Connect 4 Robot

ME 4451 Final Project Report
Fall 2017

Dr. Nader Sadegh and Dr. Wayne Book

Mukul Bhatt, Edoardo Dioni, Naud Ghebre, Nathaniel Stowe

December 5, 2017

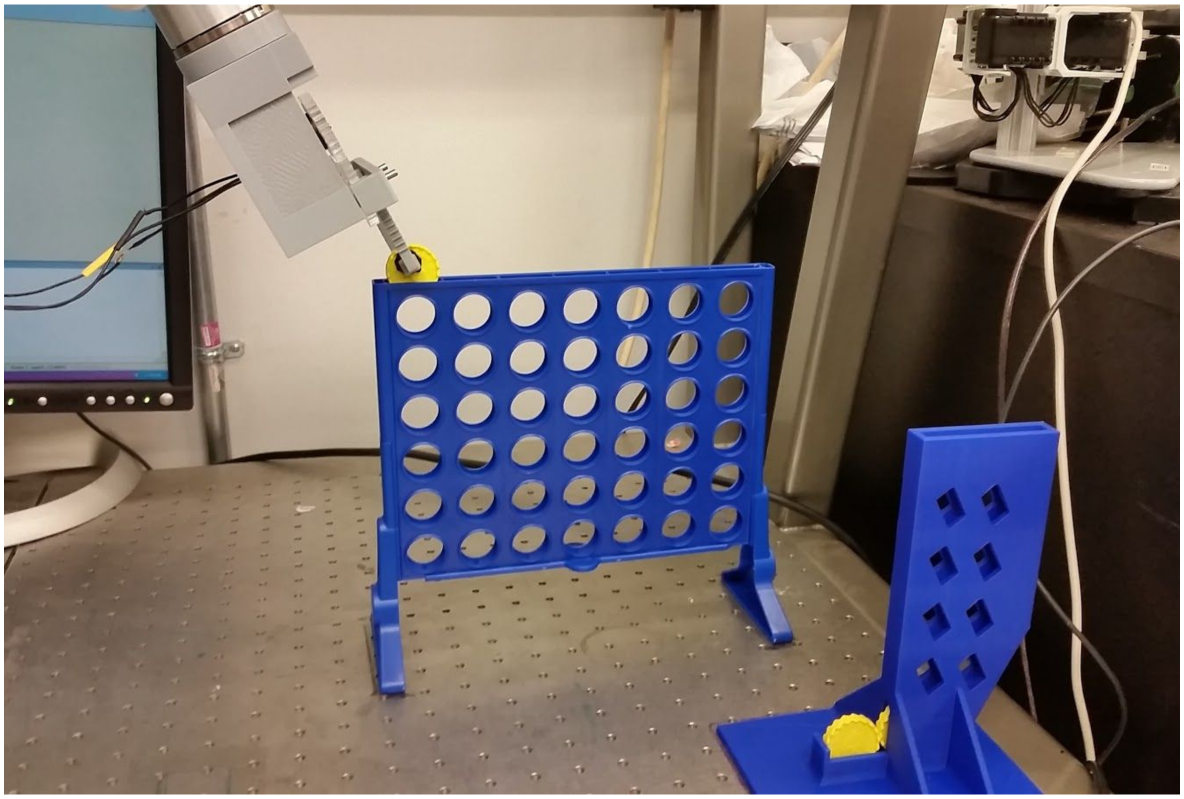


Table of Contents

1. OBJECTIVE

Design an AI player that can utilize the Epson C3 Robot to play Connect 4 against a human opponent.

2. DESIGN AND IMPLEMENTATION

2.1 DESIGN

The setup includes a camera that has eyes on the board at all times and detects when a move has been made by the opponent (human player). The robot then uses an algorithm to determine the optimal location to place its chip. The robot picks up its chip using a gripper end effector, place the chip in the desired column and then release it there. The end effector locations are provided for where to pick up the chip and where to place the chip.

2.2 VISION

The vision algorithm works through color thresholding. Assuming the background is uniform, only blue, red and yellow were filtered using the MATLAB app *Color thresholding*.

1. **Board**

   By filtering the blue of the board, it is possible to obtain a binary image, in which a rectangular geometry can be detected. Straight lines were detected using Hough transform, which implements the Standard Hough Transform (SHT). The Hough transform is designed to detect lines, using the parametric representation of a line:
   $\rho = x * cos(\theta) + y * sin(\theta)$

   The variable rho is the distance from the origin to the line along a vector perpendicular to the line. Theta is the angle between the x-axis and this vector. The hough function generates a parameter space matrix whose rows and columns correspond to these rho and theta values, respectively. The *houghpeaks* function was used to find peak values in the parameter space. These peaks represent potential lines in the input image. The best four lines were then selected assuming all the lines can have a θ value around $\pm 90°$ or $0°$ and assuming that ρ for each line is unique.

Then intersections were computed in order to obtain the edges coordinates (red circles). This resulted in four segments, the two vertical lines were divided by the number of rows of the board (six rows) and the horizontal ones were divided by the number of columns of the board (seven columns). Connecting the corresponding points of the horizontal and vertical lines, led to the grid design. The intersection of these ideal lines is important. These points are highlighted in pink in the following image.

