Proceedings of the Austrian Robotics Workshop 2014
22-23 May, 2014
Linz, Austria **Regular**

Playing Nine Men's Morris with the Humanoid Robot Nao

Sven Bock, Roland Klöbl, Thomas Hackl, Oswin Aichholzer and Gerald Steinbauer

***Abstract*— Playing games is an important aspect in human life in order to develop skills or in terms of entertainment. Games also play a major role in research such as Artificial Intelligence and Robotics. In this paper we present an approach to enable the humanoid robot Nao to play a board game against a human opponent. We discuss the challenges that arise by the task of playing a board game with a humanoid robot, provide solutions for the Nao, and, introduce our proof-of-concept implementation for the board game *Nine Men's Morris*. Finally, we will present a first experimental evaluation of the approach. The main contribution of this paper is the integration of various techniques into one real robot system, enabling it to manage a complex task such as playing a board game.**

I. Introduction

Games play an important role ever since the beginning of mankind. Apart from their obvious purpose of entertainment games usually touch other interesting aspects like exploring strategies and comparing skills with each other. Games have already been introduced within education in order to ease learning processes or developing certain skills. Playing games also plays an important role in Artificial Intelligence (AI) and Robotics. When it comes to Artificial Intelligence the major task concerning playing games is to find efficient algorithms to solve a game, whereas in the field of Robotics usually the physical act of playing and the interaction with the environment is the major issue.

A main goal of our research is to fruitfully combine diverse fields such as game theory, advanced search approaches, computer vision and mobile manipulation. In earlier work [1] we already developed an artificial robot opponent that successfully played the board game *Pylos* against humans. In the current paper we present an implementation for the humanoid robot Nao in order to allow a more appealing and more general game play between humans and robots. The approach presented in this paper is able to master the challenges of playing a board game with the Nao like 3D perception and mobile navigation for a currently simplified setup. In the paper we present a prototype implementation for the game *Nine Men's Morris*. This first proof-of-concept implementation integrates various techniques into a real robot system in order to manage playing a board game with a humanoid robot. Such an integration is hardly seen and is the key contribution of this paper.

II. Game Setup

Nine Men's Morris is a two player strategy board game. In Europe it is very popular under the name Mill(s). As a robot we use a standard Aldebaran Nao with actuated fingers.

The authors are with the Institute for Software Technology, Graz University of Technology, Graz, Austria. {thackl,oaich,steinbauer}@ist.tugraz.at

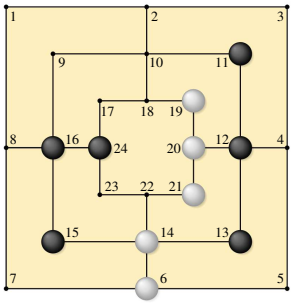


Fig. 1: Game Board and Situation from Nine Men's Morris. The numbers represent the abstract strategic positions.

In a nutshell, both players own nine tokens which are placed on 24 spots on the board. Fig. 1 shows the board and a typical game situation. The tokens of the two players are distinguished by their color. Tokens may only be placed at corners and intersections on the board. There is at most one token per spot allowed. The game has 3 phases, placing the tokens, moving the tokens to neighboring spots on the lines, and jumping (i.e., moving a token to any free spot). If a player gets three tokens of his/her color in a straight row, he/she can remove one token from the opponent. The objective of the game is to leave the opponent with fewer than three tokens, or in a position where no more legal moves are possible. Especially the three different phases of the game require quite different strategies and make it a challenge to determine the best move.

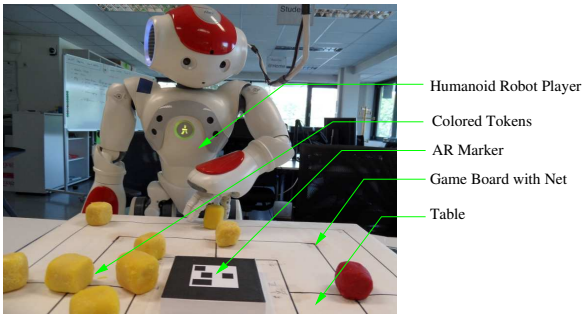


Fig. 2: Game Setup from Nine Men's Morris.

