Master Thesis

Autonomous grasping and path planning for a humanoid robot playing Nine Men's Morris

Roland Klöbl

Graz, 2012

Institute for Software Technology
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa
Second reviewer: Dr. techn. Gerald Steinbauer

Abstract (English)

Playing games has always been an important aspect in human society. Games may be used for entertainment purposes as well as for educational ones. Hence playing games was also interesting for research in the past. In the field of Artificial Intelligence (AI) it has always been an interesting aspect to create algorithms which are capable of winning against humans who are considered to be masters. Recently playing games has also become an interesting topic for researchers in the field of robotics. In addition to the creation of AI, other challenging problems like self-localization within the environment, perception of the current game status and proper reaction in terms of mobile manipulation need to be addressed.

The following thesis introduces a framework which can be used for playing Nine Men's Morris with the robot Nao manufactured by Aldebaran robotics. This particular game was chosen because it is very popular in Europe and it also features a relatively simple setup. Unlike in chess, every token has the same movement options and thus the tokens only need to be distinguished by color. The framework needed to deal with diverse challenges which may be grouped into three major tasks:

- Vision Task
- Mobile Manipulation Task
- Game Knowledge Task

Each task was realized by a different person. This thesis only focuses on the task of mobile Manipulation. The term mobile manipulation involves the entire motion control of the robot as well as the motion planning in order to reach a certain goal. Hence the presented information concerns planar navigation through a known environment and collision-free arm navigation towards a certain point in order to grab an object. Furthermore, it provides an idea about how the forward and inverse kinematics with respect to the given grab scenarios were solved.

The biggest challenge in realizing the framework was to solely use built-in sensors of Nao. The most important sensors to solve the given tasks were the two cameras located in Nao's head. However, it was not possible to retrieve viable pictures during motion due to motion blur. This made the tasks concerning mobile manipulation even more challenging. In the past, various approaches regarding how to play board games with Nao were demonstrated. However, the presented framework features a scenario which, to our knowledge, has not been used before.

The problems concerning the vision task of the framework including self-localization and general perception of the environment are addressed in a different thesis by Bock [3]. More information concerning the framework used as an interface to the game database representing the AI can be found in [19].

Abstract (German)

Spiele hatten schon immer eine wichtige Rolle in unterschiedlichen Bereichen unserer Gesellschaft inne. Neben dem Zweck der Unterhaltung wurden sie bereits für viele andere Einsatzgebiete, wie dem spielerischen Erlernen von Fähigkeiten eingesetzt. Aufgrund dieser Vielseitigkeit und der Faszination, die durch das Finden von immer besseren Spielstrategien entsteht, wurden Spiele auch ein integraler Bestandteil diverser Forschungsarbeiten. Im Gebiet der künstlichen Intelligenz (KI) galt es stets als interessante Aufgabe neue Algorithmen für die Entwicklung von Spielstrategien zu finden, die selbst bekannten Größen aus dem betreffenden Spielsektor wie dem Schachgroßmeister überlegen waren. Seit kurzem wurde auch das Forschungsgebiet der Robotik auf Spiele aufmerksam. In diesem Sektor gibt es zusätzlich zu dem Problem des Lösens des Spiels andere Herausforderungen betreffend der Ausführung des Spiels. Dies beinhaltet zum Beispiel die Lokalisierung des Roboters in seiner Umgebung, sowie das Erkennen des aktuellen Spielstatus und der physischen Ausführung der benötigten Aktionen.

Die folgende Arbeit gibt einen Überblick über das implementierte Framework, dass es dem Roboter Nao von Aldebaran Robotics ermöglicht, das Spiel Mühle zu spielen. Das Spiel Mühle wurde gewählt, weil es im europäischen Raum ein sehr bekanntes Spiel ist und außerdem einen relativ einfachen Aufbau besitzt. Anders als bei Schach sind bei Mühle alle Spielsteine gleichberechtigt und müssen daher nur anhand der Farbe unterschieden werden, um eine Spielerzuordnung zu ermöglichen. Die Probleme, welche während eines Spiels aus Sicht des Roboters auftreten, lassen sich grob in drei Gebiete unterteilen:

- Visuelle Wahrnehmung der Umgebung
- Mobile Manipulation, also physische Ausführung
- Spiellogik und Folgezugberechnung

Jede dieser Aufgaben wurde von unterschiedlichen Personen behandelt. Die vorliegende Arbeit beschäftigt sich hauptsächlich mit den Aufgaben und Schwierigkeiten im Bereich der mobilen Navigation. Dies beinhaltet die vollständige Kontrolle sämtlicher Bewegungen des Roboters, sowie das Erstellen von Plänen, um die gewünschten Ziele zu erreichen. Die Bereiche, welche in dieser Arbeit behandelt werden, sind daher die planare Navigation, also das Gehen des Roboters von einer Seite des Spielfelds auf eine andere, sowie die Planung und Ausführung der Armbewegung, um einen Stein zu greifen. Zusätzlich wird in dieser Arbeit auch erklärt wie die Vorwärts- bzw. Rückwärts-Kinematik im vorliegenden Fall gelöst wurde.

Das größte Problem an dieser Aufgabe war, dass nur interne Sensoren des Roboters verwendet werden durften. Aus diesem Grund wurden die beiden Kameras im Kopf des Roboters zur wichtigsten Informationsquelle für das Framework. Der Einsatz der Kameras wurde allerdings durch den Umstand erschwert, dass Bilder, die während einer Bewegung aufgezeichnet wurden, mit sehr starkem Motion Blur belastet waren, sodass keine verwertbaren Informationen aus den Bildern gezogen werden konnten. In der Vergangenheit wurden bereits mehrere Ansätze präsentiert, um mit Nao Brettspiele zu spielen, jedoch wurde dies noch nicht mit einem vergleichbaren Setup realisiert.

Die beiden anderen Bereiche werden in separaten Arbeiten behandelt. Sämtliche Probleme, Lösungsansätze und Details aus dem Bereich der visuellen Wahrnehmung der Umgebung können in [3] nachgelesen werden. Eine detaillierte Beschreibung bezüglich des eingebauten Frameworks für die Spiellogik, das als Interface für die dahinter liegende Spieledatenbank fungiert, werden in [19] erläutert.

Acknowledgement

I would like to thank my parents for supporting me during my entire studies and during the presented project. I would also like to thank Lena Hesse that she patiently listened to me when I didn't stop talking about the project, and her family for support and discussions on various topics. I would like to thank Gerald Steinbauer for giving me the opportunity to work on this project and for providing interesting ideas and concepts whenever needed. Furthermore, I would like to thank my proofreaders for their time, and of course I would like to thank all my friends who shared the past six years at university with me.They were a great support throughout my studies and also helped me during my master thesis by discussing the project with me and providing ideas on how to solve certain problems. Finally I would like to thank Sven Bock who worked on the vision part of the project for the great cooperation.

Roland Klöbl
Graz, 2012

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Place, Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Ort, Datum Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of humanoid robots has always been of great interest and fascination for researchers as well as for people without a technical background. Especially in movies and video games, humanoid robots are encountered in several cases. The audience always finds these robots fascinating, regardless of the purpose they are designed for.

Designing a humanoid robot itself has already been an interesting task to achieve. Due to the design these robots are expected to be able to act in typical human environments. However, researchers face a lot of problems when designing and trying to operate such robots. This includes problems based on the perception of the environment as well as acting and manipulating objects in a typical human environment. When it comes to humanoid robots, it is also of great interest to keep the movements smooth and as close to the human model as possible [8].

This master thesis focuses on the use of the robot Nao manufactured by Aldebaran Robotics. This robot has been used within the Robocup (RC) for a couple of years now. The Robocup is an organization where researchers can present their work by accomplishing several tasks. These tasks include playing soccer, navigating through cluttered environment, as well as dancing and others. Within the last couple of years, Nao has been used in order to play soccer in the humanoid league where this kind of robots are used to play soccer against each other. Thus Nao has been widely used for research which makes it obviously a good choice for further work on a related topic. The goal of this thesis was to generate a framework for the Nao robot in order to successfully play the board game Nine Men's Morris. This was a very interesting task to perform because it involves a lot of different problems to successfully achieve the goal. These problems can be grouped into three different tasks:

- Vision Task
- Game Knowledge Task
- Mobile Manipulation Task

The vision task includes the entire perception of the robot's environment. Since Nao is only capable of using one of its two cameras at a time, and moreover the regions of vision of both cameras do not overlap, there was no possibility for using stereo vision. Stereo vision is a technique which generates three dimensional information based on the data of two separate images with overlapping areas. The lack of other sensors in order to precisely localize the robot in a given environment made this task particularly interesting. Apart from the determination of the robot's current position, other problems needed to be taken into account comprising the challenge of visual occlusion and game state recognition. However, the present thesis does not provide detailed information concerning this task, since it was discussed in the master thesis written by Bock [3].

The game knowledge task refers to the problem of being able to actually play the game in terms of understanding the rules. Generally, there are two possible approaches which can be used for solving this problem. The first approach would be to generate Artificial Intelligence (AI) by formulating rules, and thus infer appropriate follow-up moves to a given game situation. The second approach would be to analyze the game in detail. This includes analyzing every possible game situation and evaluating possible follow-up moves in terms of how many moves would be necessary to win the game. The second possibility seemed promising for the task because a realization of various difficulties for the AI would be easy to achieve, just by taking e.g. the third best move instead of the best. Moreover, the analysis of the entire game was also a challenging task. The results of the game analysis are stored into a database and may be accessed through a special framework which is described in detail in [19].

The final task concerns the mobile manipulation of the robot. Since Nao has previously been used for playing soccer, the given task was of great interest because it enables a form of control the robot's arms which had not been necessary in any previous situation. The mobile manipulation task also involved to solve the forward and inverse kinematics of the robot as well as collision-free navigation through a known environment. This thesis especially focuses on the mobile manipulation task.

The master thesis is organized as follows: The next section illustrates the motivation for discussing the given topic. After that the thesis focuses on the problem statement itself. This includes a description of the game environment and some of the challenges which had to be taken into account in order to achieve the final goal of playing Nine Men's Morris. The sub-sequent part provides some background information about the techniques and concepts used in order to play this game. These include some important basics of vector analysis as well as a description of the Denavit-Hartenberg parameters which were used in order to solve the forward kinematics problem. This section also features a detailed description of the robot including available sensors and actuators. Moreover, the chapter contains a short description of the Robot Operating System (ROS) framework which was used for the implementation of the given framework. The chapter also informs about the board game Nine Mens' Morris. The next section focuses on related work, also including a discussion on the presented concepts and how they were incorporated into the framework or not implemented for the reasons listed. Section 5 describes the concepts incorporated into the framework in detail and also provides an overview of how the framework is organized. Finally an experimental evaluation of performed tests with the framework are presented, including a conclusion and an outlook concerning possible improvements and future concepts such as playing other board games as well.

1.1. Motivation

Playing games has always played an important role in human history. Their diverse purposes range from entertainment to social interaction and even education. Playing a game is all about exploring different strategies, analyzing the opponent's moves, and trying to create a situation leading to victory. Thus the field of playing games has also been a very interesting field for researchers of different areas including artificial intelligence and robotics. In terms of AI, it has been of interest because it focuses on problem analyses and proper reactions based on a given set of rules. However, in terms of robotics, different tasks need to be managed on top of that, including environmental perception and physical execution. Within the Robocup, a soccer league was established especially for humanoid robots. In this league, robots are used in order to autonomously play soccer against each other. Researchers all over the world work on different approaches to improve the skills of their robot team and therefore win the league. In the field of robotics, artificial intelligence is a key aspect as robots require a certain expertise to successfully interact. Nevertheless, there are numerous further challenges connected to the denoted topic [14].

The main motivation concerning the given problem was to fruitfully combine tasks of different areas in order to retrieve a viable solution for playing Nine Men's Morris. As already mentioned in the introduction, these tasks were the following ones:

- Vision Task
- Game Knowledge Task

- Mobile Manipulation Task

The complexity of combining the mentioned tasks to generate a framework for finally achieving the goal of playing Nine Men's Morris was a great motivation throughout the project. The robot incorporated in the presented framework was Nao. This robot was chosen because it is widely used in the field of research, it looks appealing and of course it is physically capable of solving the task.

Previously a robot arm was able to successfully play Pylos [1] against human opponents. The setup used an industrial robot arm in order to manipulate the game tokens and thus it did not look very appealing. Nao seemed to be the ideal choice in order to generate a human like environment and in addition generate a more attractive setup as well. Nao is also able to speak so it would be possible to incorporate a vocal human robot interaction based on the current game state. The robot would then resemble humans even more.

Another interesting aspect of playing games with a human opponent is the social interaction as such. In society, board games have gradually been replaced by computer games. And even if some computer games feature chats in written form or via headsets, it is still a big difference to real-life social interaction. The possibility of playing games with a robot companion is therefore great for several reasons. It would for example be possible to use the robot as companion for elderly people who live on their own, or it may act as a board game trainer for example in chess. The chess grandmasters have already been beaten by computers like Deep Blue [10]. Thus the robot would be a good training companion for players at any level with an appropriate AI. It would further be possible to train certain scenarios over and over again and thus human reactions may be examined via the analysis of the robot's actions. Achieving such goals with a real life opponent would also be great because board game training would not require a personal computer (PC) anymore.

Furthermore, playing a board game is also a small step to fully enable Assisted Ambient Living (AAL). In our aging society a lot of elderly live on their own. This number is even expected to rise in the next years. Considering this fact, there is a certain demand for technical systems in order to ease the life of these people. Robots would be a viable choice to achieve this goal. However, there are still a lot of issues which need to be solved so that robots are physically able to act in a human-like environment. AAL does not only address support services such as help with carrying certain objects but also the social aspect. If it was possible to create humanoid robots which are able to master daily human tasks, they would be expected to also incorporate social interaction and thus help people to not feel alone. As mentioned above, this aspect was another motivation for the given project.

Chapter 2

Problem Statement

2.1. Goals

The overall goal of this project was to generate a framework that enables the robot Nao to autonomously play the board game Nine Men's Morris. This goal was divided into several subgoals in order to reach the overall goal step by step.

The most important goal in the first place was to create an environment for the robot in which it is capable of fulfilling the task. Therefore the first step was to find viable dimensions for the game board. To achieve this goal, several limitations had to be considered. The most important limitations in this case were the dimensions of the robot itself, especially the dimensions of the robot's arms. These limitations made clear that the robot needed to be able to walk around the board in order to reach the different stones. If the dimension of the game board had been chosen smaller, they would have been too small for the robot's hands. The robot would not have been able to grasp single stones as it has big and simple designed hands.

Figure 2.1 shows the basic setup of the robot's environment. All values are provided in millimeter. The final dimensions of the game board are 227 mm height and 376 mm side length. Due to the placement of the table legs, the robot is able to sit by the table in the center of each side having its feet placed under the table. Being close to the board, the robot is able to reach every position on one side of the game board although the game board's dimension were defined relatively big.

In the given project the robot's only sensor in order to re-localize itself was a camera. Since Nao is only capable of using one of its two cameras at a time and the regions of vision do not overlap, it has been required to use an Augmented Reality Marker (ARMarker) for localization purposes. Therefore another big issue concerning the dimensions of the game board was the marker placement and the recognition of the stones. Since the marker is required for localization purposes, the risk of not being able to detect the marker because of token occlusion had to be minimized. In order to achieve this, the marker was placed on a wooden plate in the middle of the game board. Furthermore, the quality of the localization by using a marker highly depends on perspective distortion meaning the slant of the camera towards the marker. The closer this angle is to 0°, the more precise the result of the localization. This requires the table height to be as small as possible in order to maximize that angle when Nao stands close to the table. In our case the Augmented Reality Toolkit (ARToolkit) was used for marker detection [13].

Another important point concerning the design of the environment was the ability to detect stone positions. In order to detect stones in general a color based method was used. The computation of a real three-dimensional position for each stone requires each token to be detected separately and thus tokens shall not overlap in the segmented camera pictures. This problem also concerns the board's dimensions. Therefore the possibility of stone overlapping, the accuracy of marker information as well as the arm

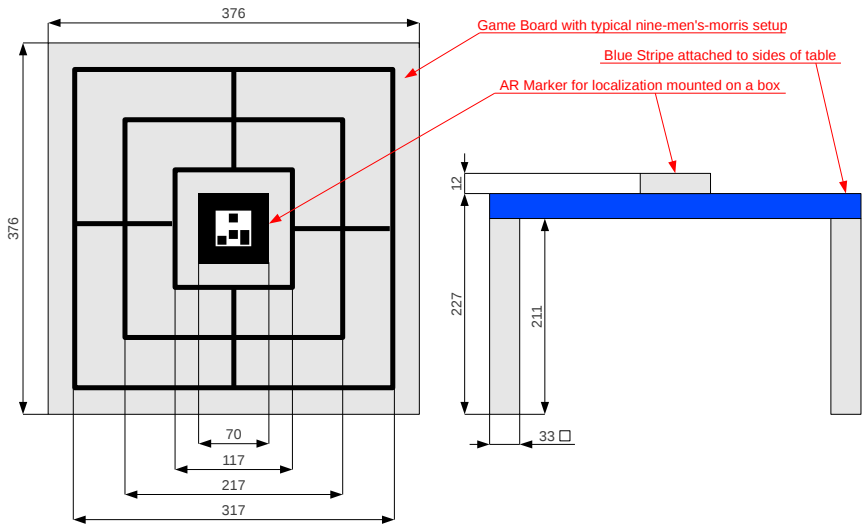


Figure 2.1.: Game setup from Nine Men's Morris

workspace of the robot needed to be taken into account in order to find appropriate dimensions for the board involved.

Apart from the design of the environment, subgoals concerning the robot's movement and perception of the environment were defined. The subgoals concerned with mobile navigation were defined as listed below:

- Find a method to compute inverse kinematics: This is required to perform a grasp. In order to grasp a token, the robot needs to know where the token is. But in fact the robot only deals with joint values. The inverse kinematics solves the problem of matching a certain goal position with corresponding joint values for each required joint.
- Navigate an arm to the point of interest without colliding with the game board, the robot itself, or any of the stones.
- Navigate the entire robot from one side of the table to another in order to grab a stone.

The subgoals regarding the vision and thus the perception of the environment are the following:

- Enable self-localization by using a marker.
- Retrieve the current game state.
- Calculate three-dimensional positions for each of the stones in order to grab them.

Even though the goals mentioned above are strictly separated into goals concerning vision and goals concerning mobile manipulation, the achievement of each goal always required a combination of motion and vision tasks.

2.2. Challenges

This section will outline the major challenges that needed to be dealt with during the implementation. This will also include some of the challenges which are mainly concerned with the vision. The solution for these will however not be presented here since they were discussed in detail by Bock in [3].

The following list gives an overview about the most important problems concerning the given task:

- Marker detection problems
- Occlusion of game tokens
- Detection of token position
- Solving the inverse kinematics
- Walking without online sensors
- Arm control omitting collisions
- Low accuracy of the odometry
- Limitations in robot control

2.2.1. Marker Detection Problems

One of the most important aspects of the entire project was to retrieve precise information about the current position of the robot in respect to the game board. This had to be guaranteed, otherwise no action would have been possible to perform without the danger of collisions. Therefore it was absolutely vital to minimize the risk of possible marker occlusions by game tokens. This was partly taken care of by mounting the marker onto a box of 12 mm height as mentioned above. However, it turned out that the marker cannot be detected if the robot stands too far away from the table. During the robot's walk, a scenario featuring such big distances to the game board may easily occur due to inaccuracy concerning walk executions. This problem was addressed by attaching a blue stripe to each side of the game board as shown in Figure 2.1. If the marker cannot be detected, the robot searches for the blue stripe and walks towards it. Finally it tries to find the marker again assuming that the chances for recognition have increased since the robot assumes to stand closer to the table than before.

2.2.2. Occlusion Of Game Tokens And The Token Position Problem

In order to successfully play Nine Men's Morris, it is necessary to exactly recognize the current game state. In the given case, the tokens are detected by their colors. In order to decrease the influence of changes of light the tokens were segmented by using the Hue-Saturation-Value (HSV) color model instead of the standard RGB color model. In case of tokens occluding each other, several tokens may be interpreted as one by the color detection. In order to recognize such situations, the size of the tokens was bound to a given value. Given that size and a certain distance between the camera and the token, an average pixel count in the image for a token can be computed. However, if the pixel count of any given token is significantly above this value, it is likely that two tokens have accidentally been recognized as one. In general this is not problematic for the game state estimation itself as shown in [3], but it is indeed relevant in terms of real position estimation.

The problem is that the current token position is estimated based on the color segmented images of the game board. The first step in recognizing the tokens is to generate a binary image which contains the tokens of one player in white while the rest of the image remains black. Then the center of each white area is estimated and finally this position is transformed into 3D space via camera transformation. Read [3] for further details. If two tokens actually form one center, the center of the white space will actually lie somewhere between these two tokens. Thus the robot would try to grasp a stone by placing its hand between the two and consequently it will not be able to grasp either of them.

In this case the problem is just a theoretical one. If the robot stands close to the game board, its camera angle is always close to 0° when the robot looks at the side of the game board where it actually stands. This means that the risk of occlusions is extremely low in this area compared to the opposite side of the game board where the risk of occlusion is considerably higher. Since the robot is not capable of grasping a token on the opposite side of the game board, it needs to walk there before grasping it. When the robot arrives at

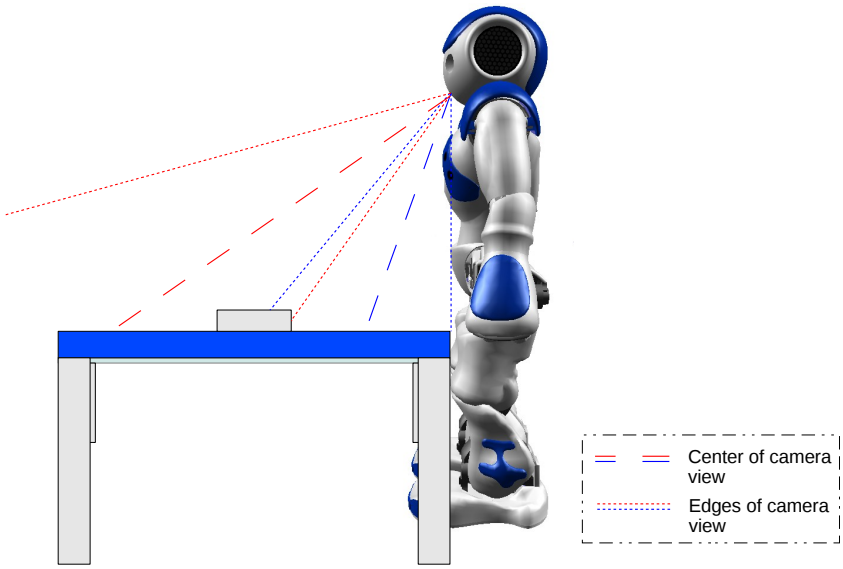


Figure 2.2.: Issue of occlusion depending on camera angle

the chosen side it scans the field again in order to compute the correct 3D position of the token, but now the camera angle is somewhere close to 0° and therefore occlusions are very unlikely.

Figure 2.2 illustrates the circumstances mentioned above. The dashed lines illustrate possible view centers of the camera perspective whereas the fine dashed lines mark the edges of the camera picture. It is obvious that the perspective distortion concerning the game board will not be that much of an issue in case of the blue camera center compared to the red camera center where occlusions are more likely to appear.

2.2.3. Solving The Inverse Kinematics

As mentioned above the computation of the inverse kinematics is required to compute the robot's joint angles based on a given point to reach. It would not be possible to automatically grasp a given object without a solution of the inverse kinematics. In general, solving kinematics of a humanoid robot is considered to be rather complicated and time-consuming because of their numerous degrees of freedom. However, solving the inverse kinematics is considerably easier if one simply focuses on either the arms or the legs instead of the entire body. Each arm of the robot features six joints whereas one of them is just used in order to open and close the hand and thus it is not relevant for the kinematic solution. The remaining five joints are placed in two separate links, the upper arm and the lower arm. The same applies to the leg. Nao's six leg joints are placed within two links, the upper and the lower leg. In this case the inverse kinematics may be computed by the intersection of two spheres as discussed in detail in Section 3.3.3. By splitting the computation in two separate smaller computations and linking them by an iterative trial and error approach, the inverse kinematics could be solved with relative ease. The algorithms used will be discussed in detail in Section 5.

2.2.4. Walking Without Online Sensors

Any time a robot moves, collision avoidance is very important. Usually there are several ways in order to achieve this. The first scenario is that the robot operates in a completely unknown environment. In this case the robot has to be equipped with certain sensors which provide online information about the environment. With the help of this information the robot may be able to avoid occurring obstacles. The second scenario is that the robot operates in a semi-static environment. This means that most of the obstacles in the robot's environment are known and over time only small changes are possible. The possible changes need to

be either predictable or noticeable by the robot. In the given task we assumed to have the latter form of environment, which was necessary because of the lack of online sensors on the robot. Furthermore, the robot is only able to re-localize within the environment by looking at the ARMarker placed in the center of the game board. During the walk it is not possible to detect the marker with the required accuracy because of delay times throughout the system architecture and the camera images themselves which are blurred by the motion. Techniques in order to reduce motion blur were tried out with limited success because the pictures were not sharp enough to retrieve information at the required level of accuracy.

As a consequence the entire walk of the robot when switching board sides was realized based on the robot's odometry data. In order to improve the accuracy of the odometry, visual odometry was incorporated into walk execution. The problems concerning walking will be presented in detail in Section 5.

