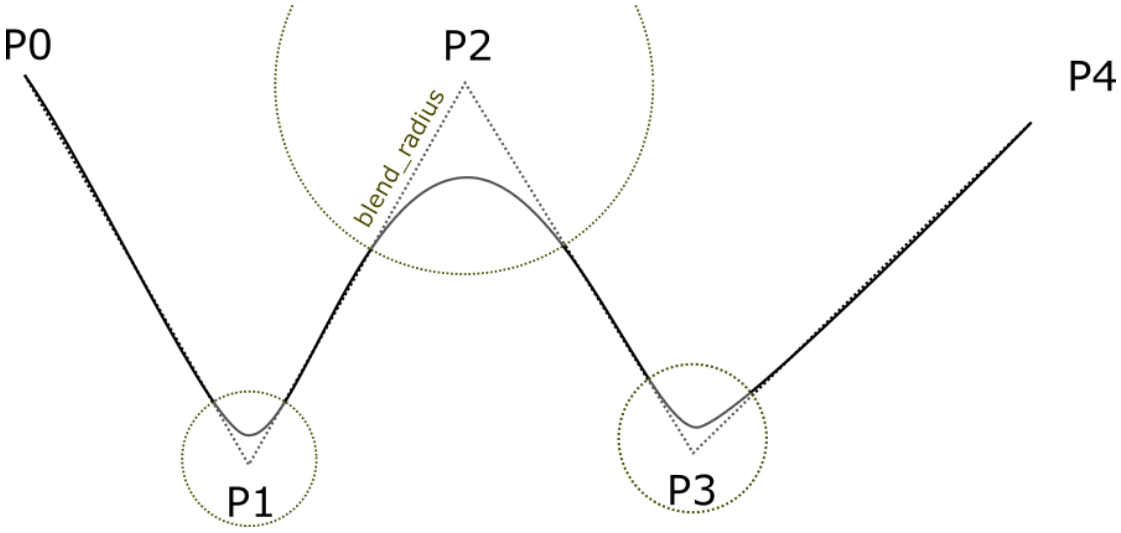

FIGURE 3.11: The blend radius is set where the two trajectories would've met. The blending is started where the first trajectory enters the radius and ends where the second trajectory leaves the radius. In-between the blending trajectory from equation 3.3 is used instead.

3.6 Timing

As explained in section 2.1 timing is an important aspect of playing a piece of music. To achieve that the keys are pressed at a certain point in time it was opted to use trajectory scaling.

The important aspect is the time needed between hitting two consecutive keys. According to this trajectory, segments are created, which consist of lifting the finger off the last key, moving the arm to the next key and lowering the finger on the next key. These segments need to be scaled according to the length of that note.

To describe the length of the note, a variable called note_divisor is used, which is 1 for a whole note, 2 for a half note, 4 for a quarter note and so on. The planning is done at maximum speed, so that scaling should always only result in slower executions. In order to determine the highest possible BPM, we need to figure out what the shortest possible value for the whole note is. This is determined by the following formula:

$$\arg\max \text{whole\_note}(\text{segment}) = \text{duration}(\text{segment}) \cdot \text{note\_divisor}(\text{segment}) \quad (3.5)$$

The desired value is the lowest possible value for the whole note, while all segments can still be scaled accordingly. For each segment, the duration is known, as well as the note_divisor. The multiplication of the duration of a segment with its note_divisor is the length it would have if it would be played as a whole note. Thus, the largest value found for the whole note among all segments is the lowest possible value for the whole note. Then the segment that resulted in the highest

possible value does not need to be scaled at all, while all other segments need to be scaled to be executed with lower speeds.

Now all segments have to be scaled in relation to the calculated value of the whole note. The positions along the trajectory will remain the same, but to slow down the movement a number of parameters need to be adjusted in the entire segment. Each waypoint has the parameter *time_from_start* which describes at what point in time after starting the execution of the trajectory this particular waypoint should be reached. In consequence the velocities and acceleration values need to be adjusted for the waypoints as well. This is done according to the following formulas:

$$\begin{aligned} \text{scale\_factor(segment)} &= \frac{\text{duration(segment)}}{\frac{\text{whole\_note}}{\text{note\_divisor}}} \\ \text{waypoint.time\_from\_start} &= \frac{\text{waypoint.time\_from\_start}}{\text{scale\_factor(segment)}} \\ \text{waypoint.velocity} &= \text{waypoint.velocity} \cdot \text{scale\_factor(segment)} \\ \text{waypoint.acceleration} &= \text{waypoint.acceleration} \cdot \text{scale\_factor(segment)}^2 \end{aligned} \tag{3.6}$$

So far only the existing waypoints get adjusted. This can result in a low density of waypoints. For example, a short trajectory with only a couple of waypoints gets scaled to over 10 seconds. This can result in less than 1 waypoint per second. As a countermeasure, new points need to be added. This can be achieved by interpolating between the already existing points after they were scaled. Quintic spline interpolation is used for this purpose. Since each waypoint consists of position, velocity and acceleration, quintic splines have to be used in order to represent all dimensions. The implementation from the *joint_trajectory_controller* package is used in this thesis (`http://wiki.ros.org/joint_trajectory_controller`, accessed on 21.08.2019). To keep a ensure a certain density of waypoints, new points are created to fill the gaps in an interval of 0.1 second when the distance between waypoints is higher than 0.1 seconds.

The result are trajectories that hit the keys at roughly the right point in time. In summary, it is now possible to execute all trajectories without stopping in-between and hit the keys at the right point in time. With that setup, it is not possible to control how the keys are pressed since that movement is created with a trajectory planner as well. Possibilities to control the behavior of the fingers is investigated in the following section.

3.7 Pressing Keys

3.7.1 Recording Training Data

As discussed in section 2.2 we can use the MIDI data as feedback. The MIDI velocity is the focus in particular in this section. The velocity is directly related

to how loud the sound is played. Pressing lighter results in a lower velocity, which results in a quieter sound. Pressing firmer, on the other hand, results in a higher velocity and in a louder sound.

The goal is to investigate the range of velocity that can be achieved with the Shadow Dexterous Hand and how it is possible to target a certain velocity when pressing. It was chosen not to use an analytical approach. This was done due to the high complexity of the dynamics of the tendon-driven system of the hand and the unknown transfer from a press to the MIDI velocity. Instead, a data-driven approach was chosen. In the first step, training data was recorded. Instead of creating a whole trajectory for moving over the key and pressing the key, the pressing was isolated. For this purpose, the trajectory interface of the Shadow Dexterous Hand was used directly. The interface uses the JointTrajectory message type, which is defined as follows:

```
Header header
string[] joint_names
JointTrajectoryPoint[] points
```

Here the points contain the following fields:

```
float64[] positions
float64[] velocities
float64[] accelerations
float64[] effort
duration time_from_start
```

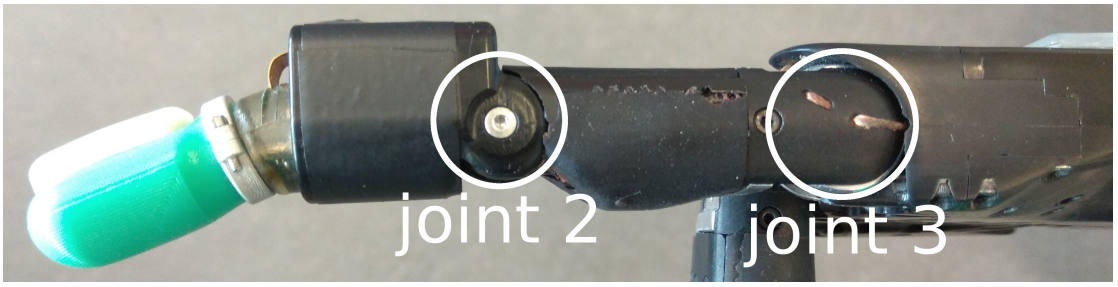

FIGURE 3.12: The joints used to record training for pressing keys, excluding the thumb.

Figure 3.12 shows the relevant joints used to press down keys. For our particular Shadow Hand, Joint 1 can not be moved, because it is blocked by the BioTac sensors. For the setup for collecting training data, the arm is moved to put the relevant finger over a key. Then the trajectory interface of the hand is used to send a command to move the finger containing values for the two joint shown in figure 3.12 and a value for time_from_start. Thus, the trajectory of the finger only consists of a single waypoint. The values for the joints and the time_from_start

are varied for each attempt to press the keys. The arm can either be moved to a different key between press attempts or the finger simply stays over the current key for each attempt. An example of an attempt looked like this:

```
joint_names = {rh_FFJ2, rh_FFJ3}
points[0].positions = {0.75, 0.3}
points[0].time_from_start = 0.8
```

This message contains positions for joint 2 and 3 for the index finger. The positions were randomly chosen between 0 degrees, which corresponds to an extended finger, and 90 degrees. The time_from_start parameter was in the range of 0.2 and 1.75 seconds.

For each attempt the following values are recorded:

```
velocity
duration_to_signal
```

The variable duration_to_signal records the time it takes from sending the trajectory until the MIDI feedback arrives. This can give us an idea, given certain parameters, how long it takes until the key actually gets hit.

3.7.2 Velocity Data Discussion

Figure 3.13 shows the plot of the recorded training data for the velocity. The values for joint 2 and joint 3 were summed up in order to create a more comprehensive plot.

The range for the velocity could not be covered to its full extent. As discussed in section 2.2 the full range spans from 0 to 127. The Shadow Hand could barely hit values above 70. There are clusters around 0 and 1. A velocity of 0 simply means the key was not reached at all, so the joint angles were just too low. A velocity of 1 was facilitated by very slow presses. Starting at a certain slowness, the resulting value will always be 1. That explains the large cluster since it encompasses a much broader range than any other value. The range between 1 and roughly 30, on the other hand, is only populated sparsely. Values in that range were hard to achieve. The range from roughly 45 up to 65 was hit very often.

For the joint angles, higher degrees are coupled with higher velocities. A velocity of 1 can only be reached with summed joint angles up to roughly 1.9. Velocities between 1 and 30 are exclusively hit in that same range as well. The higher the angles, the thinner the spread of hit velocities. The reason for the rise in velocity with higher angles is that the targets for the fingers often lie beyond the keys so that when the finger hits the key, it tries to move further and thus still has a high velocity and sometimes acceleration.

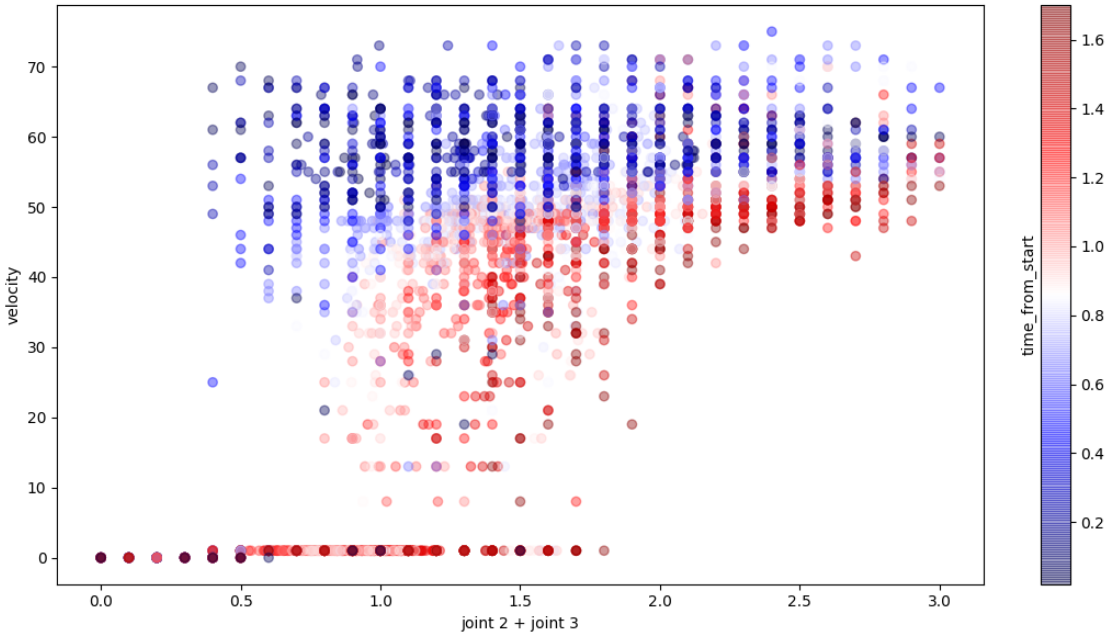

FIGURE 3.13: The plot shows the relation between the different parameters for the trajectory and the resulting velocity. The Y-axis shows the resulting velocity, the X-axis is a summation of joint 2 and joint 3 and the color coding describes the value of the time_from_start parameter.

