EnternatiNAO : Poker Master

Selin Kesler
Technical University of Munich
selin.kesler@tum.de

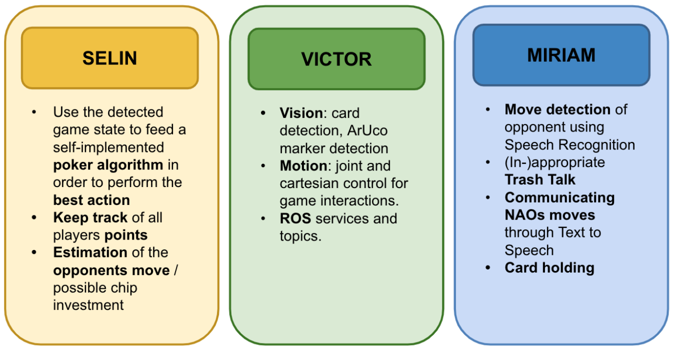
Victor Kowalski
Technical University of Munich
victor.kowalski@tum.de

Miriam Senne
Technical University of Munich
miriam.senne@tum.de

I. Introduction

The main goal of this project is programming a NAO robot, from Aldebaran robotics, who can play poker against an opponent complete autonomously. This means that it can make up its own mind about the next move while estimating the opponents next moves, it can detect and understand the cards of its own and the cards in the middle, it can raise money by pushing chips in the middle or making moves representing ’check’ and ’fold’. NAO robot is also in complete interaction with its opponent, which includes both understanding the ’game language’ such as raising money, check and also trash talking to demotivate its opponent to help with its victory.

For better understanding of the project, it is important to know the rules of the poker game and sufficient level of knowledge is assumed from this point.



SELIN

- Use the detected game state to feed a self-implemented **poker algorithm** in order to perform the **best action**
- **Keep track** of all players **points**
- **Estimation** of the **opponents move** / possible chip investment

VICTOR

- **Vision**: card detection, ArUco marker detection
- **Motion**: joint and cartesian control for game interactions.
- **ROS** services and topics.

MIRIAM

- **Move detection** of opponent using Speech Recognition
- (In-)appropriate **Trash Talk**
- **Communicating NAOs moves** through Text to Speech
- **Card holding**

Fig. 1: Task Distribution in the Team

The task distribution in the team is represented in the Figure 1. These tasks will be examined and discussed further in the following sections.

II. Game Strategy

A. Algorithm Behind The Poker Game

For an intelligent poker game playing scenario, an approach with Monte Carlo Tree Search (MCTS) algorithm has been followed. With this approach, the necessity for hard coding each possible scenario during the game is removed and adjustability for all circumstances was ensured. The principle of MCTS is introduced briefly before the detailed description of the method.

The MCTS is a powerful decision-making algorithm which made its breakthrough by mastering one of the most complicated games, Go [1], [2]. It is an iterative algorithm that searches the available decisions of the current state and updates its nodes with respect to the statistical evidence [3].

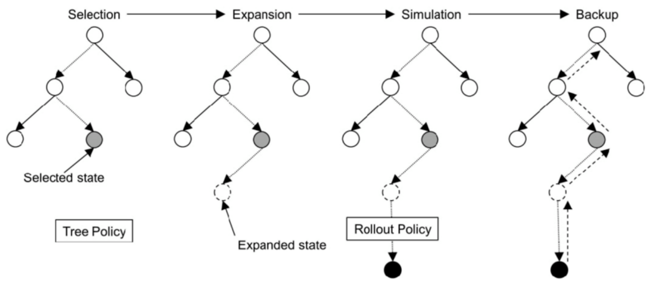


Fig. 2: MCTS four Steps [4]

Each MCTS iteration has four stages:

- Selection : The already identified tree is being searched by the algorithm starting from the root node. It iterates through the nodes of the tree via *selection policy* and selects the node if a leaf node was found or a terminal state is reached.
- Expansion : If the selection stage results with a leaf node and not with the terminal state, expansion adds a new child node to the current state which is reached by performing an allowed action in the leaf node.
- Simulation : random simulations in the given limitations are performed until a terminal state is reached as won or lost the game.
- Backpropagation : backpropagates the reward from simulations starting from the child node to the root by visiting all the nodes on its way.

