ChessBot: A Single View Perception and Manipulation System for Robotic Chess

1st Ethan Chun
EECS
Massachusetts Institute of Technology
Cambridge, USA
elchun@mit.edu

***Abstract*—Chess is a rich test-bed for human-machine interaction. In order to play, a robotic system must: (1) understand the current piece locations; (2) generate a feasible move; and (3) execute the move in the physical world. Unfortunately, current methods for perceiving chess pieces either involve custom built chess boards or a camera mounted directly above the board – both of which increase cost and physical complexity. In this paper, we subvert this hurdle and present a deep perception system to determine all piece locations from a single, side-mounted, rgbd image. Our approach, dubbed ChessBot, utilizes a fine-tuned Mask R-CNN and ICP system to perceive, parse, and reconstruct all piece locations from the single image. Integrating this perception system with an off-the-shelf chess engine and differential inverse kinematics pipeline produces our complete system. We find that ChessBot can perceive and act on roughly 86% of chess board configurations and carry a variety of chess games to completion. Additionally, we find that surveyed users enjoyed playing chess with ChessBot. Our code is available at https://github.com/elchun/ChessBot.**

I. Introduction

In order to play chess, a robotic system must: (1) understand the current piece locations; (2) generate a feasible move; and (3) execute the move in the physical world. Systems exist to solve the latter two issues – generating feasible chess moves and for moving arbitrary objects across a two dimensional board. However, the former presents more difficulty. Current board perception systems either involve custom built chess boards with integrated location sensing hardware [1], [2], or use cameras mounted directly above the board [3], [4]. While location sensing chess boards do not use external imaging, they are costly or complex and only register integer piece coordinates (i.e. what square a piece occupies) as opposed to 2d spatial locations. Conversely, systems using top mounted cameras can extract piece locations, however they are awkward to mount and have difficulty determining the type of piece. Therefore, we aim to circumvent both issues by developing a system which uses a single, side-mounted, rgbd sensor to perceive the board while being simple and cost effective. Our system should function for a wide variety of board states and be enjoyable to play against.

Our system, dubbed ChessBot, applies deep geometric perception to the issue of robotic chess, utilizing Mask R-CNN and ICP to perceive and reconstruct an arbitrary chess board (Fig. 1). Our preliminary testing demonstrates that ChessBot can perceive most chess board configurations and carry a chess game to completion. Additionally, responses from our small sample of users indicate that ChessBot is enjoyable to play against.

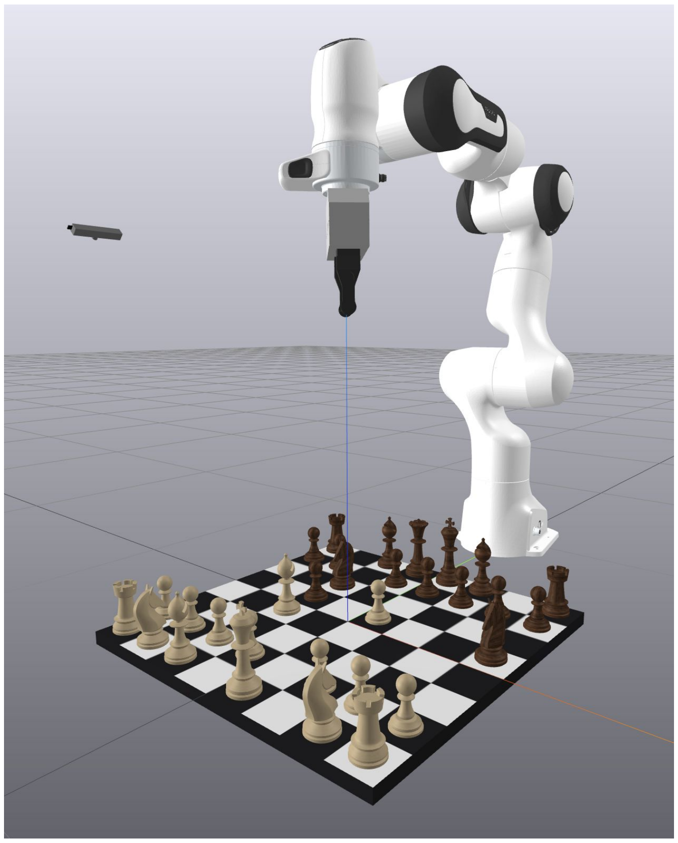


Fig. 1. Using a single, side mounted, rgbd sensor, ChessBot can perceive 86% of chess boards configurations, execute moves on demand, and complete full games of chess against a human adversary.