The most important factor in controlling the velocity is the time_from_start parameter. A lower value leads to a higher velocity because there is less time for the finger to reach the given target, meaning the finger moves with higher velocities and accelerations.

The goal is to be able to generate the parameters used to lower the finger on the key for a given velocity. Unfortunately, the function from the velocity as an input to the parameters is a multivalued function, which complicates the modeling of the underlying function.

3.7.3 Velocity to Press Parameters Prediction

In section 2.7 multivalued functions were introduced, with MDNs as a means to model these kinds of functions. A different applicable solution would be to search for parameters that fulfill a secondary criterion so that the associations from input to output become single-valued. For example, searching for the parameters for a certain velocity with the lowest possible time_from_start. However, here it was opted to use an MDN to model the entire inverse function instead.

TensorFlow [Abadi et al., 2015] with Keras [Chollet et al., 2015] were used as a framework for neural networks in general. Unfortunately, MDNs are not part of these frameworks by default. For this reason, the MDN layer for Keras provided by Martin [2018] was used additionally.

An MDN encompasses the following parameters:

```
Network Parameters:
Number of Hidden Neurons
Number of Gaussians
```

For the input of the network, we only use the velocity:

```
Network Input:
Velocity
```

The output of the network is not directly the predicted parameters for the press, but instead the parameters of the Gaussian mixture model used to model the underlying probability distribution:

```
Network Output for each Gaussian:
3 means (μ_i)
3 variances (σ_i)
1 weight (π_i)
```

There are 3 means μ_i and variances σ_i due to the 3 dimensionality of the target values we want to predict, namely the positions of joint 2, joint 3 and the time_from_start. The weight π_i indicates how likely it is that the particular input velocity is modeled by this particular Gaussian.

The following is the final output sampled from the Gaussian mixture model:

```
Output Sampled From Gaussian Mixture Model:
position of joint 2
position of joint 3
time_from_start
```

The first step to sample any values is to choose one of the Gaussian distributions. For a result with a high confidence, a Gaussian associated with a high weight π_i should be chosen. For a broader range of solutions, a lower threshold should be chosen to take more Gaussian distributions into consideration. Then the associated means μ_i and variances σ_i are used to sample from the chosen Gaussian. Once again, a solution with a high confidence can be chosen, by using the mean value for the input velocity or a broader range of solutions can be considered by taking the variances into consideration.

Figure 3.14 shows a plot of the predictions of the mixture density network using the velocities from the training data as input. The parameters for the sampling were chosen so that a broad range of solutions was generated. The resulting plot is similar to the plot with the original data in figure 3.13.

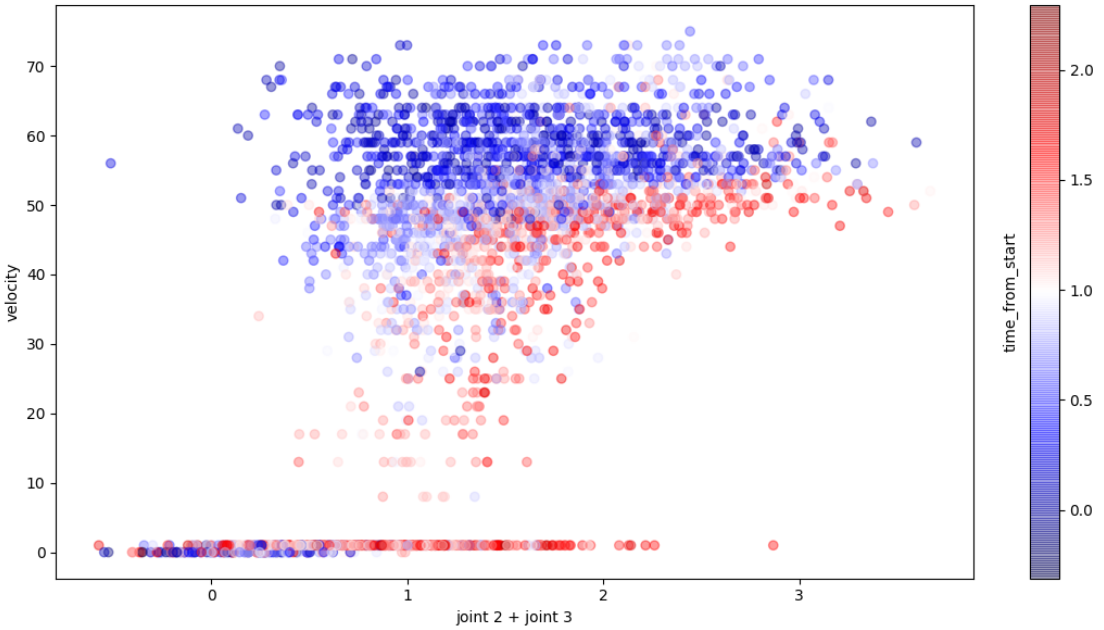

FIGURE 3.14: The data in the plot was generated by the trained mixture density network with the velocities from the original training data as input.

The result is a more flexible approach to press the keys, with more control over how loud the piano is being played. But as a side effect, the timing got less predictable because the time it takes for the finger to reach the key can differ depending on the parameters used for the press.

3.7.4 Duration to Hit Data

As discussed in section 3.7.1, not only the velocity was recorded, but also the duration until the key was hit and the MIDI data was sent.

Figure 3.15 visualizes the resulting data. A lower time_from_start always results in a lower duration, because it controls how long the finger should take to reach the given target angles. The joint angles determine the spread of the durations. Lower joint angles result in a higher spread, while high joint angles generally result in very low durations. For high joint angles, the finger needs to move quicker to reach the given target in the given time, so that the duration, until it hits the key, is lower as well.

3.7.5 Parmeters to Duration Prediction

The found data can be used to predict the duration until a key is hit given the parameters time_from_start and the positions of joints 2 and 3. Each set of parameters has one specific duration associated with it. In contrast to the previous

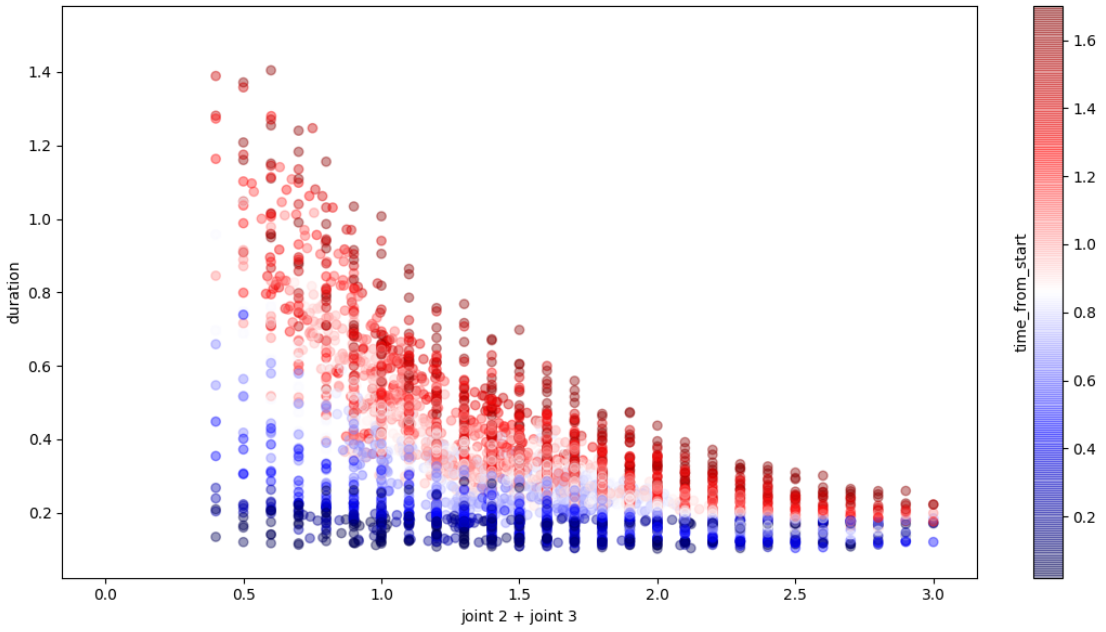

FIGURE 3.15: The Y-axis shows the duration from sending the trajectory until the key was hit. The positions of joint 2 and joint 3 are summed up on the X-axis, for the sake of comprehension. The time_from_start parameter is color coded.

problem for predicting the parameters from a given velocity, the underlying function is not multivalued, and common regression methods can be used.

For this problem, a simple neural network regression method was chosen. The network consists of two hidden layers with 64 neurons with rectified linear unit (ReLU) activation functions [Nair and Hinton, 2010].

The input is as follows:

```
Network Input:
time_from_start
position joint 2
position joint 3
```

And it predicts the following value:

```
Network Output:
duration
```

Figures 3.16 and 3.17 show predicted values from the network. The predicted values mirror the behavior of the recorded training data.

With the use of the two networks, a pipeline can be created to press the keys with a certain firmness with estimated timing.

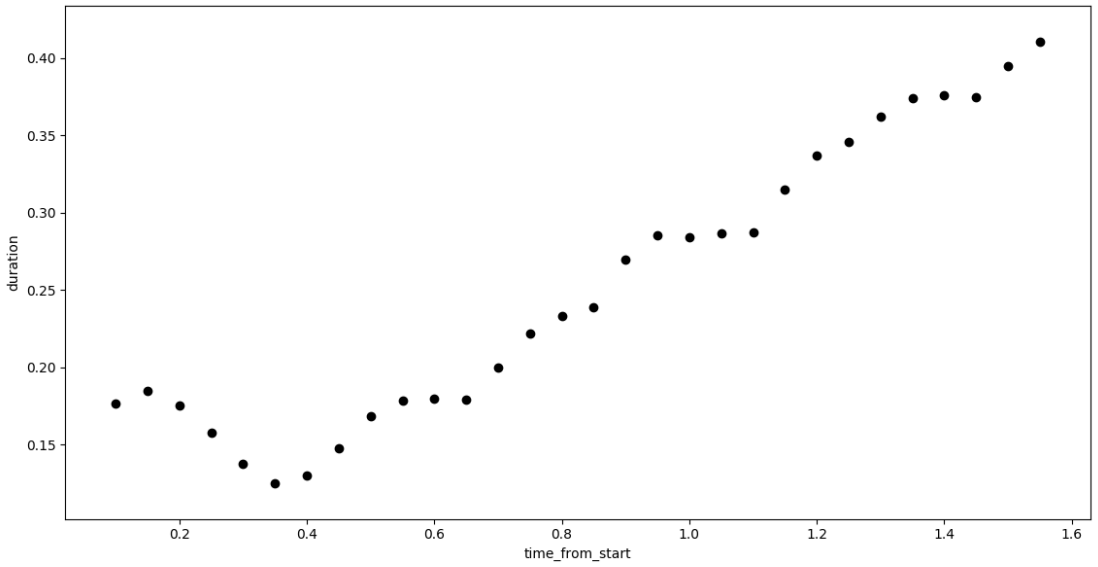

FIGURE 3.16: Plot of the predicted duration with static joint positions. The predicted duration rises with higher values for time_from_start.