In order to play the game with the Nao, we build up the physical game setup shown in Fig. 2. The setup is a compromise between limitations imposed by the robot, e.g. range of the arms, and an attractive setup for a human player, e.g. appearance and haptics. In order to allow a maximum reach of the arms the game board with a size of 34 cm

× 34 cm is placed on a table with a height of 22.7 cm. An elevated AR marker is placed in the center of the board to compensate the accuracy for the missing gyroscope in the horizontal plane. In order to allow a reliable grasping, the tokens are cubes up to a side length of 3 cm, made of modelling clay. Moreover, the tokens are colored clearly differently in red and yellow for a clear identification by the vision system, even in poor lighting conditions.

III. Game Playing

Fig. 3 shows an overview of the architecture we used to play a a board game with the Nao.

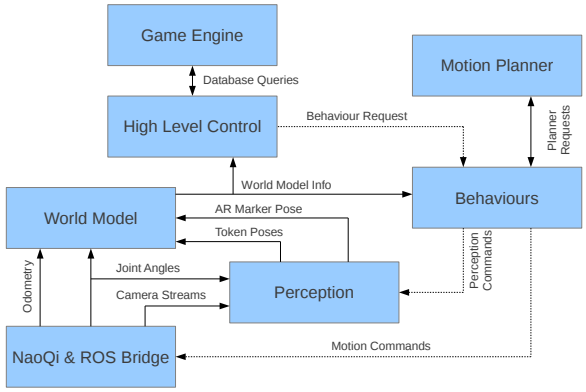


Fig. 3: System Overview. Solid connections depict flow of data. Dashed connections represents flow of commands.

In the following sections we will describe the individual modules in more detail. For the implementation we use the Robot Operating System (ROS) and NaoQi robot interface.

A. Perception

The main sensors to perceive the environment, besides the odometry, are the two cameras in the head. Unfortunately, stereo vision is not possible because the cameras do not have an overlapping region. The perception recognizes an AR marker and other features in the image that are used for localization. The result of the perception is an abstract discrete game state that consists of 24 strategic positions (see Fig. 1). Additionally there is an ideal and a real position saved for each token. Ideal positions correspond the strategic positions in torso coordinates (for placing) on the real game board, while real positions hold the information, where tokens are really located (for grapsing).

Marker and Coordinate Frames: Due to the close distance of the robot to the board only a part of it is visible at the time. This complicates the localization of the robot. As a result we use an AR marker of 70 mm side length in the center of the board. The transformation between the marker and the robot's camera is determined using the AR toolkit[1] and the ROS wrapper ar_pose[2]. The marker is 12 mm elevated from the board to reduce occlusion problems when tokens are close to the marker. The transformation is used to localize the robot and is fused into the world model together with other sensor information.

Visual Odometry: The robot Nao neither has a gyroscope for the upright z-axis nor does it provide reliable odometry due to slip during walking. In order to minimize the angular error we use visual odometry. This supports the estimation of rotations by calculating the horizontal angle difference of images. During the estimation, the z-axis of the torso is vertical to reduce the rotation to one angle. First a key point detector is applied to find interesting points in each image. Afterwards SURF descriptors [2] are computed of both images. This descriptor has been used because it combined the robustness and speed that is required to solve this task. Then matches between both images are computed using a "radiusMatch"-function that finds matches for descriptors in a given distance. The offset of the matches in pixel in the horizontal x-axis is used to determine the rotation of the robot.

Calibration and Blob Detection: The colors of the tokens are initialized at startup using the median of the colors at the area around the strategic positions 15 (first player's color) and 13 (second player's color). Using blob detection and morphological operations, a binary image of token candidates is generated.