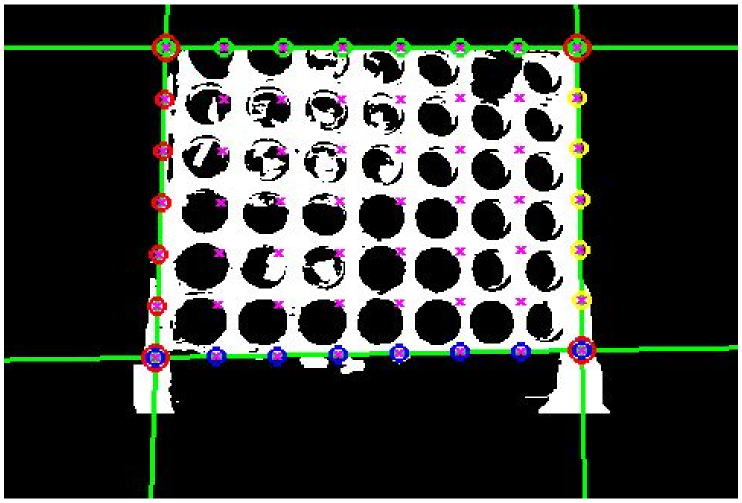


Figure 2.2-1: Blue mask output and edge detection.

These points are stored in a 7x6x2 matrix. Because of this structure, it was possible to relate the chip coordinates in the image with their actual position in the board.

2. **Chips**

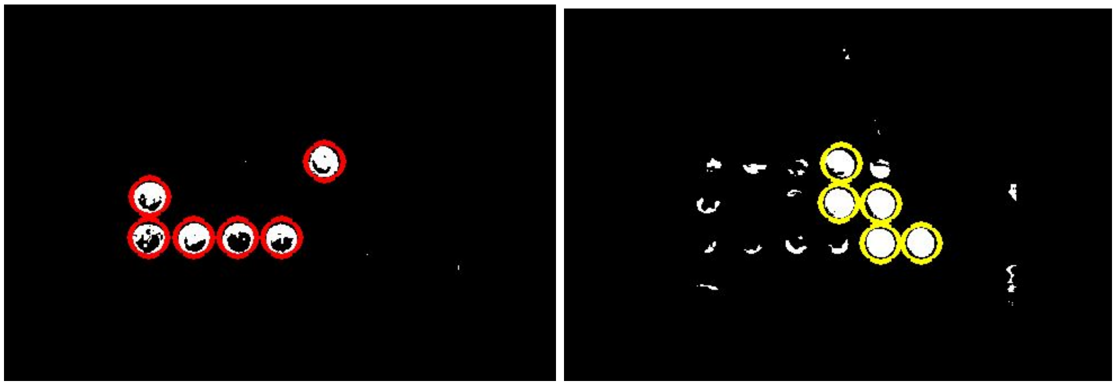


Figure 2.2-2: Red and yellow masks output and chip detection.

Red and yellow were detected by applying two color filtering masks. The two masks were generated and analyzed separately. Utilizing MATLAB function *regionprops,* the area and the center of mass for each blob in the binary picture were determined. The area was used as a filter in order to get rid of some smaller spots that appeared in the mask. The chip area is known to be much bigger than noise and therefore this criteria was used as a threshold level. In addition all the spots with center of mass, outside the board area were discarded.

In order to detect in which row and column a chip was placed, the center of mass coordinates were compared with the coordinates of the points in the pink grid (figure 2.2-1) as mentioned before. Each chip has a center of mass, which will have x coordinate larger than the grid points coordinates in all the columns prior, and y coordinate larger than the grid points in all the row prior.

3. **Output**

Vision algorithm provided an image from the camera with all the lines and points detected in superposition, so that it was possible to see, in real time, what the algorithm was detecting.

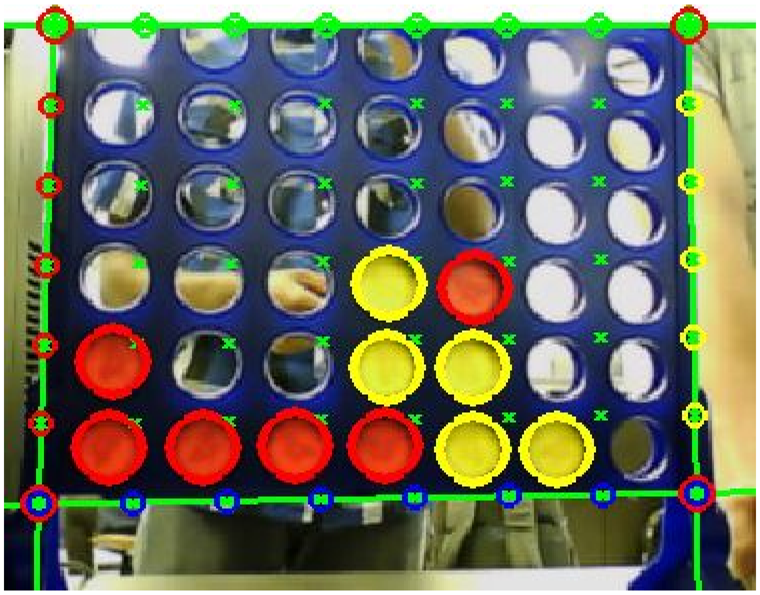


Figure 2.2-3: Final interface.