Same principles of MCTS is used for programming of the poker game. Tic-Tac-Toe algorithm with MCTS approach[1] is used as basis for the development of the code. In the beginning of the game ’Poker Board’ is initialized which includes the parameters in the Table I. Poker Board is updated after each move, both in real game and in simulations, so that the current game state and the money of the players are always tracked.

Each player may enter the game with a money of multiples of five. For simplicity reasons due to the rollout performance, the first player is set fixed to the opponent, who is forced to raise money (as in small blind). In counteract, NAO should raise the same amount of money to enter the game. After both players raised money, three cards will be opened in the middle

[1]https://gist.github.com/qpwo/c538c6f73727e254fdc7fab81024f6e1

and opponent can choose between the actions 'check', 'fold' and 'raise', where raise can only occur in multiples of five.

Poker Board	Type	Values
Tup	Str * 9	('S9', 'H5', ..., 'D7', 'D13', None, None)
Turn	Bool	True (Machine) /False (Opponent)
Winner	Bool	None/True (Machine) /False (Opponent)
Terminal	Bool	True/False
Money Machine	Int	Money left in machine bank
Money Middle	Int	Money raised in total in the middle
Money Opp	Int	Money left in opponent bank
Raised Opp	Bool	True/False
Checked Opp	Bool	True/False
Raised Ma	Bool	True/False
Checked Ma	Bool	True/False
Raised Money Opp	Int	Money opponent raised in total
Raised Money Ma	Int	Money machine raised in total
Folded	Str	None/opp (opp folded) /ma (ma folded)

TABLE I: Poker Board Parameters

The decision of the opponent is then applied to the Poker Board, which is an observation space for the NAO as its decision is based on the moves of the opponent. This state of the board is fed to the MCTS with desired number of rollouts. MCTS first searches for all the allowed actions for the given state and explores these allowed actions with the pre-defined exploration rate. After the first layer is finished exploring, children nodes of the first layer will be explored. These explorations goes on until a terminal state is reached, where a player has won or lost. In this case NAO gets a reward of + money in the middle if won or - money in the middle if lost. The values of the nodes will be updated during backpropagation step, which will have an effect on the next choosing processes of children nodes during simulations.

The game reaches its terminal state if one of the following scenarios takes place:

1) Either one of the players fold at any time step, in which case the cards in the middle are not important for the winner calculation.
2) One player is all in and the other player raises as much as the all-in-player. All the five cards in the middle will be opened if not opened yet and the best hand combinations will be compared to declare winner.
3) All cards in the middle are opened, both players checked or raised same amount of the money. Best hand combinations (5 cards out of 7 cards in total) will be calculated and the scores of these hands will be compared to declare the winner. [2] [3]

Since the the hand of the opponent is unknown during simulations, possible hands of opponent will be created in the terminal state for best hand combination calculations, so that effect of randomness in the opponent hand is taken into consideration during reward calculation.

For a better understanding of the game process, the Figure 3 represents the steps in the game as a tree structure, which also builds the foundation of the MCTS rollout steps. The nodes are color coded for players and terminal state, where each node has its own PokerBoard at each time, which is updated after each move. The inclined arrows represents other paths that could have been followed in the real game or in the simulations.