II. Related Work

A. Current Board Perception

Custom boards. Several works utilize custom built boards for determining piece location. The simplest of these, [1], [2], utilize an array of reed switches hidden beneath the board. A magnet is then affixed to the bottom of each piece. By reading the reed switch array, the system can tell which locations on the board are occupied. Then, comparing a previous and current board state while sampling the legal moves that transition between these states allows the system to calculate a probable move.

While a reed switch array can determine piece occupancy, the complexity of the board and expertise involved in setting it up is limiting to many who may wish to play robotic chess. Additionally, the binary nature of a reed switch does not permit the system to extract exact spatial locations of pieces. This makes subsequent robotic grasping of pieces difficult.

Additionally, more complex boards can track both piece location and piece type [5], however these are often prohibitively expensive.

Top mounted cameras. More closely related to our system are those that use top mounted rgb cameras and a series of computer vision algorithms to process the chess boards [3], [4]. The rgb pixel values are converted into a mask from which the relative piece locations are extracted. Edge finding algorithms may also be employed to determine the bounds of each chess board tile. As with the reed switch systems, the players move may be deduced by tracking the type of piece at each location, then comparing the difference between the previously observed and currently observed boards.

In contrast to reed switch systems, top mounted cameras are able to determine precise piece location from the masked images. However, a custom apparatus to hold the camera must be constructed which can decrease usability. Conversely, this system only requires an rgb camera, which may make it accessible for those willing to set up the camera frame.

B. Deep Perception of Arbitrary Objects

Mask Generation. While current chess perception algorithms using hardware or classical computer vision have remained unchanged over the last few decades, the field of deep geometric perception has advanced considerably. Most relevant to this paper is Mask R-CNN [6]. Mask R-CNN is a variant of the Convolutional Neural Network architecture which uses a sliding window to make inferences from images. Building on Faster R-CNN [7], Mask R-CNN is able to predict the bounding box, segmentation mask, and label for instances of a known object in a novel image. While originally trained on the COCO data set [8], Mask R-CNN can readily be fine tuned to a variety of specific tasks. The ease of tuning and quality of masks that Mask R-CNN produces makes it an appealing choice for new approaches to reading chess board positions.

Point Cloud Alignment. In addition to deep perception on images, significant literature exists for processing point cloud geometry. While works like Neural Descriptor Fields [9] leverage learned latent feature spaces to perform pose alignment among classes of objects, simpler algorithms such as Iterative Closets Points (ICP) [10] can be used when the geometry of the object to align is known. ICP minimizes the distance between nearest neighbors of an observed and reference point cloud. Since the geometry of all chess pieces will be known, ICP may be applied to transform partial point clouds into full point clouds.

C. Solving

Another hurdle to developing a chess playing robot is developing an algorithm to determine the robot's move. A classical approach to this issues utilizes a Minimax search to search over the space of possible moves and maximize the potential future gains. (1)

$$v_i = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i}) \tag{1}$$

Where v_i is the value function of agent i, a_i is the action taken by agent i, and a_-i is the action taken by agent $-i$.

More recently, the open source engine Stockfish [11] uses alpha-beta pruning to optimize a Minimax search. Stockfish is commonly used for online chess game analysis and is one of the strongest non-neural net chess solvers. Succeeding Stockfish are reinforcement learning solvers such as Deepmind's Alpha Zero [12]. These generally outperform Stockfish and other Minimax solvers, but require large GPUs to run efficiently.

We elected to use Stockfish for this system as it can outplay all humans, runs well on a CPU, and is easy to install.

D. Move Execution

Robust grasping is a rich and active field of research. For planar pick and place tasks, a simple antipodal grasping algorithm often suffices. For more complex manipulation tasks, Neural Descriptor Fields [9] or similar pose estimation architectures may be used to encode grasp across entire object classes.