The output from the algorithm was a 6x7 matrix where each entry represented a slot. A "0" meant it was empty. A "1" meant it had been filled with a red chip and a "2" with a yellow chip.

At this point the algorithm did not check if the output is realistic, physically possible or consistent with the previous states (this was done by the game algorithm).

2.3 PATH PLANNING/KINEMATICS

Displacement kinematics and path planning was employed for the Connect 4 Bot to carry out each of its plays.

The displacement kinematics considerations for the Connect 4 Bot was centered around configurations for the Epson arm that would allow the end effector to start in some start/home position (one that would be away from the board, opponent, and field of play for safety reasons), pick up a chip from the chip dispenser, and drop the chip in appropriate column. Some considerations that were taken included exploring the Configuration Space to determine joint angles that would not interfere/collide with any of the field objects over its' trajectories. Lastly, trial and error runs over the Coordinate Space allowed for determination of accurate locations for each of the columns and the next chip from the chip dispenser, as shown out in Figure 2.3-1. For this, three main path routines were identified within the code: Pick up Routine, Drop Chip Routine, and Return To Home Routine. Points in space were determined and the reverse kinematics for the arm's movements were calculated by the Epson. In the Pick Up Routine, 7 points were identified via manual manipulation of the Epson arm to define the trajectory that would allow the end effector to travel from the start position to the chip dispenser, grip the chip and wait above the Connect 4 board. When this routine reached the appropriate point in its trajectory, the program would send a servo command to close the end effector gripper. 3 Points were identified to define the trajectory from above the board to the correct column (from 1 to 7) as instructed by the Connect 4 AI player. When the appropriate column was reached by the end effector, a servo command instructs the gripper to release the chip.The last routine of a play is the Return to Home Routine, which is defined by two points that instruct the arm to return to the start position behind the board and field of play.

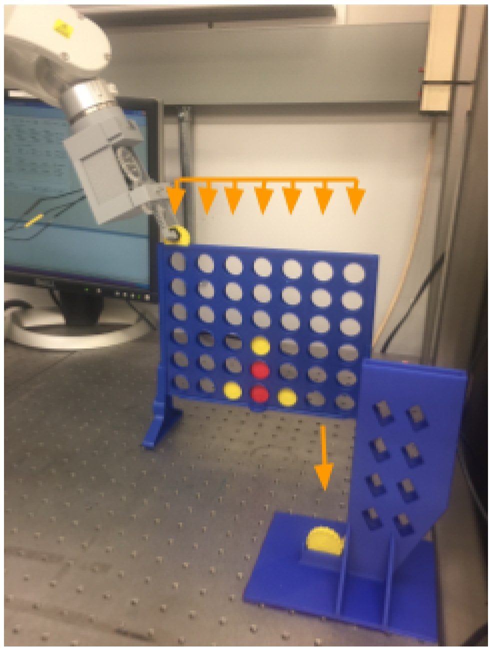


Figure 2.3-1: Points of interest to send to Epson arm.

The velocity trajection was controlled by the Epson's acceleration limited capabilities. Forty percent of the arm's max speed and twenty percent of max acceleration was used to make sure the Connect 4 Bot makes timely moves, while not posing a risk to the player and damaging the field of play. Tuning these parameters reduced wait time and provided seamless execution of its plays.

2.4 PHYSICAL INTERFACES

GRIPPER (End Effector):