2.2.5. Arm Control Omitting Collisions

In order to successfully grasp an object, the entire robot needs to be controlled in a certain way to omit collisions. Especially when it comes to humanoid robots, this may be rather complex. In general the arm navigation features challenges such as keeping the robot's stability as well as the entire body control and of course the requirement to omit collisions at all times. However, the difficulty of the task depends on the robot itself, on its environment and on the requirements concerning a certain grasp position.

The first problem which has to be addressed in terms of grasping is to define a specified pre-grasping posture. Finding a pre-grasping posture defines the problem of finding a viable posture for the robot's hand in order to successfully grasp the desired object. In the given scenario the possibilities for this posture are limited because the tokens always lie on the table and thus only postures featuring grasps from above are viable. The defined posture in this project is that the hand is required to have a certain pitch of 25° with respect to the *xy* plane.

Of course the solution of the inverse kinematics is also included in the problem of arm control since arm navigation without knowledge of the goal joint angles is not possible. The inverse kinematics needs to incorporate the limitations due to the pre-grasp posture, because the computed joint values always refer to a certain final posture of the arm.

As mentioned above the major task to grasp a token is to properly control the movement of the arm. Depending on the scenario, the required trajectories to control the motion may be computed either dynamically or statically. The dynamical approach is required whenever environment is assumed to be dynamic as well. In this case the robot has to adapt its desired trajectory due to occurring obstacles which may block the trajectory execution. However, the given scenario does not feature possible upcoming obstacles and thus there is no real need for an algorithm of this type. Apart from that the realization of such an algorithm would not have been possible because of the limitations in robot control and the problem of not being able to detect upcoming obstacles due to the lack of available sensors. Thus a semi-static trajectory generation was applied which is capable of safely navigating through the known environment. The algorithm is discussed in Section 5.

2.2.6. Limitations In Robot Control

Another goal of the project was to use the Nao's built-in interface which is called NaoQi. Therefore the presented framework can be used with any Nao robot without the need to change Nao's standard software configuration. However, the interface provided to control the robot also has its limitations. For example it is not possible to create an arm movement along a trajectory by controlling the velocities of each joint over time. The path has to be previously computed and provided in form of angle-time pairs. In the given case this form of trajectory move has turned out to be sufficient because the grab situations always look similar and it is not assumed that obstacles apart from the game board occur in the environment. In fact online collision avoidance would not have been feasible due to the lack of online velocity control.

Chapter 3

General Information

The following chapter provides background information concerning the game Nine Men's Morris, the robot as well as some information concerning the incorporated mathematical techniques needed to compute the forward and inverse kinematics.

3.1. Nine Men's Morris

Nine Men's Morris is a popular game which is also known as Mills. Picture 3.1 shows a random game state in order to provide an idea of the game setup. At the start of the game, the game board is empty and each player is equipped with nine tokens. Like in many other board games, the players' tokens have different color. The players alternately make their moves.

The game has three phases. In the first phase each player is allowed to place one token on any free position on the board. The second phase starts as soon as all tokens are placed on the board. In this phase each player is only allowed to move one of his/her tokens to a nearby free position. Throughout the entire game the player who is able to place three tokens in a row, like the white player in Figure 3.1, is allowed to take away one of the stones of the opponent. However, it is not allowed to take tokens from any combination of three tokens in a row. The third phase of the game starts when a player has only three tokens left. This player is further allowed to jump with his/her stones to any free position on the game board without the restriction of just moving to nearby positions. The goal of the game is to leave the opponent with less then three tokens or block every possible move during the second game phase.

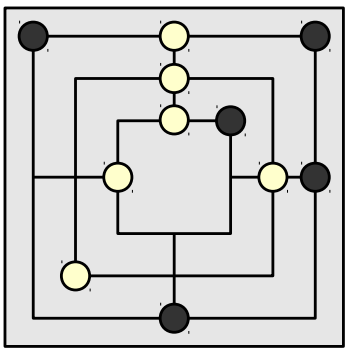


Figure 3.1.: Nine Men's Morris board layout and random token placement

3.2. The Humanoid Robot Nao

As indicated in Section 1, Nao has been used by a lot of research teams for different purposes. The most popular usage for this robot, however, was its application within the humanoid Robocup soccer league. The robot comes with several sensors and 25 actuators in order to perceive and interact with its environment. Furthermore, Aldebaran Robotics also provides software to operate the robot via a simple interface. With this software tool users can generate small programs by drawing execution objects in a certain order. In this case an execution object represents commands like stand up, sit down, talk or perform face detection. Apart from moving and providing sensor data, the robot can also speak English and French, making it even more human-like. When standing straight, the robot has a total height of 573 mm and a total width of 273 mm. Its links are organized in a human-like way, and detailed information concerning their dimensions can be found in the following two pictures.

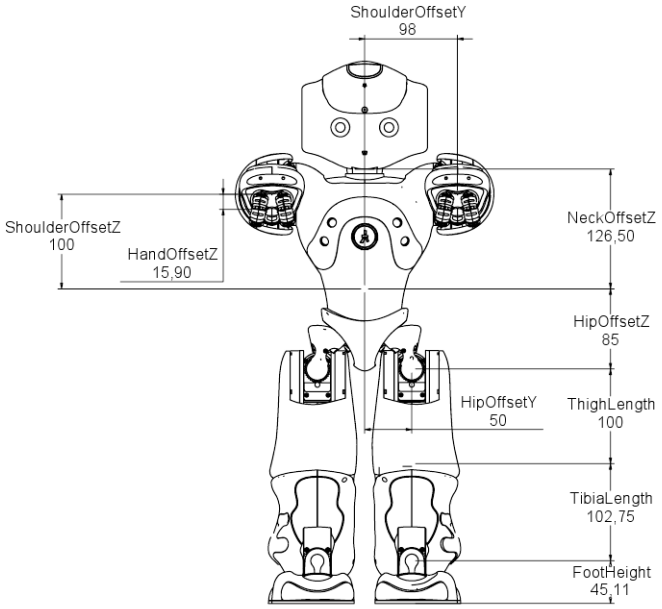


Figure 3.2.: Nao's basic link information [2]

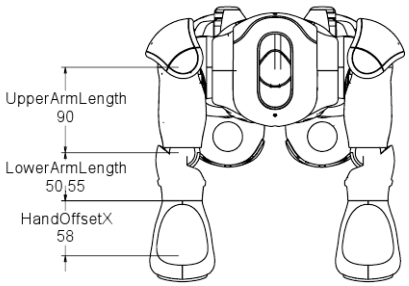


Figure 3.3.: Nao's basic link information [2]

3.2.1. Actuators

The robot has 25 joints. Two of them are used to control the head movement, six are placed within each arm and five in each leg. One joint is located in the hip, and affects both legs at the same time. Since the joint limits for both sides of the robot are similar apart from the leg, only the kinematic chains of the left side will be presented here. The correct joint angles for the right side can be found in the documentation of NaoQi [2].

As already mentioned the arm features a total of six joints. Two of them are located in the shoulder, another two of them are located in the elbow. Another one is placed in the middle of the forearm. One joint is located in Nao's wrist and it is used to open and close its hands. The entire kinematic chain for the left arm is illustrated in Figure 3.4.

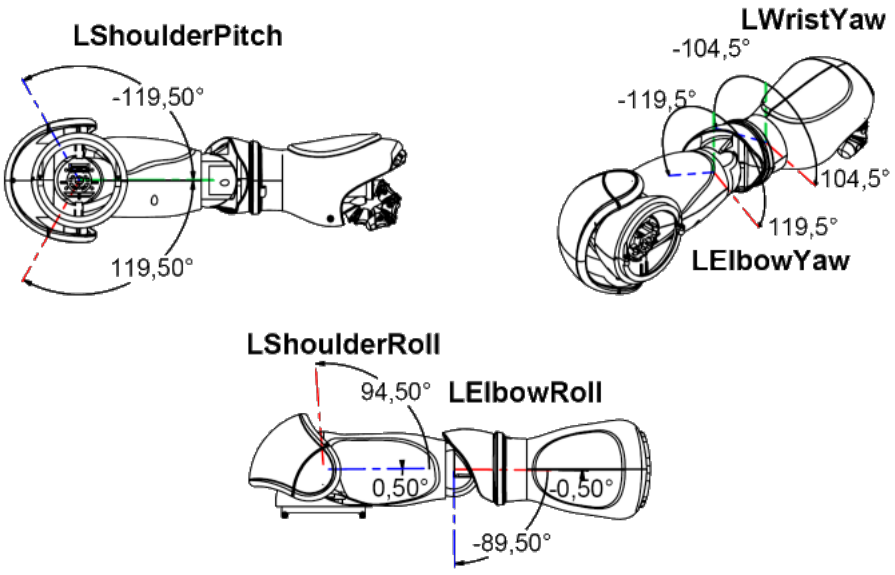


Figure 3.4.: Kinematic chain in Nao's left arm [2]

The legs of the robot also feature six joints, whereas the first joint, which is located in the hip and is referred to as LHipYawPitch and RHipYawPitch affects both legs at the same time. Apart from the HipYawPitch Joint Nao comes with two more joints located in the hip as well as one additional joint in the knee. Two joints on each leg are placed in Nao's ankle. The setup of these joints is shown in Figure 3.5 and 3.6 respectively.

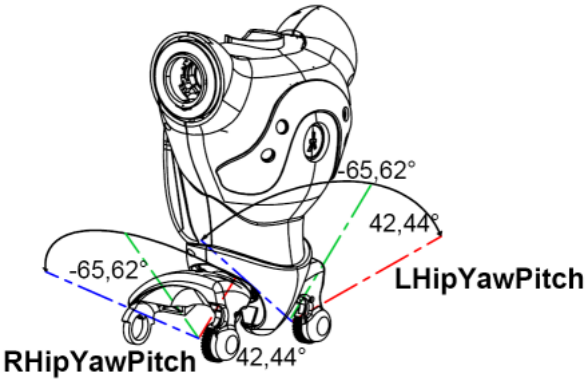


Figure 3.5.: HipYawPitch joint which affects both legs simultaneously [2]

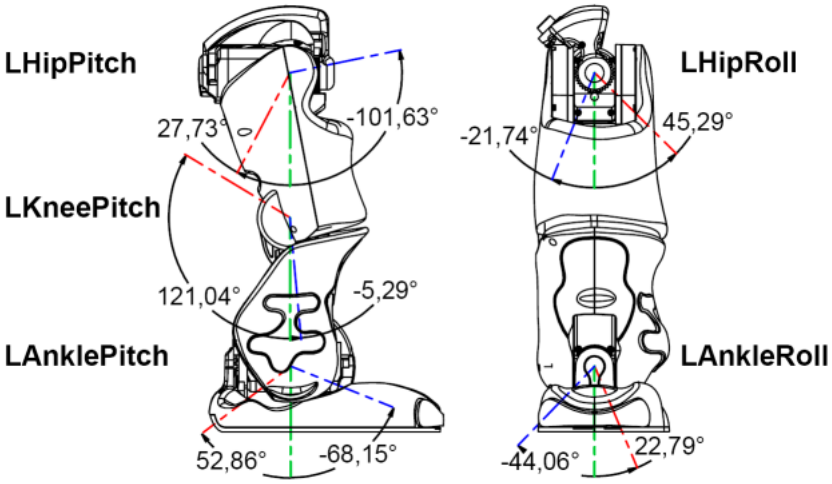


Figure 3.6.: Kinematic chain in Nao's left leg [2]

The last kinematic chain in the robot is the chain concerned with the head. This chain was especially important for the vision part of the project because Nao's cameras are mounted on its head.

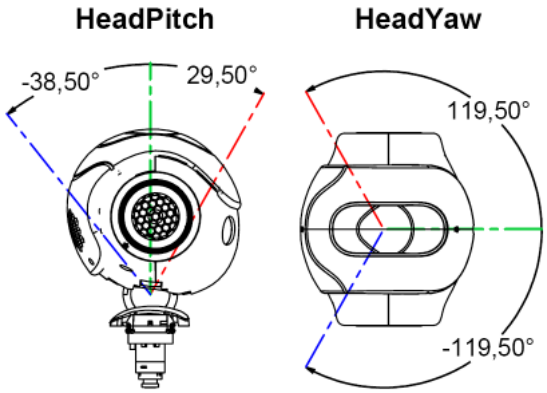


Figure 3.7.: Kinematic chain in Nao's head [2]

3.2.2. Sensors

In order to perceive its environment the robot is equipped with some sensors including force sensitive resistors on its legs, an inertial unit as well as two sonar sensors in its chest and two video cameras in its head. As mentioned above the most important sensors in the given case are the two cameras mounted in the head, shown in Figure 3.8.

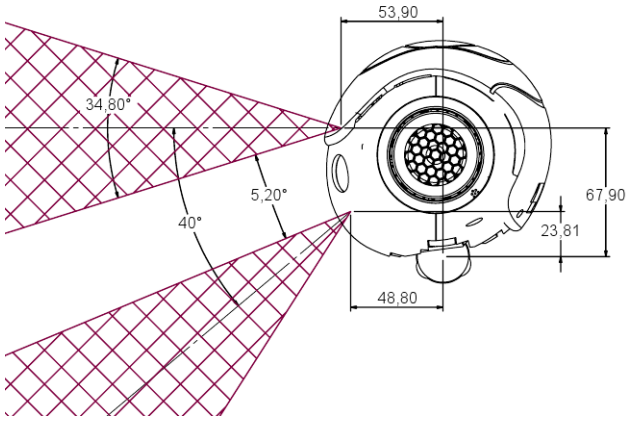


Figure 3.8.: Location and orientation of the cameras in Nao's head [2]

Apart from the already mentioned sensors, Nao also comes with eight buttons, which are used as simple robot-human interface. Three of these buttons are located in the head, one is located in the chest and the other ones are located on the front side of Nao's feet. The ones located on Nao's feet are primarily used for the walking behavior of the robot in order stop it when runs into an obstacle and thus prevent it from falling over.

3.2.3. NaoQi

The basic software installation on the robot also includes a program called NaoQi. This program represents the main interface for operating the robot. It includes methods for motion requests as well as providing access to sensor values. Since NaoQi provides some methods which really simplify certain tasks especially those concerning walking, it was decided to use it and thus not to change Nao's basic software setup.

Methods dealing with motion requests were grouped in the so-called ALMotion API. The most important methods of the ALMotion API concerning the given project are listed below.

Angle Interpolation

This function takes an array of joints and timed trajectories as parameters and executes them. It is used within the framework to execute the entire movement for grabbing a stone [2]. However, it also comes with a certain drawback. Each joint of the robot is bound to certain speed limits, meaning a given maximum speed limit and a given minimum speed limit. If the data sent to this function is not viable because the required speed to change a joint angle within a specified time is too high or too low the method does not report this to the user. In fact if the required speed is too low this may even result in violating the specified angle values instead of halting the execution when the required angle values are reached. This should always be kept in mind when using this function.

Angle Interpolation With Speed

In this case joints can simply be moved by specifying a goal angle and a desired speed relative to the maximum speed available for that given joint. This method was used whenever it was not necessary to follow a particular path e.g. for moving the head during the game board scan [2].

Walk To

This method turned out to be very valuable. It takes a position as parameter defined by x, y and θ relative to the current position as parameter and then moves the robot towards that point. In order to move, the function executes the built-in walking behavior. Given that it has not been required to plan the entire walking motion in terms of generating timed trajectories for each of the leg's joints. Furthermore, the execution turned out to be quite precise in most cases, making it a fairly good option for walking around the game board [2].

3.3. Kinematics

The term kinematics in general is always used in the context of movement. Important terms concerning kinematics are forward and inverse kinematics as well as kinematic chains.

Kinematic chains In general a kinematic chain is a set of links which are connected to each other by any type of joint. The most common type of joints in terms of robotics are revolute joints and prismatic joints. In fact most common robots solely feature revolute joints. This means that any robotic arm can be referred to as kinematic chain [9]. A humanoid robot usually features a lot of kinematic chains due to their redundancy. Such a kinematic chain for example may start in the left leg and end in the right hand. Another one may start in the left leg and end in the right one.

Forward kinematics Given a certain robot setup, and thus the kinematic chain of a robot, solving the forward kinematic task means that the position of a certain point relative to a given link of the robot is computed by providing the robots joint angles. In most cases the requirement is to compute the current position of the end-effector within a pre-defined environment coordinate frame. In general a coordinate frame is attached to each joint of the robot in order to solve that problem. Of course the transformation between the coordinate frames is related to the robot setup, but the exact placement of the coordinate frames with respect to the real position of the joint may be defined by the user. One method of defining the locations and orientations of the coordinate frames was introduced with the Denavit Hartenberg parameters which will be described in detail in Section 3.3.1. Of course the current position of the coordinate frames depends on the current position of the links, which means that their position depends on the current joint values of the robot.

Inverse kinematics The inverse kinematics describe the inverse task to the problem mentioned with forward kinematics. In this case a certain goal position in a pre-defined environment frame is given, and the joint values for the robot, enabling it to reach that point need to be computed. In addition some posture limitations concerning the goal position may be given. Such a limitation may be a certain angle restriction with respect to the environment frame. In general this task is significantly harder to achieve. The reason for this is that depending on the robot's setup, several solutions may be viable for achieving the goal depending on the number of joints and the overall degree of freedom of the robot. Since humanoid robots usually feature a considerably high degree of freedom due to their numerous joints, this task is especially hard to solve. In case of forward kinematics, there is always one solution whereas the inverse kinematics may feature a certain number of solutions, infinite solutions or no solution if the given point cannot be reached. Section 3.3.2 provides an example of how to solve the inverse kinematics of a robot featuring two links.

The following figure illustrates the kinematic chains featured within the Nao robot. In the given scenario the robot's torso is taken as basic coordinate frame and all other coordinate frames are introduced with respect to the torso. However, this does not restrict the computation possibilities in any form. In order to compute the current position of the left hand with respect to the coordinate frame of the left foot kinematic chain between left arm and torso as well as the one between torso and left foot need to be combined. The torso coordinate frame also represents the robot's main coordinate frame with respect to the environment. If the robot is re-localized in the environment, the transformation between the environment coordinate frame and the torso coordinate frame is set. Other transformations between any of the robot's frames are usually not affected by re-localization. The environment's coordinate frame is located in the center of the game board.

Figure 3.9 illustrates the mentioned coordinate frames concerning the robot. In this case each coordinate frame is labeled according to the joint name it represents. However the setup also features some coordinate frames that don not represent a joint, but are still somehow important for the project. The game board coordinate frame is placed in the center of the game board. In order to update the robot's position within the environment the transformation between the game board and the torso needs to be updated. The torso coordinate frame represents the base coordinate frame concerning the robot. This means that all the other coordinate frames concerning the robot depend on the position of this coordinate frame. This also implies that the torso coordinate frame does not represent a coordinate frame which sticks to a certain joint. In addition separate coordinate frames are introduced onto Nao's feet and hands. The coordinate frames for Nao's hands are introduced because these systems are very important for any form of grasping. Finally the coordinate frames for Nao's feet are introduced in order to compute the current position of the torso with respect to the ground.

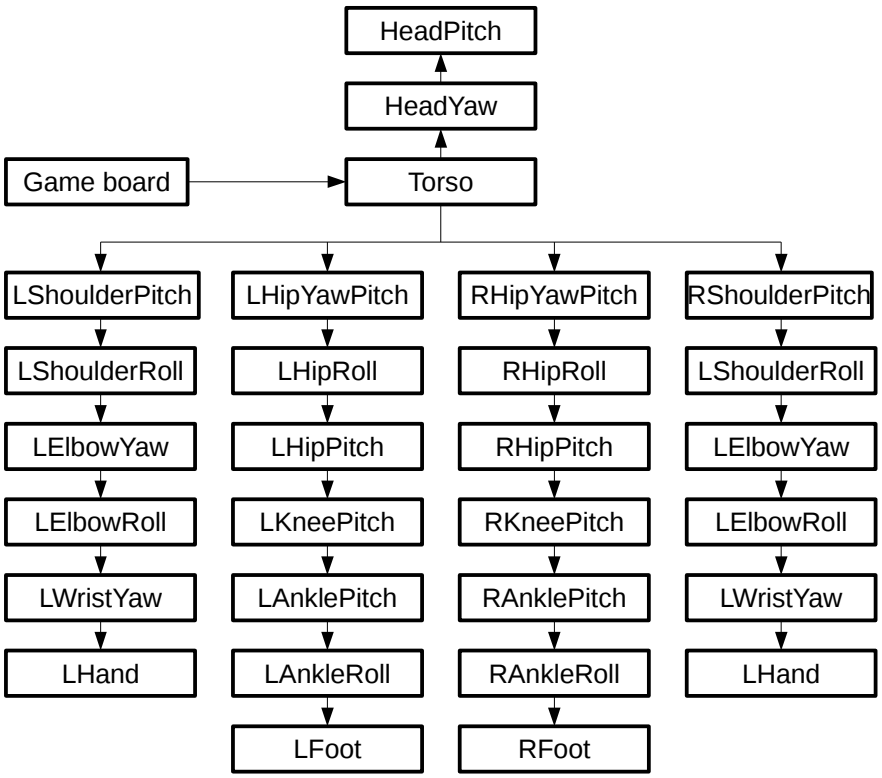


Figure 3.9.: Main coordinate frames in the project concerning the robot motion

3.3.1. Denavit Hartenberg Parameters

As mentioned above robots are usually built of kinematic chains, meaning that the robot consists of links which are connected to each other by joints. The joint enables relative movements of a certain type. The most common types of joints are prismatic joints, enabling linear movements, and rotational joints, which enable rotational movements. A good way of dealing with the robot's kinematic setup in terms of position computation is to define a separate coordinate frame for each of the robot's links. Usually the relative distances and rotation between the coordinate frames depend on the robot's hardware setup. It is obvious that there are numerous ways of introducing coordinate frames which meet this condition for a single robot setup. In terms of robotics a certain way of introducing these systems is widely used, namely the Denavit-Hartenberg parameters. These parameters provide one possibility of determining locations and rotations of coordinate frames within a kinematic chain of a robot. However, there are still degrees of freedom implemented in this method providing some flexibility about where the systems are exactly located. Hence various different sets of Denavit-Hartenberg Parameters can be found for a single kinematic chain. Each set of Denavit-Hartenberg parameters clearly describes a certain robot setup. The concept of Denavit Hartenberg parameters only supports robots featuring rotational and prismatic joints. In this case each joint provides only one degree of freedom and thus can be represented by a single axis and a joint parameter. In case of rotational joints the axis would be the axis of rotation and the joint parameter would be the angle of rotation ϕ. In case of prismatic joints the axis would be the axis of the prismatic movement and the joint parameter would be the moving distance Δ_d [12].

A set of Denavit Hartenberg Parameters contains the following values:

- α_i : Describes the angle between g_i and g_{i+1}.
- a_i : Describes the normal distance between g_i and g_{i+1}.
- d_i : Describes the distance between O_i and P_i.

In this case g_i refers to the axis of the ith joint. O_i is the origin of the coordinate frame corresponding with the ith joint and P_i is the base point of the common normal between g_i and g_{i+1}. Figure 3.10 illustrates the given parameters.

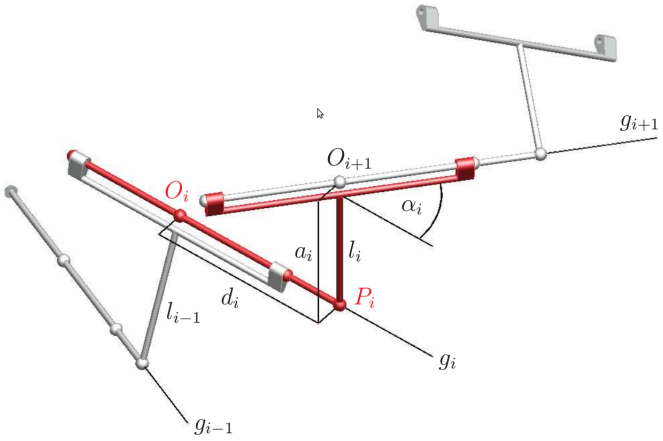


Figure 3.10.: Measuring Denavit Hartenberg Parameters [9]

Considering a robot featuring n axis the set of Denavit Hartenberg Parameters consist of

- α_i a_i with $i = 1,...,n-1$
- d_i with $i = 2,..,n-1$

Thus a total of $3 \cdot n - 4$ parameters need to be retrieved.

In order to make use of the Denavit Hartenberg Parameters for the computation of the forward kinematics problem, it is important to define the orientation of the axis for the coordinate frames. However there are

several possible ways to define the axis. One possible attempt would be to define the x-axis of a certain coordinate frame O_i to be identical with the corresponding joint axis of joint i. The z-axis however can be defined as the direction of the normal between g_i and g_{i+1}. Of course the last axis can easily be computed by computing the cross product of the other two axes [9]. However, the x- and the z-axis may also be switched. In fact the proposed Denavit Hartenberg Parameters for the Nao robot use the latter definition [2].