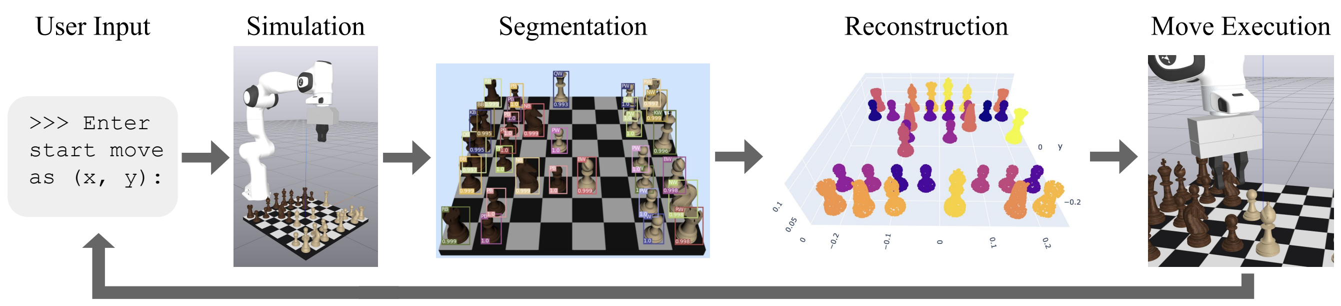
In our system, we know that all pieces must be upright. Therefore, antipodal grasping on the object point cloud would be ideal. However, for simplicity's sake, we elected to define our object collision geometry as a cylinder and grasp the mean of our perceived point cloud.

III. Methods

Reflecting the tasks that a chess robot must complete, the components of ChessBot can be segmented into perception, chess solving, and move execution. In addition, a simulated environment and user interface must be constructed to evaluate the system's capacity to play against a human player. In Section III-A, we present an overview of the simulated environment. In Section III-B, we describe the construction and training procedure for the Mask R-CNN classifier. In Section III-C, we describe the conversion of partial to full point clouds. Finally, in Section III-D and Section III-E, we describe the procedure for generating and executing chess moves from the perception data. Please see Fig. 2 for a graphical overview of the system architecture.

A. Simulation Environment

Using Drake Simulator [13], we simulate a chess board, all pieces, a Franka Panda Arm, and a Schunk WSG Gripper (Fig. 3). The chess piece models are from Rio Fortian on GrabCad [14]. The board was custom modeled. For simplicity, the collision geometry of each piece is designed as a cylinder



```
>>> Enter
start move
as (x, y):
```

Fig. 2. **System Overview** – Our simulation first prompts the user for a move. Then, ChessBot uses its rgbd sensor to capture an image of the board. Mask R-CNN is applied to segment the board into labels and masks. These are reconstructed into full point clouds using ICP. Then, Stockfish and Differential Inverse Kinematics are used to execute the move in simulation. Finally, the system repeats and the user is again prompted for a move.

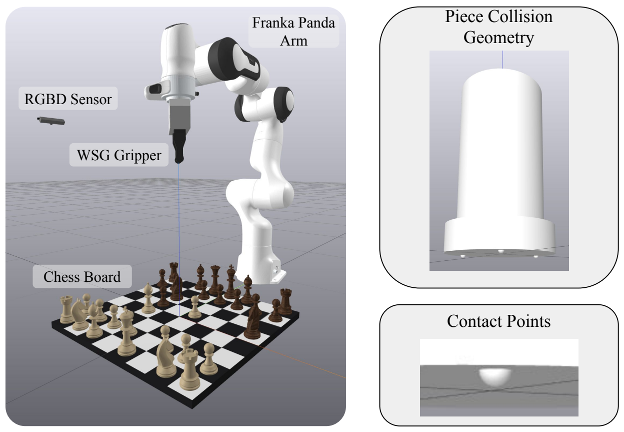


Fig. 3. **Simulation Setup** – We create the chess board, a Franka Panda Arm, and a WSG Gripper in simulation. The piece contact geometry is modeled as a cylinder with three contact points for simplicity and increased repeatability.

with three contact points on the bottom. The contact points prevent unwanted drifting of the part within the simulator. A wood texture was applied to both the white and black pieces. All pieces and boards are sized according to official chess regulation.

A Franka Panda Arm was used as the authors wanted to explore the possibility of using Drake to replace a Franka Panda Arm simulated in Pybullet [15].

We also include a single rgbd sensor to the left of the robot. The field of view was set such that no pieces are omitted from the view, yet the board occupies the majority of the image.

Additionally, we include a command line interface for users to enter their desired moves.

B. *Perception: Mask R-CNN*

In order to parse incoming rgdb images, ChessBot makes use of a fine tuned Mask R-CNN [6] Using the simulated environment, we set the chess board pieces to an arbitrary arrangement and randomly rotate each peice about its central axis, then record a rgbd image and a ground truth depth mask for the simulated rgbd sensor. We include both the piece type and color in the labels.