Token Classification: The task of the classification is to validate the detected color blobs, to assign valid tokens to strategic positions and to compute the real 3d positions of the tokens. This is done by rectifying and rotating the binarized image to fit defined masks. These masks are polygonal areas around the ideal positions on the game board (see Fig. 4b). They are used to determine if a token was validly placed and to assign it to the corresponding strategic position. The four point correspondences, required for the rectification, are chosen around the focus point of the camera in the system of the game board. After the pixel positions of all points in the camera image are computed, the image can be rectified using a perspective transform according to [3].

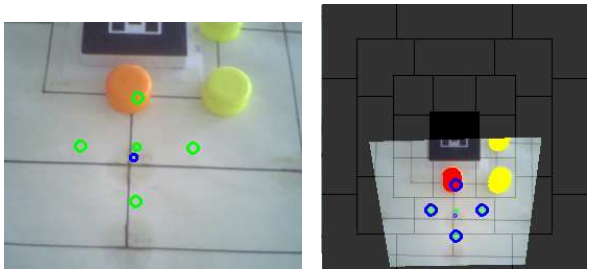
The result of the rectification procedure is visualized in Fig. 4b.

Conflicts on the board are detected if more than one token of any color lies in one mask. Fig. 5a depicts such a situation. Such conflicts occur, if tokens are placed carelessly or the robot fails to place a token correctly. In such cases there exists no clear mapping to the abstract game state. Therefore, the real 3d positions of tokens are saved and the high-level control is informed about conflicts on the board. A strategy to repair conflicts is described in Section III-D.

B. World Model

The world model stores sensor data, the board occupation, and token positions. Furthermore, it fuses different sensor data to achieve an improved localization and broadcasts all necessary transformations, e.g. torso to board. The transformation obtained from the AR marker detection is used as absolute localization of the torso w.r.t. the game board. If

[1] http://www.hitl.washington.edu/artoolkit/
[2] http://www.ros.org/wiki/ar_pose



(a) Camera image, where the focus point is surrounded by 4 rectangular points

(b) Rectified image with masks and detected tokens.

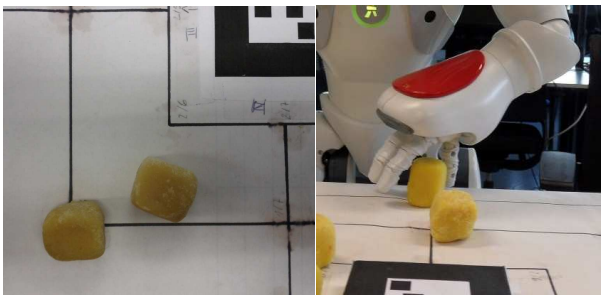
Fig. 4: Image rectification. The green points are the 4 point correspondences to the blue points to compute the projective transformation. The points are selected around the focus point of the camera a few mm above the board plane.

the marker is not detected, the robot's odometry and visual odometry are used to track the robot's position.

At each start of a game turn, the robot has to reconsider the complete board. It starts iterating through all strategic positions, updating multiple of them at once, where the majority of their mask is visible.

C. Mobile Manipulation

The task of placing and taking tokens can be seen as a mobile manipulation. Because of the limited range of the robot, we use the simplified setup presented in Section II. For the grasping of tokens we assume the pre-defined pre-grasp position shown in Figure 5b. It is defined by the x and y position of the token (taking a token) or the target position (putting a token) on the game board, a given height of 3 cm above the board, and a given pitch of the forearm of 25 °. These positions originate from the fact that the Nao has an rotational wrist and fingers with one DOF. We solve the grasping problem only for one board side and simply transform the results to the three remaining sides, assuming the robot always stands perpendicular to the board.



(a) Conflict situation.