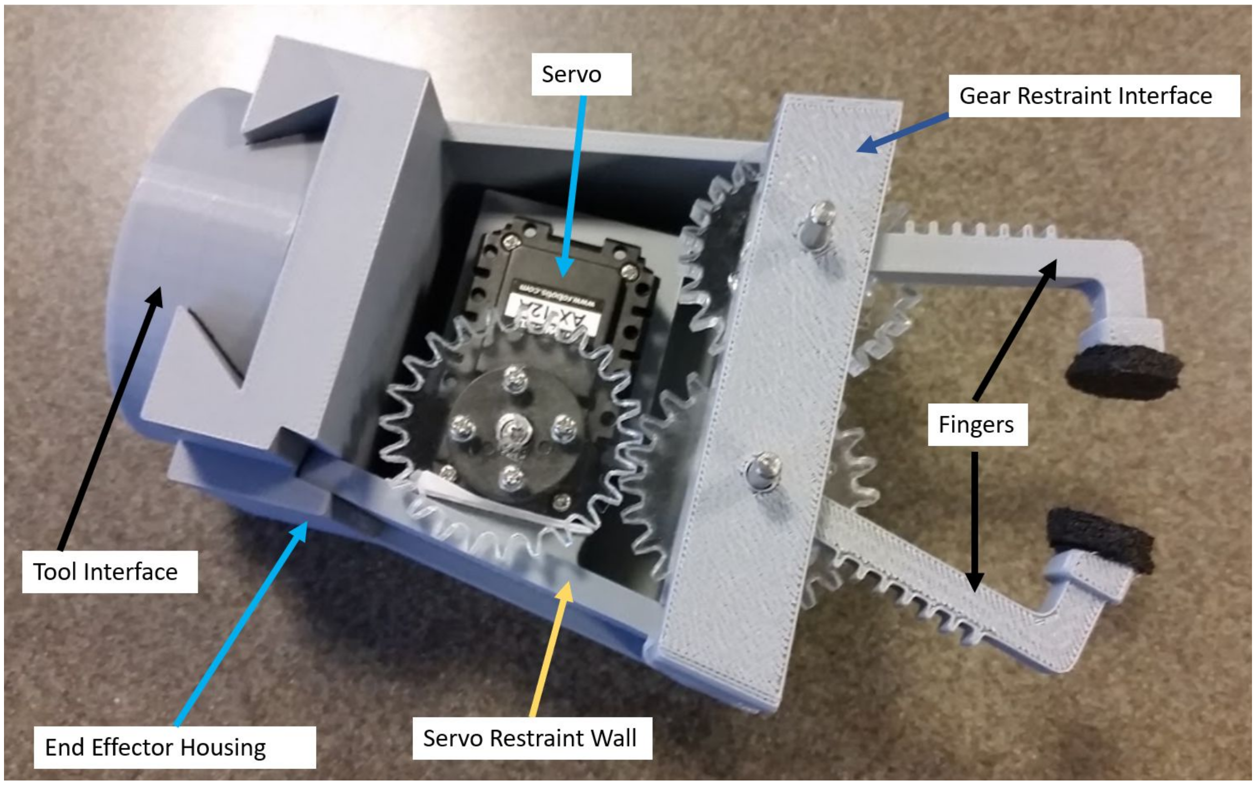


Figure: 2.4-1: Robot Gripper, View 1

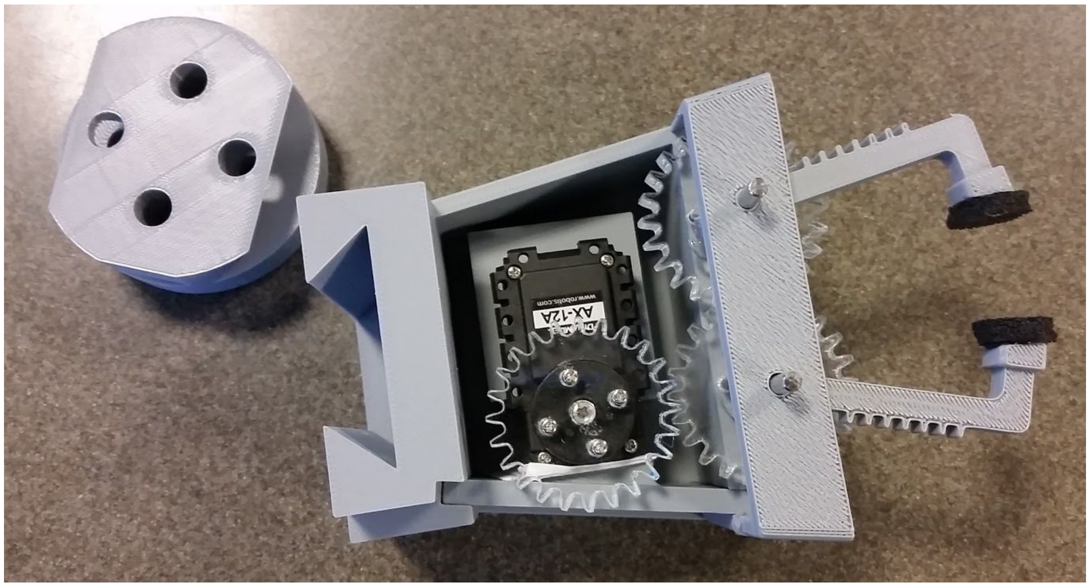


Figure 2.4-2: Robot Gripper, View 2

In order for the Epson Robot to play Connect 4, it needed an end effector with a mechanism to physically interface with the chip.

Several gripping mechanisms were brainstormed. These included: adhesive-based gripper, “spatula” gripper, vacuum gripper, angular finger gripper, parallel finger gripper. The Adhesive-based and “spatula” gripper were considered as they were the easiest to implement when picking up the chips. However, neither of these methods were chosen due to a lack of precision and repeatability that was necessary to place the chips inside the small slot of the Connect 4 gameboard. The parallel gripper design offered the benefit of ensuring motion of the fingers perpendicular to the chip when the gripper was open and closed allowing for high precision when dropping a chip. However, this design was not chosen due to the complexity of the design versus the application. The angular gripper design was chosen since it offered the best balance between precision, repeatability, and necessary complexity-to-application ratio. (For the Brainstorming sketches see the Appendix.)

The design of the angular gripper was started using the interfaces necessary. The main aspects that were required were connection with the Epson tool interface and the chip. For the tool interface a four part assembly for the housing was utilized in order to facilitate assembly and installation of the gripper onto the Epson. The tool interface (Figures 2.4-1, 2.4-2, and 2.4-3) was designed as a separate piece from the housing for ease of installation. The bolts could be mounted into this piece and the mating bolt holes on the Epson, then the rest of the housing could be slid onto this part with shims in order to prevent movement.