We repeat this 100 times to produce our training data set. Then, we tuned a pre-trained Mask R-CNN on this dataset for 4000 iterations. The CNN was pre-trained on the COCO dataset [8].

At test time, we query the Mask R-CNN to generate piece masks and predicted piece labels.

C. *Perception: Reconstruction*

Using the rgbd sensor and the Mask R-CNN masks, we generate a partial point clouds for each chess piece using the pinhole camera model [16].

$$X_c = (u - c_x)\frac{Z_c}{f_x} \tag{2}$$

$$Y_c = (v - c_y)\frac{Z_c}{f_y} \tag{3}$$

Where (X_c, Y_c, Z_c) are points in the camera frame, f_x and f_y are camera focal lengths, and c_x and c_y are the coordinates of the principal point.

To convert this into a full point cloud, we use the predicted piece type from Mask R-CNN to retrieve a reference point cloud which corresponds to the predicted object type. Then, we can use an Iterative Closest Points Algorithm (ICP) to align the reference point cloud to the sampled partial point cloud. Since all pieces must be resting on the chess board, we constrain ICP to only translate the reference point cloud in the x and y dimensions.

D. *Solving*

Once the board is reconstructed, we take the mean of each piece's point cloud to determine the square which the given piece occupies. Comparing this occupancy to the previous known board occupancies allows us to extract the user's most recent move.

While a custom chess engine could be written using a Min-Max or deep learning approach, we opted to use Stockfish, an open source chess engine commonly used in online chess games. Once the player's move is extracted, we feed it into Stockfish to determine the robot's next possible move.

E. Grasping and move execution

Using the output of Stockfish, we determine what pieces must be manipulated. Then, using the reconstructed point clouds from ICP and labels from Mask R-CNN we determine the mean location of each piece to manipulate. Finally, we use Drake's Differential Inverse Kinematics Solver to produce a viable trajectory for the Panda Arm. Once the piece is grabbed, we prompt the user to make their move, and the cycle repeats.

IV. Results

A. Experiment Design

Any chess playing robot must, above all else, be able to play chess. We therefore designed our sets of experiments to understand how well ChessBot can play. Furthermore, since we used both an off-the-shelf chess engine and an off-the-shelf differential inverse kinematics solver, the performance of these is unlikely to be unique to ChessBot. Therefore, we focus our experiments on evaluating the robustness of ChessBot's perception system and the enjoyment that a user gains from playing with ChessBot.

Robustness. Our first experiment evaluates ChessBot's robustness by answering the following: Across a multitude of games, how often does ChessBot correctly read and execute a move based on the current board state? More specifically, we evaluate the number of successful turns (one move by the player and one move by ChessBot) in comparison to the number of failed turns. The intention is to quantify the success rate of ChessBot and identify key areas of improvement.

We implemented this by replacing the user interface of ChessBot with another instance of Stockfish. Additionally, we set the experience rating of the player Stockfish to that of an average user in order to better represent the typical audience. We then had ChessBot play games against the Stockfish instance until a minimum of 100 turns had passed.

Enjoyment. Our second experiment takes a more qualitative approach. We surveyed a selection of individuals and asked, ”Did you enjoy playing with ChessBot? Why or why not?” We then tabulated extracted selected quotes to gain a qualitative understanding of a user's perception of ChessBot. The intention with this experiment is gain insight into ChessBot's interactions with human users and to understand what could be changed to improve the user experience.

The authors employed convenience sampling to find a small selection of users to play with ChessBot. This selection is unfortunately not representative of the general population, but may still provide insight into ChessBot's appeal.

B. Robustness Evaluation

As shown in Table I, ChessBot achieves a turn success rate of 0.867, indicating that the current perception system is moderately successful. The perception system is shown graphically in Fig. 4, illustrating some of potential board configurations and their relevant reconstructions.

While ChessBot can perceive a large variety of board configurations, of particular interest are ChessBot's primary failure modes: perception and kinematic.

TABLE I
AUTOMATIC EVALUATION RESULTS

	Turns	Successes	Perception Failures	Kinematic Failures
Count	105	91	9	5
Rate	1.0	0.867	0.086	0.048

We examine the number of successful and failed turns completed by ChessBot when playing against an AI chess opponent. Perception failures are turns where the incorrect user move or incorrect grasp location was predicted. Kinematic failures are when the Differential IK solver fails or when pieces are dropped unintentionally.