Transformation Basics

In order to use the Denavit Hartenberg Parameters it is necessary to have some basic knowledge about transformations and their mathematical representation in form of matrices. Further explanations and equations have been taken from [9]. In 2D any point can be represented by the following equation.

$$\mathbf{x} = x \cdot \mathbf{e_1} + y \cdot \mathbf{e_2} \tag{3.1}$$

$\mathbf{e_1}$ and $\mathbf{e_2}$ are the basic vectors of the coordinate frame. A translation adds an additional offset to a certain point. Thus the translation can be described as follows:

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.2}$$

In order to rotate a vector it is the best way to represent the vector in polar coordinates. Thus the vector components can be represented as

$$\begin{aligned} x &= r \cdot \cos(\alpha) \\ y &= r \cdot \sin(\alpha) \end{aligned} \tag{3.3}$$

where α is the current angle of the vector and r is its length. In order to execute a rotation by the angle ϕ the new coordinates will be

$$\begin{aligned} x &= r \cdot \cos(\alpha + \phi) \\ y &= r \cdot \sin(\alpha + \phi) \end{aligned} \tag{3.4}$$

Due to

$$\begin{aligned} \sin(\alpha + \phi) &= \sin(\alpha) \cdot \cos(\phi) + \sin(\phi) \cdot \cos(\alpha) \\ \cos(\alpha + \phi) &= \cos(\alpha) \cdot \cos(\phi) - \sin(\alpha) \cdot \sin(\phi) \end{aligned} \tag{3.5}$$

The new point $x*$, $y*$ is represented by

$$\begin{aligned} x^* &= x \cdot \cos(\phi) - y \cdot \sin(\phi) \\ y^* &= x \cdot \sin(\phi) + y \cdot \cos(\phi) \end{aligned} \tag{3.6}$$

Which can be further written in matrix form

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.7}$$

The matrix in 3.7 will be referred to as rotational matrix. Thus a 2D transformation which does not imply changing the lengths can be represented by a combination of a translation and a rotation.

$$\mathbf{R} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.8}$$

$$\mathbf{x}^* = \mathbf{d} + \mathbf{R} \cdot \mathbf{x} \tag{3.9}$$

R represents the rotational matrix mentioned above. This formula is still not very comfortable to use because it is not possible to merge the translational part of the calculation with the rotational part. A common way to deal with that issue and thus provide the probability of merging a rotation and a translation within one matrix is to introduce homogeneous coordinates. The concept of homogeneous coordinates introduces a third value per vector which just scales every other entry of the vector.

$$\begin{bmatrix} x_h \\ y_h \\ h \end{bmatrix} \text{ with } x_h = \frac{x}{h} \text{ and } y_h = \frac{y}{h} \tag{3.10}$$

The homogeneous coordinate h enables to merge the translation vector $\mathbf{d}$ and the rotational matrix $\mathbf{R}$ leading to the following equation. In the standard case the homogeneous coordinate h equals one for any vector which states a position, whereas its value is set to zero when it is used to define directions.

$$\begin{aligned} \mathbf{x}^* &= \mathbf{T} \cdot \mathbf{x} \\ \mathbf{T} &= \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0}^\top & 1 \end{bmatrix} \end{aligned} \tag{3.11}$$

Note that the translation vector $\mathbf{d}$ and the rotational matrix $\mathbf{R}$ are still the same as defined in 3.9.

When it comes to 3D vector analysis especially the rotation of a vector is a little more complicated. As mentioned above rotation in 2D just features one degree of freedom since the axis of rotation is always the virtual z-axis. In 3D the rotation matrix does not only deal with the rotation angle itself but also encodes the orientation of the rotation axis. In general a rotation in 3D space is defined by an axis a and an angle ϕ.

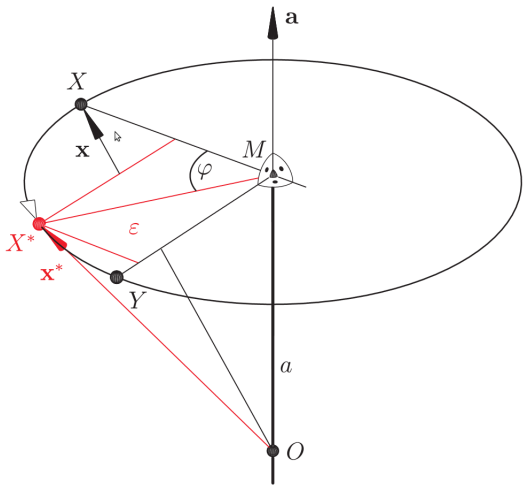


Figure 3.11.: Important values concerning a 3D rotation [9]

Figure 3.11 shows an analytical description of a rotation in 3D space. In this description x is the point that should be rotated by the angle ϕ. The result of the rotation is illustrated as x^*. In this case O is meant to be the origin of the coordinate frame. The following equations can be retrieved from the given conditions:

$$\begin{aligned} \vec{OM} &= \langle \mathbf{a}, \mathbf{x} \rangle \cdot \mathbf{a} \\ \vec{MX} &= \vec{OX} - \vec{OM} = \mathbf{x} - \langle \mathbf{a}, \mathbf{x} \rangle \cdot \mathbf{a} \end{aligned} \tag{3.12}$$

The point Y was chosen in a way that $\mathbf{a}$, $\vec{MX}$ and $\vec{MY}$ built up a valid coordinate frame in the plane ε of the rotation. Thus $\vec{MY}$ needs to be orthogonal to $\mathbf{a}$ and to $\vec{MX}$.

$$\vec{MY} = \mathbf{a} \times \vec{MX} = \mathbf{a} \times (\mathbf{x} - \langle \mathbf{a}, \mathbf{x} \rangle \cdot \mathbf{a}) = \mathbf{a} \times \mathbf{x} \tag{3.13}$$

Finally the solution vector x^* can be expressed via the vectors $\mathbf{a}$, $\vec{MX}$ and $\vec{MY}$.

$$\mathbf{x}^* = \langle \mathbf{a}, \mathbf{x} \rangle \cdot \mathbf{a} + \cos(\phi) \cdot \vec{MX} + \sin(\phi) \cdot \vec{MY} \tag{3.14}$$

which finally results in the following matrix:

$\mathbf{x}^* = \mathbf{R} \cdot \mathbf{x}$ with

$$\mathbf{R} = \begin{bmatrix} a_1^2(1-\cos(\phi)) + \cos(\phi) & a_1 a_2(1-\cos(\phi)) - a_3 \sin(\phi) & a_1 a_3(1-\cos(\phi)) + a_2 \sin(\phi) \\ a_1 a_2(1-\cos(\phi)) + a_3 \sin(\phi) & a_2^2(1-\cos(\phi)) + \cos(\phi) & a_2 a_3(1-\cos(\phi)) - a_1 \sin(\phi) \\ a_1 a_3(1-\cos(\phi)) - a_2 \sin(\phi) & a_2 a_3(1-\cos(\phi)) + a_1 \sin(\phi) & a_3(1-\cos(\phi)) + \cos(\phi) \end{bmatrix} \tag{3.15}$$

In 3.15 a_1 a_2 and a_3 are meant to be the x, y and z parameters of the rotation axis a.

This may seem complicated, but the rotation matrices become simple when it comes to rotation around any of the basic axes x, y or z. The following equations show the rotation matrices for these special cases.

Considering a rotation around the x-axis, the rotational matrix $\mathbf{R}$ is

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.16}$$

A rotation around the y-axis will be represented as:

$$\mathbf{R}_y = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{3.17}$$

Respectively the matrix for rotations around the z-axis looks as follows.

$$\mathbf{R}_z = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

In order to represent an entire transformation within one matrix again homogeneous coordinates are used. In fact introducing homogeneous coordinates works the same way in 3D and 2D and finally results in a transformation matrix $\mathbf{T}$.

$$\begin{aligned} \mathbf{x}^* &= \mathbf{T} \cdot \mathbf{x} \\ \mathbf{T} &= \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix} \end{aligned} \tag{3.19}$$

Of course this time $\mathbf{R}$ is a 3×3 matrix and $\mathbf{d}$ is a 3D vector.

Denavit Hartenberg Parameters And Forward Kinematics

Once the Denavit Hartenberg parameters are found and the coordinate frames are properly placed on the robot's links it is a straight forward problem to solve. By using the definitions concerning the directions of the coordinate frames mentioned above it is obvious that the coordinate frame O_i may be transformed into frame O_{i+1} by translating it d_i along its z-axis and a_i along its x-axis and finally rotate it by the angle α_i around the x-axis. However, the current position of the robot can be taken into account by a rotation around the z-axis in case of a rotational joint or by a translation along the z-axis in case of a prismatic joint. The entire transformation from coordinate frame O_{i+1} into O_i is computed via two transformation matrices. The first matrix will be referred to as $\mathbf{R}_{z(\phi_{i+1})}$ and includes the transformation concerning the robot's $(i+1)$th joint whereas the second matrix will be referred to as C_i and contains the transformation due to Denavit-Hartenberg rules. However the entire transformation may be computed as follows:

$$\mathbf{X}_i = \mathbf{C}_i \cdot \mathbf{R}_{z(\phi_{i+1})} \cdot \mathbf{X}_{i+1} \tag{3.20}$$

with

$$\begin{aligned} \mathbf{C}_i &= \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and} \\ \mathbf{R}_{z(\phi_{i+1})} &= \begin{bmatrix} \cos(\phi_{i+1}) & -\sin(\phi_{i+1}) & 0 & 0 \\ \sin(\phi_{i+1}) & \cos(\phi_{i+1}) & 0 & 0 \\ 0 & 0 & 1 & \Delta_{i+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.21}$$

In the given case only rotational joints are placed along the kinematic chains and thus Δ_{i+1} is zero. Considering a robot with six joints usually seven coordinate frames are used to represent the robot. The first coordinate frame is the base of the robot Σ_0 and the other six coordinate frames are sticked to each link of the robot. A transformation from the end-effector's coordinate frame into the base coordinate frame can be done by just applying the transformations step by step as shown below:

$$\mathbf{X_0} = \mathbf{C}_0 \cdot \mathbf{R}_{z(\phi_1)} \cdot \mathbf{C}_1 \cdot \mathbf{R}_{z(\phi_2)} \cdot \mathbf{C}_2 \cdot \ldots \cdot \mathbf{C}_5 \cdot \mathbf{R}_{z(\phi_6)} \cdot \mathbf{X}_6 \tag{3.22}$$

In this case ϕ_i represents the current angle of the ith joint of the robot. $\mathbf{X}_6$ is a given point in the end-effector's coordinate frame, and $\mathbf{X}_0$ is the given position with respect to the base coordinate frame Σ_0.

Table 3.1 illustrates the set of Denavit Hartenberg parameters for the Nao's left arm which was used in the given project. Note that the set features an additional parameter θ. This parameter is used to incorporate

an additional rotation around the z-axis. This rotation has been introduced in order to properly correct the basic orientation of the shoulder roll joint to be identical with the orientation proposed by the NaoQi. The table also features the parameter l_1 which refers to the robot's upper arm length.

$$l_1 = 0.09m$$

Joint Name	Joint Index i	a_i	α_i	d_i	θ_i
Shoulder Pitch	0	0	$-\frac{\pi}{2}$	0	0
Shoulder Roll	1	0	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$
Elbow Yaw	2	0	$\frac{\pi}{2}$	0	0
Elbow Roll	3	0	$-\frac{\pi}{2}$	l_1	0
Wrist Yaw	4	0	$\frac{\pi}{2}$	0	0

Table 3.1.: Denavit Hartenberg Parameters for Nao's left arm

The setup of Nao's kinematic chain implies the hand to be also located in the coordinate frame of the Wrist Yaw joint. In order to compute the current position of the hand only offsets need to be added to the wrist's coordinate frame.

If the robot's arms are assumed to be in their initial position illustrated in Figure 3.13 the corresponding orientations of the real Denavit Hartenberg coordinate frames acting behind the model are oriented as shown in Figure 3.12.

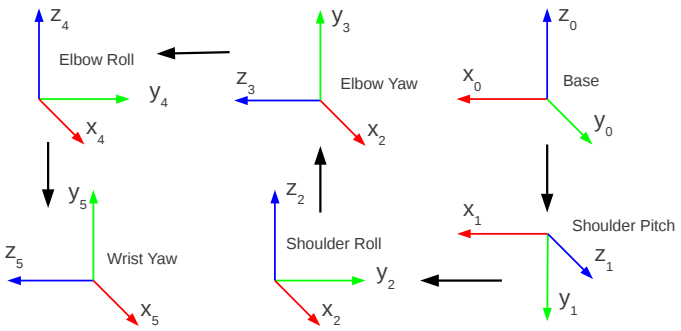


Figure 3.12.: Denavit Hartenberg coordinate frames in the left arm

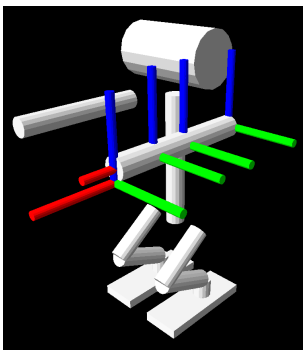
The given framework also incorporates a model of the robot. The coordinate frames in each of the joints within the model are not oriented the same way the coordinate frames of the Denavit Hartenberg Parameters are. However, in comparison to the different orientations of the coordinate frames concerning the Denavit Hartenberg Parameters, the placement and orientation of the coordinate frames within the model are rather intuitive. Hence the framework also provides the possibility to specify a certain transformation between the coordinate frame in which data is provided and the actual corresponding coordinate frame of the Denavit Hartenberg parameters. Thus it is possible to use the orientation and position of the coordinate frames provided by the model for the computation of the forward kinematics. This is obviously also an advantage in terms of computing the current position with respect to a certain coordinate frame. Without this feature each computation including the forward kinematics would require to know the orientation of the corresponding Denavit-Hartenberg coordinate frame. On top of that the internal representation of the coordinate frames would not match the ones of the model, which are visible to the user.

The following example illustrates the required transformation steps in order to compute the position of a point P, given in wrist yaw coordinates, in shoulder pitch coordinates.

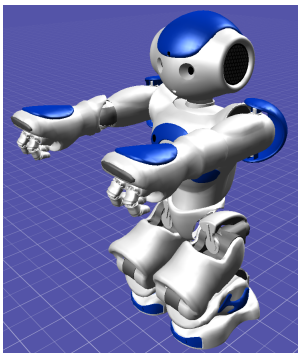
1. Transform P to the internal Denavit Hartenberg coordinate frame corresponding to the wrist yaw

2. Perform the Denavit Hartenberg transformation between the coordinate frames wrist yaw and shoulder pitch
3. Transform the result back to the model's shoulder pitch coordinate frame.

In Figure 3.13a the coordinate frames concerning the left arm are presented with respect to the robot model. The screenshot was taken by using RViz which is a visualization tool in the Robot Operating System (ROS) which was used as the basis for the given implementation. Note that there are only four coordinate frames visible because the coordinate frames of the shoulder pitch and shoulder roll joint are located in the same point and also have the same orientation. The same applies to the coordinate frames of the elbow roll and the elbow yaw joint. The coordinate system placed on the edge of the arm within the model represents the hand. In each case the red axis represents the x-axis, the green axis represents the y-axis and the blue axis represents the z-axis.



(a) Robot model and its coordinate frames visualized in RViz



(b) Current state shown with Choreographe

Figure 3.13.: Coordinate frames in Nao's arm shown with respect to the model

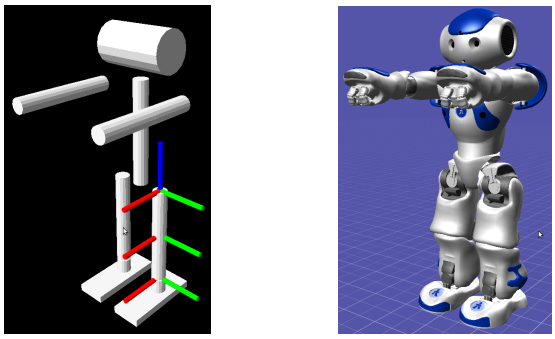
Table 3.2 shows the Denavit Hartenberg Parameters for Nao's left leg. The parameter θ is also incorporated here to match the coordinate frames introduced in the NaoQi. Additionally the table features two parameters l_1 and l_2 which represent the length of Nao's thigh and tibia.

$$l_1 = 0.1m$$
$$l_2 = 0.10275m$$

Joint Name	Joint Index i	a_i	α_i	d_i	θ_i
Hip Yaw Pitch	0	0	$-\frac{3\pi}{4}$	0	$-\frac{\pi}{2}$
Hip Roll	1	0	$-\frac{\pi}{2}$	0	$\frac{\pi}{4}$
Hip Pitch	2	0	$\frac{\pi}{2}$	0	0
Knee Pitch	3	$-l_1$	0	0	0
Ankle Pitch	4	$-l_2$	0	0	0
Ankle Roll	5	0	$-\frac{\pi}{2}$	0	0

Table 3.2.: Denavit Hartenberg Parameters for Nao's right arm

In Figure 3.14a the coordinate frames incorporated into the model are presented. If the robot is stands straight each coordinate frame has the same orientation. If the Denavit Hartenberg computation is used in order to make any computation, the framework automatically transforms the provided position with reference to the model's coordinate frames into the internal Denavit-Hartenberg coordinate frames. Figure 3.14 shows the coordinate frames of the model from the framework as well as the realistic model incorporated into Aldebaran's software tool Choreographe.



(a) Robot Model and its coordinate frames visualized in RViz

(b) Current state shown with Choreographe

Figure 3.14.: Coordinate frames in Nao's leg shown with respect to the model

3.3.2. Analytic Solution Of The Inverse Kinematics Using Vector Analysis

Different approaches may be used to solve inverse kinematics. There are approaches that incorporate iterative configuration searches, as well as analytical calculations. However, if the kinematic chain is short, analytical calculations may be the better choice because they are relatively easy and fast to compute. Solving the inverse kinematics in a fast way is a very important issue, especially when it comes to path planning. Within a dynamic path planner, the kinematic needs to be evaluated at each step of the plan in order to generate a viable path through the given environment.

The following example gives an idea about the solution of the inverse kinematics of a robot featuring two joints and two links.

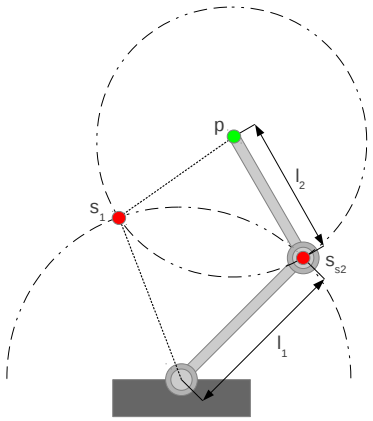


Figure 3.15.: Example for solving inverse kinematics

Figure 3.15 shows a robot with two links having the lengths l_1 and l_2. The robot features two rotational joints in order to move the links. In further explanations the base of the robot will be referred to as its shoulder, the second joint will be referred to as its elbow and the end-effector will be referred to as its hand. Given a goal position P (green circle) for the end-effector in the world frame, angles for the shoulder joint and the elbow joint shall be computed. Therefore two circles have been overlapped showing possible positions for the robot's elbow. The first circle is centered within the shoulder joint of the robot and has the radius l_1 whereas the second circle is centered in the given goal position and has the radius l_2. The result of intersecting these two circles are possible positions S_1 and S_2 for the elbow, indicated by the red circles. By knowing these positions the angle values for each joint can be computed. This step of the calculation

may show that possible positions for the elbow may turn out to be not viable due to joint angle limits. In order to solve the given problem the two circles can be represented by the formulas:

$$l_1^2 = x_1^2 + y_1^2$$
$$l_2^2 = (x_2 - p_x)^2 + (y_2 - p_y)^2$$

The intersection points can be found by setting

$$x_1 = x_2$$
$$y_1 = y_2$$

Since the computation involves two quadratic equations the resulting formula for either x or y is rather complex. An easier way to compute the intersection points is the computation by vector analysis.

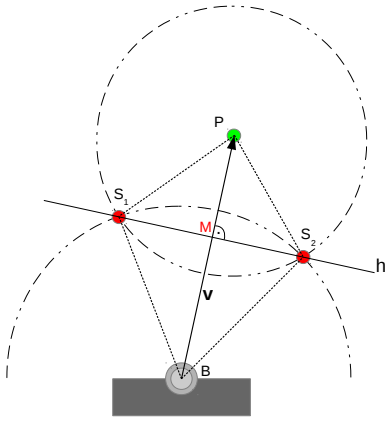


Figure 3.16.: Intersection of circles via vector analysis

Figure 3.16 shows the most important points and lines for the analysis. The two solution points obviously lie on a line h which is normal to the vector $\mathbf{v}$. In order to compute the positions of S_1 and S_2 it is important to know where exactly the intersection point of h and $\mathbf{v}$ lies. The distance between B and M can be retrieved via Pythagoras.

$$\mathbf{v} = \vec{BP}$$

The normal of a vector in 2D space can be retrieved by switching the vector components and inverting one of their signs. Thus vector $\mathbf{h}$ represents the direction vector of the line h.

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$
$$\mathbf{h} = \begin{bmatrix} v_y \\ -v_x \end{bmatrix}$$

The next step is to evaluate the distance d between B and M by solving the following equation system:

$$\|\vec{MS_1}\|^2 = l_1^2 - \|\vec{BM}\|^2$$
$$\|\vec{MS_1}\|^2 = l_2^2 - \|\vec{MP}\|^2$$
$$\|\mathbf{v}\| = \|\vec{BM}\| + \|\vec{MP}\|$$

The result for d is

$$d = \frac{l_2^2 - l_1^2 - \|\mathbf{v}\|^2}{2 \cdot \|\mathbf{v}\|}$$

The point M can be easily computed by setting the vector's **v** length to d and add it to the position B. In order to compute the position S_1 and S_2 the distance $\|\vec{MS_1}\|$ needs to be evaluated at first.

$$\|\vec{MS_1}\| = \sqrt{l_1^2 - d^2}$$

Finally the vector's **h** length has to be set to $\|\vec{MS_1}\|$ and S_1 and S_2 can be retrieved as follows:

$$S_1 = M + \mathbf{h}$$
$$S_2 = M - \mathbf{h}$$

In order to intersect two circles in the same plane in 3D space, only a small adaption has to be made in comparison to the mentioned 2D approach. In 3D space a circle is defined by a center point C, a radius r and a plane on which the circle is located. Of course the plane is defined by its normal vector **n**. Considering the intersection of two circles within the same plane ε the only thing that needs to be changed is the estimation of the vector **h**. In 3D the vector does not only need to be normal to the vector **v** but also to the normal vector **n** of the common plane ε. Thus vector **h** can be retrieved by computing the vector product of these two vectors.

$$\mathbf{h} = \mathbf{v} \times \mathbf{n}$$

All other computations do apply to 3D space the same way they apply to 2D space.

This form of solving the inverse kinematics is possible if only two links and a certain base are used to define a kinematic chain. This also applies for kinematic chains in three-dimensional space where the possible movement of the shoulder may be represented by more than a single joint. In three-dimensional space it is obvious that possible elbow positions may be computed by the intersection of two spheres which are centered in the shoulder with respect to the goal point. Intersecting these two spheres may result in different types of solutions:

- No intersection: The goal point cannot be reached because it is too far away
- Single point of intersection: The distance to the goal point is exactly $r_1 + r_2$
- Circle of intersection: The result of the intersection is a circle which implies an infinite number of solutions. If there are no more limitations to the goal position in terms of orientation or joint angles, any solution on this circle is viable and one may be picked randomly.

3.3.3. Solving Inverse Kinematics

As mentioned in Section 3.2 Nao features 25 joints in total. Since the joints controlling the head motion does not affect grasping only 23 joint values are left to compute. Furthermore, each of Nao's hands features one joint just for opening and closing the fingers. Thus these joints are also not important for the solution of the inverse kinematics which gives a total of 21 relevant joints for the computation. This number can be further reduced to a total of 16 joints by taking only one arm into account, which is sufficient to grasp a stone. However, in general this assumption is not always viable since the second arm may be used to provide a better stability or to support the grasp. Finally it is possible to reduce the given number of 16 joints to ten joints by reducing the degrees of freedom concerning the legs. In the given thesis the robot has to operate over a planar ground. As the robot is generally stable concerning the required grasp positions, it is assumed that both legs are used in a symmetric manner. In fact the two legs do not have the same joint limits. Thus only the smaller limits of both legs are used. Considering these assumptions, the kinematic chain includes ten joints. However the task in order to solve such kinematics is still not easy, because it features a total of five links from Nao's feet to Nao's hands. Thus it was decided to split the kinematics in two separate easier kinematics, and then merge the results in order to retrieve a feasible grasp position.

The first part of the kinematics focuses on Nao's legs. The input for this computation is a certain torso position relative to the legs. Based on this information it computes feasible joint values for each of Nao's legs. The second step is to compute the corresponding shoulder position via the forward kinematics using the mechanism of Denavit Hartenberg Parameters. The final step is to compute the inverse kinematics for Nao's arm by using the given shoulder position and the desired goal position as input and compute the corresponding joint values. Since both kinematic chains are just made by two links, the method mentioned in 3.3.2 which incorporates sphere intersection can be applied.