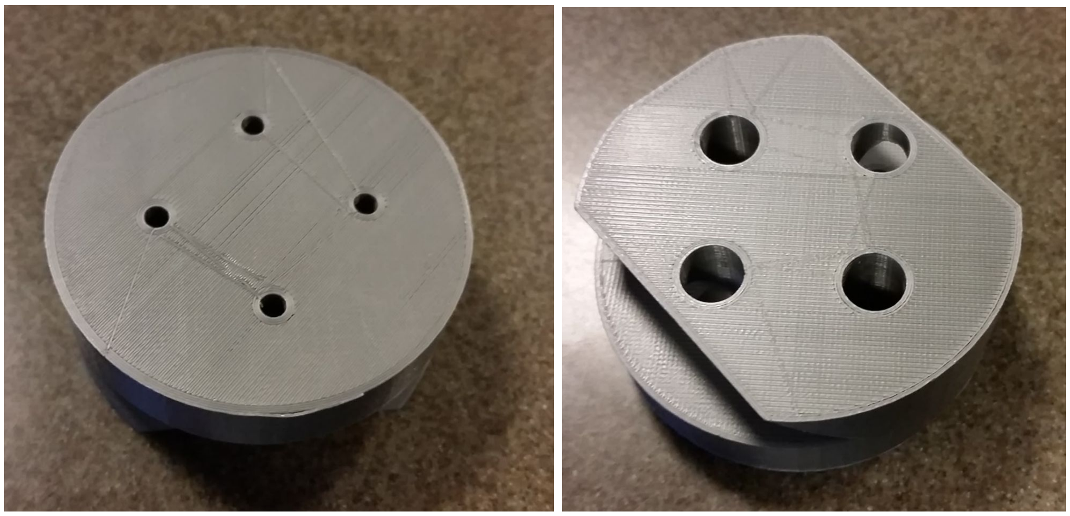


Figure 2.4-3 Tool Interface (top and bottom views)

The housing had a built in servo holder, that the servo can slide into for assembly. A servo restraint wall (Figure 2.4-1) was added in order to hold the servo in place. The final part of the housing was the gear restraint interface, which slid onto the dowel pins once the gears and fingers were installed, to prevent undesired gear motion.   There was an initial housing design used that was redesigned due to the servo restraints not having a sturdy connection to provide enough reaction forces to hold the servo in place. The previous design can be seen in the Appendix.

The housing, tool interface, fingers, and gear restraint were designed by the group and 3-D printed. The gears were designed by the group using an online gear profile generator and then laser cut.

Chip Dispenser

In order for the gripper to grab a chip in a manner that it could drop it into the Connect 4 board, the chip needed to oriented vertically. In order to provide chips in the necessary orientation, a chip dispenser was designed to constantly keep one chip ready in a particular position for the gripper to pick.

The chip dispenser was designed and 3-D printed by the group (See Figure 2.4-4).

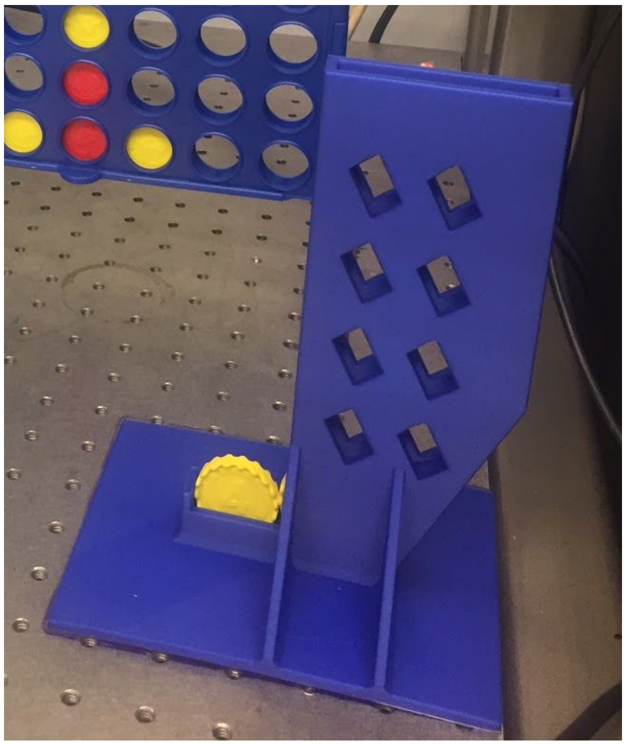


Figure 2.4-4: Chip Dispenser

2.5 AI ALGORITHM IMPLEMENTATION

The algorithm employed in the Connect 4 Bot implementation is loosely based upon the popular minimax algorithm. The minimax algorithm essentially evaluates a simulated game for each of the available play positions and determines the optimal play position based on the evaluation of each possible consecutive play. The complexity of the minimax algorithm is based upon a characteristic game parameter known as depth. The depth parameter controls the level of search when determining the optimal play position. For example, a depth of 2, simulates the game for each available play position for both the human play and the robot for two game plays. Such an operation is shown in Figure 2.5-1 (see Appendix) for the implementation of minimax in a reduced complexity game like tic-tac-toe. The depth parameter exponentially increases the number of computations and decision trees. For this reason, the traditional minimax algorithm, though extremely effective, is often used with a low search depth to balance gameplay intelligence and number of computations. In an attempt to resolve some of the issues with the minimax algorithm, a simpler method of recursion was employed to reduce computation heft and provide sufficient intelligence to beat most human connect 4 players. The connect 4 algorithm implemented in the project is based upon a simple heuristic of chip density and utilizes 3 main methods.