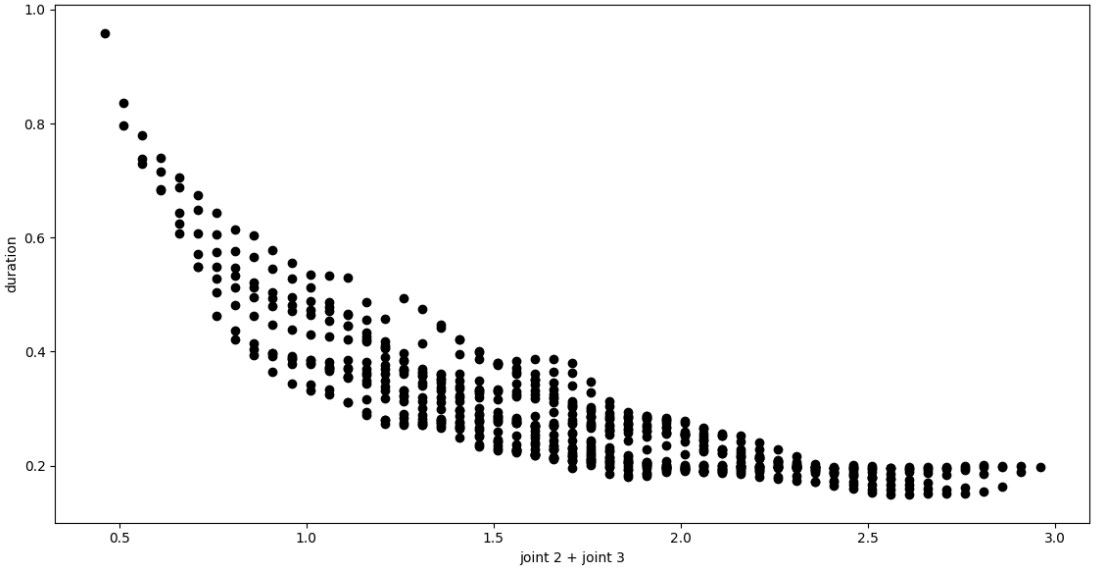

FIGURE 3.17: Plot of the predicted duration with static time_from_start. Larger joint positions result in the predicted duration being lower.

3.7.6 Pressing with the Thumb

The thumb has a different structure compared the other four fingers on the Shadow Dexterous Hand. In light of this difference, the approach to pressing the keys was slightly adjusted. Instead of using two joints to press down the keys, only one joint is used. For the thumb, in particular, the relevant joint is joint 4. Otherwise, the same methods are used to predict the parameters from a given velocity and predict the duration until the key is pressed.

Due to these circumstances, the resulting velocities are lower than they are for the rest of the fingers.

So far, the pressing itself has been the focus. Another important aspect is the kinematics of the hand, meaning moving the arm, hand and fingers so that the requested keys can be reached.

3.8 Hand Kinematics

The movement of the hand while playing piano is an important factor. This section revolves around the kinematics of the hand regarding the reaching of keys on the piano. In general, the hand and its fingers should be positioned above the keys, so that only small movements are needed when going from one target to the next.

3.8.1 The Thumb

As discussed in section 3.7.6, it is possible to press keys with the thumb as well. However, the thumb is situated differently than the rest of the fingers. It is shorter so that when the other fingers are extended and hovering above the keys, the thumb is still far away from the keys. In consequence, the hand needs to be repositioned any time the thumb is supposed to press any key.

A different finger posture is needed, so that all fingers, including the thumb, can be positioned above the keys at the same time.

Figure 3.18 shows the alternative finger posture. With the use of this posture, it is possible to press a key with the thumb and use any of the other fingers to press a different key without much movement in between. The pressing can be learned just as it was for extended fingers.

Many piano pieces require either passing the other fingers over the thumb or passing other fingers under the thumb. Figure 3.19 shows an attempt to get the index finger into a pose where it can pass over the thumb. Here the joints of the fingers are already at their limits, and the index still does not fully reach the key next to the thumb. So, due to the kinematic limits of the hand, this technique was not implemented.

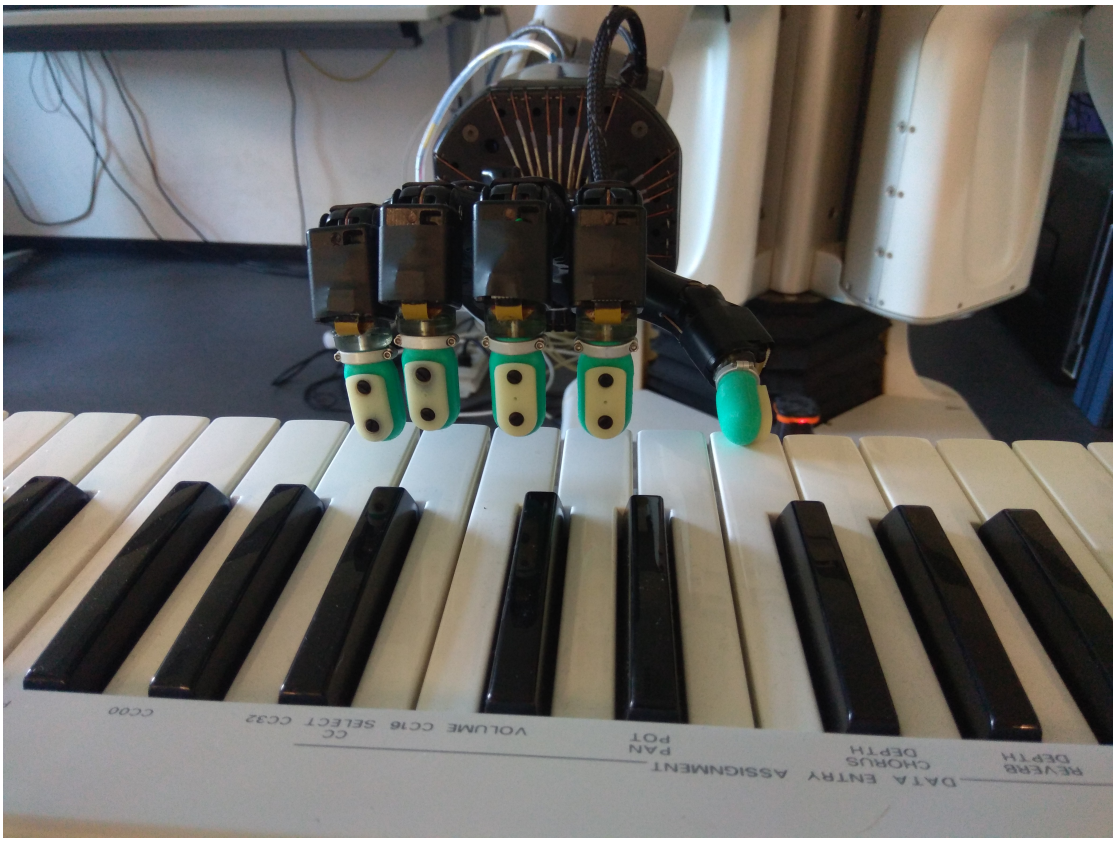

FIGURE 3.18: The alternative finger posture. It is used to get all fingers into a position where they hover above the keys, ready to press them down with small movements from one key to the next.

3.8.2 Pressing Black Keys

The black keys on the piano are positioned differently from the white ones. As with the thumb, longer times for repositioning the fingers above the black keys would be needed in case all fingers are extended and placed above the row of white keys. Once again, the alternative angled finger posture can be used to angle the fingers used to play the white keys and leave the fingers supposed to play the black keys extended. The extended fingers can easily be placed above the black row of keys, while the angled fingers sit above the row of white keys. Hence, no repositioning is needed, and playing keys consecutively becomes faster.

3.8.3 Staccato and Legato

Staccato and legato describe different styles of playing musical notes. Staccato describes playing the notes in a detached manner and legato means playing notes in a smooth manner. For more details see section 2.1.

FIGURE 3.19: Example of passing the Shadow Hand's index finger over the thumb. Here the joints of both finger are at their limits, and still, the index finger can not fully reach the key next to the thumb.

In order to model the difference between the styles in our pipeline, it is necessary to make it possible to control how long a key is pressed and to start pressing keys, while the previous key is still being pressed.

Pressing a key longer can be achieved by delaying lifting the finger. As discussed in section 3.9 all finger presses and lifts get added to the trajectory segments. So, to keep the finger down longer the lift is just applied at a later point in time. This still leaves the problem of starting to press while the other finger still sits on the previous key.

Usually, the hand will move to ensure that the finger that is supposed to press the next key is placed above its target with no regard for placing any other fingers. Thus, the finger used for pressing the previous key will move around in that phase.

It is possible to provide bio_ik with multiple goals. So instead of just using a goal to place the next finger above the key, another goal is used to keep the last finger in its place.

The different styles of staccato and legato can be replicated with the resulting pipeline.

3.8.4 Inverse Kinematics Analysis

Inverse kinematics is used to bring the hand into a pose, where the fingers can reach the requested keys when pressing down. Here, bio_ik [Ruppel, 2017] is used in particular. This framework is optimization-based and allows for custom-made goals.

The simplest approach to solve this task is by using a pose goal. The pose goal is supplied with a frame, which matches the given pose in the resulting configuration. Here's an example of the variables needed to construct that goal for the index finger to hover over the key C3:

```
frame_name = "rh_fftip"
pose.frame_id = "c3"
pose.orientation.x = 0.5
pose.orientation.y = -0.5
pose.orientation.z = -0.5
pose.orientation.w = 0.5
pose.position.x = 0.03
pose.position.y = 0.0
pose.position.z = 0.03
```

The orientation is given as quaternion and is chosen so that the finger is hovering horizontally over the key. The position is given in the frame of the key with the unit of meters.

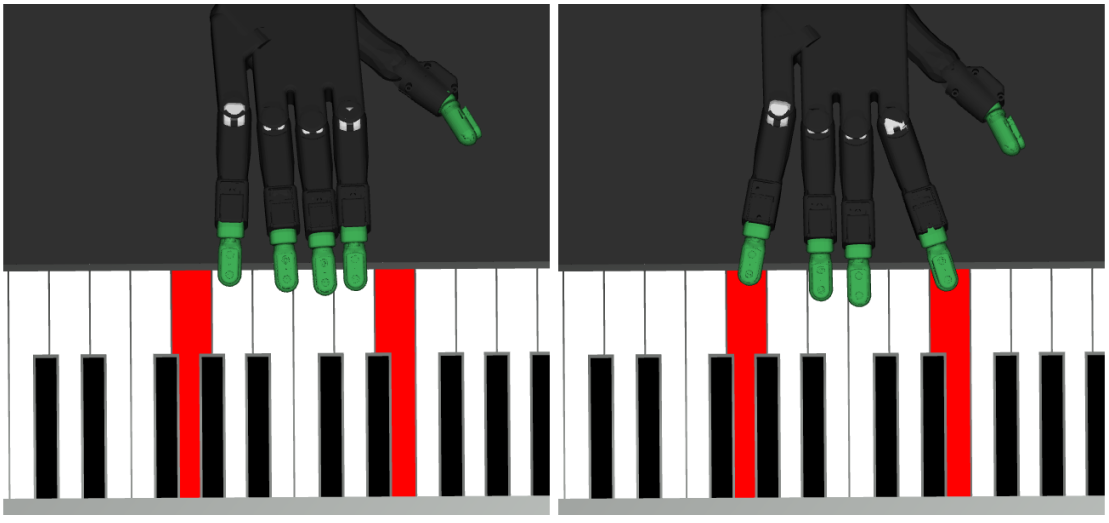

FIGURE 3.20: On the left is the attempt to position the index and little finger over the red keys with the standard pose goals. The fingers are not in a position, where the keys can be pressed. On the right image, both fingers are correctly placed above their corresponding target keys with the use of the custom-made goal. The weights were chosen so that the position of the fingers are regarded to be of higher importance than the orientation of the fingers.