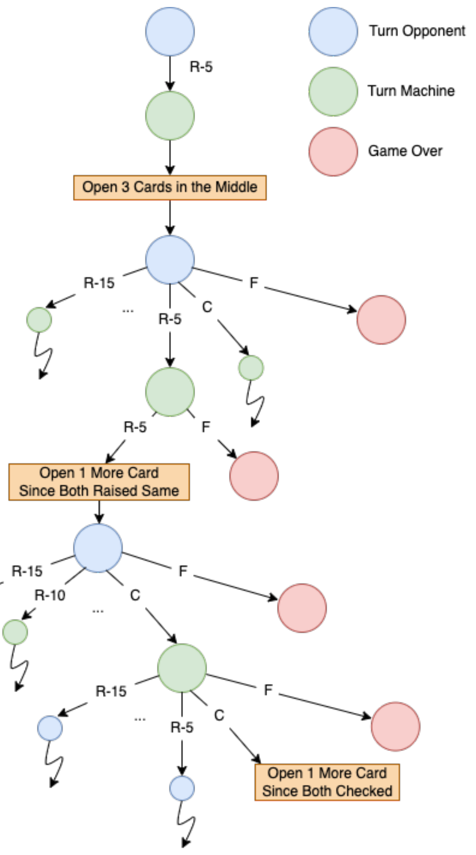

Turn Opponent
Turn Machine
Game Over
R-5
Open 3 Cards in the Middle
R-15
...
R-5
C
F
R-5
F
Open 1 More Card Since Both Raised Same
R-15
R-10
...
C
F
R-15
...
R-5
C
F
Open 1 More Card Since Both Checked

Fig. 3: Poker Logic

It is important to notice that the movements of the players are restricted by the movements of the other player. Another crucial feature is hidden under 'Open 1 More Card' state. For real game flow, it means that one more card should be opened in the middle by the dealer. Distinctively, for simulation part it means that all the possible cards that might be opened in the middle should be calculated by removing the cards in the middle and cards of NAO from virtual deck. All of the possibilities for the new middle card should be considered as child nodes.

B. *Estimation of Opponents Move*

It was desired that the NAO was able to predict the possible movement of its opponent. Although NAO already takes numerous combinations of opponent movements into

[2]https://gist.github.com/dirusali/e1184ddec664a96bdcfe36377155c19bfile-hand$_{t}ypes - py$

[3]https://gist.github.com/dirusali/bbd2039c7ce3b3a904606c0d60bb6a3bfile-score$_{h}ands - py$

consideration, a specific method has been implemented, which has the function of predicting the opponents move from opponents perspective. A copy of the current PokerBoard will be created in which the turn will be set to the next player. The aim of this was using the same simulation functions of NAO for the opponent with feeding the information of opponent as 'own player'. Since the cards of opponent is not known to the NAO, they should be simulated at the desired rate. In order to not to block the game with these simulations, ten different card combinations for the opponent has been decided. Each of these ten combinations go through the rollouts where each children node is saved with their scores from simulations.

Opponent Levels	Action Choice from Rollouts
Professional	Best action
intermediate	Random between best and second best
Beginner	Random between second and third best

TABLE II: Opponent Level Declaration

For NAO, best child from rollouts is chosen after the rollouts, which might differ in opponent move estimation case if desired. Three levels are build for possible action choice; professional, intermediate and beginner. The action choice process is represented in the Table II.

III. Communication Skills

While having a good hand in poker is an obvious way to win a game, it also involves a lot of luck. But luck alone is not sufficient. The goal in poker is to win as many chips as possible, but if one only raises chips when having a good hand, every other player knows to fold. Therefore, it is important to not reveal this information. This is achieved by bluffing. Bluffing is pretending to have a good hand although this is not the case. Even with a very bad hand, a game can be won if the bluffing game is so good that all the other players fold. That is why it is also important for NAO to master this kind of skill.

The following sections explain how NAO is able to understand the opponent's moves, how it decides to use trash talk to intimidate and confuse the other player, and will also provide an overview of NAO's general communication.

A. *Understanding Moves*

For detecting the opponent's move, the ALProxy SpeechRecognition API is used. It requires to create a vocabulary that includes all words or sentences, that the robot is supposed to understand. As can be seen in Figure 4, this list includes all possible moves, which are *check* (C), *fold* (F), *all in* (AI) and *raise* (R). When the opponent raises, NAO listens again to identify the number of chips that are being raised, as his move is strongly dependent on it. Apart from the moves, the opponent himself can respond with trash talk to intimidate NAO. The vocabulary chosen for this has been divided into two sub-lists, depending on the mood of the opponent. Since it is also possible to pretend to have bad cards even though they are good, the trash talk can come from one of these two lists: [”you will lose”, ”you suck”] or [”I am so bad”, ”I am losing”].