The first function named testAlgorithm is designed to impulsively decide where the robot should play based upon the relative occurrence of the robot's play chip. Each consecutive chip in a row, column, or diagonal is counted with a value of 1. When a winning configuration is anticipated through the simulated play at a given available play position, a value of 999 is used for the position's point score for the player and a value of 1000 is awarded for the robot's winning position score. In this way, the algorithm prioritizes a win by the robot over a player win. This simple heuristic alone, is insufficient to beat the most novice player of Connect 4 and also reveals an underlying flaw to the full implementation overall. The underlying flaw in this heuristic is that certain play positions are often evaluated as being optimal when they do not advance the robot to a position that can lead to a feasible win.

For example, Figure 2.5-2 (see Appendix) shows that using the testAlgorithm function, the most optimal play position is on row 5, column 1. In reality, this play position is not a worthwhile play, because a winning configuration is unlikely to occur as one of the diagonals is already limited by the corner restriction of the connect 4 board. It would be far more advantageous to play elsewhere to maximize the opportunity to develop row, column, or diagonal winning configurations. To enhance the capability of the first heuristic, the function also employs a simple comparative analysis. For each available play position, a point matrix is updated with a new value for the point density of player chips and robot chips. Once all of the available positions have been evaluated. The optimal play position is determined based upon the best play for the player and for the robot. If the point evaluation for the player's best play position is higher than that of the robot, the algorithm will recommend the robot to play in the position that yields the highest point density for the player. In other words, the algorithm recommends a block/defensive strategy. Likewise if the robot's best play position is higher than that of the player, the algorithm will recommend the robot's best play position. In other words, the algorithm recommends an attack/offensive strategy. The case where the optimal play positions for each player have the same point evaluation, the algorithm always works to advance the robot.

The second function named testAlgorithm2, enhances the algorithm by increasing the predictive capability of the system. This function uses the first function to simulate the optimal play positions for the user and the robot using the point density heuristic. The testAlgorithm2 function evaluates each available play position and simulates an entire game to completion. Because the function is based upon the heuristic of point density, the player and robot positions are assumed throughout the game tree. The distinction between the minimax algorithm and this implementation is more clearly shown in terms of number of iterations. The implementation used here is linear in its operation scope whereas the minimax algorithm is exponential. The point density heuristic essentially limits the algorithm to only one uniquely defined game simulation for each available play position. As a result, the decision tree can be thought of as having a maximum of 8 completely separate branches. This is opposed to the minimax algorithm where each play (or joint) has a separate branch composed of joints that form other branches.

Figure 2.5-3 (see Appendix) attempts to highlight the distinction between the two approaches. Both figures simply demonstrate the trend that each algorithm follows. Ultimately, the number of joints is dependent upon the number of empty board positions for the minimax algorithm. As stated before, because of the exponential growth of the minimax computations, the depth parameter limits the number of considered possibilities. The number of joints using the implemented algorithm is based upon the number of plays left in the game until completion. Completion is defined as a win, loss, or game tie. The testAlgorithm2 function uses a determination of positive and negative life cycles to provide a play recommendation. These life cycles correspond with the number of iterations that occur before game completion. A positive life cycle is defined as the number of iterations before a given play position is simulated to result in a win for the robot. A negative life cycle is defined as the number of iterations before a given play position is simulated to result in a tie or loss for the robot. Positive and negative life cycles are represented as positive and negative integers respectively.

The objective here is to utilize this higher order heuristic to determine the optimal play position. The goal of the testAlgorithm2 function is to select a play position that prioritizes a robot win in as few iterations as possible whilst maximizing the game duration in the case that the robot cannot win. This translates to prioritizing play positions with the minimum number of positive life cycles and maximum number of negative life cycles. The third function named testAlgorithm3 utilizes the second function to simulate the consecutive play positions of the user and robot for each available play position. This function uses the same high level heuristic of life cycles and increases the predictive capability of the system. When the gaming implementation is based on this function, the robot will often avoid at all cost losing positions and plays with good intelligence overall while still minimizing the number of iterations.

2.6 ANIMATED EXPRESSIONS and SOUNDS

The animated expressions used in the connect 4 implementation attempt to explore the areas of social robotics and human psychology. The animated expressions are creations of Čežek, Dušan, and Nikola Markovic [3]. The robotic face oscillates between 11 unique faces. The face that is shown during gameplay is dependent on a game state evaluation. The evaluation is determined through the use of an emotion score. The developed emotion score parameter is used to detail the degree to which the robot is winning or losing. Figure 2.6-1 (see Appendix) demonstrates the varying emotions associated with each emotion score. Figure 2.6-2 (see Appendix) shows the displayed face when dynamic lighting conditions cause vision processing instability. The objective here is to develop a dynamic evaluation of robot emotion that conforms to the expectation of the user as the game develops. The functionality of the animated expressions during live gameplay is often limited due to the fact that the connect 4 algorithm often prevents the expression function from reaching the extremities of negative emotion. The evaluation of the emotion score is based on a conversion of the maximum number of consecutive chips for each available play position. The emotion score conversion based on this evaluation is as shown in Table 2 (see Appendix). The advantage of such a conversion is shown when the scores for each available play position are summed. For example, an emotion score of 2020 is clearly recognized as a game state where a given player is bound to win in one of two distinct winning combinations and two other available play positions where there are two consecutive chips. The emotion function is called after each play. Thus, the robot tends towards negative emotion after a user play and towards positive emotion after a play by the AI.