This type of goal works well to position one finger above keys, but leads to problems when multiple fingers are involved, as is, for example, the keys with chords and the legato style. The orientation is a hindrance to find valid solutions. The framework does support returning approximate solutions, but as can be seen in the left image in figure 3.20 the fingers are not over the targeted keys in the returned configuration.

As mentioned, it is possible to use custom-made goals. The function of the orientation was modified so that up to a certain difference to the given target, the resulting values are lower:

```
if(orientation_difference < acceptable_range)
    orientation_difference = orientation_difference * 0.1

return orientation_difference + position_difference
```

Since bio_ik is optimization-based, it is possible to define the importance of a goal via weights. Here, the importance of the position of the finger is higher than the importance of the orientation. The goal is adjusted as follows:

```
if(orientation_difference < acceptable_range)
  orientation_difference = orientation_difference * 0.1

orientation_difference = orientation_weight * orientation_difference
position_difference = position_weight * position_difference
return orientation_difference + position_difference
```

The result of the right image in figure 3.20 was generated by using a weight of 60 for the position and a weight of 1 for the orientation. With the modified goal, both keys can be reached.

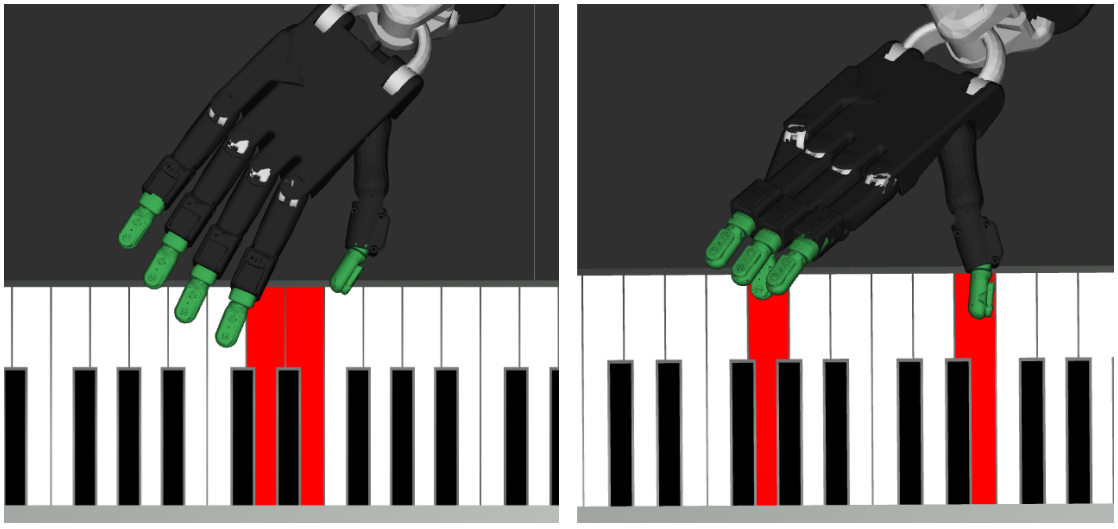

FIGURE 3.21: An example of a successful solution and a failed solution for pressing two white keys at the same time, using the thumb and an extended finger.

An alternative pose for the fingers when using the thumb was presented in section 3.8.1. Figure 3.21 shows a case where no solution could be found for pressing two white keys using extended fingers. The figure also includes an example where a solution was found, albeit it is less intuitive than a solution with angled fingers as it is shown in figure 3.22.

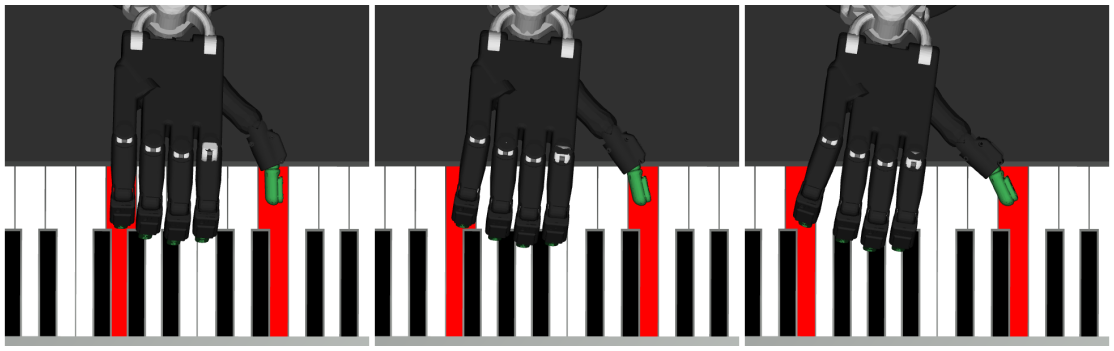

FIGURE 3.22: C3 and A3 can be reached comfortably with the thumb and little finger. In the middle image, C3 and B3 can still be reached when using the thumb and the little finger. In the image on the right when choosing C3 as a target for the thumb and C4 as a target for the little finger, the thumb would hit adjacent keys when pressing down.

Figure 3.22 shows the limits for reaching two keys at the same time. While C3 and A3 can be reached comfortably, the thumb is already slightly angled when the little finger targets B3 and when the target changes to C4, the thumb would hit adjacent keys when pressing down.

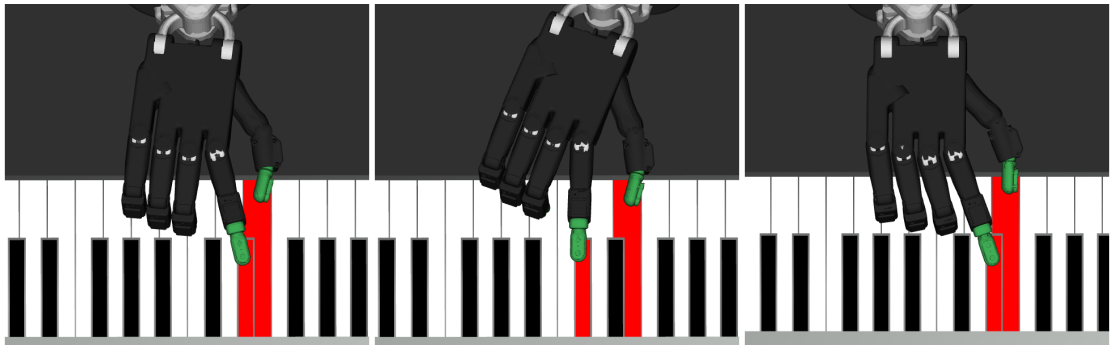

FIGURE 3.23: In the image on the left the thumb hovers over C3 and the index finger over C#3. The distance between the keys is very short so that both fingers have to be angled to reach their targets. In the image, in the middle, the thumb hovers over C3 and the index finger over D#3. In the image on the right the thumb is in a better position with a higher weight for the orientation goal.

Figure 3.23 shows the solutions for pressing a white key with the thumb and a black key with the index finger. Here the index finger stays extended, to better

reach the black key. In the first two examples, the thumb is angled in a way, so that it gets very close to the adjacent keys. The image on the right shows an example where a higher weight for the orientation goal of the thumb leads to a better positioning above the key compared to the solution shown in the image on the left.

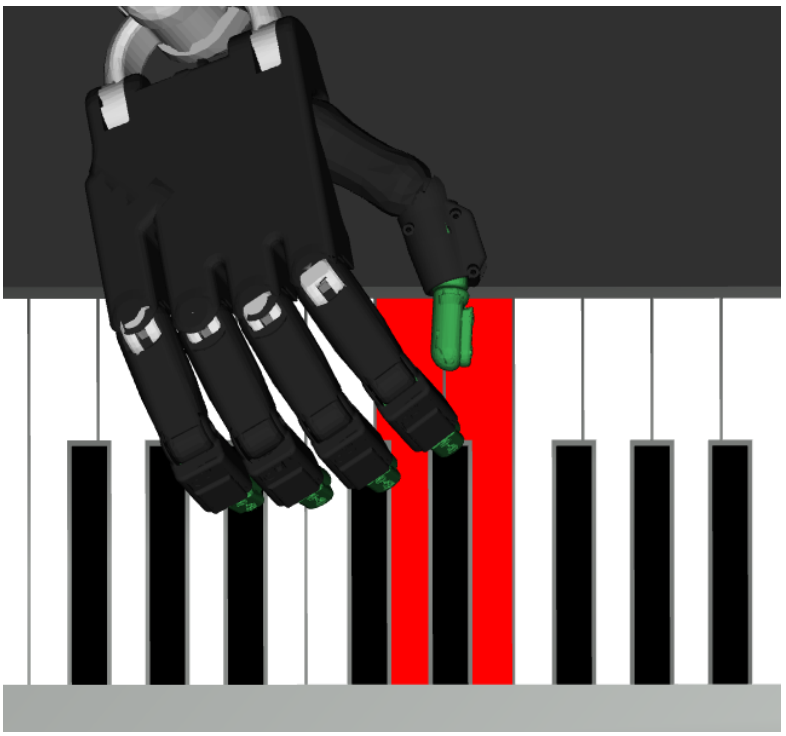

FIGURE 3.24: No acceptable solution is found for passing the index finger over the thumb.

Figure 3.24 shows that no passable solution is found for passing the index finger over the thumb. As discussed in section 3.8.1 this is due to the kinematics of the hand.

3.9 The Pipeline

The different systems are put together to create a full pipeline. The pipeline needs some kind of input. This is realized with a ROS action server, using the ROS actionlib (`http://wiki.ros.org/actionlib`, accessed on 20.08.2019). The input for the action server is defined as follows:

```
Pipeline Input:
PressGoal[] press_goals
---
Pipeline Result:
MoveitErrorCodes result
```

Each goal contains the following values:

```
PressGoal:
string[] target_keys
string[] fingers
uint32[] target_velocities
float64[] press_time
string[] angled_fingers
uint32 note_divisor
```

Here *target_keys* lists the keys, that are supposed to be pressed for that particular segment. The values of *fingers* determine which fingers should be used to press the keys and *target_velocities* which MIDI velocity the press should ideally result in. The values in *press_time* describe how long the fingers should be kept on their respective keys after pressing. The array *angled_fingers* lists all fingers that should be angled for the current segment. And at last *note_divisor* determines the divisor of the current note, for example, the value 4 would correspond to a quarter note.

Figure 3.25 shows an overview of the different nodes and how they work together. The main node is *PressKeysNode*. It provides the previously discussed action server. With the given *target_keys* and *fingers* IK solutions are generated with the fingers over their corresponding keys. This is done with bio_ik. The PTP planner introduced in section 3.5.1 is used to create the different trajectory segments moving the arm from target keys to target keys without moving the fingers yet.

The *PressControllerNode* handles generating the press parameters. It uses the *PressNN_Node* for this purpose, which uses the trained networks to predict the best parameters. The *PressControllerNode* also handles saving any training data. For that purpose, it requires MIDI data. The *MidiPublisherNode* listens to the MIDI port the keyboard publishes its data on and converts the data into a general ROS message. Then the *MidiNode* converts the bytes in the MIDI message into concrete keys and velocities.

The press parameters are given to the *PressKeysNode*. With those parameters, the finger presses and lifts are added to the trajectory segments. The segments are connected to create the entire final trajectory. At the end, MoveIt! [Sucan and Chitta, 2011] is used for execution.

The parameters for each press only contain the end positions as well as the *time_from_start*. But in order to add the movement of the finger to the existing trajectory, values in between the start and end positions need to be calculated as well. Quintic spline interpolation is used once again to interpolate between the two states. This is done with the implementation from the *joint_trajectory_controller* (`http://wiki.ros.org/joint_trajectory_controller`, accessed on 21.08.2019).