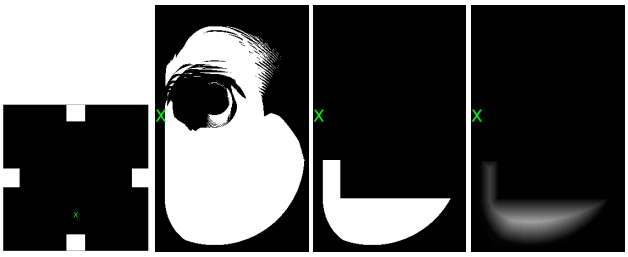
(b) Pre-grasp position for tokens.

Fig. 5: Token perception and manipulation.

The assumption that the robot stands always on an even surface and that both feet are always parallel allows us to decompose the complex manipulation planning into three simpler parts that can be handled separately. The three parts comprise of (1) selection of a suitable robot position and walking to it, (2) selection of a suitable torso posture, and (3) planning the arm movement to the pre-grasp position.

Dealing with the complete kinematic model of the Nao with 25 joints is very complex for the task. Using the above decomposition, the problem reduces itself to two simple kinematic problems. One problem is the inverse kinematics of the torso posture with only 5 joints assuming the mirroring of the joint angles between the two feet. The other problem is the inverse kinematics for the pre-grasp position starting in the shoulder joint featuring 5 joints as well.

The planning for the mobile manipulation comprises an off-line and an on-line stage. The off-line stage iterates over a grid with robot positions and a set of torso postures. It marks a grid cell with 1 if there is the possibility to grasp a token at a given position from the cell center and with 0 otherwise. The result of this calculation for the left arm depicted in Fig. 6b. We have to calculate and store the grid only once for the grasp position $\{x_{gp} = 0, y_{gp} = 0\}$ and one board side as we can later translate it to the actual grasping position and rotate it for the other board sides. Moreover, we simply mirror the resulting grid to get the results for both hands. A function $solvableIK(x,\ y,\ h,\ \phi,\ x_{pg},\ y_{pg})$ checks if the pre-grasp position $\{x_{pg},\ y_{pg}\}$ is reachable for a given robot position $\{x,\ y\}$ and a given torso posture $\{h,\ \phi\}$. The variables h and ϕ denote the height of the center of the torso and the pitch of the torso, respectively.



(a) Game board obstacle grid.

(b) Entire workspace for the left arm

(c) Combined workspace

(d) Distance transform

Fig. 6: The figures show the grid evolving steps. Invalid positions are colored in black. All figures are shown from an overhead perspective. Fig. 6a depicts the obstacle mask representing the table.The white areas are possible because the robot may bend its knees under the table. Fig. 6b shows the entire workspace of the robot. Fig. 6c depicts the overlay of workspace and obstacle mask. Fig. 6d shows the distance transform performed on the overlay. The white the pixel is shown in this grayscale image, the farther away are all invalid torso positions. The green cross depicts the desired pre-grasp position. The robot stays orthogonal to the table.

The on-line stage is shown in Algorithm 1. The algorithm uses a desired grasp position, the grasp position and obstacle grid, and the information which hand to use as input. It guides the robot to a position and posture that allows the

robot to grasp a token at the desired position. First the algorithm selects the board side that minimizes the requested walking. Then the grasp and obstacle grids are transformed to the requested grasp position and the selected board side. The intersection of these two grids provides robot positions from which the desired grasp can be reached without colliding with the table. An example for such a grid is shown in Fig. 6c. The target robot position is selected from a distance transformation of the combined grid. The distances reflect the smallest distance to a neighboring cell where the robot is not able to grasp the token. Fig. 6d depicts this transformation. Selecting the maximum yields a high possibility that the robot is still able to grasp even if the robot does not reach the position exactly due to shortcomings in the navigation.

Once the robot finished its path it checks if the robot is able to grasp the token from the actual position. If this is not the case, again a target position is selected and a path is calculated. Once the robot reached a position where it is able to reach the pre-grasp position it selects the torso posture $\{h, \phi\}$ that can reach the position and maximizes an objective function (line 17). Usually, we use a higher value for α than for β in order to prefer higher torso positions (better camera view on the board) over a mostly upright position (more stable posture).