Experimentally, the childlike animated expressions invoke an association of gentleness and joy to users. This smooths the machine human interface during gameplay. Most individuals interacting with the machine are smiling though their entire interaction is with inanimate objects. To complement the human machine interface, the emotion function plays humourous music and displays the associated emotion based on emotion score ranges. For emotion scores between 777 and 1777, a warning sound is played to recognize the potential win by the user. For emotion scores above 1777, the AI emotes extremely positively or negatively. This is due to the fact that emotion scores above this critical threshold of 1777, imply the robot is headed for an imminent win or loss by the next play. In the case of an imminent robot win, the unsuspecting player is greeted by randomized tunes that playfully mock the user. This randomization decreases predictability and gives each user a personal experience. Upon a robot win, the user is presented by a standard winning song. In the case of an imminent robot loss, the player is greeted by tunes of sadness expressed by the robot. When the robot losses, the animated face emotes extremely negatively and the program plays a tune of defeat. The recurring value of seven is a direct consequence of the number of columns in a standard game board. It is likely the case that without such human-like details, the abrupt motions of the Epson C3 robot along with the intelligence of the AI and its emotions would invoke a sense of fear in users.

2.7 GENERAL PROGRAM FLOW (IMAGE FILTERING & TIMING)

With all of the major components of this system fleshed out above, the interfacing between the components are described here to portray the entire program flow.

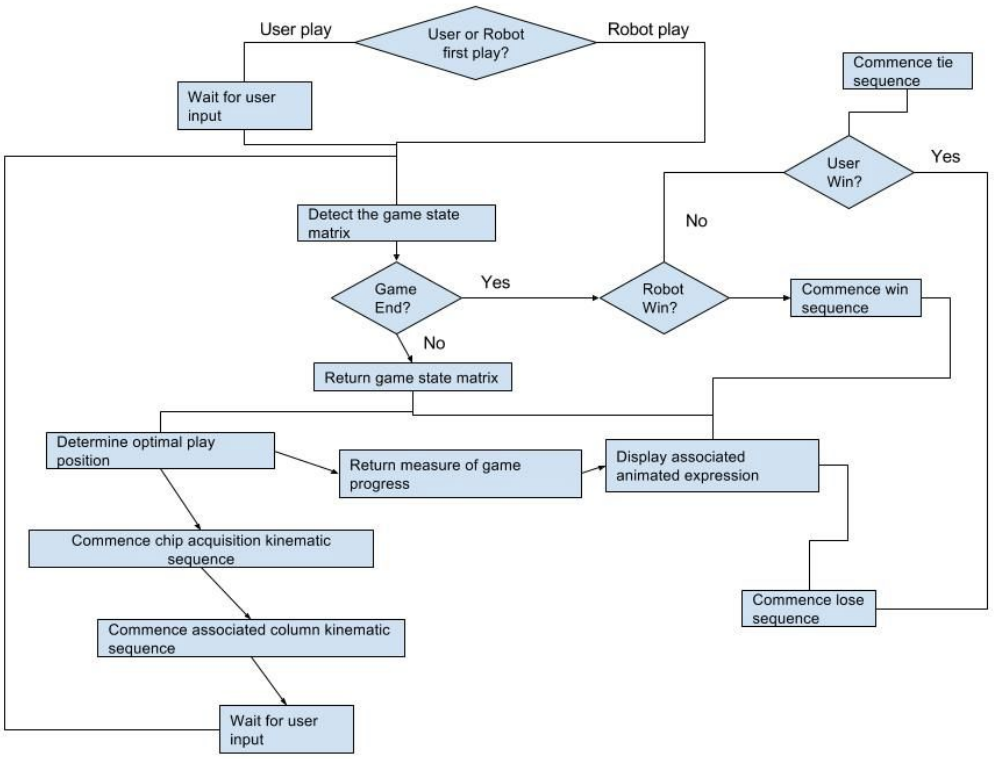

User play
User or Robot first play?
Robot play
Wait for user input
Commence tie sequence
User Win?
Yes
No
Detect the game state matrix
Game End?
Yes
Robot Win?
Commence win sequence
No
Return game state matrix
Determine optimal play position
Return measure of game progress
Display associated animated expression
Commence chip acquisition kinematic sequence
Commence associated column kinematic sequence
Commence lose sequence
Wait for user input

Figure 2.7-1: Programming Flowchart

Figure 2.7-1 above shows a more in depth look at the program flow. The entire game is run on a main while loop that continually checks whether the game is over. The program first acquires a new game board state as determined by the VISION component. There are checks made to determine whether the updated board change is valid, including whether the human player placed the correct chip in. The vision system is error prone due to noise and lighting in the lab, so obvious changes that do not make sense are discarded. For instance, fIltering techniques got rid of blobs that were either not in the board region or detected in between valid chip positions on the board. Another example, if the board matrix returned to the game loop shows an empty play position in between two player chips in a vertical column, the matrix is discarded and the program waits for the next loop to acquire another game state. Next, the updated internal matrix is passed into the AI ALGORITHM component, which is responsible for determining the Connect4 Bot's next move. After the column for play is determined, the internal matrix is updated and a dropChip routine is called, which is responsible for sending to the Epson the correct points in space for the column, as determined by the PATH PLANNING/KINEMATICS component. The points sent to the Epson are executed one by one until a chip is in the intended column and the arm is back at the home state. The program then reenters the next loop.