As soon as NAO receives the information that it is the opponent's turn, he starts listening. If the recognized word or phrase falls below a certain threshold, which is set at 0.4, he listens again and ignores the current input. If he recognizes a move with a high confidence, he passes this information on to the poker algorithm to compute his next move. This process is illustrated in Figure 4 as a flowchart, where red indicates actions of NAO, blue of the opponent and green are part of his vocabulary, as previously discussed.

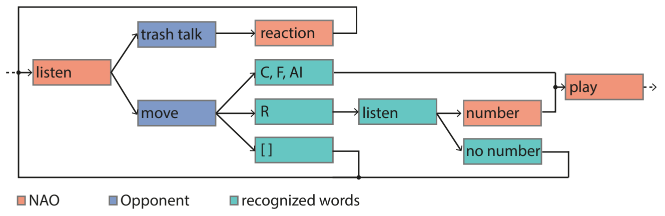


Fig. 4: Flowchart of how NAO listens and chooses to react to the opponent. The words in green indicate the possible moves of the other player and stand for check (C), fold (F), all in (AI) and raise (R). Red are actions of NAO and blue are possible linguistic inputs of the opponent.

B. *Trash Talk and Bluffing*

To give NAO the ability to bluff and do trash talk, it is important to let it seem as natural as possible. To do so, he has to react appropriately. For this task, a dictionary was created which is built as illustrated in Table III. The dictionary keys represent scenarios that initiate reactions from NAO and are therefore referred to as *triggers* in the following. The items of the dictionary consist of a list of strings, where each string represents an appropriate response to the situation. At least one of the implemented phrases is listed in the table. At various points in the poker game, different triggers are called and one of the associated phrases is randomly selected and then propagated to a function that uses the ALProxy TextToSpeech API, to enable NAO to say it out loud. To prevent repetition, sentences are removed from the dictionary as soon as they are used. If the match goes on for a long time and a list empties, NAO will no longer say anything in the event.

The *beginning* trigger is called at the start of each game. NAO randomly selects a phrase and intimidates the opponent before they even know whether they have a good hand or not. By selecting an element of *me all in*, he lets the opponent know that he chooses to raise all his remaining chips. *reaction mean* and *reaction sad* are triggered by trash talk of the opponent, when listening, as mentioned in Section III-A. At the end of each game, the robot calculates the winner and chooses an appropriate comment by selecting either an element of *won* or *lost*. Knowing the winner of the last game, NAO can choose a new opening phrase for the game, depending on the last outcome by choosing the corresponding trigger

Key	Item
beginning	”Look at these cards! I’ll show you my hand and still win.”, ...
me all in	”All in baby!”, ”All in.”, ...
reaction mean	”Bring it, don’t sing it.”, ...
reaction sad	”Get better, not bitter.”, ...
won	”Do you need a tissue?”, ...
lost	”Oh man, I thought we were playing blackjack.”, ...
begin second game + won last	”You seem to really want to get your ass kicked again.”, ...
begin second game + lost last	”Last time was only luck. This time will be different.”, ...
random	”Your mom called. She says you left your game at home.”, ...

TABLE III: Trash Talk Dictionary Setup

begin second game + won last or *begin second game + lost last*. So far, it has taken events to trigger bluffing or trash talk from nao, but human players do not need an occasion for inappropriate comments. Therefore, the robot should also possess this property and randomly use trash talk during its turn. The phrases used for this are stored as elements of the trigger *random* and are triggered with a probability of 50%. This gives NAO the appearance of being conscious and skillfully trying to manipulate its opponent.