Algorithm 1: reachPreGraspPosture

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 $\{h_r, \phi_r\} \leftarrow$
$argmax_{\{h,\phi|solvableIK(x_1,y_1,h,\phi,x_{gp},y_{gp})\}} \alpha \cdot h + \beta|\pi - \phi|$
18 moveTorso(h,ϕ)
19 moveArm(h,ϕ,x_{gp},y_{gp})

D. High Level Control

The high-level control is represented by the state machine shown in Figure 7. Transitions are triggered by events. The state machine invokes behaviors according to its actual state.

The state machine comprises the following states:

- **init** initially localizes the robot, calibrates the color, and asks the user for game information if a saved game is continued.
- **get game state** acquires the current game state.
- **get next move** requests the next move from the database, based on the current game state.
- **execute moves** executes the selected move.
- **wait for player** waits for the user finishing her move.
- **finished** either one of the players won or the user aborted the game by touching the robot's head.
- **help** an error occurred during execution. The robot waits for a human to resolve the problem.

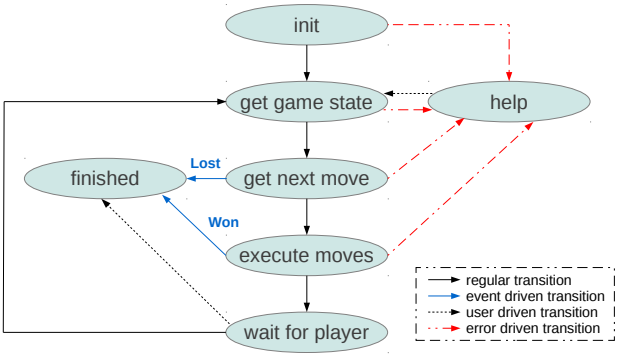


Fig. 7: Main state machine.

Achieving the different subtasks incorporates sending requests to the planners as well as communication with the game engine.

E. Behavior Execution

The framework provides a number of behaviors. Their execution is invoked by the high-level control.

Higher Level Movement provides methods to easily move to a given torso position, control the robot height, and relax the whole robot.

Look At is used to look at strategic positions and to find the marker. The marker is searched at the current position and at the position the marker was last seen. If the marker is still not found the robot begins to move its head in a circular pattern to find the marker.

Color Calibration initializes the player colors at the start of the game during the initialization phase. It uses head motion and vision services.

Analyze Field is a behavior that scans the complete board. It requests the positions to look at from the world model and moves the head to the most interesting position. It activates the vision to update the game state in the world model.

Place/Take/Move executes manipulations at a given token position. These manipulations include walking to the position, the place/take/move action, and a visual confirmation with a possible repair.

Error Recovery is applied in case the confirmation of a place/take/move action fails, i.e., the token is not at the desired position. First the entire game state is updated again. If the token is not located at the desired position after a place action we can distinguish two cases. If the token is still on the table but rolled to an already occupied position we receive a conflict. Such a conflict can be resolved by moving the actual token to its desired position. If we receive no conflict

there are two possible reasons. Either the token rolled to an unoccupied position or the token rolled off the table. Both cases can be detected by comparing the actual game state with the previous one. For the former case we recognize an additional token of the given color and simply move it to the desired position. For the latter case the robot asks for human assistance as it is not able to resolve this issue by itself.

F. Game Engine

The idea behind the game engine is to pre-process all optimal moves off-line, so that during a game play of the Nao only marginal computer resources are needed for generating the move itself. This is important since performing the interaction with the environment requires most of the resources of the Nao.

To solve the game play we follow an approach that uses an idea which is called *dynamic programming* in algorithm theory. There, every game position which might show up in a game is evaluated only once and stored together with its evaluation value. This usually saves the exponential overhead in an game tree approach. Of course this approach comes at a price: a lot of memory or disk space is needed to store the positions and their evaluation, the so-called *state-space*. Thus, it is still not feasible for games like chess or go. But fortunately Nine Men's Morris has a rather huge game tree complexity, but a limited state-space complexity, which makes dynamic programming applicable.