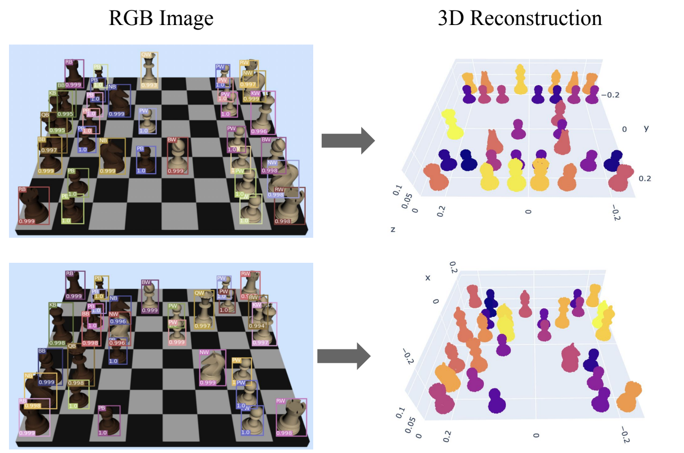


Fig. 4. **Perception Results** – ChessBot can reconstruct a wide range of piece types and locations. This enables ChessBot to perceive and act on the majority of states that arise in a typical chess game.

The perception failures are significantly higher for knights than for other pieces. This may be due to the short stature and relatively complex geometry of a Knight. We also note that pieces may be entirely occluded in some board positions. For example, consider a Queen in between a pawn and the rgbd sensor. Both the occlusions and piece complexity may explain how perception failures account for roughly two-thirds of all failures.

Furthermore, we found that kinematic failures were often correlated to perception failures. For example, if the location of a piece was observed to be translated from the true location, the piece may be placed at a corresponding incorrect location which could lead to gripper collisions.

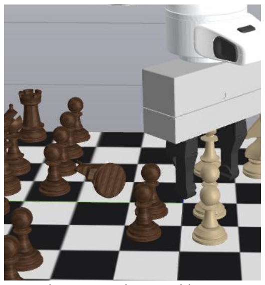
Please see Fig. 5 for an example of kinematic and perception failures.

C. Human Evaluation

In our brief user experience survey, one user commended that she ”loved it!” This implies that ChessBot provides an enjoyable user experience. However, a user also commented that ChessBot was ”not too fast – it definitely gave me a lot of time to think...” indicating that some improvements in speed may be needed.

V. Discussion

On a whole, the authors aimed for ChessBot to demonstrate the feasibility of single view chess playing robots. Both our robustness testing and user studies indicate that ChessBot is


Kinematic Failures

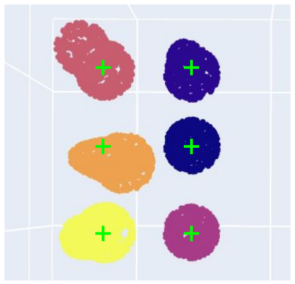

Perception Failures

Fig. 5. **Failures** – On the left, a kinematic failure has resulted in a piece being knocked over. On the right, a perception failure is shown. Green crosses indicate the ground truth centers of each piece, however the orange piece is incorrectly perceived as translated from the ground truth position. Both types of failures can impact game performance; however clever move recognition algorithms may circumvent these shortcomings.

indeed able to accomplish this goal. However, ChessBot is not without its limitations. Section V-A discusses some of the important successes from ChessBot. Section V-B outlines some use cases – chess specific and otherwise – for ChessBot. Section V-C describes some of ChessBot's limitations and potential remedies for them.

A. Successes

First and foremost is the user satisfaction from playing with ChessBot. Our subjects reported that they enjoyed playing with ChessBot, implying that ChessBot as a concept is a fun system, as well as a potential means of building public interest in robotic manipulation and deep perception.

Furthermore, the high turn success rate indicates that it is feasible to use a single view rgbd sensor to capture chess board information. This, combined with the ability to reconstruct objects based on their target labels, would by highly useful for any other chess playing robotic systems that wish to minimize physical complexity.

B. Use Cases

While we demonstrate ChessBot with a robotic arm, an important use case for ChessBot may be as pure perception system. Given recent advances in rgbd cameras (and depth sensing phone cameras), it may be feasible to use a packaged version of ChessBot as a means to record moves during chess tournaments. The single-view design would simplify setup while the availability of mobile depth cameras may reduce startup costs. Furthermore, the lack of custom board requirements would allow users to play on any chess board.