Thus a solution for the inverse kinematics concerning a given point can be found in the following manner:

Algorithm 1: compute inverse kinematics

input : $\{x,y\}$ ... current ground position of the torso,
$\{h_{t,min}, h_{t,max}\}$ ... height limits of the robot,
$\{\beta_{t,min}, \beta_{t,max}\}$ ... pitch limits of the torso,
$\{\Delta_h, \Delta_\beta\}$ ... increments for height and pitch
output: *best_pos* viable joint values

```
1 Positonsp  for h = h_t,min : Δ_h : h_t,max do
2       for φ = β_t,min : Δ_β : β_t,max do
3           if solveableIK(x,y,h,β) then
4               p.addPostion
5           end
6       end
7 end
8 best_pos = selectBestStoredPos
```

The limits for the torso's pitch β_{Torso} were chosen as

$$\beta_{Torso_{min}} = -10^\circ$$
$$\beta_{Torso_{max}} = 30^\circ$$

Whereas the limits for the torso's height h_{Torso} were chosen to be

$$h_{Torso_{min}} = 0.259$$
$$h_{Torso_{max}} = 0.332$$

The algorithm stores all joint values for each posture in reach of the goal position as well as the corresponding torso height and pitch. Finally the positions are compared in terms of required torso pitch and

height. A possible formula for the evaluation of these positions is presented here.

$$eval = a \cdot h_{Torso} + b \cdot \frac{1}{abs(\beta_{Torso}) + 1} \tag{3.23}$$

By varying the values for a and b either positions with lower pitch and thus higher stability or higher positions in terms of torso height providing a better view can be preferred. Although the overall robot stability depends on the current robot pitch β, stability is not a serious issue at any time. Thus no additional stability control had to be integrated into the system. However the current evaluation process does not exactly incorporate the formula mentioned above, but still illustrates the basic idea behind the built-in selection criterion.

In order to successfully grasp a token the robot's hand has to have a pitch value of 25° with respect to the xy plane in the game board coordinate frame. Note that because of the way Nao's arm is constructed, the entire forearm has to feature the given angle. Such a grasp position is illustrated in Figure 3.17. In general a position which is feasible to successfully grasp a token is referred to as pre-grasp position. In the given case it was relatively easy to find a feasible grasp position. However, this task can be also very complicated. Hence finding such positions for a given robot hand and element to grab has always been a widely discussed topic in research. [15] introduced a framework named "GraspIt" which was especially designed to compute possible pre-grasp positions based on user created models of robot hands and objects to grasp.

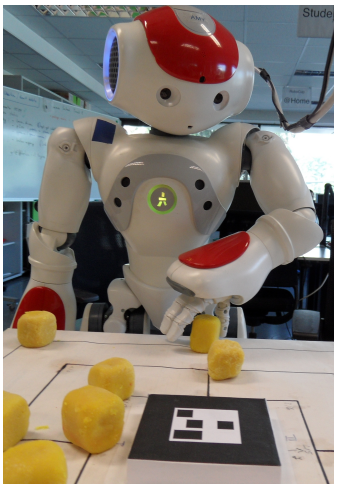


Figure 3.17.: Viable pre-grasp position

The inverse kinematics for the legs was solved without taking care of the joints HipYawPitch and HipRoll. These two joints were not required to be taken into account considering the selected limits for the torso's pitch and height. As mentioned above two parameters, namely the torso height and the torso pitch, may be specified in order to compute the inverse kinematics. These parameters are applied to the coordinate system located in Nao's torso. The kinematics was solved by intersecting the two circles shown in Figure 3.18. The radius r_1 corresponds to the length of Nao's thigh and r_2 corresponds to the length of Nao's tibia. The kinematics was solved by computing the required distance between the hip joint and the foot.

$$dist_f = h_{Torso} + \cos(\beta_{Torso}) \cdot dist_{TorsoHip}$$

With this information the required distance between Nao's hip joints and its ankle joints can be computed.

$$dist_{ankle} = dist_f - h_{Foot} - dist_{TorsoHip}$$

Since the ankle is always exactly beneath the hip, the exact ankle position can be computed. Finally the intersection between the hip circle centered in the hip joint and the ankle circle centered in the ankle joint were computed. In a standard case the result yields two possible positions. Given this positions the joint angles for the Hip Pitch and the Knee Pitch and the Ankle Pitch can be calculated. With the computation result the joint value for the hip joints is increased by the given angle β_{Torso} in order to achieve the given pitch.

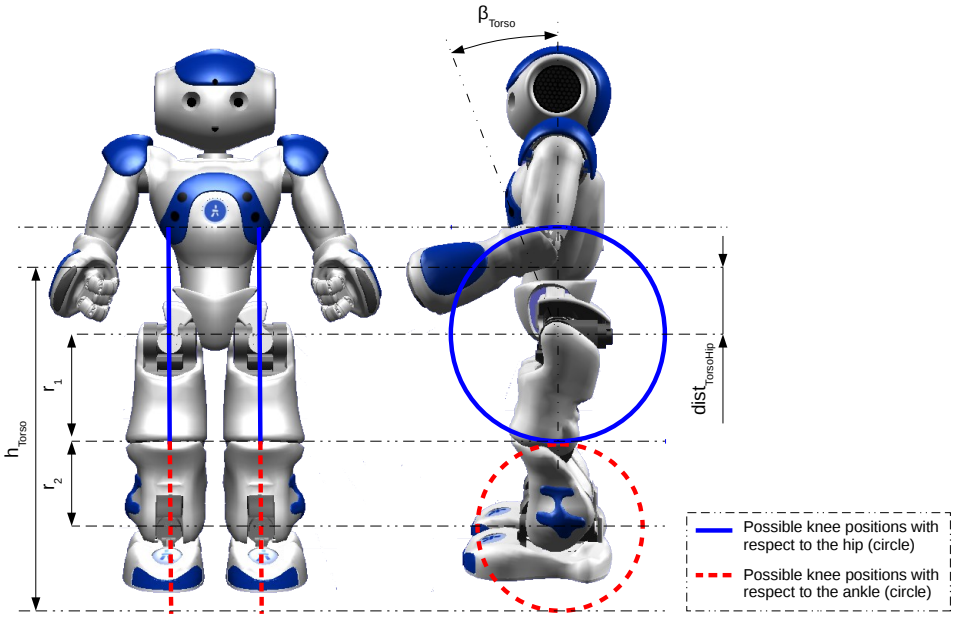


Figure 3.18.: Basic concept behind solving the inverse kinematics for the legs

The solution of the inverse kinematics concerning the arm turned out to be more complex, since each joint within the kinematic chain played an important role in order to reach the desired pose. Figure 3.19 gives an idea about how the inverse kinematics was computed. The black sphere indicates possible positions for the elbow with respect to the shoulder. The possible positions are assumed to be on a sphere's surface without any restrictions.

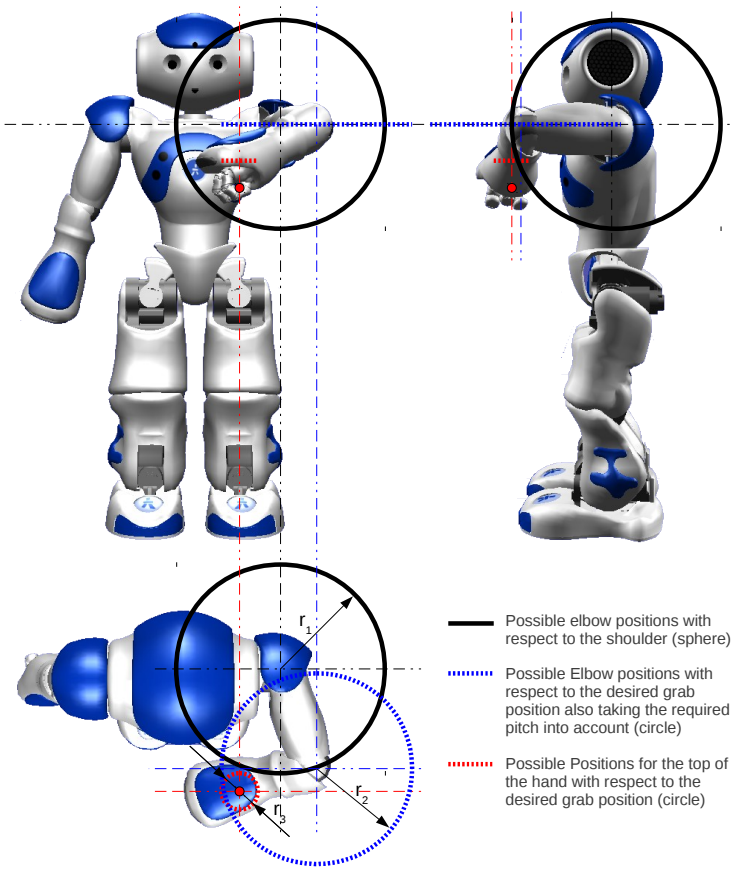


Figure 3.19.: Basic concept behind solving the inverse kinematics for the arms

In order to evaluate possible elbow positions with respect to the goal point, the structure of Nao's forearm has to be analyzed. As shown in Figures 3.2 and 3.3 the hand is not located directly in the arm but has a little z offset with respect to the wrist's coordinate frame. Considering the fact that a forearm pitch of $\alpha = 25°$ is required to grasp a stone, the possible positions for the top of the hand are placed on a circle. Figure 3.20 illustrates that. Each drawing shows the arm featuring the shoulder (S) and elbow (E) and desired goal point (G). However in the left scenario the required pitch is $\alpha = 25°$ whereas in the right scenario it is $\alpha = 0°$. If the goal point is pinned in space, the arm would be rotated around the goal point. The elbow would move along the blue circle. Due to the way the arm is manufactured, the top of the arm also moves in a circle in case of a required pitch other than $\alpha = 0°$. However the radii for the hand and the elbow can be easily computed in both cases.

In both scenarios presented in Figure 3.20 as well as in Figure 3.19 r_2 refers to the radius of the elbow whereas r_3 refers to the radius of the point on the top of the hand. These radii of course depend on the required value for α as indicated in Figure 3.20. The z offset of the hand will be referred to as z_{Hand} and the total length of the forearm will be referred to as l_{Hand}.

Case one requires a pitch of $\alpha = 25°$

$$r_3 = z_{Hand} \cdot \sin(\alpha)$$
$$r_2 = l_{Hand} \cdot \cos(\alpha) - r_3$$

And the corresponding heights h_2 and h_3 are

$$h_3 = z_{Hand} \cdot \cos(\alpha)$$
$$h_2 = l_{Hand} \cdot \sin(\alpha)$$

If the torso does not stand straight, the torso angle needs to be taken into account as well by subtracting the torso pitch from the desired forearm pitch. Furthermore, the computation requires a reference coordinate frame in order to properly setup the circles shown in figure 3.20. A proper setup of the circle the always keeps it in a plane parallel to the plane of the game board. Since the kinematics has no information about any orientation of objects in the robot's environment, the corresponding coordinate frame of the game board needs to be handed over to the kinematics.

Case two requires a pitch of $\alpha = 0°$. In this case the computation is very simple because the radius r_3 is equal to zero and r_2 is exactly the length of the forearm. No heights are needed to be computed because the height is exactly the value of z_{Hand}.

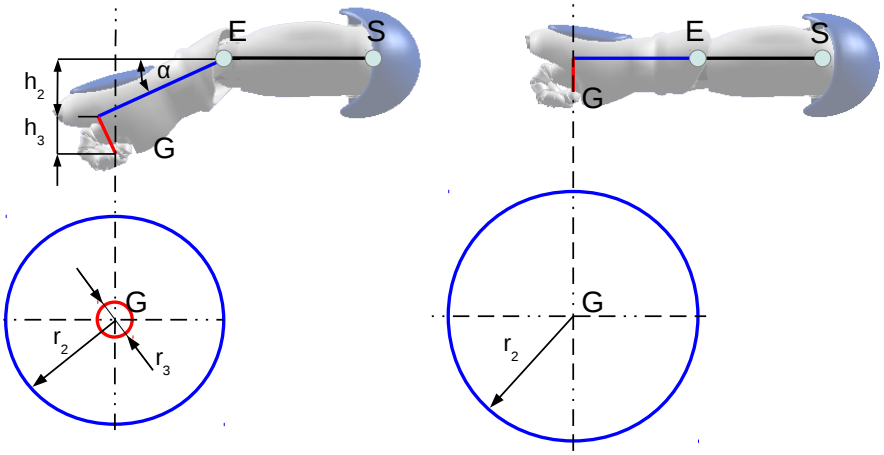


Figure 3.20.: Effects of the pitch on the possible elbow positions with respect to the goal position.

3.3.4. Solving The Inverse Kinematics For The Head

Another topic which is closely related to inverse kinematics is to compute the angles for the head joints in order to look at a certain point. The given point needs to be provided with respect to the torso coordinate frame. The point to look at will be exactly located in the center of the image if the joint limits allow for this. The given framework features three different solutions for this task. Two of them are required regarding the two cameras placed in Nao's head, and the third solution enables Nao to fake watching the given 3D point. Considering the setup of Nao's head people who don not know where the cameras are located in the head would probably assume that they are in the eyes. In fact if the Nao focuses on a certain point with respect to its eyes the resulting pose looks far more natural than by focusing on it with one of the two real cameras. Thus the third solution may be used to make the robot look more human-like if no camera image is actually needed in that specific situation. In each of the three cases the computation for the head's yaw ϕ_H is computed the same way as shown in Figure 3.21. In order to compute the angle ϕ_H the goal point P is projected onto the xy plane.

Figure 3.21 also shows a plane ε which is placed along the viewing vector v. The following figures and explanations concerning the computation of the head's pitch value in order to look at P are based on the plane ε. In terms of computation there is no difference for the pitch calculation when focusing on a point

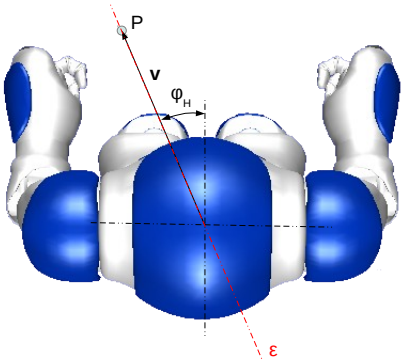


Figure 3.21.: Measurement of joint value head yaw with respect to a point P to look at

with Nao's eyes or with Nao's upper camera apart from an additional z offset. This offset describes the z difference between the upper camera and the eyes of the robot.

Figure 3.22 shows important angles and vectors in order to compute the head's pitch value for either the upper camera or the eyes. The vector g is known because the goal point P has to be provided with respect to the torso's coordinate frame. Hence g can be computed based on the given coordinates and the z offset of the neck with respect to the torso. Because this vector is known it is possible to measure the angle β between g and the standard orientation of the x-axis of the robot's head. After some computations the pitch value ψ_H can be found as follows:

$$\psi_H = \frac{\pi}{2} + \beta - \alpha \tag{3.24}$$

Note that since β was computed with respect to the x axis its value is negative in the given scenario.

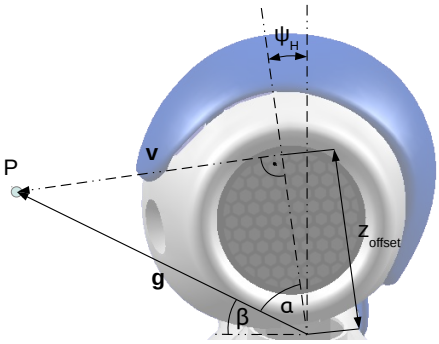


Figure 3.22.: Important values for the computation of the head's pitch in case of the upper camera or the eyes.

In case of the bottom camera an additional x offset is relevant as well as the fact that the camera has a pitch value of 40°. Figure 3.23 shows some important values in order to compute the required head pitch ψ_H. In this case β was also measured between the known vector $\mathbf{v}$ and the standard x-axis of the robot's head. The value for ψ_H can be found with the following formula:

$$\psi_H = \frac{\pi}{2} + \beta - \delta - \sigma \tag{3.25}$$

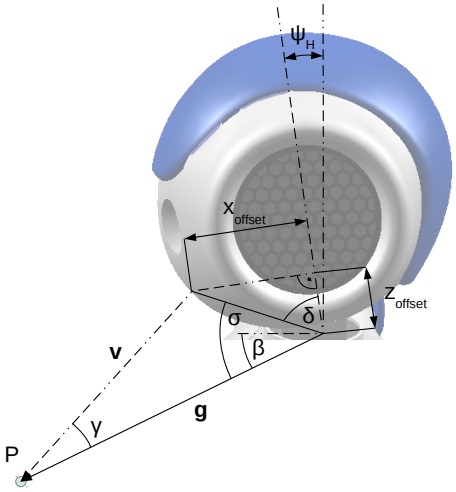


Figure 3.23.: Important values for the computation of the head's pitch in case of the lower camera.

3.3.5. ROS - Robot Operating System

The entire framework presented in this thesis was realized by using the Robot Operating System (ROS) created by Willowgarage [24]. The ROS framework features lots of important tools concerning several issues of robotics. These include tools for arm navigation as well as planar navigation and different drivers for certain hardware. Moreover, tools in order to visualize important robot data are included. The framework also features wrappers for other common libraries like OpenCV, the Orocos Project or the ARToolKit library [16, 18, 13]. Basically the tools within the ROS framework are organized in a hierarchy using three levels. The lowest level includes the executables themselves which are referred to as nodes throughout the entire framework. Some nodes addressing similar problems may be encapsulated within so-called packages which represent the medium level in the ROS hierarchy. And similar packages may be encapsulated if they address similar topics in stacks which represent the highest level.

The whole concept of ROS is based on the idea of having several independent processes (nodes) running at the same time in order to achieve a common goal. ROS also provides several ways of exchanging information between active processes. The supported concepts feature:

- Messages
- Services
- Actions

The message concept is used to provide general information within the framework which does not require any form of confirmation. One example for the use of messages may be the provision of sensor information. Any other node in the framework may get the message and use it in any possible way, whereas the sensor has no need to wait for a feedback about how the information was processed. It just keeps on providing the information. The service concept enables a kind of handshake between several nodes. Usually there is a service server and one or more clients. Clients send requests to the server. The important thing is that the clients are put to sleep when the request was sent. Further the server processes the request and when this is done it provides some form of results for the clients and it also wakes the client processes. The final concept is similar to the service concept. There is also a client that needs some information from a server. The difference in this case is that the client is not put to sleep. It may continue its work and it will receive a notification as soon as the server finishes the request and presents the results. Typically this concept is used when the server's action requires a great amount of time, or if the client is able to progress with other computations without immediately knowing the results from the server.

Chapter 4

Related Work

The following chapter will give an overview about related work concerning mobile navigation as such, especially focusing on planar motion planning and general motion planning for humanoid robots in order to grab objects. Furthermore, some other examples of using the Nao robot in order to play games will be mentioned and differences between the mentioned examples and the given tasks will be pointed out.

4.1. Mobile Manipulation

As mentioned in [21, 7] humanoid robots are highly complex systems due to their numerous degrees of freedom. This complexity is the main reason for making any sort of planning algorithm very time consuming. However, common solutions often imply the use of probabilistic approaches like Rapidly exploring Random Trees (RRT) or Probabilistic Road Maps (PRM) [21].

As mentioned above mobile manipulation usually concerns the entire movement of the robot. This also includes grasp actions while changing the robot's localization by walking.

Since humanoid robots are meant to support humans in their typical environment, some typical human tasks have gained great interest in the field of humanoid robotics. One of these typical tasks is passing through a door. This task has received a lot of attention because it features some problems concerning robot navigation which are considered to be fairly difficult. First the robot has to manipulate an object, which is the door in this case, while walking. Furthermore, the robot has to navigate through a narrow region without colliding with other objects, and finally stability constraints have to be met at all times [8].

This problem for example was addressed in [6] where an approach for task planning while walking was introduced. That approach assumed that the robot knows how certain objects like doors should be manipulated. In this case the task for the robot was to walk through a closed door. The planning itself was realized in a two-step strategy. In the first step, a simple plan was made where the robot was just represented by a bounding box and workspace of the robot's arms was represented by two circles. Based on this information a plan was created to fulfill the required steps for passing through a door. In the second step, the plan for the simplified model was refined in order to retrieve whole body trajectories which made it possible to pass through the door under given conditions [6].

Another attempt was based on the assumption that the robot has to walk through a swing door. The basic concept beneath the trajectory planner was an artificial potential field [APF] algorithm which was further evolved with a genetic algorithm (GA). In this particular case the GA was used in order to retrieve an optimal trajectory according to an user defined fitness function [8].

In both cases the task could be fulfilled within a simulated environment.

In the given thesis grasp and walking actions are however not carried out at the same time but one after another. Thus the further presented work either addresses the planning and execution of walks or the performance of grasp actions.

4.2. Walk Planning For Humanoid Robots

How walk planning for humanoid robots is realized, highly depends on the environment of the robot. If the robot has to navigate through rough terrain, computations concerning stability and foot placement will be very difficult compared to an environment only featuring planar ground. Since the ground in the robot's environment in case of this work is assumed to be planar, the related work further presented focuses on navigating in such terrain.

[21] proposed a two-step planning algorithm for walk execution. In the first step, the robot will be represented by a bounding box which slides on a plane. Thus the problem of finding a certain trajectory to navigate between obstacles on the plane is reduced to a 2D problem with 3D constraints. The bounding box has three degrees of freedom x,y and θ which define its position at any time. Searching for the path requires collision checks with the 3D robot representation in order to retrieve a 2D path on the plane. In the second step, the robot's footprints along the path trajectory are computed. The joint values for the robot's legs can be obtained by combining the information of the calculated footprints and the height of the robot's waist. Since this step produces real joint values, stability issues are also taken into account here. Afterwards the trajectory is refined in order retrieve a more efficient trajectory according to a given cost function [21].

This proposal was actually tested on a real robot, but unfortunately the paper does not mention any sensors used, so it is not quite clear how the robot was able to localize itself within the environment.

One of the biggest problems concerning the goal of the work presented in this thesis was that the Nao robot is actually not equipped with a laser or any other sensor which enables retrieving precise environmental data while moving. Thus it is not possible to re-localize the robot during the walk. The only way the robot is able to re-localize itself is by finding the marker. During the walk the robot solely relies on its odometry and thus bigger security distances were required in order to perform a collision-free planar navigation. However the basic idea of planning a trajectory by representing the robot as a sliding box on a plane was taken into account in the presented work. The only difference is that in the presented work additional information concerning the environment is given, which made it possible to reduce the 3D bounding box to a 2D rectangle. Instead of computing the collisions of this rectangle with known obstacles in the occupancy grid, the obstacles on the map were dilated in order to gain security distance to the robot.

4.3. Path Planning For Grasping

Previous papers addressed the problem of planning paths for robot arms in dynamic environments. In terms of humanoid robots, the complexity of solving this kind of problems is usually higher since stability and positioning of the lower body's joints also need to be taken into account. In addition, these robots usually feature a higher degree of freedom due to their numerous joints. In the presented work it turned out that in general the required grasp actions don not require robot positions putting the robot's stability at risk. This is the main reason why it is possible to plan grasp actions just based on the upper body of the robot, and moreover only using the arm which is supposed to perform the grasp.

Safe navigation through partially unknown territory cluttered with obstacles may be realized by using various forms of planning. One attempt is to use a simple model of the robot. It was shown that in general it is better to use simple representations of the robot in order to reduce the required computation time for collision checks. In order to safely navigate through the environment, the robot is assumed to possess detailed information concerning the current state of the world. This includes precise knowledge about where obstacles are located in the robot's workspace at each and every time step. At every execution of the main loop the current situation is evaluated, and possible collisions are calculated. Given that knowledge, constraints

for the motion speed for each of the robot's links are extracted. As a result numerous possible velocities are extracted and the best is selected according to an optimization method. This method minimizes the error between current end-effector velocities in comparison to the desired ones. This attempt was realized in real time, and tested on a pair of robots with interfering work spaces. One of the robots was equipped with the collision avoidance algorithm whereas the other just followed a programmed trajectory [5].

One of the problems concerning the mentioned algorithm is that it requires precise knowledge of the workspace at any time. In the given case, the only available information concerning the robot's environment is the robot's position with respect to the game board. Although the game board is assumed to be the only obstacle, the information about the robot position is not exactly known at all times. In fact it is only precisely known as long as the marker is visible and the robot does not move. However, another problem concerning the realization of this algorithm would be that the robot's interface does not feature changing joint velocities during execution. Furthermore, the algorithm is based on the idea that the algorithm is always executed within a small period of time. Since the ROS framework is no real-time framework, a periodic update within hard time limits could not have been guaranteed either. This fact finally made the method impossible to implement for the given purpose.

Another method enabling the robot to safely navigate through an unknown terrain is proposed in [20]. This method is also based on having precise information about the environment. In this case the goal is interpreted as an attractor whereas the obstacles are interpreted as repellors. Based on this information a vector field is generated which is further used to adapt the current link velocities to prevent collisions.

This attempt was also not taken into account because of similar reasons as mentioned above. Considering the task of playing Nine Men's Morris the grasp actions are always quite similar. In general the robot is assumed to stand in front of the table with its torso oriented orthogonal to the game board. Consequently the grasp actions differ only in the choice of the arm that should be used to achieve the grab and in the robot's distance to the game board. Thus a general path was formed which only depends on a few parameters like the height of the robot's torso at the time when the arm starts to move. The use of this simple method turned out to be sufficient to generate a collision-free arm trajectory.

4.4. Other Related Work

There have been some other attempts of playing board-like games featuring Nao which can be seen in [22, 11]. Even though the problems look similar to the one presented in this theses at first glance, there are certain differences.

[22] realized Nao playing domino with a special setup. The stones are placed in front of the robot and the robot decides which stone should be taken next due to the current game situation. When the decision is taken the robot uses one of its arms to indicate which stone should be used in order to continue the game. Finally, the human player has to take the stone and place it accordingly. In this case the main focus of the work is clearly the recognition of the stones and the proper reaction carried out by the Nao. In our case the robot actually places the stones, which required dealing with problems like self localization, mapping and path planning which might not have been necessary in the domino scenario.