For Nine Men's Morris we generated all possible positions which can show up during any game, of course taking symmetry, rotation, and reflection (inner and outer cycles can be exchanged) into account. This results in a data base of over 19 billion positions, needing approx. 19.6 GB of disk space. Using the dynamic programming approach described above we evaluated all positions, and stored whether it is a first player win, a second player win, or a draw. In case one of the players can force a win, we also stored in how many half moves she can guarantee that win.

The data base provides a simple interface for the other components of the playing Nao. A query can be sent for a position which is currently considered. The database can check whether it is a valid position and, if true, provides possible optimal moves with additional information. In this way it is also easy to check whether a board which is recognized only with a certain probability could in fact be the result of a legal move of the opponent. Thus the actual board can even be determined if the vision of the Nao delivers several different candidates — a kind of consistency check.

IV. Experimental Results

We implemented a prototype version of the playing Nao using ROS and NaoQi. Because of the size of the game database and the computational demands of the perception we run these parts on an external PC and communicate to the robot over Ethernet. The following tables show the experimental results of performing small subtasks which are essential for playing Nine Men's Morris. Table I focuses on the placement of tokens. This task is considered to be the easiest one. However, in two cases the token was placed too far away from its ideal position such that it was classified as not correct. In one of these cases the repair strategy was able to resolve the initial error. In all the other cases the robot was able to successfully place the tokens on the correct positions in one try.

		Place & Repair		$\sum$	%
		Success	Fail		
Place	Success	28	0	28	93.33
	Fail	1	1	2	6.67
$\sum$		29	1	30	
%		96.67	3.33		

TABLE I: Test: Token placement. In one case an initial placement failure could be repaired using the repair strategy.

Table II illustrates the results concerning the grasping actions. This task also requires a correct classification of the token and self-localization. The robot is capable of grasping a token in most cases. However, the grasp fails in some of the cases at the first try. This happens, as the robot is not always able to perform a reliable grasp because of the limited hand. Although, following the repair, the success rate can be significantly improved.

		Recognition		$\sum$	%
		Success	Fail		
Grasping	Success	18	2	20	66.67
	Fail	10	0	10	33.33
$\sum$		28	2	30	
%		93.33	6.67		

TABLE II: Test: Token grasping. In two cases an initial recognition failure could be repaired using the repair strategy.

Table III shows the results for the classification of single tokens and recognizing the entire game state. The table features game scenarios with a low number of tokens as well as scenarios with 9 tokens per player (see Fig. 8). It has to be mentioned that these results were gained by just scanning the game board scene without having any knowledge of previous game states. Classification errors mainly occurred on the opposite side (w.r.t. the robot) of the game board.

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 III: Test: Get game state.

Table IV shows the results of walking to a different side of the board while avoiding collisions. Due to the lack of sensors providing on-line data while walking, the success rate of this task highly depends on the robots accuracy while executing motion commands. A re-localization while walking parallel to the board is not possible, as the shoulder of the robot is blocking the view to the board. Furthermore the marker is very small and difficult to detect under perspective view. Without the marker the robot rarely found back to the

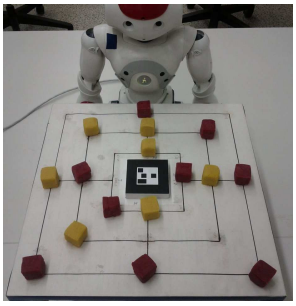


Fig. 8: Example game situation used in the evaluation.

board. Therefore we detect now the edges of the board to give the robot a course estimation where it is. This estimation allows us to approach to the board until, the robot finds the marker again. This additional procedure greatly increased the robustness of the walking procedure. In case the marker was never lost, the robot approaches to the board in one complete move which results in slight collisions due to the inaccuracy of the odometry.

