**Team 9 Final Project: ChessBot**

Tarun Narayan, Nico Ranabhat, Com Boonkert

**Introduction**

ChessBot is a robotic arm able to move chess pieces through user command, with the hope of eventual integration with StockFish, a chess engine that can calculate the best move. The goal of ChessBot is to be an over-the-board training tool with varying difficulty levels (through the use of StockFish) since playing over the board vs. playing online is a completely different type of competitive game. Though this was our secondary idea, we believe that this project is more doable in our short time frame, will be challenging, and super fun. Additionally ChessBot has a combination of both computer science and mechanical tasks, allowing us to play to our strengths and split up the work equally.

**Design**

In terms of the design process we split the project into mechanical and code design. Upon visual inspection, it was glaringly apparent that our robotic arm couldn’t reach pieces on a full size chess board. Using the techniques learned in class we were able to map out our robotic arm’s reachable workspace and found that an 18x18 cm board would work best. Based on this known area we subdivided it into 64 squares of equal area and found online that a piece’s base diameter should be at most 75% of the square [1]. This came out to a piece diameter of 2 cm for our board but we decided to go for a rectangular prism piece instead of the standard piece shape since it will be easier for the robot to grab accurately and 3D print. After we made the piece in SolidWorks we realized that the robot can easily knock over pieces and because of dead reckoning, will drift over time resulting in improper piece placement. To help combat this mechanically, a 1.1 cm circle with a 0.1 cm cut was put in the bottom of the piece so a magnet of that size could be glued to the bottom. The idea was that if we put magnets on the board and pieces, the likelihood of the pieces being knocked over would go down and they would snap in place when nearby, hopefully combatting drift. With this in mind, the board was created with circular holes to hold the magnets instead of the standard squares. Both were printed using PLA material with a 20% in fill since they don’t need to be super durable. After they were printed the magnets were glued into the respective slots.

For the software side we split the code into inverse kinematics control and gripper functionality. For inverse kinematics, the first challenge was to modify the “pen holding” robot inverse kinematics to one where the robot was able to extend its wrist joint out past a vertical orientation. This would allow the robot to reach locations further away from its base. To derive these inverse kinematics, we first had to define the locations of all points on the chessboard. This piece of code mapped square locations (e.g. ‘B5’) to x-y-z coordinates in real space. Once we had a mapping for all the squares on the board, we had to determine the robot arm joint angles that would result in the end of the arm reaching any given endpoint in the workspace. To do this, we utilized the SciPy optimization library in Python to numerically solve the inverse kinematics problem. The solver struggled to find a solution at first, but after removing unnecessary parameters and tightening the bounds, it was able to converge and successfully output the joint angles for the corresponding locations of all squares on the chessboard.

After solving the inverse kinematics problem, we created a script to determine a set of waypoints the arm would visit when making a chess move. The waypoints included an ‘origin’, locations above and on the starting piece, and locations above/on the ending location. We edited the ‘manual\_endpoint\_locations.py’ file to follow this determined path when given user input.

Once we determined the arm path for any given move, we just needed to figure out how to actuate the gripper to grab the chess piece. We found out that the robot arm’s gripper can be commanded similarly to the other servos for the arm joints simply by appending a specific gripper angle to the arm angles. Knowing the gripper could be commanded in this way, we needed to make sure the arm motion would pause when the gripper was actuating so that it could accurately grasp an individual piece. This required making additional changes to the ‘manual\_endpoint\_locations.py’ file.

**Testing**

After testing the arm with the board, we noticed some problems with the motion of the arm and gripper right away. Often the arm would undershoot the starting position as well as overshoot the ending position, all while the gripper would not close tight enough. After doing tests with the board being moved to different positions, we realized that it wasn’t just a placement issue. It turns out that while the board was thought to be the correct size, the measurement from the central axis of the robot to the board was potentially incorrect. We ended up having to take time to change the virtual workspace dimensions while shifting the board around until we were able to get accurate movement of the arm. Once this was done, we focused on fixing the gripping problem. At first we thought that the pieces were too slippery and the gripper couldn’t get any traction on them. However, after fixing the virtual workspace and aligning the board correctly, and trying different gripper angles, we were able to correct it.

**Result**

We were able to get ChessBot to grab and place pieces correctly given user input of starting and ending squares (e.g. A1 to H8). Due to time constraints we were unable to integrate StockFish with the current code. However, our ChessBot should be able to pick up a piece from any square to another square on the custom board, given there is a manual input from the user.

**Discussion**

The main thing we learned about this project was different ways to compute inverse kinematics. This project gave us some experience with MoveIt, a software used to compute inverse kinematics for any robot given a URDF file with the dimensions and specifications. Though we weren’t able to successfully use Moveit, the experience working on it was valuable and showed us what solutions are out there.

If we had more time for this project we would probably invest more time into developing the board and chess pieces. Since we had time constraints we decided to choose a design that was simple and slightly rushed. The magnets were .1 cm, so to allow them to fit in the hole properly we made the magnet hole dimension .11cm which created some problems for us. We weren’t able to get the magnets to fit in the hole perfectly at first leading to a row of lopsided magnets. To make sure the entire board didn’t have this problem we had to cut off extra material at every hole to press fit the magnets in.

Additionally the magnets in the board, while a good idea, were extremely annoying to work with. We purchased these magnets online thinking they would work perfectly but when they arrived we realized that they were extremely strong. If we weren’t careful and the magnets were close together, they would connect to each other at high speed and force that they would instantly break. At one point we also made the mistake of putting the incorrect polarity of the magnet facing up. Instead of using magnets, a major improvement would be utilizing computer vision to detect all the pieces on the board. That way the robot arm can make sure it grabs/places the pieces correctly every time without bumping other pieces. Additionally, the robot could see what the opponent’s move was and use StockFish to play the best move. This would take a decent amount of time but would dramatically improve the robustness of this project.

**Contributions:**

|  | **Tarun** | **Nico** | **Com** |
| --- | --- | --- | --- |
| **Concept/Planning/Research** | **33** | **33** | **33** |
| **Design - Hardware** | **70** | **0** | **30** |
| **Design - Logic** | **30** | **30** | **40** |
| **Programming** | **5** | **85** | **10** |
| **Presentation** | **50** | **20** | **30** |
| **Report** | **33** | **33** | **33** |

**Code:**

The code for this project is available at <https://github.com/nranabhat/Chess-Playing-Robot>. To run the code, launch the ‘robot\_manual\_nocmd.launch’ file and run the ‘manual\_endpoint\_locations.py’ node.

**References**

[1]“Chess Size Guide - The Regency Chess Company, The Finest Online Chess Shop,” *www.regencychess.co.uk*.https://www.regencychess.co.uk/size\_guide.html#:~:text=There%20is%20an%20official%20tournament (accessed Dec. 13, 2023).

‌