The *MidiPublisherNode* reads the MIDI messages sent by the keyboard on a MIDI port. The *MidiReaderNode* reads the *midi_topic* and publishes the messages once again on a different MIDI port. With the use of this method, it is possible to replay the MIDI messages with a rosbag (`http://wiki.ros.org/rosbag`, accessed on 16.09.2019) and still generate sound. The software synthesizer FluidSynth

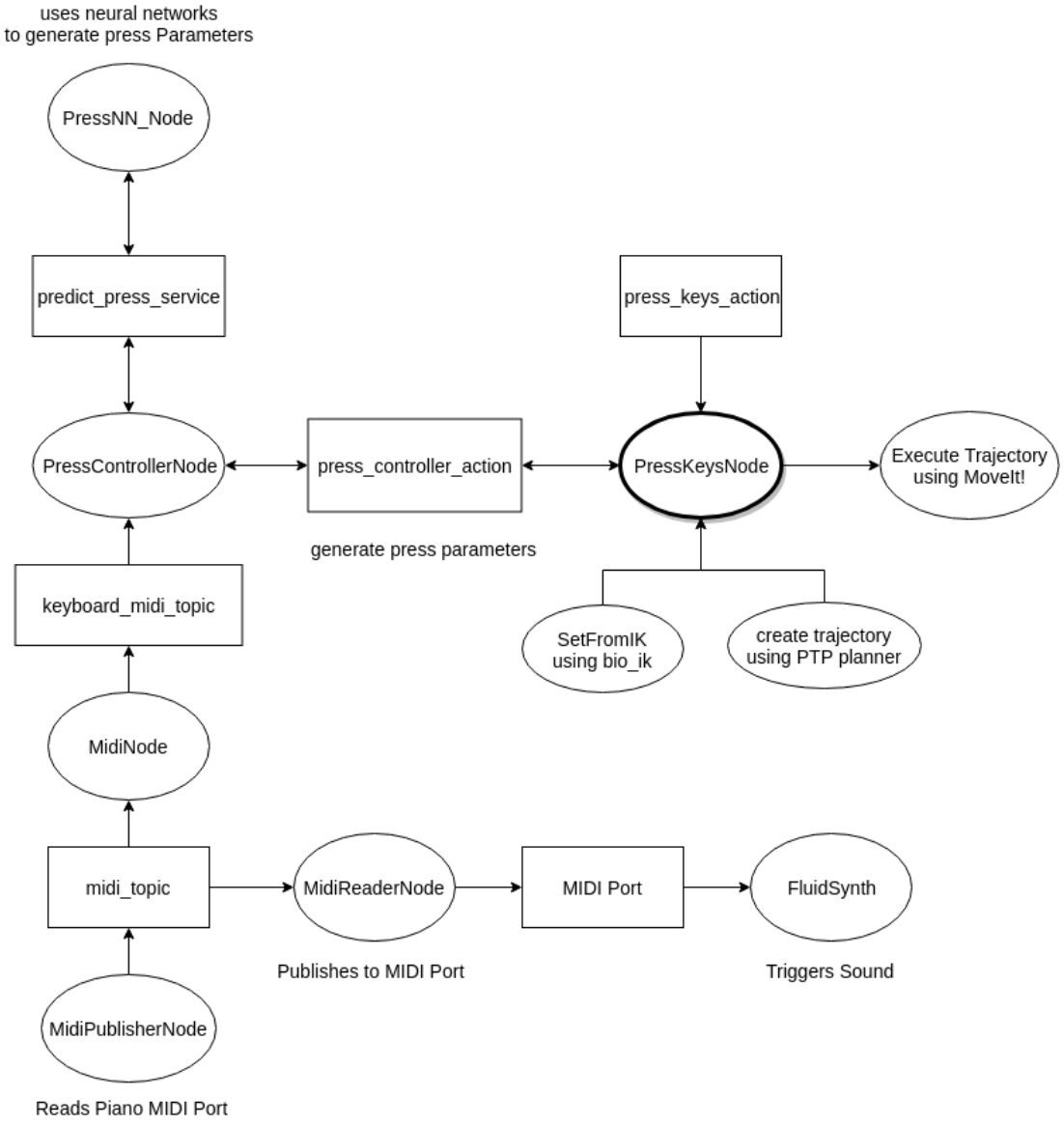

FIGURE 3.25: *PressKeysNode* is the main node. It gets its input as an action server. *PressControllerNode* generates the press parameters using the *PressNN_Node*. It also handles generating training data using the MIDI data it receives from the *MIDINode*. The *PressKeysNode* generates the trajectories using IK solutions from bio_ik [Ruppel, 2017]. Then it creates trajectory segments with the PTP planner and connects them. The execution of the trajectory is done using MoveIt! [Sucan and Chitta, 2011].

(`http://www.fluidsynth.org/`, accessed on 16.09.2019) is listening on that port and triggers the sounds.

Chapter 4

Experiments and Results

4.1 Velocity Precision

With the method shown in section 3.7.3, it is possible to predict parameters for pressing keys from a given target velocity. The following experiment is supposed to test how well the target velocities can be matched.

In the setup of this experiment, one of the fingers is moved over a white key. Then, the parameters get predicted with a rising target velocity and are executed. The resulting velocity from the MIDI data is compared with the given target. For the predictions, the most likely parameters are used, meaning the parameters are sampled from the mean value of the Gaussian with the highest weight.

Figure 4.1 shows the result for a single iteration. As discussed in section 3.7.2, the velocity of 1 is easy to achieve. So, it is not surprising that the result here matches its target exactly. For 5 and 10, the results are still close to the wanted velocity.

This is followed by a range of values that were missed by a larger margin. From 15 to roughly 40, the targets were mostly overshot. This phenomenon can be explained by the few training data points in this area. This was discussed in section 3.7.2 as well. Values in this target range are difficult to hit, and it can be assumed that even small differences in the pose of the hand above the key can lead to large differences in the result.

In the range from 40 to 60, the results are close to the wanted targets. This is another range, with lots of training data points. Values above 60 could not be achieved in the shown iteration. Here the density of training data points was lower once again, due to the difficulty for the hand to reach such high values.

For the sake of comparison, figure 4.2 shows the result of an untrained human trying to hit a key with rising velocity. It is of notice that the range from above 1 to 40 couldn't be hit either. A difference can be found when comparing the highest resulting velocities. While the Shadow Dexterous Hand was not able to

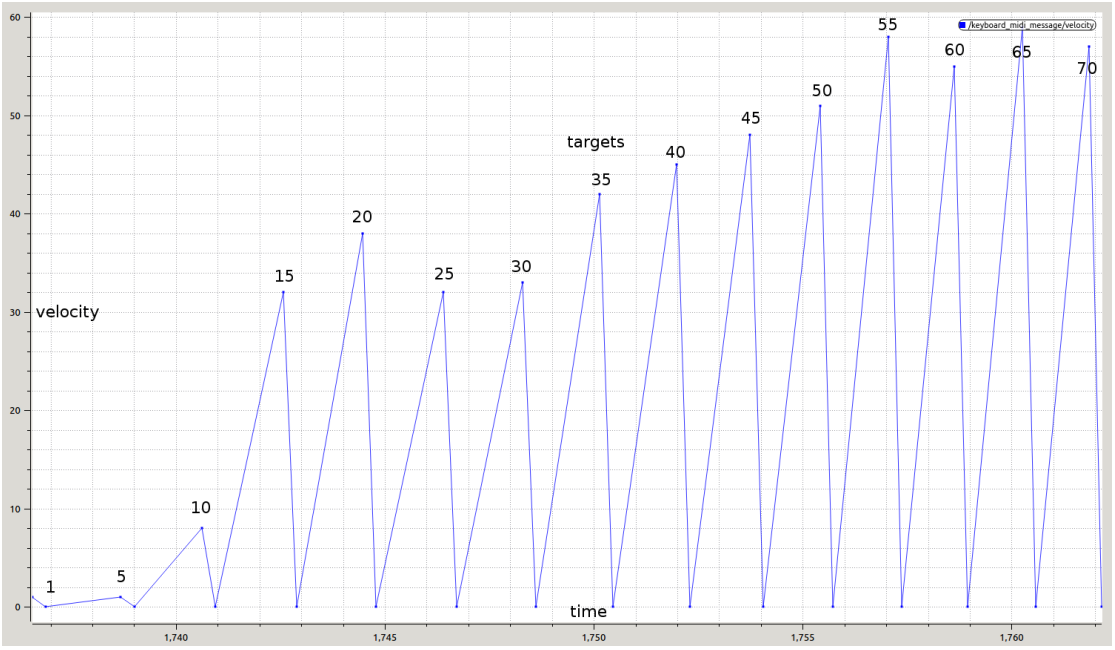

FIGURE 4.1: The result of the experiment for the precision of hitting target velocities. The numbers above the data points are the target velocities.

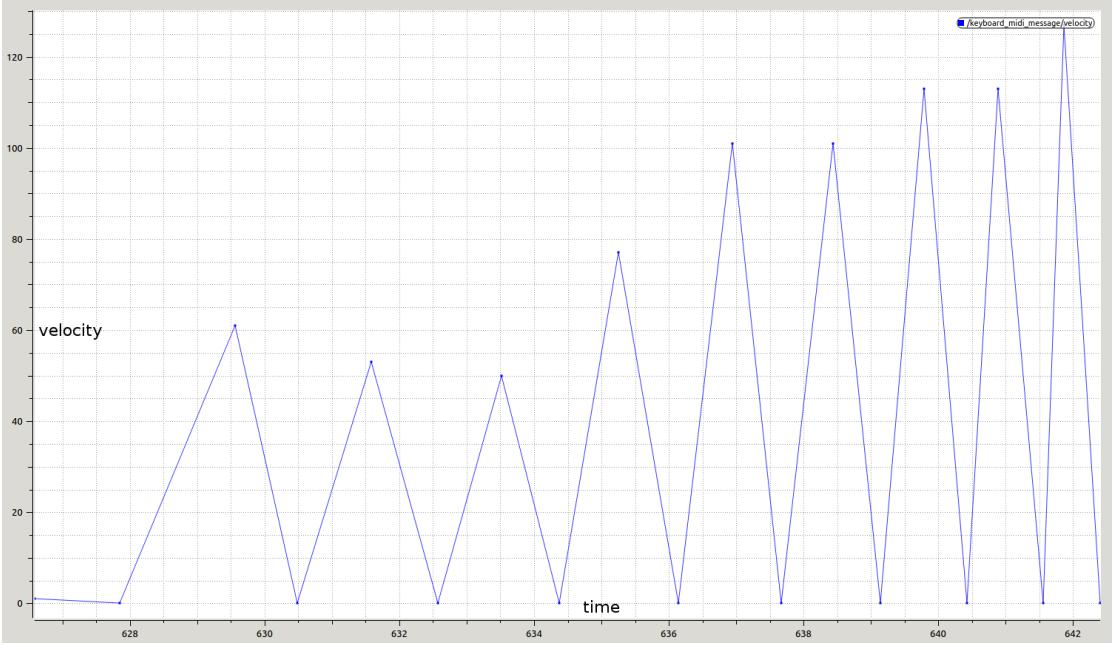

FIGURE 4.2: Result of an untrained human to hit a key with rising velocity.

achieve any values above 60, it is no problem for a human to reach the maximum value of 127.

The experiment was done with the index finger, middle finger, ring finger and little finger for the keys C3 to G3. For each combination, the experiment was executed 5 times. For the index finger on D3, in particular, the experiment was even repeated 19 times.

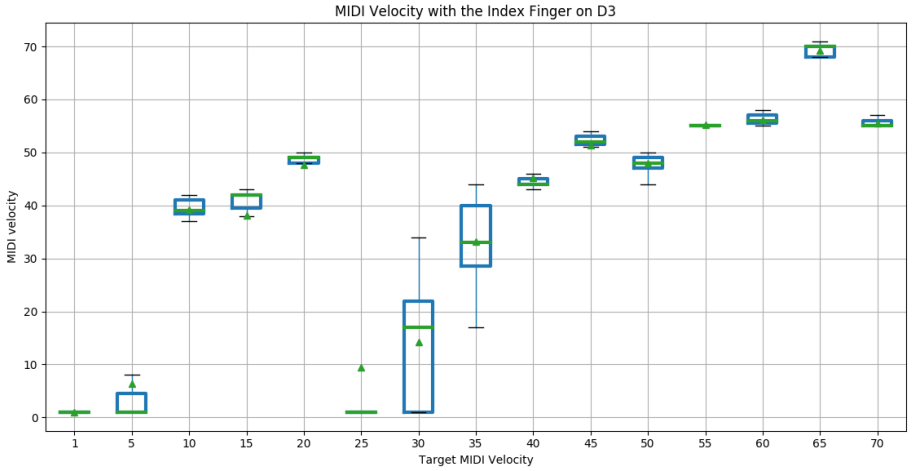

FIGURE 4.3: Example box plot of pressing the key D3 with rising target velocity with the index finger. The green line signifies the median. The blue box signifies the 1. and 3. quartiles. The whiskers are the interquartile range times 1.5. The triangle signifies the mean value.