[11] made the Nao play Connect Four. At startup the robot is placed far away from the game looking towards it. Because of the distance the robot is able to see the entire game at once. Thus it is capable of constantly scanning the game for changes concerning its status. Furthermore, it seems that the game itself is only tracked by its special optical setup, and thus no marker is required in order to localize the robot with respect to the game. As soon as the human opponent places a stone into one of the slots, the robot automatically recognizes that the opponent has finished his/her turn and walks towards the game. When it is close to the game it gets a token from the human opponent and places it in the desired slot by realizing a vision-based control of its hands. Finally, it steps back again to scan the game state while the opponent is taking his/her next turn.

One of the main differences of this scenario in comparison to the one presented in this thesis is that because of the basic setup of Nine Men's Morris it is not possible to see the entire game state all at once.

Therefore, the robot is not capable of automatically reacting to the opponent's move. One possibility of solving this problem would be to continuously check the game board for changes. One problem with this is that it would require the robot to stand all the time, because otherwise the angle of view would be too small to retrieve reliable data. This would require the robot's joints to be consequently stiff leading to higher joint temperatures. It was tried to avoid such a scenario because high temperatures lead to faulty executions and above that joints tend to loose their accuracy.

Another difference is that in the Connect Four realization the angle between the camera and the hand is rather good in order to achieve visual controlling of the arm. In case of our setup some of the stones are not even visible when grasping them because they are completely occluded by Nao's hand. Given that it was impossible to realize visual control of the arm in our case.

Chapter 5

Playing Nine Men's Morris

The following chapter gives an overview about the implemented framework. In the first section general information about the system design as well as information concerning the system's high level tasks will be presented. In the following sections the tasks concerning the mobile navigation will be discussed in detail. Further information concerning vision tasks and problems is presented in [3].

5.1. System Overview

Figure 5.1 shows the main parts of the given system design as well as the most important data transfer channels. These parts are:

- Game Engine (GE)
- High Level Control (HLC)
- Planners (PL)
- Motion Control (MC)
- Perception
- World Model (WM)
- NaoQi and ROS Bridge

5.1.1. Motion Control

This is the smallest part within the project. The main goal was to generate an interface which translates ROS messages into NaoQi method calls. In order to generate an interface with high re-usability a message format was incorporated which was able to hold data for different NaoQi movement tasks. The supported tasks include movement along a timed trajectory, movement to given angles at a certain speed as well as movement to given angles within a certain time. Since the task of providing perception data is different to the task of sending a motion request, the message type is only capable of dealing with different forms of motion requests and does not also support retrieving sensor values. In order to communicate with the robot the following two task groups are supported by different scripts:

- Actuator requests meaning the execution of physical movement
- Sensor requests meaning retrieving of sensor information

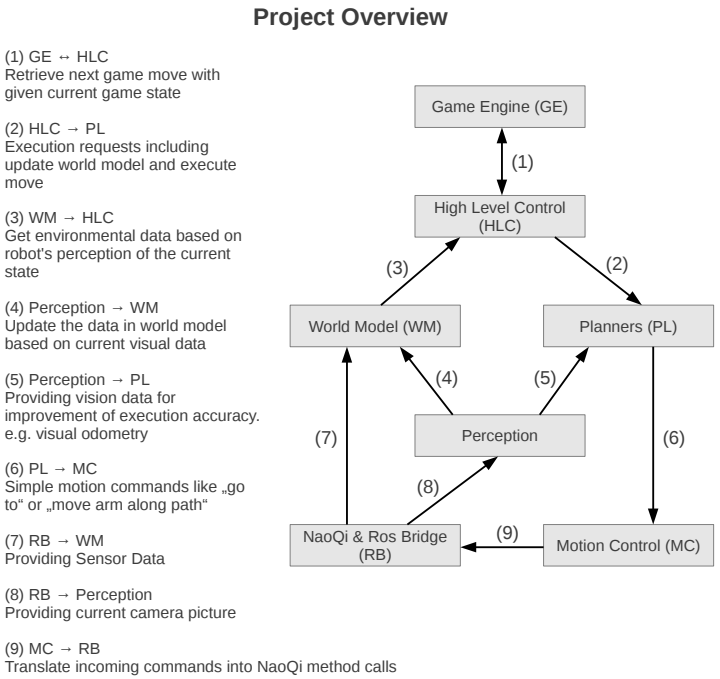


Figure 5.1.: System architecture

The reason for splitting the tasks into different separate scripts is simple. On the one hand it was required that the robot told the system when the motion request had been fulfilled. On the other hand an update of sensor values within a specified time interval was required. Due to the split approach it was possible to realize the actuator requests as blocking calls and at the same time retrieve sensor values by different scripts. A different approach would have been to just send the requests to the NaoQi in a non-blocking manner and to use each sensor evaluation loop to also watch the status of active actuator requests. However, this approach seemed to be more time-consuming so it was decided to split the NaoQi interaction in several separate programs.

5.1.2. World Model

The world model stores the entire knowledge about the robot's environment and the robot itself. This includes the robot's position as well as its current joint states and the current game state, the real 3D position of each stone and additional status information which for example also includes a status flag to document whether the robot is currently moving or not.

The information retrieved from the robot's sensors are updated within a certain time interval. In order to reduce the amount of requests to the NaoQi and as a consequence keep the work load for the robot's processor as low as possible, the time intervals were designed to be flexible. For example if the robot is currently moving, the time interval for the joint requests are shorter compared to the robot standing still. ROS [24] features a node called RViz which provides the possibility of visualizing all the different data the world model manages. The visualization of this data is shown in Figure 5.2.

The robot model in Figure 5.2 was taken from the Nao stack incorporated in the ROS system [23]. Although it is a very simple model it features each of Nao's joints and is therefore capable of properly visualizing the current state of the robot. The picture incorporates every information stored in the world model apart from the information whether the robot currently has foot contact, meaning whether it stands on its feet, or not. The latter information is important for some operations throughout the framework which

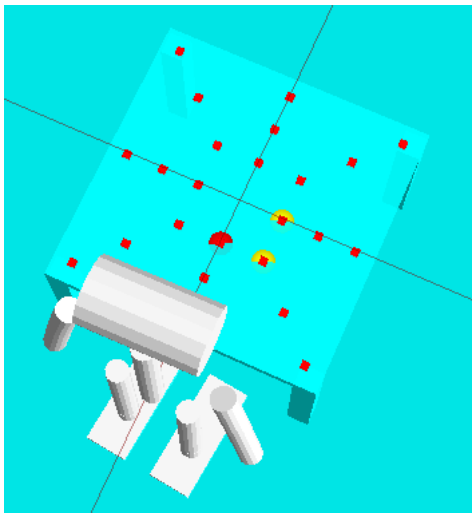


Figure 5.2.: World model data visualized within the RViz node of ROS

will not be executed if the robot is not standing. This was incorporated to reduce the risk of damaging the robot.

5.1.3. Perception

The perception part of the project incorporates actions related to the visual sensing of the robot. This includes the robot's localization, visual odometry, determination of the current game state as well as the computation of the real positions of the tokens on the game board.

The determination of the current game state works as follows: The world model holds the information concerning the 24 possible token positions on the game board as current game state. For every possible position a certain number representing the player ID from the player who placed his/her token on the position is stored. This also includes a value for positions which are not occupied. Moreover, each position holds a time stamp which is updated each time the position information is updated by the vision input. If the game state has to be re-evaluated, the robot starts to look at the oldest positions first and then iteratively updates the game board until all points are updated within a certain time interval. Since the camera is mounted on Nao's chin as shown in Figure 2.2 and Nao always stands while retrieving the game state, one camera image usually does not only update a single position but several. In general, the tokens do not necessarily lie directly on the ideal board positions. For correct classification of tokens which are not in ideal board positions, the current camera image is rectified and compared to previously generated masks. Each of these masks represents a possible area for a token at a specific position to lie on. Full details on the implementation and realization of this technique can be found in [3].

5.1.4. Game Engine

The game engine represents the AI of the robot. It provides an ordered list of possible moves to follow the current game state. The list is ordered by the quality of a move which is measured in the least required moves to win the game. This ordering provides the possibility of introducing different levels of difficulty just by randomly selecting a move between e.g. the first two listed entries or between the fifth and the tenth entry. However, in order to provide this data the game was completely pre-calculated in game state space and results were stored in a database. Since Nine Men's Morris is a game which features different game phases and thus completely different strategies for each phase, but it is rather limited when it comes to state space, this choice was made. For detailed information concerning the framework in order to access the data read [19].

5.1.5. High Level Control

The High Level Control (HLC) manages the entire application flow in order to successfully play the game. This includes sending database queries to the game engine as well as sending execution queries to the planners and requiring the world model to update its current environment status. The simplified state machine can be represented as follows:

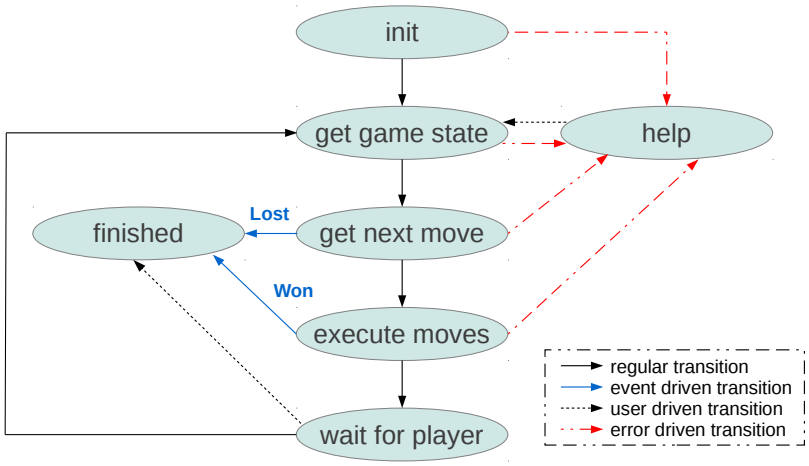


Figure 5.3.: Main state machine

The most important states in Figure 5.3 are "Get Game State" and "Execute Moves". Both actions are realized within the planner level of the project and are thus only started by request of the HLC. Although the planner level is capable of providing some form of self-repair in case an error occurs it is not capable of dealing with any kind of error scenario. In case an error cannot be repaired by the planner an error code is returned and the general reaction of the robot is to ask its opponent for help. If the error is cleaned, the system starts all over again with the same move and tries to execute it again. Due to the flexibility in the framework and the system design the robot will continue its game as if no error had occurred. Table 5.1 shows possible error scenarios and the possible reactions. In general errors may of course also occur during the repair phase within the planner level. In this case the robot will also be set to the help state.

Error description	Error context	Is self repair possible in general
Stone placement failed	Grab Execution	If stone lies on the table yes, otherwise no
Stone grab failed	Grab Execution	If stone lies still on the table yes, otherwise no
Marker not found	Walk Execution	Try blue stripe detection, if that fails no
Illegal game status detected	Perception	Re-check game status if error still occurs, no

Table 5.1.: Possible error scenarios

As mentioned above the standard reaction of the robot facing a non-repairable situation is to ask the human opponent for help. The required help may include repositioning the robot namely closer to the table, rearranging misplaced stones or just re-executing the same movement if the stone fell from the table.

The state machine also features a state called "wait for the player". In similar works [11] the robot continuously checked the game state for changes and therefore knew when the opponent had finished his/her move. One reason for this being possible was that the robot was able to see the entire game state at once. In the given thesis this is not possible due to limitations of the camera itself and the maximum distance between the camera and the game board. Hence continuously checking the board for changes concerning the game state would also require the robot to keep certain joints stiff while moving the head. If the joints are permanently stiff for a certain time, they get very hot and thus it is more likely that the

robot's movement accuracy becomes worse. The given framework automatically forces the robot to sit down and clear the stiffness on each of its joints in order to keep the temperature in the joints low. Hence the framework forces the robot to sit down to prevent it from falling over due to the cleared stiffness. In the resulting position the distance between the camera and the board game is smaller than it is when the robot stands and therefore the quality of the game state perception is significantly lowered. Due to that reasons it was not possible to automatically detect changes on the game board. If the opponent finishes his/her turn he/she is required to touch Nao's head sensor buttons in order to tell the robot that his/her move is finished. In [11] the game board is a vertical game board and thus Nao was able to step away from the board and watch the game status without having its stiffness set during the whole time.

5.1.6. Planners

The planner level of the framework manages the different requests from the high level control. Possible requests are listed below:

- Scan game status
- Calibrate colors
- Find marker
- Execute move

Each of the tasks mentioned in this list features the use of physical motion as well as perception of the environment via the camera information. "Scan game status" is used to retrieve the game status. This features standing up and scanning the game status by looking at different positions and updating the game information which is stored in the world model. The scan order for positions is not a hard coded procedure but it prefers to look at positions that have not been updated for a longer period of time over those which have been updated more recently.

"Calibrate colors" is usually only required once per game. Since the perceived color of the game tokens and the blue stripe attached to the side of the board may vary due to the present lighting of the scene it was decided to automatically select the colors based on the tokens placed on pre-defined spots on the game board. In order to calibrate the color for the blue stripe, a part of the blue stripe was also attached to Nao's shoulder. When the color calibration is started, the robot looks at previously defined positions and stores the information about the colors of the tokens which are placed there as well as the color of the blue stripe attached to its shoulder.

"Find marker" is also only required by the HLC during the startup phase. This is used to localize the robot within the environment after the startup routine is finished. Further re-localizations are performed within the execution of game moves because the need for re-localization is always closely bound to the action the robot has to perform.

"Execute move" performs all the steps required in order to successfully perform an entire move in Nine Men's Morris. This includes walk planning, walk execution, grasp planning, grasp execution as well as re-localization and computation of real 3D positions of the tokens. This is the most complex task in the given framework. If a single Nine Men's Morris move is represented by the basic actions grasping, placing and walking, it may feature anything from one single place action up to two grasp actions, one place action and three walking actions. However, the latter case may only occur in the last game phase and even then it is not very likely.

The planner also analyses and optimizes the given move in terms of execution with the help of the world model. In general the execution time and also the uncertainty of execution success rise with the amount of required walks during the movement execution. Thus the goal in terms of optimization is to select the way of execution which minimizes the amount of required walks in order to get both, a quicker and safer execution.

Consider the following scenario shown in Figure 5.4: The robot should move the black token in its front to the position indicated by the arrow. If the robot uses its right hand to move the token, it may move it without any form of walk movement, whereas it has to walk if it would try to fulfill the task with its left hand. In order to simplify the arm decision making, the robot is only allowed to grasp the positions on the left with the left arm and those on the right with the right arm. Positions in the middle of each board side, however, may be grabbed with both arms. Thus using the left hand here would require the robot to walk to the right board side in order to place the token.

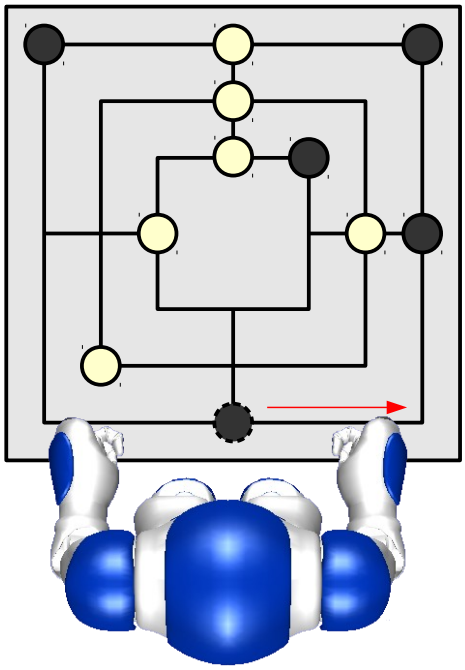


Figure 5.4.: Example movement

5.2. Planar Motion Planning And Execution

As mentioned above the robot needs to walk around the table to successfully play the game. Planar motion planning and execution addresses the problem of creating a plan and execute the required walking actions. Safe navigation around the table without colliding with known obstacles requires the generation of a walk plan. In chapter 4.2 several techniques for humanoid robot navigation are shown. However neither of this work addresses the problem of localization within the given environment. In fact the knowledge concerning the current position and orientation of the robot is the key to solving this task properly. Without the position knowledge it is obviously hard to safely navigate even through known environment. As mentioned above in the given project the only way to get information about the robot's position is to look at the marker. Thus a safe navigation would require to keep the marker in Nao's view as long as possible. However, this is not possible due to several reasons:

1. Required board distance. Safe navigation requires a minimum distance to the game board at all times. The distance ensures that the robot is able to freely rotate in any direction without colliding with the board. This restriction is required for keeping the planning algorithm as simple as possible. In this case the drawback is that a bigger distance to the marker also lowers the chance of being able to recognize it.

2. Nao's view. The robot is only able to successfully recognize the marker within certain limits for the head's rotation around the z-axis, which is carried out by the joint HeadYaw. These limits occur because of the way the robot is constructed. If the head was rotated beyond the given limits Nao's camera actually points into the robot's shoulder and hence the view to the game board is blocked.
3. Walking direction. The execution of the walk is carried out by two basic movement commands. The first command rotates the robot towards the goal point, whereas the second command walks the required distance. The reason for implementing the movement execution in this way was to keep the movement execution as simple as possible. Considering a path around the table and the given restriction for the walking direction the robot obviously cannot keep the marker in its sight at all times.
4. Motion Blur. Even if the marker could have been kept in the vision at all times, the pictures of Nao's camera would still not be capable of performing proper localization because of motion blur. Thus the robot has to stand still while looking at the marker in order to get precise sensor information.

5.2.1. Plan Generation

The plan generation was realized based on a potential field approach. In such approaches the environment is divided into small chunks. In our case the known environment was split into chunks of five mm side length. After that a certain cost value was assigned to each chunk. The cost value represents the risk of the movement resulting in a collision while walking over that chunk. This means that the standard idea would be to set the cost for chunks which happen to be within obstacles to infinite and those which are in free area to one. In order to assign the values properly, the algorithm requires an obstacle map of the given environment. A start and end position within the environment need to be defined in order to plan a path. Based on the cost values the way featuring the lowest total cost value is computed between the start and the target point. Of course the minimum cost value needs to be one. This also ensures that shorter paths are preferred over longer ones. Since this algorithm does not consider restrictions to robot movement in any form, the possibility of freely move from one chunk to another has to be guaranteed by the way the cost values are defined. As mentioned above the robot has to keep a certain distance to the game board at all times to be able to freely rotate without colliding with the table. This has been achieved by two additional steps. The first step is that the planner only creates paths from and to points with a minimum distance to the board. The second step is to enlarge the obstacles in the given obstacle map by a certain value depending on the robot's dimensions and create a plan based on the enlarged map. In addition the cost values of chunks increase by a linear function when getting closer to an obstacle. Apart from the game board a fence obstacle was introduced around the game board in order to keep the robot from moving too far away from the board.

Figure 5.5 shows an example path around the table.

The blue line in Figure 5.5 illustrates the created plan. As mentioned above the created plan has a resolution of 5 mm. Due to this high resolution it cannot be simply passed to the NaoQi because the robot is just not capable of walking such short distances. On top of that the execution time would be significantly raised if each subgoal was executed separately. Thus subgoals along the plan were extracted with the help of a simplification process. Considering a plan to walk straight for 1 m there would theoretically be no difference in walking the whole distance at once or dividing it into several pieces and then execute the path by walking to the given subgoals. In order to extract appropriate subgoals from the path, a minimum distance Δx_{min} as well as a maximum angle $\Delta\alpha_{max}$ difference between to subgoals were introduced. The latter restriction is important in order to prohibit the robot from walking into the edges of the game board. The simplified path is illustrated in Figure 5.5. The red crosses mark the subgoals whereas the black arrows illustrate the path which will be executed by the robot.

One can easily see that the robot actually gets closer to the board when executing the simplified path than it would get if the real path was executed. This was also taken into account within the process of enlarging the obstacles.

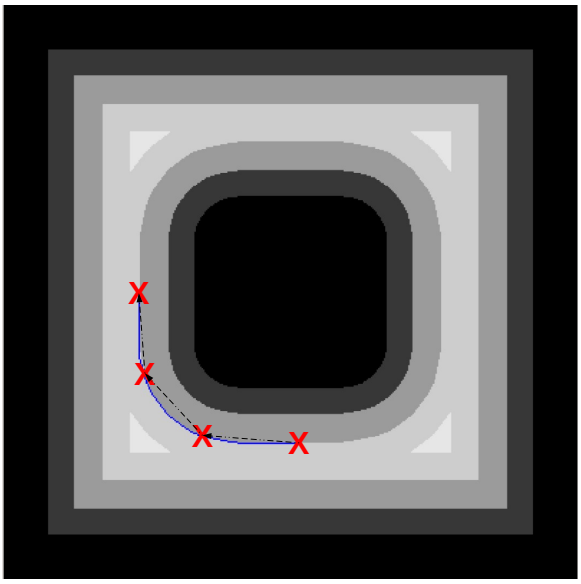


Figure 5.5.: Cost map featuring enlarged obstacles and the extracted path

5.2.2. Plan Execution

As mentioned above it is not possible to estimate the robot's current position based on sensor information while walking. Thus the only way to estimate the robot's current position is odometry. A robot's odometry is the robot's prediction about where it is actually located based on internal sensor values like joint encoders. When it comes to humanoid robots, the odometry for the current robot position based on encoder values concerning the walking process is hard to compute because in general this movement features a lot of joints to be operated in a synchronized manner. On top of that the robot's feet may also slide on the ground which cannot be detected by the robot by any means. Considering that the Nao's built-in odometry is rather good, its accuracy however still highly depends on the environment's ground material.

As mentioned above the walk towards any subgoal consists of two basic movements:

1. Rotate towards the goal point
2. Walk straight forward until the goal point is reached

In order to improve the odometry's accuracy concerning the robot's rotation, visual odometry was incorporated into the framework. If the robot needs to rotate by a certain angle α, first it rotates its head by the given angle. Since the encoder values are considered to be of high precision, the robot is now assumed to look exactly in the direction it should look at after the rotation. Then an image of the current view is taken and the head is rotated back to its initial position. After that the torso rotates by α and another image is taken. Then the two images are compared in terms of the position of certain feature points. The result of this comparison allows the robot to correct its rotation if necessary. The implementation of the visual odometry is further discussed in [3].

Figure 5.6 shows an example of such an image comparison in terms of visual odometry.

If the robot is actually required to walk to another side of the board, it usually sits in front of the board. As mentioned above it is not possible to generate a plan with the implemented method based on the current position of the robot because its possible movements are restricted due to its current distance to the board. Hence the first action of the robot is to step away from the board. After that a plan is created based on the new position leading to a position on the other side of the table. Of course the goal position of the plan also incorporates the mentioned minimum distance to the board. Thus another action has to be executed in order to move closer to the board.

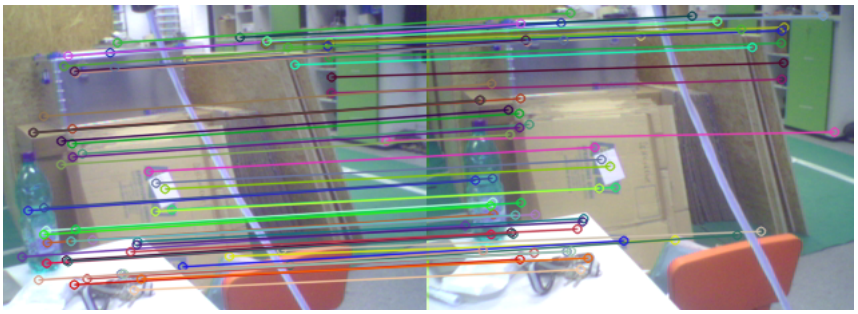


Figure 5.6.: Picture comparison for visual odometry

5.2.3. Navigation While Standing Close To The Board

If the robot walks next to the board its possible types of movement as well as number of feasible robot poses at the goal position are limited. Figure 5.7 illustrates this. If the robot would walk the same way as it does by executing a standard walk, it would collide with the table. In the illustrated scenario the robot would have been required to walk sidewards to prevent a collision. Another advantage of walking sidewards in the given case is that the robot actually keeps the marker in view all the time, and is therefore easily capable to re-localize itself after reaching the target point. If the robot had a different orientation, its shoulder may probably block its view of the marker.

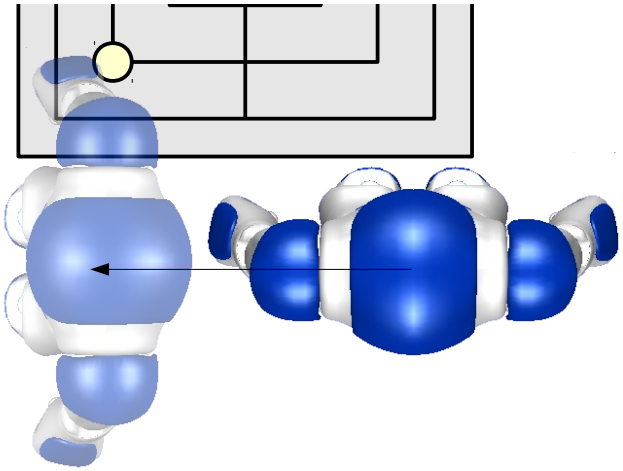


Figure 5.7.: Robot colliding with the game board if move command was executed in the regular way

Any required movement of the robot in order to safely navigate close to the table may be executed by the following three motion commands:

- Δ_x along x-axis
- Δ_y along y-axis
- Δ_θ around z-axis

In the given thesis the rotation by Δ_θ around the z-axis was executed twice, considering the current robot position x_R, y_R, θ_R and a goal position x_G, y_G, θ_G. First a rotation around the z-axis was executed in order to adjust the robot's position in a way that its torso is oriented exactly normal to the board. Hence following movements along the x- and the y-axis are automatically oriented normal to the board or along the board side. Thus the initial rotation did also simplify the computation for the remaining required movements Δ_x

and Δ_y. When the goal point was reached another rotation around the z-axis was executed in order to reach the desired goal orientation θ_G.