C. Communicating Moves

As previously mentioned, the ALProxy TextToSpeech API is used to let NAO talk. Besides the already mentioned talking events, he should also be able to communicate his moves. Whenever he has calculated his best next move, he tells his opponent about it before he makes the move. When he has made his turn, he lets the opponent know that he can make his move by saying: ”Your turn” and then starts listening. If he didn’t recognize an action correctly, he asks the other player ”Can you say that again?” and listens once more. If the game state requires players to show their cards, NAO communicates that as well. He then announces each player’s score and the resulting winner. When the game is over, he politely asks for another round.

IV. Setup Card Holding

NAO’s ability to hold cards is severely limited as he only has three fingers per hand, which he can only open and close. Therefore, this section discusses three approaches for the card holding and presents the final design.

A. Regular Cards

The first, optimal and probably most obvious approach is to place regular sized cars directly in NAOs hands. Surprisingly, he was able to hold the required two cards correctly. The problem was to find a solution to recognise the class of each card. For several reasons, this approach required a deep learning algorithm for card recognition. For solutions whose implementation would have remained within the scope of this work, Python 3 was required, which was not possible to match with the setup provided.

B. Card Holder

The second approach is a 3D printed card holder, which NAO is able to hold in his hand. An illustration of it can be seen in Figure 6. The cards are attached in the centre of the shield, which is coloured black to maximize the contrast between the cards and the holder. The reason why this is important arises from the classification algorithm which is discussed in Section V-B. Although the results were promising, this approach could not be used definitively because the print is too heavy for NAO. He is able to hold it, but in certain important movements he lost the holder. Due to time constraints, the print could not be adjusted and run again.

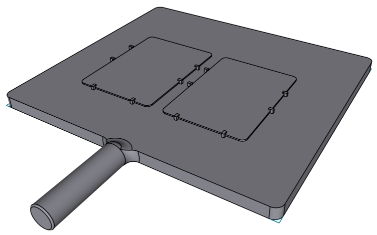


Fig. 5: 3D Model of Card Holder

C. Black Background

To retain the black background, smaller poker cards are used, which are attached to a larger black paper. This is done twice, for both playing cards. With this setup, the robot can hold the cards in its hand as in the first approach and the black background of the second approach, which is so important for the recognition algorithm, is preserved. The setup can be seen in the following Figure

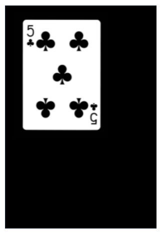


Fig. 6: Final Setup for the Card Holding

V. Vision

The vision submodule was responsible for taking care of the required detection tasks of the project. Those consisted in detecting the playing cards as well as an ArUco marker used for the calibration of chips’ positions.

A. Cameras setup

In addition to both cameras from the robot, it turned out to be necessary to set up an external camera pointing at the game table. Each camera's role was defined as follows:

1) **NAO's top camera:** detection of cards on the robot's hand.
2) **NAO's bottom camera:** detection of the ArUco marker.
3) **External camera:** detection of cards on the table.

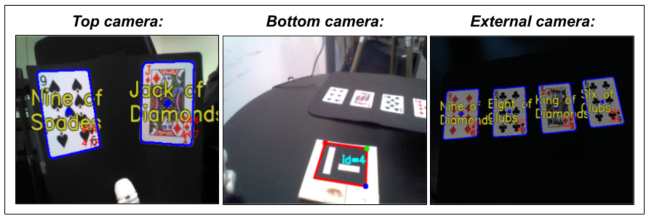


Fig. 7: Camera views

B. Card detection

The chosen card detection algorithm was based on the project hosted on [5], which is to a great extent similar to the more detailed work present on [6].

The algorithm can be divided in two parts, namely the generation of templates of ranks and suits for comparison with an arbitrary card to be detected and the detection (matching) per se.