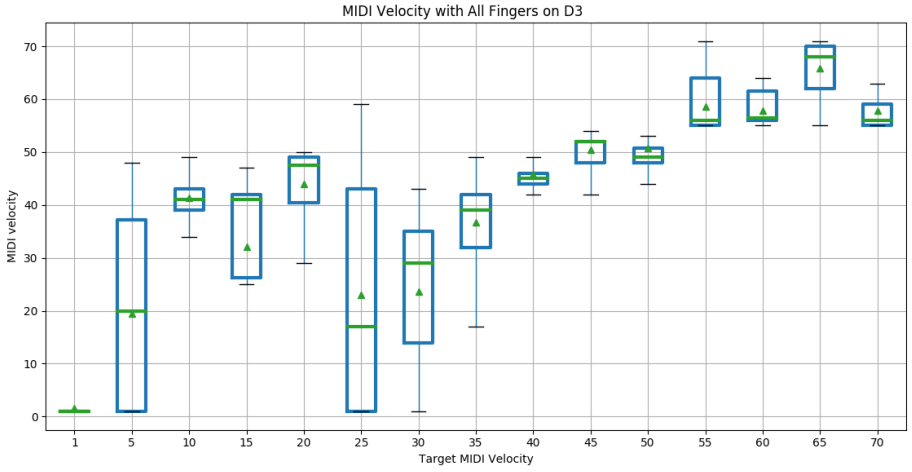

FIGURE 4.4: Example box plot of pressing the key D3 with rising target velocity with all fingers. The green line signifies the median. The blue box signifies the 1. and 3. quartiles. The whiskers are the interquartile range times 1.5. The triangle signifies the mean value.

Figure 4.3 and 4.4 show the variance when pressing the keys. In the first figure, only the index finger was used. There is a low variance for velocities around 1 and velocities above 40. Once again, values in the area from 10 to 35 were missed by a large margin. The variances for target values of 30 and 35 are remarkable. Here the mean value for the resulting velocity lands in the range of those values hard to achieve. As one would expect, these circumstances result in high variances.

Figure 4.4 shows box plots for D3 as well, but now including the velocities generated by all fingers. Obviously, the variance has risen in general. Once again, the variance is the highest for values in the area of velocities that are hard to hit.

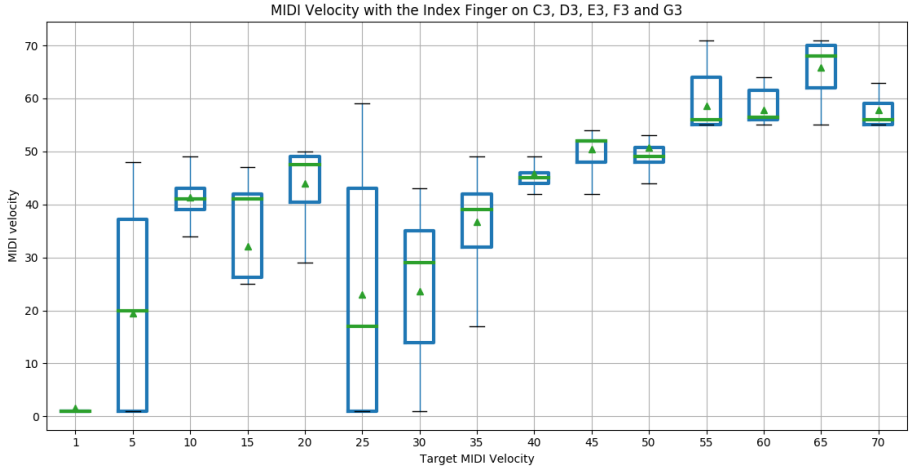

FIGURE 4.5: Example box plot of pressing the keys C3, D3, E3, F3 and G3 with rising target velocity with the the index finger. The green line signifies the median. The blue box signifies the 1. and 3. quartiles. The whiskers are the interquartile range times 1.5. The triangle signifies the mean value.

Figure 4.5 shows the variance for the keys C3, D3, E3, F3 and G3 when pressed with the index finger. Once again the variance is rather high in the usual region, but still rather low for the velocity 1 and velocities above 40. From this result, one can infer that there are no large differences in velocity output between the different keys.

4.2 Timing Precision

Achieving a certain timing for hitting the keys was previously discussed in section 3.6. The following experiment shows the difference in timing with the described methods applied and without them.

In this experiment setup, the entire pipeline for playing the piano is used. The index finger hits C3, then the middle finger hits D3, and then the ring finger hits C4. As there is a gap between D3 and C4, the hand needs to be repositioned

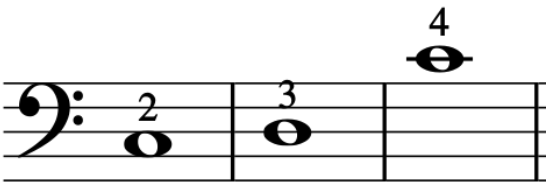

FIGURE 4.6: Notes played in the first timing experiment. All notes have the same length.

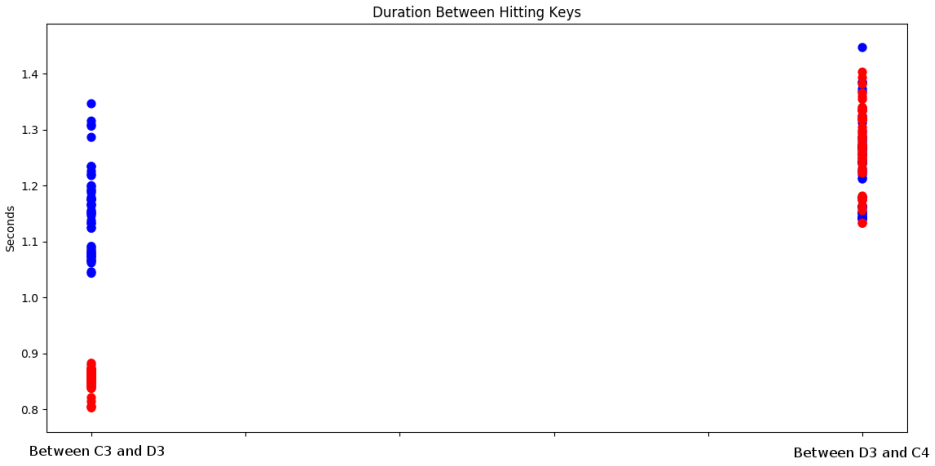

FIGURE 4.7: Scatterplot of the data generated by the experiment for playing three keys with even timing. Here blue dots were generated with trajectory scaling and red dots were generated without trajectory scaling.

in between these steps. Figure 4.6 shows the musical score for this experiment. All keys are supposed to be hit with the same time interval in between. The experiment was repeated 50 times.

The expected result would be a significant difference in the timing when playing without any trajectory scaling, and a constant time span between hitting keys with trajectory scaling.

Figures 4.8 show the results of the experiment. The values shown are the difference in elapsed time between playing C3 and then D3 and the elapsed time between playing D3 and then C4. With scaling the timing remains even, even though the hand needs to be repositioned between steps. The mean error stays below 0.1 seconds. Without scaling, there is a mean difference of more than 0.4 seconds.

The goal of the next experiment is to test how well quarter, half and whole notes can be played with the use of trajectory scaling. For this purpose the keys D3, E3 and F3 were played repeatedly, where D3 should be played as a quarter note with the index finger, E3 is supposed to be played as a half note with the middle finger, and F3 is supposed to be played as a whole note with the ring finger. Figure 4.9

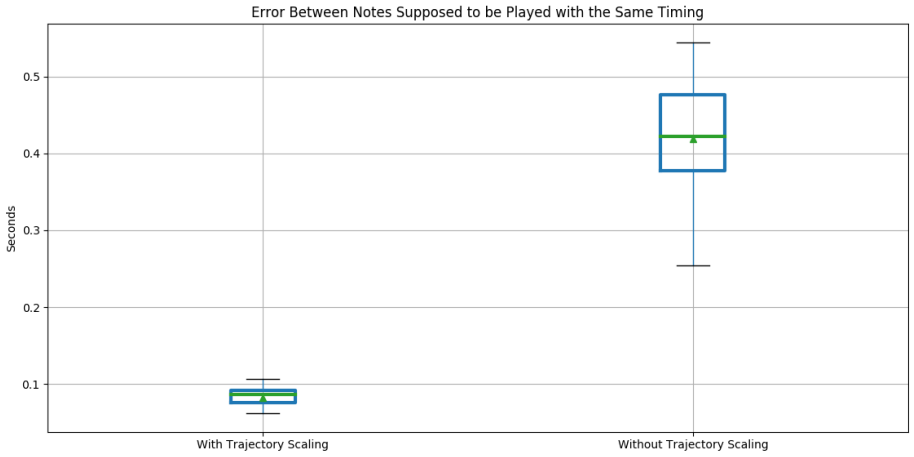

FIGURE 4.8: Comparison of the results with and without trajectory scaling. The error is calculated from the difference in elapsed time between playing C3 and then D3 and the elapsed time between playing D3 and then C4. The green line signifies the median. The blue box signifies the 1. and 3. quartiles. The whiskers are the interquartile range times 1.5. The triangle signifies the mean value.

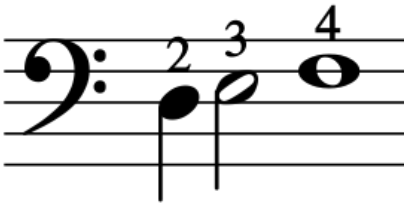

FIGURE 4.9: Notes played in the second timing experiment. All notes have a different length.

shows the experiment in musical notation. This was done 4 times in one iteration of the experiment, with 10 iterations in total.

Figure 4.10 shows an example of the durations between hitting the keys for a single iteration. Figure 4.11 shows a boxplot of the error, given the duration for the quarter note as ground truth. The mean error for the half note and the whole note lies at roughly 0.13 seconds. There is a bit more variance in the results for the whole note.

Friberg and Sundberg [1993] investigated the just noticeable difference (JND) for the time displacement of a tone in a sequence. Six notes are played in a sequence where all of them are equally spaced, except for the fourth one. Listeners were asked to move the fourth note so that the spacing is equal for all notes. They found out that the JND is about 10 ms for any notes shorter than around 240 ms and around 5% of the duration for any notes longer than that. For the first

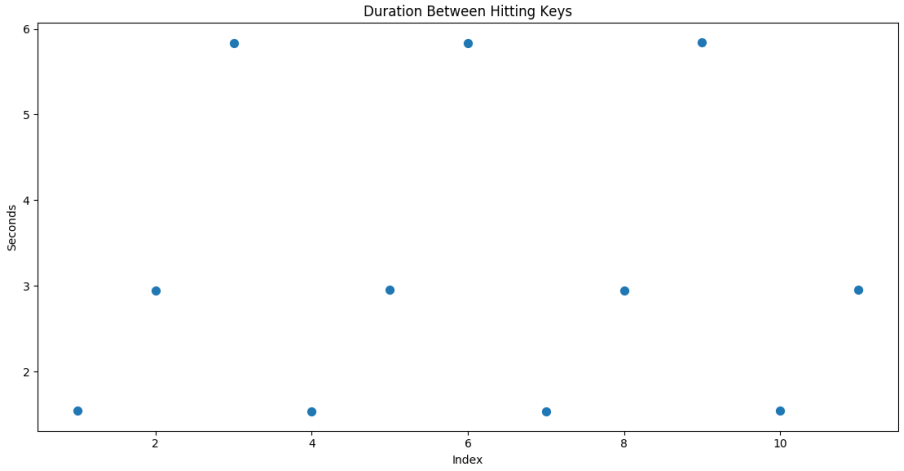

FIGURE 4.10: Example of one iteration of the experiment where quarter, half and whole notes were played repeatedly.