Before a movement towards a position close to the board is executed, the position is evaluated. If the desired position is close to the board and also implies a goal orientation which would actually result in the robot's arm colliding with the board, the command is not executed. This evaluation process analyzes the goal position based on the information of the obstacle map shown in Figure 5.8. There is a small region R in the center of each side allowing the robot to step closer to the game board. This is possible because the robot's feet are placed under the table. At the edges of the game board, the robot cannot walk that close to because of the table legs.

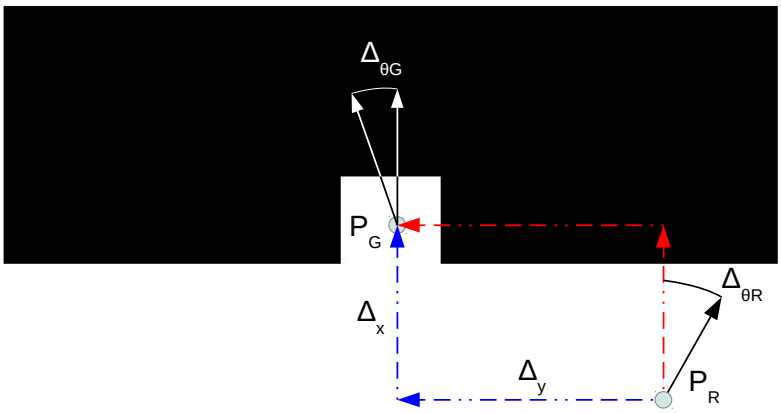


Figure 5.8.: One side of the game board obstacle map including movement parameters and effects of x,y movement order

As mentioned above the execution order of the given movement commands depends on the goal position. If the goal position is located in the mentioned region R, whereas the start position is not, the correct execution order is:

1. $-\Delta_{\theta_R}$: Orient the robot towards the table
2. Δ_y: Move to the goal y position (Sideward motion).
3. Δ_x: Move to the goal x position (Forward motion).
4. Δ_{θ_G}: Rotate to desired orientation

In this case the y motion has to be executed before the x motion in order to omit collisions. These motions refer to the current robot coordinate frame and thus Δ_x, Δ_y and Δ_θ need to be transformed from the game board frame, which is used to define positions within the environment, to the current torso coordinate frame. If the robot has to step to a position outside R when it is actually located within R the execution order of the x and y motion need to be swapped. After finally reaching the new position, the robot relocates itself and verifies if the goal position has actually been reached. If the point has not been reached another motion is computed based on the current information. Figure 5.8 also illustrates the effects of changing the execution order.

5.3. Arm Motion Planning

In the given scenario the robot has to perform certain actions with its arms. Of course neither of these actions should lead to collisions with the robot itself or collisions with the game board. The first attempt to achieve this goal was to create a dynamical path planning tool to enable the robot to compute non-colliding

trajectories for its arms. Given the goal joint values, the current joint values and a given angle resolution for each joint, it is possible to extract such a trajectory by generating sub-steps due to the resolution towards the goal configuration. At each step a collision detection has to be performed. The direction for each joint in order to retrieve the next subgoal may be chosen based on the goal information or simply by random selection. A big issue here is how to perform collision detection. ROS already features collision detection for any previously created robot model. However, the robot model of Nao included in the Nao stack of ROS [23] is too simple for performing a realistic collision detection. Moreover, the generation of such a dynamic plan including collision detection may require a lot of time depending on the angle resolution and the level of detail of the model.

Due to these reasons the different actions the robot has to perform were analyzed. Each action required a movement from the front of the table to the top of the table or vice versa or it may just require a movement taking place entirely above the table depending on where the hand is located when the action is performed. Moreover, the robot will also stand in front of the table and its torso will always point towards the table. The maximum angular fault $\Delta\Phi_{max}$ illustrated in Figure 5.9 is limited because of the way the movements close to the board are executed.

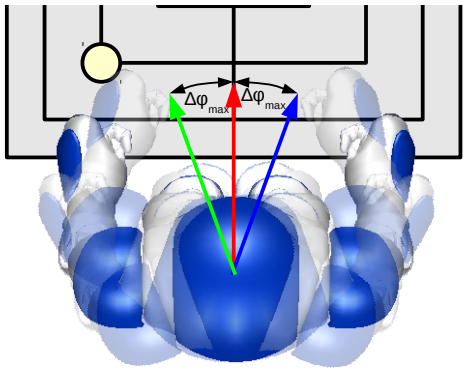


Figure 5.9.: Assumed angular position tolerances before grasp actions

Because of the given facts it is possible to realize a form of semi-static trajectory which is both fast to compute and viable in terms of collision avoidance. This means that a general trajectory for movements taking place above the table as well as for movements starting in front of the table and ending on its top and vice versa was setup by defining sparse support points for the robot's elbow. During tests it turned out that the insertion of a single support point was sufficient in order to omit collisions. The location of this support point depends on the current height of the robot's torso at the start of the movement and also on whether the entire arm movement occurs above the table or also includes bringing the arm to the top from the front of the board. The height of the point was chosen to be that far above the table that collisions between the hand and the game board were not possible because of the limited length of the robots lower arm. As mentioned in section 3.3.3 the robot's hand has to have a pitch angle of 25° relative to the *xy* plane. Considering this limitation and the computed support point for the elbow, trajectory collisions are very unlikely. During the test phases of the project the robot never collided with any object in as a result of the arm navigation.

The implemented semi-static trajectories are fast to execute but they do not provide the possibility of successfully executing different grab scenarios. However, they turned out to be sufficient to achieve the goal and also fast to compute which made them the best choice for the given task.

Grasping a token with Nao's hands requires the robot to place its hand above the token and then lower it before its fingers are finally closed. The position requirements need to be fulfilled any time during the lowering process. This prohibits the robot from colliding with other objects. Due to the limited degrees of freedom of the robot's hand it cannot be guaranteed that the requirements are met at all times if the robot actually moves the hand for the lowering. Because of that it was decided that the robot places its hand above the stone, thus clearly taking the position requirements into account, and then lowers the entire torso

by bending its knees. When the desired position is reached, Nao closes its fingers and tries to grab the token. All in all the grab action requires the following actions to be executed:

- Move arm above the token by making use of the presented trajectories
- Bend knees in order to lower the hand
- Close fingers to grasp the token
- Raise torso by stretching the knees
- Move arm to standard position above the table.

5.4. Position Planning

The term position planing addresses the problem of finding a valid position in order to grasp a token from a specific position in the environment. In section 3.3.3 the algorithm for computing the inverse kinematics was presented. This algorithm evaluates a position x, y of the robot and tries to find valid joint values in order to grasp a certain goal point. If the method fails to find a solution, a different position has to be computed which actually enables the robot to grasp the given position. This chapter presents the implemented method in order to find such a position.

If the planner level of the environment has to grab a certain position it actually always knows on which side of the game board the robot has to be in order to compute a valid grasping position. Considered that side is known a simple iterative search for feasible positions to grasp a point over the area around the table was implemented. Each position was evaluated with the algorithm presented in 3.3.3 and thus a solution could always be retrieved. Depending on the resolution of the search algorithm, this operation can be rather time consuming. The first attempt in order to accelerate this algorithm was to keep the area resolution at a low level of 1 cm in both x and y direction and furthermore stop the algorithm as soon as an appropriate position was found. The problem with this attempt can be easily identified. If the algorithm stops its execution at the first possible grasp position, this also means that the selected point is only about 1 cm away from a position from which the desired point cannot be reached. Since the robot is not able to navigate to a certain point without any tolerance it may actually end up standing on a point from which grasping the desired point is not possible. Thus the procedure starts all over again. In this case the robot now stands very close to the calculated goal point in the previous loop. However, since the algorithm is deterministic and the point to grab has not changed its position within the environment, the robot tries again to navigate exactly to the same point again. The problem is that the robot is simply not able to navigate to such short distances. If other points next to the goal point are also not feasible for grasping the desired token, it is very likely that the robot will end up repositioning itself several times without any chance of fulfilling the task. Considering the fact that the search algorithm takes at least 20 seconds even with the rather big resolution, the results of this approach were poor.

Thus a different approach needed to be incorporated which dealt with the problem of standing close to points which are actually not feasible for grasping the desired point and which is furthermore faster than the standard position iteration. In order to achieve these goals, it was decided to pre-compute possible robot positions. The grasp action as such is always nearly the same in terms of execution. As mentioned above the robot stands straight in front of the table and the torso always points towards the table at every grasp action. Even the restrictions for the grab posture, namely the 25° mentioned above, do not depend on where the token to grasp actually lies. In a standard case the workspace of a robot features all points that may actually be reached by the end effector. In the given case the opposite task was realized. A certain position was chosen and then all possible grasp positions with respect to that given point were evaluated and stored within a picture. The result of such a picture is shown in Figure 5.10. The white pixels in the picture are possible grasp positions, whereas the black pixels are not. The green cross is there to show the point which was tried to grab at any position in the picture.

This picture features a total area of a width of 60 cm and a height of 45 cm which is quite big considering the fact that the game board has a side length of 37 cm. However, the computation of such a workspace

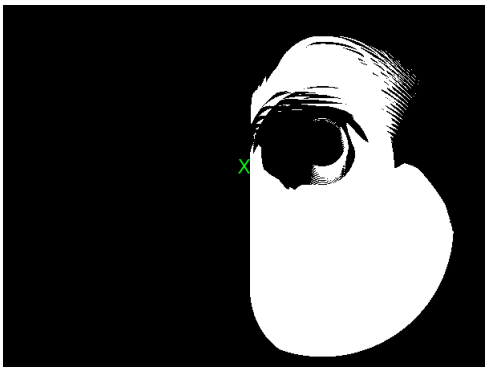


Figure 5.10.: Computed viable robot positions (white) in order to grab a token at a given position (green cross)

picture requires a lot of time since it was computed with a very high resolution of 1 mm in each direction. Of course the picture is not capable of storing the corresponding joint angles as well but this was not necessary because the evaluation of a single position only requires a couple of milliseconds. The algorithm for the picture generation can be found below.

Algorithm 2: Pre-calculated viable robot grasp positions

input : $\{x_{min}, x_{max}\}$ ... position limits x direction,
$\{y_{min}, y_{max}\}$ ... position limits y direction,
$\{h_{t,min}, h_{t,max}\}$ ... limits of torso height,
$\{\Phi_{t,min}, \Phi_{t,max}\}$ ... limits of torso pitch,
$\{\Delta_{xy}, \Delta_h, \Delta_\Phi\}$ ... increments for position, height and pitch
$\{x_{pg}, y_{pg}\}$ ... desired pre-grasp position
output: a robot grasp position grid *RGP*

```
1  for x = x_min : Δ_xy : x_max do
2      for y = x_min : Δ_xy : y_max do
3          RGP[x,y] = 0
4          for h = h_t,min : Δ_h : h_t,max do
5              for ϕ = Φ_t,min : Δ_Φ : Φ_t,max do
6                  if solveableIK(x,y,h,ϕ,x_pg,y_pg) then
7                      RGP[x,y] = 1
8                  end
9              end
10         end
11     end
12 end
```

In the created picture the robot's orientation was always assumed to be as follows: The x-axis of the robot's coordinate frame points vertically up in the picture and the y-axis points horizontally to the left. The assumed orientations of the robot depend on the side the robot currently is at. Figure 5.11 shows the assumed coordinate frames depending on the board side as well as the main coordinate frame of the game board.

In order to use the created picture to retrieve a possible grasp position for a given point provided in the base frame, the picture has to be transformed in a way that it actually fits the robot's orientation depending on the side. After that transformation the picture has to be translated in a way that the desired point to grasp overlays the given point. Finally the obstacle, namely the game board, needs to be incorporated into the picture in order to prohibit the robot from picking positions which are located within the obstacle. The result of that merging process is a picture which holds possible positions in order to grasp the goal position. As mentioned above it is necessary to ensure that the robot picks a position that is as far away from positions from which the token cannot be grasped as possible. By ensuring this, the risk of moving to a point which is actually not a possible grasping point due to walk tolerances is minimized. This is realized

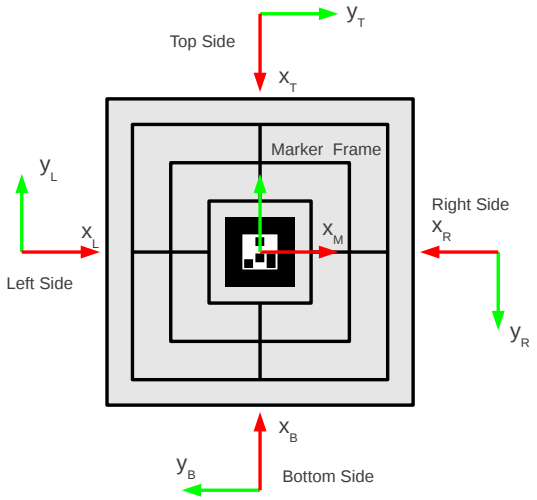


Figure 5.11.: Assumed orientations for the robot on each side of the board

by performing a distance transformation on the just created picture. Within a distance-transformed picture, each pixel's value is defined by the distance to the next black pixel. Since the distance information is now encoded in the picture, the point with the biggest value has to be chosen. Each picture created during the process is referred to as grid in the following explanations.

The entire process of generating the different grids can be represented by the following algorithm. Of course a single grid is only able to represent possible grasping points for either the left or the right arm.

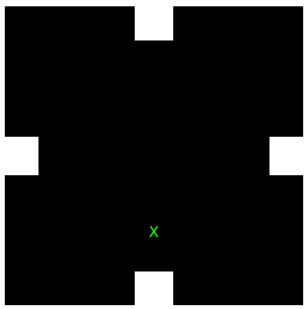
Algorithm 3: Reach pre-grasp posture

input: $\{x_{pg}, y_{pg}\}$ ... desired pre-grasp position
RGP ... robot grasp position grid
GBO ... game board obstacle grid
hand ... which hand to use

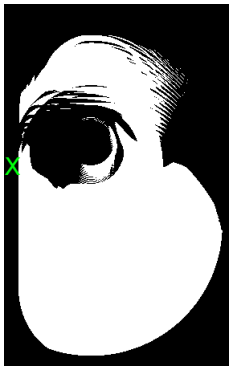
1 $side \leftarrow$ selectSide(x_{pg},y_{pg},*hand*)
2 $TRGP \leftarrow$ transGraspGrid(*RGP*,*side*,x_{pg},y_{pg})
3 $TGBO \leftarrow$ transObstacleGrid(*GBO*,*side*,x_{pg},y_{pg})
4 $CG \leftarrow TRGP \cap TGBO$
5 $DCG \leftarrow$ calcDistanceGrid(*CG*)
6 $\{x_r, y_r\} \leftarrow argmax_{\{x,y\}} DCG[x,y]$
7 $\{x_0, y_0\} \leftarrow$ getRobotPosition()
8 $p \leftarrow$ planPath(x_0,y_0,x_r,y_r)
9 executePath(p)
10 $\{x_1, y_1\} \leftarrow$ getRobotPosition()
11 **while** $CG[x_1]][y_1] = 0$ **do**
12 $\{x_r, y_r\} \leftarrow argmax_{\{x,y\}} DCG[x,y]$
13 $p \leftarrow$ planPath(x_1,y_1,x_r,y_r)
14 executePath(p)
15 $\{x_1, y_1\} \leftarrow$ getRobotPosition()
16 **end**
17 $\{h_r, \phi_r\} \leftarrow argmax_{\{h,\phi | solvableIK(x_1,y_1,h,\phi,x_{gp},y_{gp})\}}$ evaluatePosition()
18 moveTorso(h_r,ϕ_r)
19 moveArm(h_r,ϕ_r,x_{gp},y_{gp})

The pictures resulting of each step are shown in Figure 5.12. Figure 5.12a shows the obstacle mask for the game board. As mentioned above the robot is able to walk closer to the board at the center of each side than it is on the edges of the game board. The reason for this is the placement of the table legs.

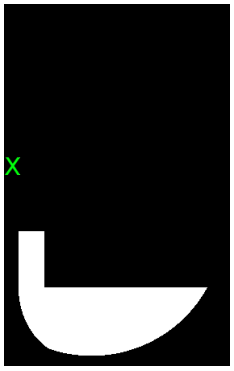
The pre-calculated workspace pictures only represent a single grab height. This was sufficient in the



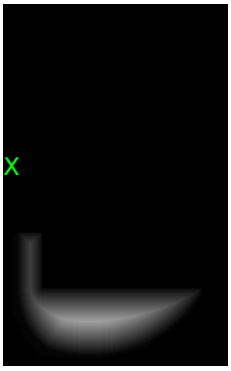
(a) Game board obstacle grid



(b) Entire workspace



(c) Combined workspace

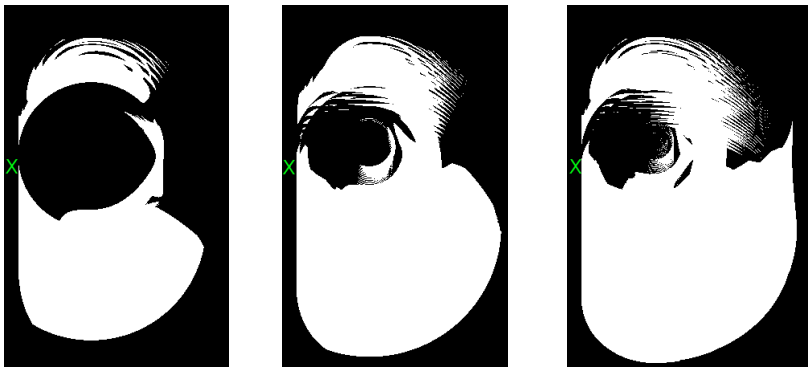


(d) Distance transform

Figure 5.12.: These figures show the grid evolving steps during the algorithm from above. The first picture shows the mask representing the game board obstacle. Figure 5.12b shows the entire workspace of the robot. Figure 5.12c shows the computed overlap of workspace and game board. Fig. 5.12d shows the distance transformation performed on that overlap figure. The green cross is the desired point to grasp.

given case because the tokens always lie on the game board. However, in order to analyze the effects of changing the grab height for the pictures, various pictures were created. Figures 5.13 show the viable grab points for the left arm at different heights between a range of -3 cm to +3 cm whereas 0 would be exactly the board height.

Figure 5.13 shows that the amount of possible grasping points within the height range of $-0.025m - 0.025m$ rises as the height rises. This means that when it comes to grasping, the robot has a higher chance of actually standing on a position which enables the robot to grasp the given point if the table is higher. But increasing the height does not only have advantages. In terms of vision a bigger distance between the camera and the game board would ease the process of estimating the current game state and reduce the risk of occlusions. Considering this trade-off the selected game board height seemed to be a good choice.



(a) Possible grasp positions at board height -0.025 m

(b) Possible grasp positions at board height (0 m)

(c) Possible grasp positions at board height +0.025 m

Figure 5.13.: These figures show the grid evaluated at different grab heights

5.5. Human Robot Interaction

As mentioned above a certain human-robot interaction has to take place in order to enable the robot to play the game. One part of this human-robot interaction is that the human opponent needs to touch Nao's head when he/she has finished his/her move. Another important form of human interaction happens during the first phase of the game. In this phase the robot has to place its stones on the game board one by one. The first idea was to provide a certain place in the environment where the robot is supposed to find its tokens. But this would have demanded a lot of execution time since the robot most likely would have had to walk to that place several times. Thus a different approach was realized. The robot opens its hand and asks its human opponent to place a stone in it. When its hands are open, the robot constantly watches its fingers and automatically detects when a stone is placed in its hand. As soon as the token has been recognized the robot automatically closes its fingers and places the token in the desired position.

In this case the detection was realized by making use of masks. The picture the robot took of each hand while watching it was taken as a model to create a binary mask of Nao's fingers. If however a certain number of pixels in this mask matched the token color, the robot suspected that a token had been placed in its hands and the fingers were closed. For more details on the mentioned masks and how they were designed read [3].

The same applies to the situation when the robot successfully placed three tokens in a row and thus is allowed to take away a token from its opponent. In this case the robot grasps the opponent's token and raises its hand. Finally it opens its fingers and advices its opponent to take away the token.

When it comes to gaming, emotions often play a certain role. In order to win a game players develop different strategies and depending on the strategy, yielding to either win or loose the game, the player may be frustrated or happy. Since Nao is also capable of speaking and features a built-in tool for speech recognition and face detection such a human-robot interaction may be incorporated into the framework as well. Thus the robot would be capable of reacting to an opponent's move by commenting on it and as a consequence generate a more human-like feeling to its opponent. However, the current state of the framework does only involve very little human-robot interaction and it does not imply reacting to anything the opponent does during the game.

5.6. Towards the framework

The following section provides an outline of how the framework was designed and where certain features can be found within the project. As mentioned above the framework was completely realized within the

ROS framework [24] and furthermore implied communication with the NaoQi [2]. Due to this fact the entire project was designed as a ROS Stack called nao_game_play featuring the following packages:

- nao_game_db
- nao_high_level_control
- nao_motion
- nao_msgs
- nao_planners
- nao_vision
- nao_world_model

5.6.1. Nao Game Database

Overall Purpose This package incorporates the interface to the game database as well as a data base simulator which allows to manually execute a certain move. Thus this package manages the game intelligence.

The nodes incorporated in this package are:

- Database communicator: Main interface to the game database
- Database simulator: Simulation of the database. This node enables the user to manually execute certain moves e.g. for test purposes.

5.6.2. Nao High Level Control

Overall Purpose As mentioned in Section 5.1.5 this package is responsible for the overall organization of the execution. It mainly executes high level commands by sending requests to other packages. The package also features nodes to separately test certain high level requests.

The nodes incorporated in this package are:

- High Level Control: This node includes the main loop to play the game.

5.6.3. Nao Motion

Overall Purpose The Nao Motion package features the entire communication with the NaoQi in terms of motion commands. All nodes which incorporate communication with the NaoQi were written in Python in order to avoid problems with possibly occurring different supported versions of the boost library [4].

The nodes within this package are:

- Nao Movement: This is the main communication node and the only relevant Python node. Other Python nodes within that package were only implemented for test purposes, or are previous versions of this node, which are not supported by the framework anymore.
- Nao Motion Client: This node was incorporated into the package in order to test the execution of possible move commands via the NaoQi. It provides a text-based user interface for sending different commands. In order to test the commands the Nao Movement node has to run.
- Teleop Nao Joy: This node is a slightly modified version of the teleop node in the official Nao stack. It enables the user to operate the robot via joypad.

5.6.4. Nao Messages

Overall Purpose This package was created in order to manage certain data types, including services and messages as well as constants which were required throughout the entire project. It furthermore features some libraries which are capable of performing certain movements like looking at a specific 3D point or performing other general purpose issues which are required by several packages as well.

Since this package was only created to manage data which is required by many nodes the package does not incorporate any nodes.

5.6.5. Nao Planners

Overall Purpose Nao Planners is the biggest package within the framework. It incorporates the inverse kinematics, the forward kinematics, the entire path planning for the walk and the arm navigation as well as the code in order to find a marker and perform blue stripe detection. Nearly any of the mentioned tasks was individually tested by also incorporating several tests into that package.

The package incorporates the following nodes:

- Motion Planner: This node holds the entire logic for performing a move, finding the marker, calibrating colors or analyzing the game board. Thus this node represents the heart of the project.

The motion planner handles nearly all HLC requests. The biggest part of the motion planner is the execution of a certain move. As mentioned above basic moves were defined inside the motion planner such as grasp, place and move. Any move occurring in a single game of Nine Men's Morris may involve one ore more of these basic moves. However, the decomposition of a game move into basic moves is carried out in the HLC and thus the motion planner only has to execute basic move commands. The following outlines which actions need to be performed in order to perform such a basic move.

Execute A Place Command

A place command basically represents the standard move in the first phase of the game. In this case the robot opens its hand at the beginning and waits for the human opponent to hand over a token. Then the token is placed on the game board. The steps to achieve this goal are:

1. Place Position Evaluation

   Input The position where the robot should place the token and Nao's current board side.

   Output A side of the game board where the robot needs to go and a corresponding arm index in order to place the token on the given position.

   Explanatory notes As mentioned above the algorithm to solve that problem focuses on minimizing the required walking distance. Furthermore, it is carried out in the world model because the world model has the entire information concerning the game board design. Thus the motion planner just sends a request to the world model in order to retrieve the data.

2. Walk to board side

   Input A specified board side

   Output The robot reaches the center of the given board side.

   Explanatory notes This task is entirely realized by the Walk Planner. It features close to board navigation as well as generating and simplifying the plan as well as walking execution itself. Errors may occur due to different reasons including failing to find the marker. The robot always centers itself on a board side within this execution.This step does not require that the robot reaches a certain position which is however viable for placing the token on the desired position.

3. Walk to a position allowing the robot to place the token without further movement

   Input The current robot position within the environment

   Output A position which is viable for placing a stone on the desired position

   Explanatory notes As mentioned above the robot does not necessarily stand on a position where placing is possible. If it is not possible, another position is searched for with the use of the algorithm presented in 3. In rare cases the robot may have to execute several moves in order to place itself properly. This may especially occur if the number of available positions is very limited and the goal position is close to the current position. The robot is not capable of performing very small steps properly.