Moving outside the field chess, single-view point cloud completion may be helpful for developing household robotic agents. If a user has a limited number of dishes, for example, a household robot could store a reference point cloud for all objects. Then, when the robot is tasked with washing the dishes, the robot could use a similar pipeline to ChessBot to classify, reconstruct and reason about the objects from a partial or occluded view.

C. Limitations and Next Steps

Misplaced Pieces. Not every system is perfect, and ChessBot is no exception. Of particular interest are the perception failures. Among these, ChessBot will either estimate that a piece is in a different location than it actually is, or miss an occluded piece entirely. While translated pieces may be more difficult to solve, disappearing pieces can likely be fixed by refining the move prediction portion of ChessBot. Instead of simply comparing past and current boards, ChessBot could compare past and current boards with the added assumption that the user's move is legal. This would reduce the number of moves that may arise from a piece disappearing and may remedy the problem entirely.

Interestingly, ChessBot is unique among chess perception systems for having the issue of occluded pieces. Both top mounted cameras and custom built boards can perceive all pieces regardless of location. The authors, however, think that the increased simplicity of ChessBot makes up for this performance limitation.

Speed. Another common complaint among robotics system is speed. ChessBot, like many other robotic perception systems, is rather slow. Predictions take 12 seconds on average, and move execution can take up to a minute afterwards. Optimizing the deep perception system to operate on lower resolution images may help with speed, as might increasing the computing power available to the simulator (to help speed up move execution).

VI. Conclusion

We introduce ChessBot, a single view, deep perception, system for playing robotic chess. Through ChessBot, we illustrate that such a system is capable of perceiving most chess boards and playing full chess games, providing a path towards simpler chess board perception systems. Furthermore, we demonstrate user interest in ChessBot, and potential applications to chess and beyond. We also acknowledge the limitations of ChessBot, including occluded pieces and slow processing time. We propose that future work may remedy these issues through move based heuristics to circumvent occlusions or with a higher throughput, lower precision, deep perception system. We hope that the methods presented in ChessBot may prove useful to the further development of chess playing robots and to the field of robotic manipulation as a whole.

References

[1] yousifnimat. "Chess robot : 7 steps (with pictures) - instructables." (), [Online]. Available: https : / / www . instructables . com / Chess - Robot/ (visited on 12/11/2022).

[2] F. A. T. Al-Saedi and A. H. Mohammed, "Design and implementation of chess-playing robotic system," vol. 5, no. 5, p. 9, 2015.

[3] J. Meyer. "Raspberry turk." (2017), [Online]. Available: http://www.raspberryturk.com/ (visited on 12/11/2022).

[4] P. K. Rath, N. Mahapatro, P. Nath, and R. Dash, "Autonomous chess playing robot," in *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, New Delhi, India: IEEE, Oct. 2019, pp. 1–6.

[5] DGT. "The DGT electronic chessboard USB – chess house." (), [Online]. Available: https://www.chesshouse.com/collections/dgt-electronic-chess-boards-pc-connect/products/the-dgt-electronic-chessboard-usb (visited on 12/11/2022).

[6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask r-CNN," *CoRR*, vol. abs/1703.06870, 2017. arXiv: 1703.06870.

[7] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-CNN: Towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. arXiv: 1506.01497.

[8] T.-Y. Lin, M. Maire, S. J. Belongie, *et al.*, "Microsoft COCO: Common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312.

[9] A. Simeonov, Y. Du, A. Tagliasacchi, *et al.*, "Neural descriptor fields: SE(3)-equivariant object representations for manipulation," *CoRR*, vol. abs/2112.05124, 2021. arXiv: 2112.05124.

[10] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, Quebec City, Que., Canada: IEEE Comput. Soc, 2001, pp. 145–152.

[11] stockfish. "About - stockfish - open source chess engine." (), [Online]. Available: https://stockfishchess.org/about/ (visited on 12/11/2022).

[12] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. arXiv: 1712.01815.

[13] R. Tedrake and the Drake Development Team, *Drake: Model-based design and verification for robotics*, 2019.

[14] R. Fortian. "Chess board and pieces | 3d CAD model library | GrabCAD." (), [Online]. Available: https://grabcad.com/library/chess-board-and-pieces-3 (visited on 12/11/2022).

[15] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[16] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.