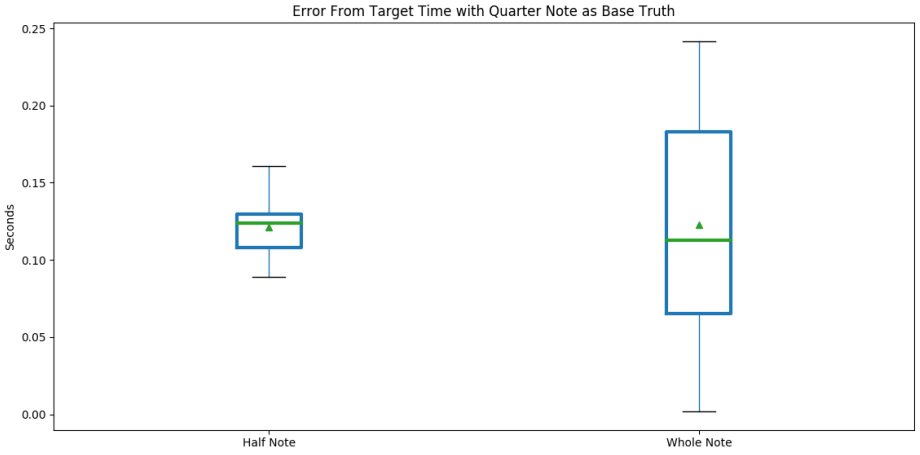

FIGURE 4.11: Error from the target time for half and whole notes, with the played quarter notes as base truth. The green line signifies the median. The blue box signifies the 1. and 3. quartiles. The whiskers are the interquartile range times 1.5. The triangle signifies the mean value.

experiment, of this section the duration of the notes was around 1000 ms to 1400 ms, with an error of about 100 ms. This error is about twice as large as the JND found by Friberg and Sundberg [1993], which is around 50 ms.

4.3 Trill

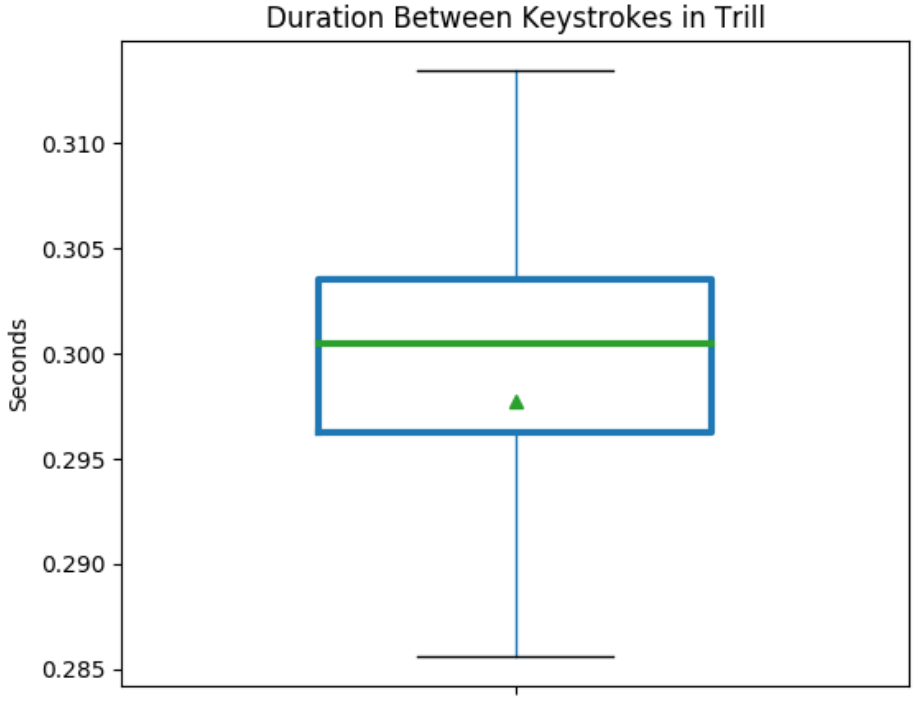

FIGURE 4.12: Box plot of the duration from hitting a key to hitting the next key when playing a trill. The green line signifies the median. The blue box signifies the 1. and 3. quartiles. The whiskers are the interquartile range times 1.5. The triangle signifies the mean value.

A trill is the rapid alternation between two adjacent notes. For this experiment, two adjacent white keys were played repeatedly in quick succession using the index finger and middle finger. The focus of this experiment is the duration from hitting a key to hitting the next key.

In each round of the experiment, 36 data points were collected. With 5 rounds in total, this results in 180 data points.

Figure 4.12 shows the result as a box plot. The median sits at roughly 0.3 seconds, with only small variations within the data.

4.4 Piano Pieces

So far, the experiments were used to evaluate the components of the pipeline. This section evaluates playing an entire piece. It was opted to play the German children's song "Alle meine Entchen" ("all my ducklings"). An interesting aspect here is the beats per minute that can be reached. The song was played in two different variants. One starts playing with the thumb and then uses the angled index finger to play the next key. All following notes are played with the fingers in the extended position. The second variant ignores the thumb all-together and only uses four fingers.

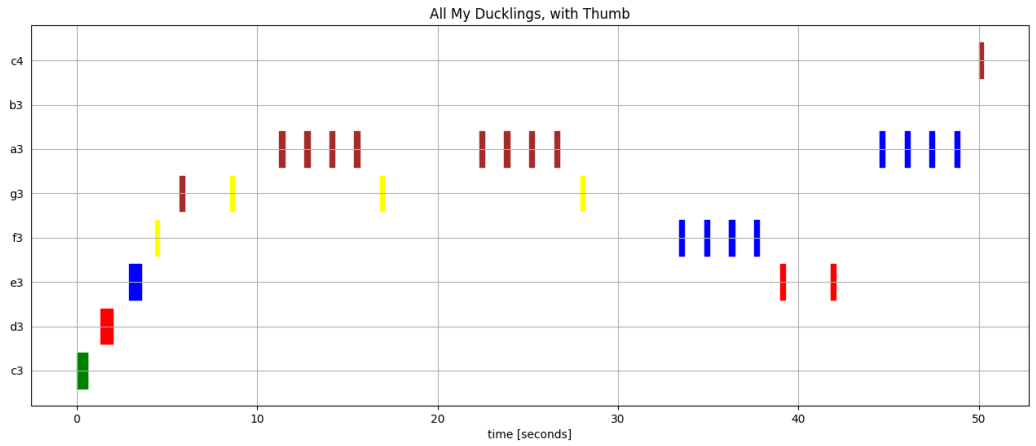

FIGURE 4.13: Diagram of all my ducklings being played with the thumb at the beginning. Each bar starts with the "note on" MIDI signal and ends with the corresponding "note off" signal.

Figures 4.13 and 4.14 show the result for both variants with trajectory scaling. The bars show when, which key was pressed. The biggest difference is the time needed to play the whole song. The variant with thumb needs almost twice as long to play the entire piece compared to the variant without the thumb. For both variants, it was opted to scale it with as many beats per minute as possible. For the variant using the thumb, the biggest issue is the repositioning of the finger from angled to extended. It is too slow, and thus the entire piece needs to be scaled accordingly. It becomes very apparent in figure 4.15. The piece is played with the thumb, but without trajectory scaling. The longest pause is between the second and third note, where the fingers move from angled to extended.

For the variant, without the thumb, a quarter note is about 0.7 seconds long, which equates to roughly 85 beats per minute.

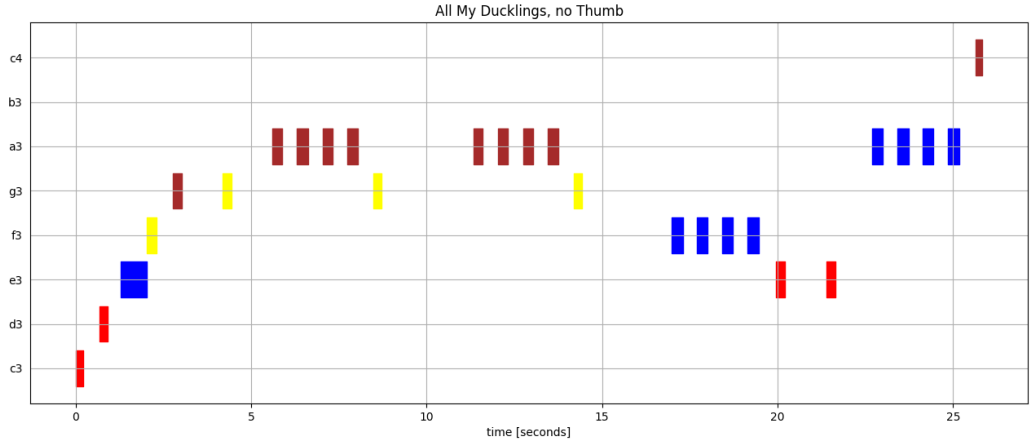

FIGURE 4.14: Diagram of all my ducklings being played without the thumb. Each bar starts with the "note on" MIDI signal and ends with the corresponding "note off" signal.

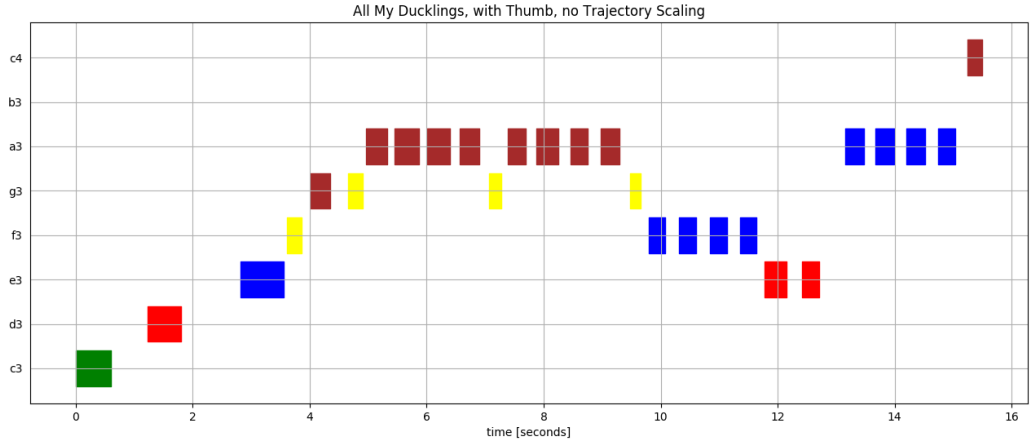

FIGURE 4.15: Diagram of all my ducklings being played with the thumb at the beginning without any trajectory scaling. Each bar starts with the "note on" MIDI signal and ends with the corresponding "note off" signal.

Chapter 5

Conclusion

In this thesis, a software pipeline was created to play piano with the Shadow Dexterous Hand mounted on a PR2. It is possible to localize the piano and press any key on it with any finger on the hand.

Playing with a specific volume and playing with the correct timing were attempted to be implemented. As the experiments showed, it is not possible to precisely hit a certain velocity with the presented pipeline. In certain ranges, it is possible though to hold a certain velocity. It is also possible to at least either play very silent or with moderate volume. The timing was not exact and had a median offset of roughly 0.1 seconds. Another factor is the beats per minute, or how fast pieces can be played. Only for the trill roughly 200 beats per minute could be reached, which would be considered very fast. However, even pieces with low complexity, like all my ducklings, are only possible to be played at around 80 beats per minute, which is considered to be slow. When the thumb is used in addition and the hand changes from angled fingers to flat fingers while playing, the reachable beats per minute fall even more.