4. Open hand

   Input -

   Output The robot moves its arm to a certain position and opens its hands waiting for a token to be placed in it.

   Explanatory notes This task also requires the motion planner to create a semi-static path as mentioned in 5.3. In the standard case the hand is initially placed on the side of the torso and needs to be navigated above the table in order to reach the desired goal pose. The robot also moves its head in order to see its opened hand.

5. Wait for token

   Input The camera image

   Output Visual confirmation that the token is in the robot's hand.

   Explanatory notes The camera image is searched for the token within a specified mask. If the token is in the hand the next step is started.

6. Close fingers

   Input Token placed in open hand

   Output Hand closed

   Explanatory notes This task just features a simple motion call to close Nao's hand.

7. Place token

   Input The token is safely in Nao's hand

   Output The token is placed on the desired position

   Explanatory notes The robot moves the arm to place the token in the desired position. Again semi-static paths are used for the generation of a collision-free trajectory. The position is always reachable because this was proven in previous steps.

8. Relax arm

   Input -

   Output The robot moves its arm to the side of its torso.

   Explanatory notes This task also involves semi-static trajectories. The robot autonomously moves its arm back to the position where it started from. The reason for this is that the arm is actually blocking Nao's vision of the game board. In some cases leaving the arm above the table could render the visual confirmation of the placement impossible.

9. Visual confirmation

   Input The position where the token was placed

Output Visual Confirmation about the placement

Explanatory notes Nao looks at the position where it has just placed the token. If the token is located close to the desired position the task is completed. Otherwise the robot starts a repair procedure. It searches for the token on the field. If the token is found on any other position the robot autonomously goes to the position, grasps the stone and places it exactly at the same location as in the first run. If the token cannot be found on the field, it most likely dropped during the arm motion. In this case no self-repair is possible.

In most cases an error during any of the steps produces a fatal error leading to the help status in the HLC. In steps were repair is possible the motion planner tries to repair the situation once. If this fails this also leads to the help status in HLC.

Execute A Take Command

A take command is always required if Nao successfully places three tokens in a row. This task involves taking a stone and offering the stone to the opponent who is further asked to take the token away.

1. Take Position Evaluation

   Input The position of the token which should be taken by Nao and Nao's current board side

   Output A side of the game board where the robot needs to go and a corresponding arm index in order to take the token from the given position.

   Explanatory notes This task is the same task as already mentioned in 5.6.5. The only difference is the purpose of the movement which does not affect the explained the algorithm itself.

2. Walk to board side

   Input A specified board side

   Output The robot reached the center of the given board side

   Explanatory notes Further explanations to this task were presented above in 5.6.5.

3. Refine Position

   Input The position of the token as ideal position on the game board

   Output The real position of the token on the game board

   Explanatory notes In this task the real position of the token is estimated. Based on the information concerning the center of the detected token in the image, a real position with respect to the marker is computed. For exact details on how the position is refined read [3]. It is important to mention that this position is always estimated after the robot has reached the correct side of the game board. This highly improves the position evaluation since the effects of perspective distortion are limited because of the camera angle. Another advantage of this execution is to reduce the risk of occlusion. This is also closely bound to the camera angle as shown in 2.2.

4. Walk to a possible grasp position

   Input The current robot position within the environment

   Output A position which is viable for grasping the token from the given position

   Explanatory notes This task is also similar to the place action described in 5.6.5.

5. Grasp token from a position

   Input -

   Output Nao holds the grasped token.

Explanatory notes Since this task involves movement it automatically incorporates semi-static trajectory motion. It is important to mention that even the position refinement does not guarantee a successful grasp action. In fact the grasp action may fail due to various reasons including the lack of separate finger control, finger weakness and the entire hand setup. As mentioned above visual occlusion of the grab is not possible because of the hand occluding the fingers and the token in several cases.

6. Check token and wait for confirmation

   Input The assumption that the robot successfully grasped the token.

   Output Visual confirmation of the grab

   Explanatory notes The robot moves its arm towards its face and looks at its fingers. If the token is in its hand the grasp is confirmed, otherwise it has failed. In this case the implementation of self-repair would also be possible, but it has not been incorporated yet. The idea would be that the robot checks the game board for differences between the last game state and the current state in order to find the missing token. If the token slipped out of Nao's hands during the movement a repair would not be possible without the help of the human opponent. However, it would be possible if the token has not been moved because Nao's thumb missed the token in the first place. In this case probable correction scenarios would need to be evaluated in order to successfully grab the token when executing the next try.

7. Open hands

   Input The token is currently in Nao's hand

   Output The opponent removes the token

   Explanatory notes The robot raises the arm, opens its fingers and asks the opponent to remove the token. The hand is visually observed and as soon as the token is removed the robot continues with its next step. During the evaluation phase of the project, it turned out that in some cases the robot was not able to grasp the token correctly. When it opens its fingers, the token may automatically drop. This was however not detected and would also be interpreted as viable removal.

8. Relax arm

   Input Nao's hand is not holding the token anymore

   Output Nao's arm is moved to its relax position at the side of Nao's torso.

   Explanatory notes This has been integrated just to ensure that the starting scenario is the same as the ending scenario concerning the robots state. Moreover this guarantees that none of Nao's arms is blocking the visual perception.

Execute A Move Command

Executing a move command is somehow like a place and grasp action without the user interaction but including the restriction of using the same hand for both motions. This is especially relevant for the algorithm which computes the required board sides and the arm index for this task. Algorithms which were presented above are not further presented here. Only their basic idea is presented by stating Input and Output of the steps. These may refer to data as well as to a current robot's state.

1. Take Position evaluation

   Input Position of the token to grab and Nao's current board side

   Output Board side of where to go and arm index

2. Walk to board side

Input Current Position

Output The robot reached the center of the given side

3. Refine Position

   Input An ideal token position on the board

   Output The real token position on the board

4. Walk to a possible grasp position

   Input The current robot position within the environment

   Output Nao is in a position which is viable for grasping the token at the given position

5. Grasp token at the given position

   Input -

   Output The robot holds the token

6. Check token and wait for confirmation

   Input Current camera image

   Output Grab is confirmed

7. Relax arm

   Input Arm is above the table

   Output Arm is moved to the side of Nao's torso

8. Place position evaluation

   Input The given position where the stone should be placed and the given arm index and Nao's current board side

   Output A board side in order to place the stone properly

   Explanatory notes In this case the robot already holds the stone in one of its hands. Of course the same hand needs to be used in order to place the stone. In this case the arm index is not changed, but the board side is selected based on the given arm index.

9. Walk to board side

   Input Current position

   Output The robot reached the center of the given side

10. Walk to a position where placing is possible

    Input The current robot position within the environment

    Output A position which is viable for placing the token at the given position

11. Place token

    Input -

    Output The robot does not hold the token anymore

12. Relax arm

    Input Arm is above the table

    Output Arm is moved to the side of Nao's torso

During the second phase of the game, it is only allowed to perform moves to nearby positions on the game board. In this case the algorithm, which was introduced in order to minimize the number of required walks, always finds a solution which requires one or less walks. Because reaching neighboring positions on one side is always possible to be realized by using one hand without any movement. During the last game phase where the player is allowed to jump, more than one walk may be required in order to execute a move command.

5.6.6. Nao Vision

Overall Purpose This package is responsible for all issues that are only concerned with the visual perception and image processing. Thus it is responsible for retrieving the camera image from the robot, the creation of binary images in order to find the player's tokens as well as the blue stripe. Furthermore, it is responsible for detecting the marker and also providing the transformation between the marker and the camera of the robot. Another purpose of this package is to classify the segmented tokens in terms of position and thus transform the retrieved image data into game board information which is further sent to the world model.

This package involves the following nodes:

- Blob Detection: This node is responsible for transferring the current camera images into binary Blob images [Blob = "Binary Large OBject"]. Based on these images the position of a certain token is evaluated. Furthermore, this node is responsible for checking the robot's hand for a token in case the robot wants to get a token from its opponent.
- Binary Image To Board Info: This node transfers the binary image data into real board info data. This includes the position classification for visible tokens and this is done by image rectification and mask comparison [3].
- Augmented Reality Pose (ARPose): This node is responsible for marker detection and broadcasting the transformation between camera and marker throughout the entire framework.

5.6.7. Nao World Model

Overall Purpose This package is responsible for collecting data from different nodes and furthermore generally providing this data to all nodes in the framework. The data managed by the world model includes the obstacle map of the environment (occupancy grid map), the current joint state of the robot as well as sensor values, the 3D representation of the environment, the current robot and vision status in terms of the current robot movement or vision, as well as a broadcast the entire set of robot related transformations incorporated in the framework.

The package features the following nodes:

- Nao Get Data: This node was written in Python because it directly interacts with the NaoQi. It collects sensor information from Nao's buttons as well as the information from Nao's foot contact sensors. Furthermore, it provides an interface in order to set the colors of Nao's LEDs or force it to speak a given text.
- Robot State: This node was also realized in Python for the same reasons. It consequently requests the current state of the robot's joints from the NaoQi within a certain time interval. However, the time interval is flexible and will be shorter if the robot is currently moving.
- World Model: This is the main node. It provides all the information described above by collecting data from different nodes and packages. This node includes the generation of the occupancy grid map as well as the generation of 3D objects to represent the environment in RViz.

Chapter 6

Experimental Evaluation

In this thesis an overview of a framework which enables the robot to play Nine Men's Morris was presented. The framework was realized within the ROS environment and further made use of Nao's built-in interface NaoQi. It is not possible to run the framework entirely on the robot because of the size of the database representing the game logic and the overall computational demands of the framework. Thus the only operations executed on the robot's CPU concern the processing of the requests to NaoQi. Hence the robot needs to be connected via a network to a computer. Throughout the project various problems concerning the network latency occurred when Wlan connection was used. In fact it was not possible to play the game using the robot because the transmission of data between the main computer and the robot required too much time. The entire framework was realized by Bock and myself. Since each mentioned task in the experimental section features parts of the vision as well as tasks referring to the mobile manipulation, the test results presented in this section concern the entire framework. Thus the results presented here are also illustrated in [3].

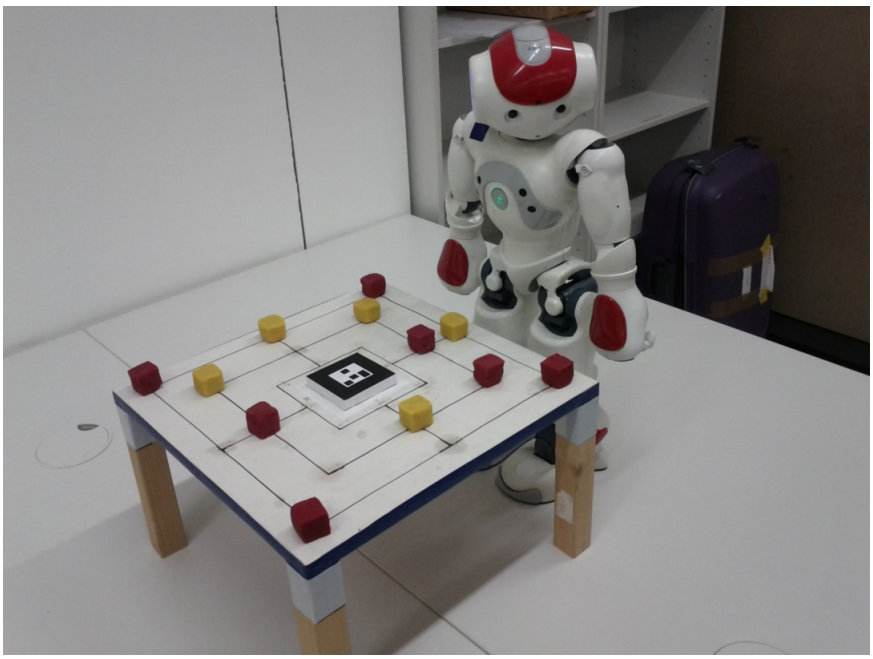


Figure 6.1.: Real robot setup

Figure 6.1 shows the general setup as it was built for the project. The material chosen for the tokens was modeling clay. There were many reasons for this decision. In fact the robot's fingers are not that strong since they are only controlled by a small cable pull inside the robot's wrist. Thus a material was required which was soft and sticky. On top of that the form of the tokens can be easily modified to meet additional requirements if the are made of modeling clay. Of course other advantages of using modeling clay are that it is easy to produce and it is also easy to generate tokens with different colors.

In this setup block shape tokens were used. As mentioned above the overall goal of the framework was to successfully play the board game Nine Men's Morris against a human opponent. In the given case the AI of the robot was represented by a database. However, this database was not finished at the time this thesis was handed int. Hence it was not possible to test the process of playing an entire game autonomously. In order to test the robot's ability of playing the game, a database simulator was incorporated into the framework. This simulator allows humans to enter the moves the robot is supposed to perform. The simulator features a text-based interface as shown in Figure 6.2.

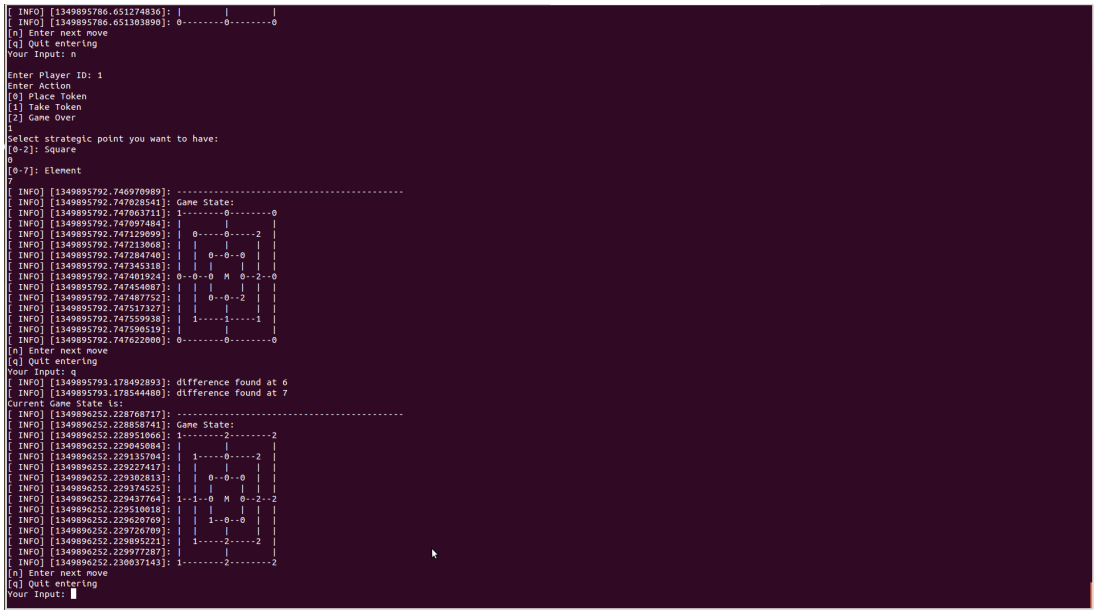


Figure 6.2.: Interface of the data base simulator

The interface features a representation of the current game state as well as the performance of the two basic types of interaction needed to play Nine Men's Morris, namely placing and taking a token. A token movement may be realized by first taking the token from its initial position and then placing it in its goal position. The database is also capable of noticing if a game is won or lost. This function was also incorporated into the simulator by providing a corresponding option.

As mentioned above the ROS environment also provides the possibility of visualizing project data. In the given case the entire framework and all the relevant data is visualized using the RViz node of ROS. The virtual representation of the current state is shown in Figure 6.3.

Figure 6.3 also features a simple robot model which was taken from the nao_common stack of the ROS framework. As mentioned in previous sections, the world model requests the current joint values of the robot within a certain time interval and provides the result to the incorporated model. Thus the model represents the current state of the robot any time. The red squares on the game board indicate ideal token positions on the game board. A certain value for each position is stored which either refers to one of the players or to no one occupying the respective position. If a player's token is placed on a certain position and the current game state has already been evaluated, the token is drawn on the corresponding position on the game board. On the right side of RViz the current camera view overlain by the data is shown. It is easy

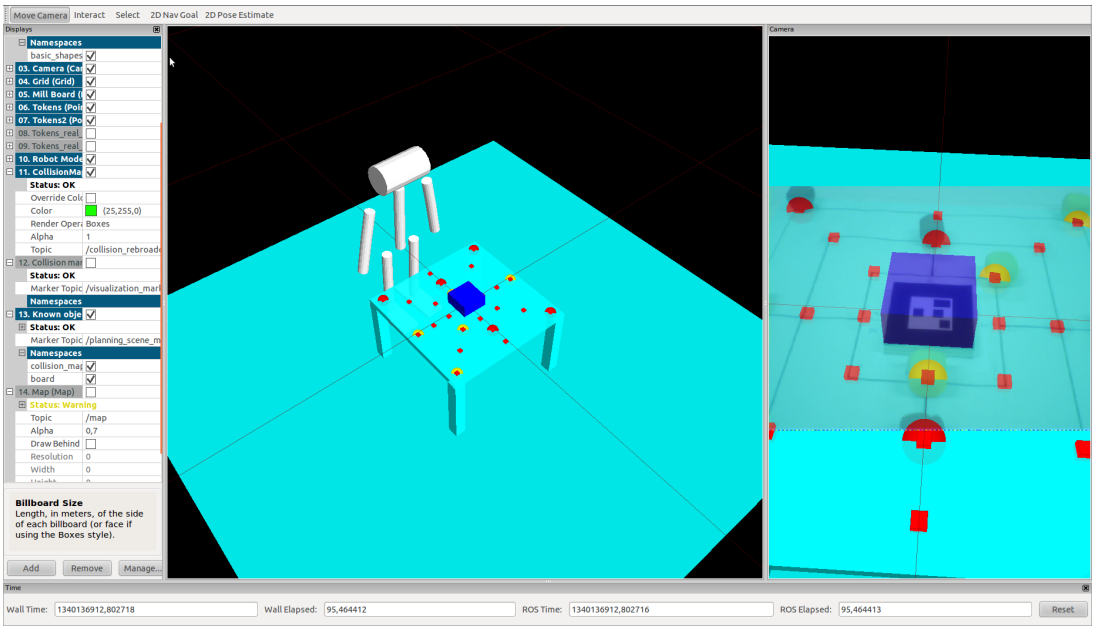


Figure 6.3.: Internal representation of relevant data using RViz node of ROS

to see that the tokens which are currently within the robot's sight were correctly classified. The blue square shows the estimated position of the marker.

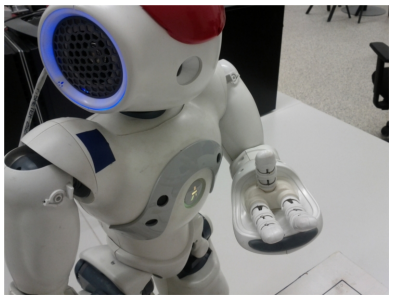
However, with the help of the mentioned interface, it was tried to play the game various times. Because of problems concerning the robot's joints it was not possible to finish a game. In the tested scenarios the human opponent quickly made his/her move and thus did not allow Nao's joints to cool down after its previous move. This lead to serious issues after a couple of moves. Apart from loosing execution accuracy the robot was not capable of performing walk requests anymore. In the mentioned cases it seemed that the robot lost the stiffness in some of the joints in the lower part of its body. As a consequence the walk execution failed and the game could not be finished. The first time this error occurred, eight robot moves were finished. But it was not possible to continue the test even after spending some time on relaxing the robot's joints. The robot always lost the joint stiffness in its legs after a couple of moves. After some research it turned out that the power cable needs to be unplugged while operating the robot. In fact the power cable was always plugged in to test the robot's functionality for a certain period of time. The error did not occur in several tests after the cable was unplugged. However, there was not enough time to proof the link between the error that had occurred and the power cable being plugged in.

In each of the mentioned test games human help was required at some point. In order to get accurate test results the robot actions required for playing the game were decomposed into the basic actions. Every single action was tested several times to evaluate the robot's overall performance on playing Nine Men's Morris. The evaluated basic tasks were:

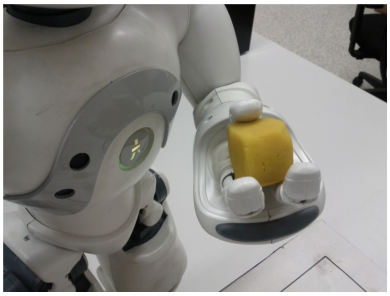
- Placing a token
- Taking a token
- Game state recognition
- Walking

6.1. Placing A Token

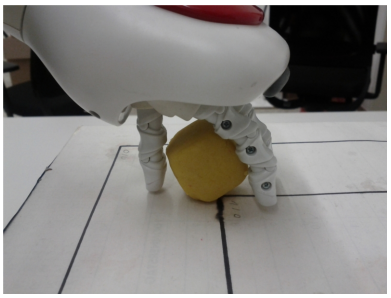
There are two actions concerning the placement of a token while playing Nine Men's Morris. The first type of placement occurs during the first phase of the game. In this case the user is required to place a token in the robot's hand, which is further placed on the game board. The second type of placement may occur in any other game phase. This type features the placement of a token after it being grasped by the robot. The major difference between these two tasks is that in the first case the human opponent is responsible for placing the token properly in Nao's hand, whereas in the second case the placement of the token in the hand depends on the quality of the grasping action performed previously. Since the grasping was tested separately the test concerning the placement of a token just focused on token placement after the stone was placed in Nao's hand by a human. In comparison to the other tasks required for playing the game, this task is considered to be the easiest.



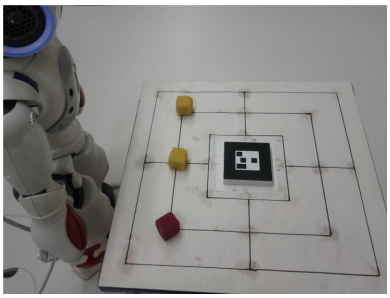
(a) Waiting for a token



(b) Recognized token and closed fingers



(c) Placing the token



(d) Game board after placement

Figure 6.4.: These figures illustrate the execution steps while grasping. Figure 6.4a shows the robot waiting for a token. In Figure 6.4b the token had already been recognized in the robot's hand and its fingers were closed. Figure 6.4c shows the placement action itself. It can be imagined that if a token is placed in a bad way in the robot's hands, it may consequently be placed in a bad way on the game board. If the token is placed one of its edges, it may even flip over and thus the resulting position may be considerably far away from the desired position. Finally Figure 6.4d shows the game state after the placement.

Table 6.1 shows the results of placing a token. The process was tested 30 times. As mentioned above in each case the human opponent placed a token in Nao's hand. Because of that the token was successfully detected within the robot's hand in every case. As table 6.1 illustrates the placement was successful in 28 out of 30 tests. In one case the token was placed on one of its edges. When the robot opened its hand the token tilted over, away from the desired goal position. In this case visual confirmation for the token position failed leading to a scan of the entire game board to find the missing token. The robot found the misplaced token, grasped it and placed it again on the desired position, this time with better results. In this case the position and thus the placement of the misplaced token was automatically repaired by the robot. However, in one case the token placement was also not accurate, but still the robot was able to visually confirm the placement. But later on the robot was required to change the board side. Since it was not possible to correctly classify the token in question from the other board side, this case was counted as repair fail. As already mentioned the repair process was not started in this case because the visual position

confirmation did not fail.

		Repair place		Σ	%
		Success	Fail		
Place	Success	28	0	28	93.33
	Fail	1	1	2	6.66
Σ		29	1	30	
%		96.66	3.33		

Table 6.1.: Test results for placing a token

6.2. Taking A Token

Taking a token does also involve two different ways of execution. The first way applies if the robot successfully manages to place three tokens in a row, and is therefore allowed to take one of the opponent's tokens. In this case the action after the grasp action is to check if the token is in the robot's hands and then open the fingers and asks the opponent to take away the token. In this scenario the robot does not continue before the opponent has taken away the stone. The second type of grasping comes up in the second and third phase of the game when the robot needs to move a certain token. In this case the robot takes the token and checks if the grasp was successful. After the confirmation of the grab the robot continues with any action needed at that moment in order to place the token on the desired position.

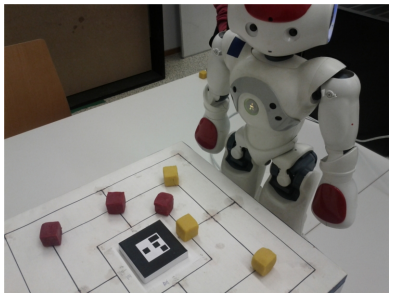
Grasping is considered to be the harder task because the real position of the token has to be computed first. In addition the computed position needs to be quite accurate in order to successfully grasp a token. Since the robot has only three fingers which cannot be controlled independently, the thumb is required to be placed properly. If the robot's thumb misses the token, the grasp fails. Based on a single image it is hard to determine the token's real 3D position with the required accuracy. As mentioned before the camera image is segmented by color in order to retrieve a binary image for each player's tokens. In the resulting image the tokens are represented by white Blobs (Binary Large OBjects). In order to compute the real 3D position of a token, the center of this Blob is estimated. The 3D position is computed based on the 2D position in the picture. Details on that computation are shown in [3]. However, the retrieved center of the Blob may be close to the real center of the token or not. The accuracy of the computed position depends on the current camera angle. However, the given scenario does not allow visual confirmation of the grasp because in many cases the desired token is completely occluded by the robot's hand. This also made it impossible to compute any form of correction for the token position if a grasp fails. It is not possible to detect how the robot has to correct the positioning of its arm in order to successfully grasp the token.