		Walk			$\sum$	%
		Success	Slight collisions	Fail		
Marker	Found	10	4	0	14	60.87
	Lost	9	0	0	9	39.13
$\sum$		19	4	0	23	
%		82.61	17.39	0		

TABLE IV: Test: Walk around the board.

V. Related Research

There are several other approaches for playing games with humanoid robots. For instance the work of [4] presents a humanoid robot that is able to play ping pong against a human opponent. Their greatest challenges are the fast accelerations of the arm and the balance of the robot.

A work that fits better to our task is developed by a french company named HumaRobotics [5] who presented an autonomously *connect-4* playing Nao. Their result is impressive because they need no marker, external PC and recognize when a human player finished its turn to play their game. However, they can see the whole gameboard which is orientated upright. This results in minimal projective distortion, the board can be scanned continuously to recognize human action and the board serves as a landmark with known size and background. Additionally they do not grasp tokens and they can observe the token while throwing it into the board.

In 2011 [6] proposed an approach of the Nao grasping tokens on a board. This approach is similar to our work because they face the same problems as we do, like limited processing power, inaccurate repositioning, reduced stiffness on continuous operation, limited grasping ability, and self occlusion of the target position. They use visual servoing to move the thumb, which is essential for a successful grasp to a target position above the object. They project the thumb onto the table surface to estimate the hand position.

In [7] an approach of a real-time SLAM with a single camera computed on a desktop PC is proposed. They focused on natural long term features and a motion model to reduce motion drift. They inspired our work for projections from the image in the 3D scene and the visual odometry.

Recently in [8] the authors presented an integrated approach for the Nao that allows it to grasp general objects like a cup. The approach combines object recognition based on stereo vision, a grasp quality estimation, and an A*-based motion planner. Although, the performance of this approach is impressive it has the drawback that a special version of the Nao with an integrated stereo setup is used.

VI. Conclusion and Future Work

In this paper we present an approach for playing board games with the humanoid robot Nao. A proof-of-concept implementation addresses all challenges arising from playing a game with a humanoid robot, such as perceiving the game board, deciding the next winning move, and mobile manipulation of the tokens. The approach is a combination of a strong game engine and a robotics framework. The main components of the architecture are a competitive game engine, a reliable 3d scene recognition, and a simplified manipulation approach. The major contribution of this paper is the integration of various techniques in one real system to allow it to solve a complex task. First experiments show that the robot is able to autonomously play the game quite stable. Moreover, first empirical figures are given for the performance of the needed individual capabilities.

In future work we will aim at a native implementation that runs completely on the robot. Furthermore, the individual skills have to be improved. Mainly we think of a probabilistic recognition approach for the tokens and the integration of more advanced planning techniques to master dynamic environments as well. Finally, we will extend our approach to other games like chess. Especially this is a real challenge because the tokens are not homogeneous anymore.

References

[1] Oswin Aichholzer, Daniel Detassis, Thomas Hackl, Gerald Steinbauer, and Johannes Thonhauser. Playing pylos with an autonomous robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2507–2508, 2010.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008.

[3] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. pages 33–35. Cambridge University Press, 2003.

[4] Yichao Sun, Rong Xiong, Qiuguo Zhu, Jun Wu, and Jian Chu. Balance motion generation for a humanoid robot playing table tennis. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 19 –25, 2011.

[5] HumaRobotics. Nao plays... connect 4, 2013. Online: http://www.generationrobots.com; accessed June 26 2013.

[6] Thomas Höll and Axel Pinz. Vision-based grasping of objects from a table using the humanoid robot nao. Austrian Robotics Workshop, 2011.

[7] Andrew J. Davison. Real-Time Simultaneous Localization and Mapping with a Single Camera. In *Proceedings. Ninth IEEE International Conference on Computer Vision*, pages 1403 – 1410 vol.2. IEEE, 2003.

[8] Judith Müller and Udo Frese and Thomas Röfer. Grab a Mug - Object Detection and Grasp Motion Planning with the Nao Robot. In *IEEE-RAS International Conference on Humanoid Robots*, 2012.