For the low bound of roughly 70 for reachable velocities it should be noted, that the hand was used with lower limits for velocity and acceleration, than technically possible. This choice was made due to concerns about the durability of the hand for higher values. With these circumstances in mind, it can be concluded that the hand is not able to exert forces similar to a human hand with these limits. For an untrained human, it was possible to easily reach the maximum MIDI velocity, while the hand could barely reach a velocity of 70. The significant deviations from the target MIDI velocities could be explained by a number of reasons. Mixture Density Networks might not be a well-suited model for this problem. It is also possible that deviations in the localization led to differences in the pose of the finger above the keys. Another source of error might be the collected training data.

The low values for reachable beats per minute depend on multiple factors. The first factor is the arm the Shadow Dexterous Hand is mounted on. It is responsible for bringing the hand into the correct position to be able to play the keys. Its limits

in velocity and acceleration are a critical factor in this context. For comparably high values, the arm would stop abruptly when reaching the goal, which would result in the hand shaking considerably and thus lower the accuracy. For this reason, the values were chosen to be set lower. Mounting the Shadow Dexterous Hand on a different arm, for example, a UR10 [Ravn et al., 2014], would result in faster arm motions with higher acceleration and thus have a positive influence on the reachable beats per minute. The software side is another factor. Here the IK solutions are of considerable interest. For reaching the next keys as quickly as possible, the difference of the joint configurations should be as low as possible. The focus for the IK solutions in this thesis was on reaching certain targets with the fingers, with the distance between joint configurations only being a secondary consideration.

It was also possible to implement playing staccato and legato. So it is possible to replicate some basic techniques used by piano players with the Shadow Dexterous Hand. But there were also limiting factors. First of all, it turned out that the calibration of the hand was often not accurate enough to hit the keys. The keys are a bit wider than the fingers of the hand, so arguably a decent accuracy is needed, but still at a level that many robotic systems are able to achieve.

The hand is modeled after a human hand, so it is interesting to compare it in the aspect of kinematics. The span of the hand is similar to that of a human, so it is able to cover a similar range of keys to a human. But for example, reaching with the thumb over the index finger proved to be hardly possible with the Shadow Dexterous Hand, while it is a technique commonly used by human piano players.

So, while the Shadow Dexterous Hand is modeled after the human hand, there are some essential differences, which make it hard to replicate techniques used by human piano players and lead to a worse performance compared to human piano players in some areas.

In terms of characteristics of the particular task of playing the piano, one unique aspect is the timing. As an example, grasping does only require a motion of the arm and hand, but the speed with which these motions are carried out is not of relevance. The trajectories only have to adhere to the velocity and acceleration limits of the joints. For the task of playing piano, it is of importance to press the keys at a certain point in time, and here timing and how fast trajectories are executed do matter. In the context of this thesis, trajectory scaling was used, to be able to execute trajectories, so that pressing the keys occurs at a certain point in time. The results of the experiment showed that this technique is a suitable solution for this task, albeit the slight offset. Piano playing is well suited as a benchmark for the aspect of timing in robot motions, as it can be used to compare the errors between techniques.

Chapter 6

Outlook

The most important shortcoming for playing the piano well is the accuracy for hitting the correct keys. This is directly related to the shortcomings of the current calibration. Recalibration with the method provided by the Shadow Robot Company was attempted in the course of this thesis, but the result proved to be far from ideal. The first step to improve general performance would be a more reliable calibration method.

A different possibility to solve the issues with accuracy would be the use of the BioTac sensors [Fishel et al., 2015] mounted on each finger of the Shadow Dexterous Hand in combination with MIDI feedback. BioTac sensors are tactile sensors. The fingers could be set on top of the keys and use the generated data to estimate on which key the fingers are currently positioned. The sensors could also be used to predict whether a press would only target the desired key or if it would hit any of the adjacent keys as well. This approach would require for it to be possible to recognize the transitions between keys with the data produced by the BioTacs. For the placing of the hand on top of the keys, the data of the BioTacs could be used as well. For the current pipeline, targets are defined $3cm$ above the keys. Especially the little finger hangs lower than the other fingers, due to additional joints at the palm, so that the targets needed to be set at that height to avoid collisions with the keyboard. Using the data of the BioTacs to recognize when the keys are touched, but in a way that the hand could still be moved above the keys, would solve this problem as well. As a side-effect, the fingers would require less movement for a successful press, which would result in less time needed to press the keys.

In the last chapter, it was also discussed how targets with substantial differences in the joint configurations lead to longer pauses between hitting keys. For this purpose, IK solutions would need to prioritize to keep the difference between these configurations at a low level. For the aid in finding solutions with low differences in joint space, it would be helpful to have a broad range of valid solutions to choose from as possible. Otherwise, a goal that is added to keep the distance between configurations low will still result in large differences. The targets to place the fingers above the keys should cover as many solutions as possible from which it is

possible to press the key. This would involve tolerance in height above the keys, tolerance in the placement along the key, and tolerance in orientation. To a certain extent, these tolerances are already part of the current pipeline used, but it could certainly be extended to cover a range of solutions as broad as possible.

The fingering of piano pieces was solved by assigning a finger to each note. In the context of robotics, it would be an interesting task to find the optimal fingering for a given piece in order to play a piece as quickly as possible.

No data produced by humans was considered in this thesis. The simplest form would be recording MIDI data from a human performing a piece on the piano. Certain characteristics could be extracted and replicated with the robot. For the current system looking at staccato and legato would be a compelling use case. In the current system, each note has the time it should be pressed assigned to it. These values could be extracted from the data generated by humans to evaluate how well the system is able to replicate this aspect of piano playing.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Alford, A., Northrup, S., Kawamura, K., Chan, K., and Barile, J. (1999). A Music Playing Robot. In *Proc. of the Conf. on Field and Service Robots*, pages 29–31.

Barraquand, J. and Latombe, J. . (1991). Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2328–2335 vol.3.

Bartels, R. H., Beatty, J. C., and Barsky, B. A. (1995). *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann.

Batula, A. M. and Kim, Y. E. (2010). Development of a Mini-Humanoid Pianist. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 192–197.

Beeson, P. and Ames, B. (2015). TRAC-IK: An Open-Source Library for Improved Solving of Generic Inverse Kinematics. In *Proceedings of the IEEE RAS Humanoids Conference*, Seoul, Korea.

Bishop, C. M. (1994). Mixture Density Networks.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. springer.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Fishel, J., Lin, G., and Loeb, G. (2015). *BioTac® Product Manual*. SynTouch LLC.

Friberg, A. and Sundberg, J. (1993). Perception of Just-Noticeable Time Displacement of a Tone Presented in a Metrical Sequence at Different Tempos. *The Journal of The Acoustical Society of America*, 94(3):1859–1859.

Ha, D. (2015). Mixture Density Networks with TensorFlow. *blog.otoro.net*.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hughes, J. A. E., Maiolino, P., and Iida, F. (2018). An Anthropomorphic Soft Skeleton Hand Exploiting Conditional Models for Piano Playing. *Science Robotics*, 3(25).

Jaju, V., Sukhpal, A., Shinde, P., Shroff, A., and Patankar, A. B. (2016). Piano Playing Robot. In *2016 International Conference on Internet of Things and Applications (IOTA)*, pages 223–226.

Koenig, N. and Howard, A. (2004). Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3.

Kuffner, J. J. and LaValle, S. M. (2000). RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2.

L. Finn, D. (2004). MA 323 Geometric Modelling Course Notes: Day 09 Quintic Hermite Interpolation.

LaValle, S. M. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, Computer Science Dept., Iowa State University.

Li, Y. and Chi-Yi Lai (2014). Intelligent Algorithm for Music Playing Robot — Applied to the Anthropomorphic Piano Robot Control. In *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pages 1538–1543.

Li, Y. and Chuang, L. (2013). Controller Design for Music Playing Robot — Applied to the Anthropomorphic Piano Robot. In *2013 IEEE 10th International Conference on Power Electronics and Drive Systems (PEDS)*, pages 968–973.

Li, Y. and Huang, C. (2015). Force Control for the Fingers of the Piano Playing Robot — A Gain Switched Approach. In *2015 IEEE 11th International Conference on Power Electronics and Drive Systems*, pages 265–270.

Li, Y. and Lai, C. (2016). Development on an Intelligent Music Score Input System — Applied for the Piano Robot. In *2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pages 67–71.

Li, Y. and Li, T. (2017). Adaptive Anti-Windup Controller Design for the Piano Playing Robot Control. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 52–56.

Lin, J.-C., Huang, H.-H., Li, Y.-F., Tai, J.-C., and Liu, L.-W. (2010). Electronic Piano Playing Robot. In *2010 International Symposium on Computer, Communication, Control and Automation (3CA)*, volume 2, pages 353–356. IEEE.

Lloyd, J. and Hayward, V. (1993). Trajectory Generation for Sensor-Driven and Time-Varying Tasks. *The International journal of robotics research*, 12(4):380–393.

Martin, C. (2018). Keras Mixture Density Network Layer. `https://github.com/cpmpercussion/keras-mdn-layer`.

MIDI Manufacturers Association (2014). The Complete MIDI 1.0 Detailed Specification. Third Edition:334. Document Version 4.2. The specification and all related practices are described as of 1996.

Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: an Open-Source Robot Operating System. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan.

Ravn, O., Andersen, N., and Andersen, T. (2014). *UR10 Performance Analysis*. Technical University of Denmark, Department of Electrical Engineering.

Ruppel, P. S. (2017). Performance Optimization and Implementation of Evolutionary Inverse Kinematics in ROS. Master's thesis, Universität Hamburg, Germany.

Shadow Robot Company (2013). *Shadow Dexterous Hand E1 Series (E1M3R, E1M3L, E1P1R, E1P1L)*. Shadow Robot Company.

Solis, J., Koichi Taniguchi, Takeshi Ninomiya, Tetsuro Yamamoto, and Atsuo Takanishi (2008). Development of Waseda Flutist Robot WF-4RIV: Implementation of Auditory Feedback System. In *2008 IEEE International Conference on Robotics and Automation*, pages 3654–3659.

Starke, S., Hendrich, N., Krupke, D., and Zhang, J. (2017). Evolutionary Multi-Objective Inverse Kinematics on Highly Articulated and Humanoid Robots.

Sucan, I. A. and Chitta, S. (2011). *MoveIt*. `http://moveit.ros.org` [Accessed: 21.08.2019].

Sugano, S. and Kato, I. (1987). WABOT-2: Autonomous Robot with Dexterous Finger-Arm–Finger-Arm Coordination Control in Keyboard Performance. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 90–97. IEEE.

Topper, A. J. and Maloney, T. (2019). Piano-Playing Robotic Arm.

Wee, C. C. and Mariappan, M. (2017). Finger Tracking for Piano Playing Through Contactless Sensor System: Signal Processing and Data Training Using Artificial Neural Network. In *2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, pages 41–45.

Weinberg, G. and Driscoll, S. (2007). The Design of a Perceptual and Improvisational Robotic Marimba Player. In *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pages 769–774. IEEE.

Willow Garage (2012). *PR2 User Manual*. Willow Garage.

Yamamoto, K., Ueda, E., Suenaga, T., Takemura, K., Takamatsu, J., and Ogasawara, T. (2010). Generating Natural Hand Motion in Playing a Piano. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3513–3518. IEEE.

Zhang, A., Malhotra, M., and Matsuoka, Y. (2011). Musical Piano Performance by the ACT Hand. In *2011 IEEE International Conference on Robotics and Automation*, pages 3536–3541. IEEE.

Zhang, D., Lei, J., Li, B., Lau, D., and Cameron, C. (2009). Design and Analysis of a Piano Playing Robot. In *2009 International Conference on Information and Automation*, pages 757–761. IEEE.

Selbstständigkeitserklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang 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.

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

Hamburg, den 17. September 2019

_____

Benjamin Scholz