1) **Template generation:** for each of the 13 ranks and then each of the 4 suits, the following set of steps was conducted:
   a) Get a capture from the top camera.
   b) Apply the first blur and threshold operation, for getting contours present on the image and identifying which of them is a card.
   c) Flatten the image section containing the card.
   d) Crop upper left corner showing the cards rank and suit.
   e) Apply the second blur and threshold operation, for getting contours of rank and suit.
   f) Crop either the rank or the suit, depending on the current iteration of the procedure.
   g) Apply bounding rectangle to get a rimless image.
2) **Matching:** for each frame coming from the video stream:
   a) Go through same preprocessing steps a) to g) from template generation to first identify the presence of cards and then get their ranks and suits ready for comparison with the templates.
   b) Given the rank and suit of one or more query cards (i.e. the ones to be detected) found on the image, compare them with all the templates of ranks and suits (by doing a simple subtraction of binary pixels) and find the ones that differ the least.

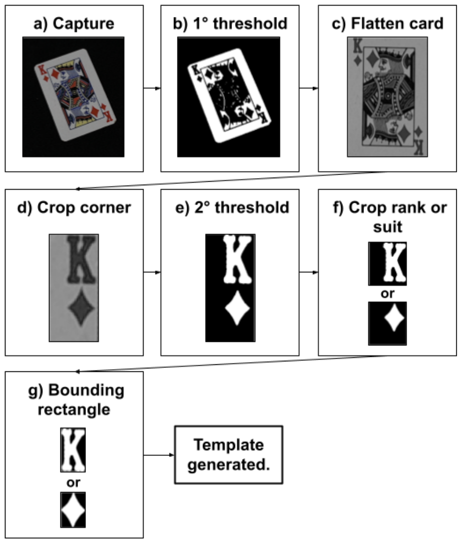


Fig. 8: Template generation

   c) Return the results both visually in the original image and also through the correspondent ROS [9] topics for communication with further modules.

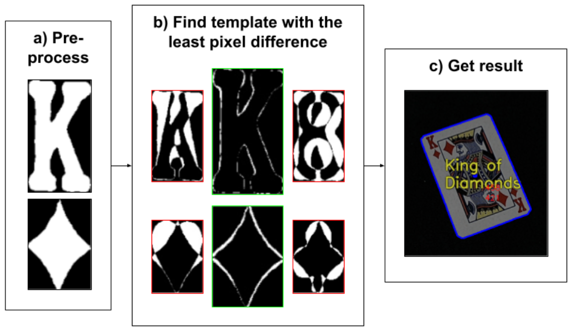


Fig. 9: Matching

C. ArUco marker detection

The function of the ArUco marker [7] [8] was setting a 3D reference point for the positioning of the robot's chips.

For that purpose, the marker was detected by the bottom camera, and its 3D position was obtained. This position was originally in the robot's bottom camera optical frame, then transformed to the bottom camera frame through the rotation matrix built up from observation on *RViz* and lastly to the robot's torso frame through the homogeneous transformation matrix provided by NAOqi's *ALMotion API*

With the marker position in terms of the torso frame, the robot's chips would be positioned relative to it in order to allow a more robust execution of raise movements, as further described in section VI-B.

VI. Motion

The motion submodule was responsible for executing the game interactions using NAO's end effectors, through NAOqi's *ALMotion API*.

It can be split into two parts, namely joint and cartesian control. The main criteria for assigning each movement one type of control was the necessity of flexibility and adaptation to different initial states, as instanced next.

A. Joint control

This first type of control was used for the movements classified as fixed or constant for any game setup, viz.:

- Draw cards
- Check
- Fold
- Show cards
- Rest arms
- Nod

and so on. Three of those are depicted in Figure 10.