Another problem concerning the grasping occurs if the grasp was successful but the robot only holds the token with two fingers. In such scenarios it is possible that the token falls out of the robot's hand during the arm movement. On top of that the recognition of the token in the hand may fail because the token is not fully visible in the robot's hand.

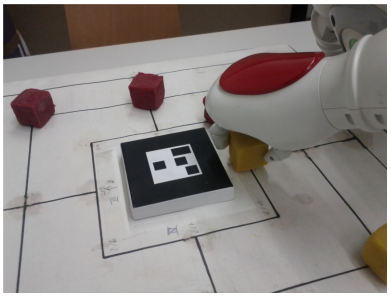
Table 6.2 illustrates the results of the grasp action. A total of 30 tests were carried out. As shown in the table 18 tests were entirely successful. This means that the token was grasped successfully and it was also recognized in the robot's hand after the grasp. In ten cases the grasp entirely failed. In these cases the thumb missed the token and thus it was not possible to grasp it. The recognition of the tokens in the hand always detected correctly that the token has not been grasped. In two of the mentioned cases the grasp worked, but the token was only held in the hand by a slight edge and thus the recognition did not report the token to be in the robot's hand.

		Recognition		Σ	%
		Success	Fail		
Grab	Success	18	2	20	66.66
	Fail	10	0	10	33.33
Σ		28	2	30	
%		93.33	6.66		

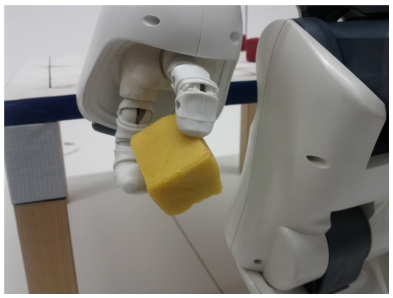
Table 6.2.: Test results for grasping a token



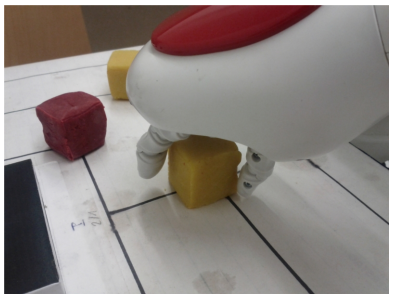
(a) Game state before movement of the token



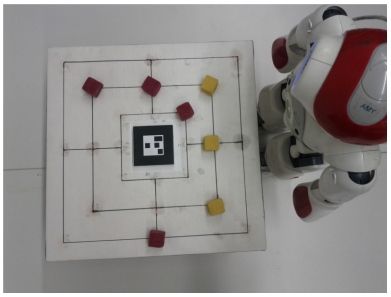
(b) Grasping the token



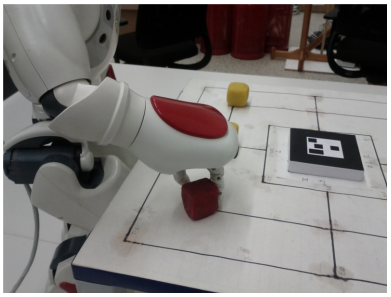
(c) Holding the token during examination of goal position



(d) Placement of the token



(e) Game state after movement of the token



(f) Example grasp fail with thumb missing the token

Figure 6.5.: These figures illustrate major execution steps when moving a token. A token movement incorporates proper grasping and placement of the token. Figure 6.5a shows the game state before the movement is executed. Figure 6.5b illustrates the grasping. In this case the grasp is executed in a very good way since the token is held by all of Nao's fingers. Figure 6.5c shows the hand holding the token with the arm being placed at the side of Nao's torso. As mentioned above this was necessary in order to avoid the blocking of Nao's vision by its arm. Figure 6.5d demonstrates the execution of the placement in order to finish the move. Figure 6.5e shows the result of the game state. Figure 6.5f shows an example of a grasp fail with the thumb missing the token.

6.3. Game State Recognition

The game state recognition is very important concerning the goal of playing the game autonomously. If a certain token is not recognized but this token is required to be moved next by the robot, the robot will not be able to execute the given move. However such misclassification may be identified in the future with the help of the database. The database is expected to recognize if a certain game state is not possible to be followed up with another one. Given the previous and the current game state a request can be send to the database in order to validate the current game state. If this validation fails, the robot needs to estimate the game state again to get better results.

Table 6.3 shows the results of classifying tokens and estimating the current game state. In order to test this several scenarios featuring game states with different numbers of tokens between five and eighteen were created. Of course the classification task is not an issue related to the mobile manipulation task but the results presented here are meant to be results for the entire framework. As mentioned above, tasks concerning vision are further described in [3].

Number of tests		38	%
Number of classified tokens	Correct	359	98.63
	Wrong	5	1.37
Game state without any misclassification	Correct	33	86.84
	Wrong	5	13.16

Table 6.3.: Test results for getting the game state

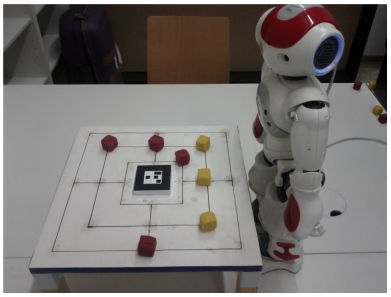
6.4. Walking

The walk of the robot mainly consists of four parts. These include stepping away from the board, walking to the desired side of the game board, re-localizing its position when the desired board side is reached and finally stepping closer to the board. The entire process of walking was tested 23 times. In case the robot is required to walk to the opposite side of the board, the walk is split into separate walks by only allowing the robot to walk to a nearby side of the game board. This aspect was incorporated in order to improve the execution accuracy. In some cases, when the robot reaches the desired side, the re-localization may fail because the robot ends up too far away from the game board to recognize the marker. In this case the detection of the blue stripe attached to each of the game board sides is started. As soon as the blue stripe is recognized the robot walks towards it. After getting closer to the board the re-localization process starts all over again because it is now assumed that the robot is able to find the marker due to the shorter distance between the robot and the game board.

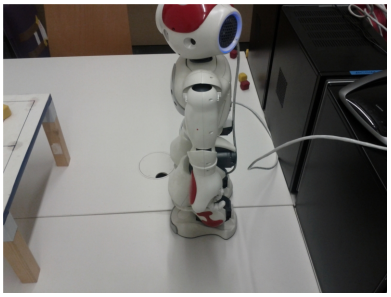
Table 6.4 depicts the results of the walk task. In 19 cases the walking execution was successful. However, in nine of these cases the marker was not recognized after reaching the desired side of the game board. Nevertheless, the walks were finished successfully by using the blue stripe detection. In four cases slight collisions with the robot's arm and the game board occurred. In these cases the accuracy of the odometry was for no apparent reason lower than in the other cases, because all tests took place in the same environment and thus on the same environment ground material. However, neither of these collisions did affect the game state itself nor the continuation of the movement execution and therefore these collisions were not counted as walk fails.

		Walk			Σ	%
		Success	Slight collisions	Fail		
Marker	Found	10	4	0	14	60.86
	Lost	9	0	0	9	39.13
Σ		19	4	0	23	
%		82.61	17.39	0		

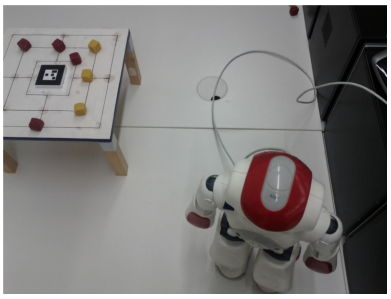
Table 6.4.: Test results for walking



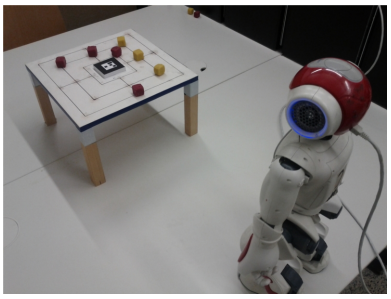
(a) The robot stands close before walking procedure starts



(b) Nao walked away from the game board



(c) The robot walks to the desired board side



(d) Nao reached the desired side and starts walking closer to the board

Figure 6.6.: These figures illustrate the walk execution. The robot starts the walk from a position like the one illustrated in 6.6a. The first step during the walking process is to step away from the board in order to establish the plan creation. The position after stepping away is shown in 6.6b. Figure 6.6c illustrates some state during the walking process. In the last picture the robot has already reached the goal side. The final step which needs to be executed after reaching the position shown in 6.6d is to step closer to the table after the robot has re-localized itself.

6.5. Evaluation Of Implemented Algorithms

In this section the algorithms concerning mobile manipulation are evaluated. This includes the algorithm for finding a solution for the inverse kinematics on a given point and the evaluation of the algorithm for creating the workspace pictures.

6.5.1. Evaluation Of The Kinematics

As mentioned in Section 3.3.3 the algorithm for solving the inverse kinematics selects torso postures in an iterative manner and computes corresponding joint values for each posture. Afterwards this each posture is evaluated and the best posture according to a certain evaluation function is chosen.

Since the posture is only limited to a selected pitch and height value, the algorithm has an execution time

of $O(h \cdot \alpha)$. The values for h and α depend on the defined height and pitch limits and on their corresponding resolution. In the present case these values are:

$$\begin{aligned} h &= \frac{(h_{max} - h_{min})}{\Delta_h} = \frac{0.333 - 0.259}{0.01} = 7.4 \\ \alpha &= \frac{\alpha_{max} - \alpha_{min}}{\Delta_\alpha} = \frac{30 - (-10)}{1} = 40 \end{aligned} \tag{6.1}$$

The algorithm was tested on an Intel(R) Core(TM) i7 CPU with 2.8 GHz using 4 GB of RAM and a 32 bit Ubuntu 11.10 installation. In order to test the algorithm the entire framework was required to run. The required times were estimated with the ROS built-in time measurement. The test was split into six parts. For each hand the time needed to compute joint angles for positions on the three different squares of the game board was evaluated. The naming of the squares are shown in 6.7. The current evaluation function focuses on keeping the torso in an upright position. Thus the algorithm stops the execution if a solution is found featuring a torso pitch of $\alpha = 0°$. Since such solutions are in general more likely at positions in the outer squares of the Nine Men's Morris board, the execution time is significantly lower when evaluating positions on these squares. This issue is also illustrated in table 6.5. The presented times in this table represent average execution times of a total of 200 executions.

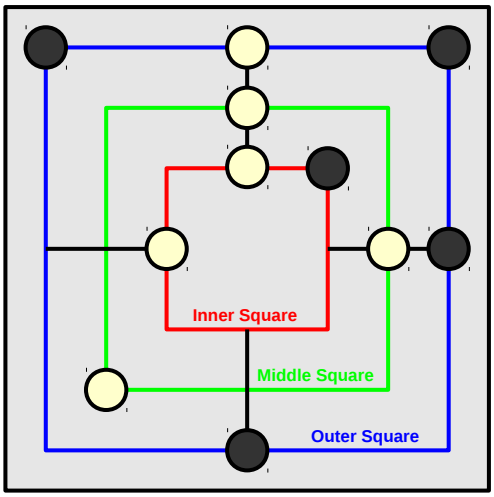


Figure 6.7.: Nine Men's Morris square naming

Used Arm [m]	Used Game Square	Required Time [msec]
Left Arm	Inner Square	15.371
Right Arm	Inner Square	15.383
Left Arm	Middle Square	26.831
Right Arm	Middle Square	26.075
Left Arm	Outer Square	63.395
Right Arm	Outer Square	64.172

Table 6.5.: Results for algorithm solving the inverse kinematics

6.5.2. Evaluation Of The Algorithm For The Creation Of Workspace Pictures

In Section 5.4 the algorithm for the creation of workspace pictures was illustrated. As mentioned previously the algorithm evaluates the kinematics for an area featuring 0.6 m length l and 0.45 m width w in front of a board side. This means that the execution time for this algorithm can be stated as $O(h \cdot \alpha \cdot x \cdot y)$. The values for α and h are the same as the ones mentioned in Section 6.5.1. The remaining values for x and y depend on the user input concerning the corresponding resolutions. For the evaluation of the algorithm a resolution of 0.001 m per pixel was selected.

$$\begin{aligned} x &= \frac{l}{\Delta_x} = \frac{0.6}{0.001} = 600 \\ y &= \frac{w}{\Delta_y} = \frac{0.45}{0.001} = 450 \end{aligned} \tag{6.2}$$

Table 6.6 depicts the results of the picture creation. The execution of this algorithm also requires the entire framework to run. It was executed on an Intel(R) Core(TM) i7-2 CPU with 2.2 GHz using 4GB of RAM and a 32 bit Ubuntu 11.10 installation. As the table shows, the creation of the pictures concerning the right arm required nearly twice the amount of time compared to the pictures related to the left arm. It was not possible to find the reason for this phenomenon at the time this thesis was handed in. In general the algorithms should require nearly the same time for both hands since the pictures are symmetric. The game board height mentioned in the table is given with respect to the real game board height. The required times were estimated with the ROS built-in time measurement.

Game board height h [m]	Used Arm	Required Time [sec]
-0.03	Left Arm	8791
-0.03	Right Arm	17593
-0.02	Left Arm	8650
-0.02	Right Arm	17155
-0.015	Left Arm	8480
-0.015	Right Arm	16931
-0.010	Left Arm	8320s
-0.010	Right Arm	16613
-0.005	Left Arm	8153
-0.005	Right Arm	16240
0.005	Left Arm	7778
0.005	Right Arm	15532

Table 6.6.: Results for algorithm for creation of workspace pictures

Table 6.6 also shows the effects of having a smaller workspace in comparison to a bigger one concerning the execution time. In 5.4 the pictures at different heights were compared. It turned out that the amount of feasible points to grasp a certain position rises with the height of the table within a certain range for h. However, the algorithm has been optimized in order to take care of that. The goal of this algorithm is to create a picture which distinguishes between positions from which a certain point may be grasped and positions where such a grasp is not possible. Thus the algorithm does not search for the optimum joint values to meet an evaluation function as realized within the algorithm mentioned in 6.5.1. The algorithm mentioned in this section immediately jumps to the evaluation of the next point if a valid joint configuration was found in order to grasp the given point from the current position. However, if no valid joints can be found for a certain position, the algorithm is required to take any possible configuration in terms of torso height and pitch into account. Hence the evaluation of pictures with more possible grasp positions requires less time than the evaluation of pictures with fewer possible grasp positions. This fact can be illustrated by comparing the required time of either the left arm or the right arm at a height of -0.03 m with the time

required at a height of 0.005 m. Since the latter one incorporates a higher number of valid grasp positions, the execution time is lower.

Chapter 7

Conclusion

The framework and basic concepts presented in this thesis represent a first step towards enabling humanoid robots to interact via board games with a human opponent in a human-like environment. Therefore the framework solely relies on the robot's built-in sensors. Due to the size of the framework and especially the size of the database representing the artificial intelligence to play Nine Men's Morris, it was not able to execute the entire framework on the robot's CPU. In addition the Wlan adapter of the robot could not be used because of the higher network latency. Nevertheless, various techniques were incorporated in order to reduce the required network traffic, but with only limited success. Any movement command sent to the robot via Wlan required a couple of seconds before the execution started. The transfer of images from the robot to the framework over Wlan took even longer. Thus it was not possible to play the game using Wlan, which meant that the robot was required to be attached to the network cable at all times.

During the development various challenges were met which were not thought of in the first place. One of the biggest issues in terms of robot movement was to solve the inverse kinematics. The first attempts in order to solve the inverse kinematics made use of the "IKFast" module of OpenRAVE [17] or the "Kinematics and dynamics library" (KDL) of the orocos project [18]. In both cases it was not possible to retrieve viable solutions. Hence it was decided to implement an own solution. However the attempts including OpenRAVE and KDL were only applied to a model of the robot's arm which just features 5 degrees of freedom.

Another challenge was to generate a walking plan which avoids collisions with obstacles by just relying on the robot's built-in odometry. The first approaches did not incorporate visual odometry and thus the results were not satisfying. The robot collided with the table in too many cases although the security distance to the table was increased several times. During the tests it also turned out that the accuracy of the robot's odometry highly depends on the ground material and stability. Another issue concerning execution accuracy was related to hot joints. When the robot's joints were required to be stiff for a longer period of time, they became hot and thus lost accuracy. In some cases this effect led to serious stability issues while walking and the robot had to be shut down in order to cool its joints. Therefore the framework automatically relaxes the robot's joints whenever possible which usually happens to be at the end of the robot's move. Of course a quick opponent needs to keep that in mind because if the robot has to be stiff all the time because the opponent performs its moves so quickly the issue may arise anyway. In order to improve the accuracy of the robot's odometry the concept of visual odometry was incorporated into the project. This technique turned out to be very valuable since the amount of collisions was significantly reduced by then. However, it was not possible to completely avoid them. In general most of the time only slight collisions with the game board and the robot's hand occurred which did neither affect the game situation nor the execution stability of the framework.

The most challenging task which needed to be dealt with turned out to be the grasping of tokens. Due to recognition tolerances and the way the robot's hands are designed, a successful grasp was hard to achieve.

One big problem in this context was that no method was found which was capable of significantly improving grasping results. In other approaches in which the Nao was used for playing games like [11] it was possible to visually confirm the position where the stone needed to be placed. However, as far as we know no implementation exists so far which features safe grasping of standard game tokens like in the presented scenario. In general visual confirmation is hard to apply in the given case. This is because in most cases the robot's hand completely occludes the desired token when grasping it. Another issue concerns the hardware. The fingers of the robot seem to be rather fragile and on top of that all fingers of one hand are controlled by a single cable pull located in the robot's wrist. This means that the fingers cannot be controlled independently. If the hand is placed over a token and the robot's thumb misses the token the grasp automatically fails. Another point is that a correction is also hard to implement because as already mentioned the grasp action cannot be visually observed. So if a grab fails, it is not possible to retrieve any information about how the robot has to correct the given grasp position in order to try it again and grab the token successfully.

Given all these problems the robot is still able to play the game even though some moves during execution may fail due to the mentioned reasons and thus the help of the human opponent is required. The given framework represents a first step towards the generation of a general framework enabling Nao to play different board games. The presented concepts and solutions have been tested several times and have been considered rather stable, however there is still room for improvement. Finally it should be mentioned that the database computation was not finished at the time this thesis was handed in. Thus it has not been tested in combination with the framework. However, the framework included also a database simulator which was used to control the robot's movement in some test games.

7.1. Future Work

The presented framework provides a basic program which enables the robot Nao to play the board game Nine Men's Morris. However, there is still room for improvement in terms of execution stability but also in terms of flexibility.

When it comes to flexibility other games like chess may also be incorporated into the framework. Of course this would require some adjustments to the framework. In case of chess the tokens need to be distinguished not only by color but also by form. Apart from that it is not sure if the robot is physically able to grasp the different types of chess tokens. Another game which may be incorporated into the framework is Connect Four since [11] has already presented an approach with the Nao successfully playing that game. One advantage of this game is that the robot is actually not required to grasp game tokens, but in contrast the placement action is required to be rather precise because of the physical game setup of Connect Four.

As mentioned in the given thesis the solution for the arm navigation was semi-static. In order to provide greater flexibility especially when considering the implementation of other board games, a dynamic path planner would be a better choice. The ROS environment currently features an arm navigation stack which is capable of dynamic path planning. However, this was not incorporated into the framework because in the given case the semi-static trajectories turned out to be a good choice. As far as we know, the arm navigation stack has not been used for creating trajectories for a humanoid robot so far. Furthermore, the tutorials within the ROS Wiki [24] mainly address the use of the arm navigation stack for Willow Garage's own robot PR2, which is not a humanoid robot. Given these facts the use of the arm navigation stack has not been taken into account yet.

Apart from that the recognition of the token position may be improved by finding a solution to somehow implement visual grab confirmation before the arm is raised from the game board. Such a mechanism would highly improve the success rate of the grasping actions as a whole. In terms of kinematics the given solution is also closely bound to solving the given problem. Thus the kinematics is only capable of dealing with different pitch and roll requests for the forearm. In order to provide greater flexibility in this part of the project further functions within the kinematics class need to be incorporated as well.

Another issue within the current framework is the lack of real human-robot interaction. In order to improve the look and feel of playing board games with a robot, incorporating several possible reactions to certain movements of the player would also be a good idea.

In terms of playing Nine Men's Morris a possible improvement would also be to enable the robot to grasp tokens from a token stack during the first phase of the game. As mentioned above this would of course lower the execution time, but still it is an interesting task to think of since it features token recognition and grasping from a location other than the game board. Another possible improvement would be to enable the robot to actually pass the token from one hand to the other one. In some cases this could lead to a speedup in terms of execution and on top of that it could reduce the walks necessary to perform a certain move.

Of course the walk execution may also be improved. The execution of rotational and translational moves are strictly split in the given setup. However, combining them in order to make the resulting walk trajectories more human-like would also be an interesting task.

Bibliography

[1] AICHHOLZER, O., DETASSIS, D., HACKL, T., STEINBAUER, G., AND THONHAUSER, J. 2010. Playing pylos with an autonomous robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2507–2508. (Cited on page 3.)

[2] ALDEBARAN. 2012. Nao software documentation. Online. https://developer.aldebaran–robotics.com/doc/1–12/index.html Accessed 26.9.2012. (Cited on pages 12, 13, 14, 15, 16, 19, and 55.)

[3] BOCK, S. 2012. Autonomous nine men's morris position and game state recognition using a humanoid robot platform. (Cited on pages i, iv, 1, 6, 7, 39, 41, 46, 54, 58, 61, 63, 67, and 69.)

[4] BOOST. 2012. Boost library. Online. http://www.boost.org/ Accessed 3.10.2012. (Cited on page 55.)

[5] BOSSCHER, P. AND HEDMAN, D. 2009. Real-time collision avoidance algorithm for robotic manipulators. In *Technologies for Practical Robot Applications, 2009. TePRA 2009. IEEE International Conference on*. 113 –122. (Cited on page 37.)

[6] DALIBARD, S., NAKHAEI, A., LAMIRAUX, F., AND LAUMOND, J. 2010. Manipulation of documented objects by a walking humanoid robot. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. 518 –523. (Cited on page 35.)

[7] DALIBARD, S., NAKHAEI, A., LAMIRAUX, F., AND LAUMOND, J.-P. 2009. Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. 355 –360. (Cited on page 35.)

[8] DIGIOIA, G., ARISUMI, H., AND YOKOI, K. 2009. Trajectory planner for a humanoid robot passing through a door. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. 134 –141. (Cited on pages 1 and 35.)

[9] GFRERRER, A. 2008. Kinematik und robotik. Online. http://www.geometrie.tugraz.at/lehre /KinematikRobotik/Kinematik_und_Robotik.pdf. (Cited on pages 16, 18, 19, and 20.)

[10] HSU, F.-H. 2004. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, Princeton, NJ, USA. (Cited on page 3.)

[11] HUMAROBOTICS. 2012. Nao plays... connect 4. Online. http://www.generationrobots.com/site/program–nao–robot/index_en.html. (Cited on pages 37, 42, 43, and 76.)

[12] HUSTY, M., KARGER, A., SACHS, H., AND W., S. 1997. Kinematik und robotik. Springer Verlag. (Cited on page 18.)

[13] KATO, H. AND BILLINGHURST, M. 1999. Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. In Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on. *Augmented Reality, International Workshop on 0*, 85–94. (Cited on pages 5 and 34.)

[14] KITANO, H., ASADA, M., KUNIYOSHI, Y., NODA, I., OSAWA, E., AND MATSUBARA, H. 1997. Robocup: A challenge problem for ai and robotics. In *RoboCup*, H. Kitano, Ed. Lecture Notes in Computer Science, vol. 1395. Springer, 1–19. (Cited on page 2.)

[15] MILLER, A. AND ALLEN, P. 2004. Graspit! a versatile simulator for robotic grasping. *Robotics Automation Magazine, IEEE 11,* 4 (dec.), 110 – 122. (Cited on page 29.)

[16] OPENCV. 2012. Opencv library. Online. http://opencv.org/ Accessed 26.9.2012. (Cited on page 34.)

[17] OPENRAVE. 2012. Openrave library. Online. http://openrave.org/ Accessed 3.10.2012. (Cited on page 75.)

[18] OROCOS. 2012. The orocos project. Online. http://www.orocos.org/ Accessed 26.9.2012. (Cited on pages 34 and 75.)

[19] REGENFELDER, O. 2010. Implementation of games in a generalized framework for solving combinatorial games using the state space. In *Faculty of Computer Science, Graz University of Technology*. (Cited on pages i, iv, 2, and 41.)

[20] REIMANN, H., IOSSIFIDIS, I., AND SCHONER, G. 2010. Generating collision free reaching movements for redundant manipulators using dynamical systems. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. 5372 –5379. (Cited on page 37.)

[21] THOMAS MOULARD, FLORENT LAMIRAUX, P.-B. W. 2010. Collision-free walk planning for humanoid robots using numerical optimization. Toulouse, France, Online. http://hal.archives–ouvertes.fr/hal–00486997. (Cited on pages 35 and 36.)

[22] TOMLINSON, S. 2012. Human-robot interaction: Dominoes player. Online. http://mipal.net.au/video.php. (Cited on page 37.)

[23] WILLOWGARAGE. 2012a. Ros - nao common stack. Online. http://www.ros.org/wiki/nao_common Accessed 1.10.2012. (Cited on pages 40 and 49.)

[24] WILLOWGARAGE. 2012b. Ros - robot operating system. Online. http://www.ros.org/wiki/ Accessed 26.9.2012. (Cited on pages 34, 40, 55, and 76.)