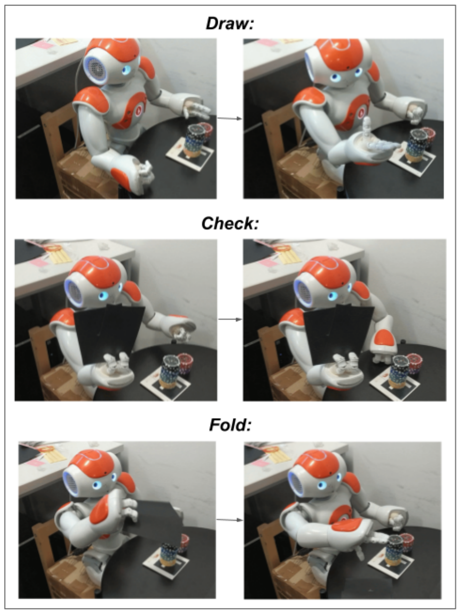


Fig. 10: Joint control

B. Cartesian control

Cartesian control, on the other hand, was used for changeable movements. More specifically, the raise movements depended on the positions of the chips, and these again on the position of the marker. Therefore, a desired position for the left hand was set with respect to the previously recorded position of the marker. An example can be seen in Figure 11.

Hereafter, some remarks are made about the logic underlying the chip positioning.

1) Chip Matrix: A 4×3 matrix C keeps track of the presence of chip blocks in each position. The block right above the marker is represented by C_{11}. The rows (i) represent the height and the columns (j) the stack.

Given the torso frame (origin on robot's chest, x axis pointing forwards, y axis pointing to the left and z coordinate pointing upwards), the recorded marker position can be represented in it as:

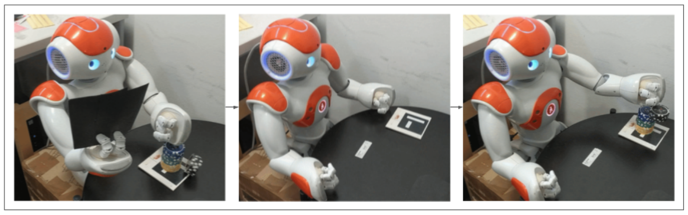


Fig. 11: Cartesian control

$$[x_m, y_m, z_m]^T$$

To make the arm reach a goal block, the correspondent vector would simply write as:

$$[x_m, y_m + a \cdot j, z_m + b \cdot i]^T$$

Where a and b are constants chosen with respect to the stack spacing and the height of one block, respectively.

Besides, after a raise movement, the pushed chips entries are updated to zero in the matrix.

2) Least Movements: When a raise move is performed, it is done in such way that requires the minimum amount of movements. E.g. push three chip blocks at once instead of pushing one chip three times for raising fifteen.

3) Random Choice: There normally are multiple ways to raise a certain amount, which are all taken into account by the robot, that chooses one at will.

4) Filling: When the robot wins a round, the received chips are stacked from the pile closest to the marker to the furthest one. Thus the positions are known and only the amount won is necessary for filling the chip matrix with ones.

VII. System architecture

To sum up and give a bird's-eye view of the overall structure, the system's architecture is presented in Figure 12. It gives an overview of inputs and outputs of the system and the ROS processes involved.

Furthermore, the presented architecture can be interpreted as of two distinct types, related to each output module.

Regarding the movement, the system can be seem as a *deliberative* one, in the sense that it plans, thinks then acts. The first step is devising the game strategy, considering the drawn cards, the cards on the table and estimating the opponent moves. Subsequently, an action is chosen, which is expressed by a physical movement.

The speech generation module, in contrast, relates to a *reactive* architecture. The approach here is simply reacting to events such as speech input from the opponent.

VIII. Issues faced

Some noteworthy issues faced during the development of the project are quickly mentioned below, along with the solutions found:

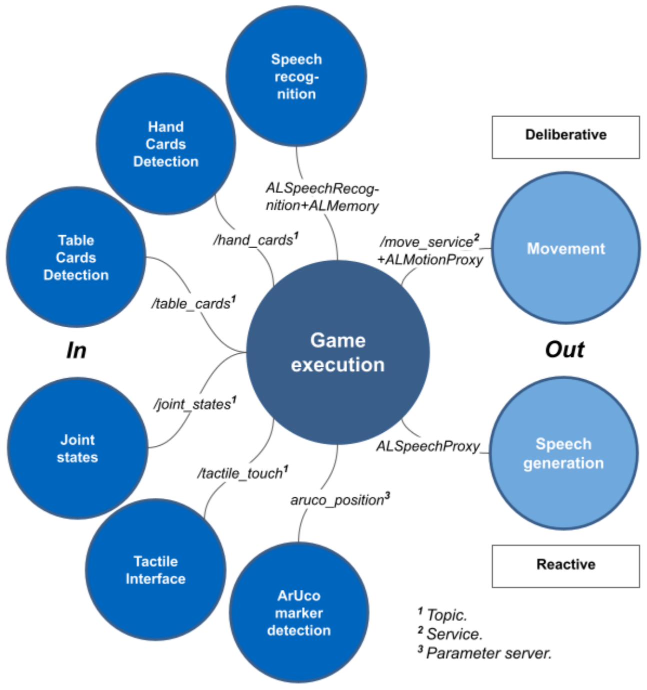


Fig. 12: System Architecture

- Dependence of appropriate light conditions for cards recognition: manually setting the exposure and other camera parameters.
- Low quality of NAO's cameras: using an external camera for cards standing further away (on the table).
- Inaccurate movement of the end effectors: taping together blocks of five chips.
- Short range of the arms: restricting the chip positions to a few stacks close to the robot.
- Overheating of the joints: programmatically disabling stiffness on exit of motion script.
- Limited grasping capability: pushing chips instead of grasping, black paper background instead of 3D printed holder.
- ROS lack of support for Python 3: implementing a detection algorithm solely dependent on the OpenCV library, instead of using more robust (You Only Look Once) YOLO detection.

IX. Outlook

This project is open to many improvements as far as the imagination goes. An Example would be the implementation of a face/emotion detection algorithm, which should predict the opponents emotional state as an input for the simulations. This detection of emotions can also be used in the detection of the opponents level, which would be a helpful feature for the opponent move estimation.

The simulations may be parallelized for a better estimation rate. Since multiples of the rollouts may be done while not blocking the game process, certainty of the chosen action would raise.

References

[1] Levente Kocsis and Csaba Szepesvári, "Bandit based Monte-Carlo Planning", In: ECML-06. Number 4212 in LNCS, 2006, 282–293, Springer

[2] Silver, David and Huang, Aja and Maddison, Christopher and Guez, Arthur and Sifre, Laurent and Driessche, George and Schrittwieser, Julian and Antonoglou, Ioannis and Panneershelvam, Veda and Lanctot, Marc and Dieleman, Sander and Grewe, Dominik and Nham, John and Kalchbrenner, Nal and Sutskever, Ilya and Lillicrap, Timothy and Leach, Madeleine and Kavukcuoglu, Koray and Graepel, Thore and Hassabis, Demis, "Mastering the game of Go with deep neural networks and tree search", 2016, 01, 484-489

[3] Maciej Świechowski and Konrad Godlewski and Bartosz Sawicki and Jacek Mańdziuk, "Monte Carlo Tree Search: A Review of Recent Modifications and Applications", 2021, 2103.04931

[4] Duarte, Fernando and Lau, Nuno and Pereira, Artur and Reis, Luís, "A Survey of Planning and Learning in Games", 2020, 06, volume 10, Applied Sciences

[5] EdjeElectronics. (2018). EdjeElectronics/opencv-playing-card-detector: Python program that uses opencv to detect and identify playing cards from a picamera video feed on a Raspberry Pi. GitHub. Retrieved February 5, 2022, from https://github.com/EdjeElectronics/OpenCV-Playing-Card-Detector

[6] Zheng, Chunhui, and Richard Green. "Playing card recognition using rotational invariant template matching." In Proc. of Image and Vision Computing New Zealand, pp. 276-281. 2007.

[7] Romero-Ramirez, Francisco J., Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. "Speeded up detection of squared fiducial markers." Image and vision Computing 76 (2018): 38-47.

[8] Garrido-Jurado, Sergio, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Rafael Medina-Carnicer. "Generation of fiducial marker dictionaries using mixed integer linear programming." Pattern recognition 51 (2016): 481-491.

[9] Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System." In ICRA workshop on open source software, vol. 3, no. 3.2, p. 5. 2009.