An issue that was encountered during the interfacing between the different components was in making sure the servo commands being sent to open and close the gripper were being executed at the right time. Often times the command would get sent and be executed a lot sooner than expected, since the commands are sent and queued at the Epson, instead of being executed in real time, instruction by instruction. The approach taken to get around this was to implement time delays between the moment servo commands are sent and the moment they'd be executed. Despite it not being the most elegant solution, it was viable given the repeatability of the system and how precise and accurate the Epson robot is.

3. RESULTS AND DISCUSSIONS

During the project presentation, the machine performed well. There were two instances where the robot missed chip placement in the appropriate column (see Section 4 Learning Experiences for discussion on accuracy) and one instance where the program did not exit properly after a user win condition. Though this was the case, the machine was robust enough to allow for such minor issues without propagating error throughout all aspects of the machine. The game state based progression of the game worked flawlessly and responded according to the dynamic user play wait periods. The vision system worked well under the changing lighting conditions due to crowd shadow movements around the vision system's primary lighting source, however, a background was necessary in order to guarantee an optimal performance of the algorithm. The kinematic elements of the robot performed well to swiftly obtain a chip with limited jerk and minimal system vibration. Overall, the system performed well and according to expectation.

Alternative Designs for Robustness

Some alternative designs that were considered but not implemented include using a vacuum end effector to grip the chips. The reason for this consideration was to simulate an actual Connect 4 game between players, where a chip dispenser is not available, thus the chips would be laying flat on the table. With the use of another camera that has eyes on the table and chips sprawled about, a more robust design would allow the Epson arm to detect its own chips and pick it up from anywhere to make its next move.

4. LEARNING EXPERIENCES

End Effector

During the build and design process, there were several lessons that the group learned. The first area was manufacturing and design tolerances. The end effector gears were designed for tight tolerance in gear mesh including their location with respect to each other on the housing. However, due to the manufacturing tolerance of the laser cutter with which the gears were built, there was additional spacing between gear teeth in the final product. This did not seem to be an issue as the gears still mated together well. However, when utilizing the gripper, it was noticed that the additional gear space led to increased backlash in the gears reducing the repeatability and accuracy of the mechanism. If this was redesigned, the gear teeth and their pitch diameter would be designed such that the thickness of the laser would be taken into account in the part's GD&T.

To compensate for unknown situations with how the gripper will grip the chip every time, compliance was added to the fingers using foam padding. A thick foam was utilized to have large compliance. However, when testing the robot and gripper, it was discovered that the larger compliance resulted in reduced accuracy when dropping the chip into the Connect 4 slots. For future redesign, smaller foam padding should be utilized

During the first move of the Epson with the end effect, the end effector broke (see Appendix for picture) because it collided with the wall. The tool interface part broke however, the end effector remained unharmed. It was discovered that having a 2 part assembly with the end effector was a good design model as it allowed for a weaker part to sustain the damage and thus allow the more critical pieces and the Epson robot to not undergo damage in case of collision with the environment.

Machine Vision

Once the vision algorithm had been tested for the first time, it was evident that an image from a camera was much different from a static image. Lighting conditions and reflections were causing a challenge. Color thresholding in fact was strictly related to external conditions and needed to be recalibrated every day or even many times per day. A solution to this problem was the tuning of the camera input. Through a preliminary filtering of the input image, it has been possible to obtain a bright and clean image, much less related to the temporary conditions. For future implementation, a specific light source would be utilized to light up the game board, in order to reduce the effect of ambient lighting conditions on the vision system.

Algorithm

During testing of the robot, the algorithm performed with reasonable gaming intelligence. Though this was the case, the robot was not situationally intelligent. As a result, the algorithm implementation was only moderately robust. A situationally intelligent machine would be sufficiently robust to correct itself for error. For example, during testing the chips would sometimes miss the identified column by the AI. In such cases, a situationally aware robot should re-attempt to place a chip in the correct column and proceed only after its success. The general flow of the current program only proceeds once the game state matches that of the AI’s internal representation. In this way, after a chip miss, human intervention is necessary to progress the game. Another way in which the algorithm can become more situationally aware is to keep track of the number of chips used throughout the game and prompt the user to reload the chip dispenser at the appropriate time. A more situationally aware algorithm would also be able to detect if the chip dispenser is depleted and halt any kinematic execution if the board is being modified.

APPENDIX

Contents:

References:

[1] “Connect Four Artificial Intelligence (AI).” Assignment A4: Connect Four AI, www.cs.cornell.edu/courses/cs2110/2014sp/assignments/a4/A4ConnectFour.pdf.

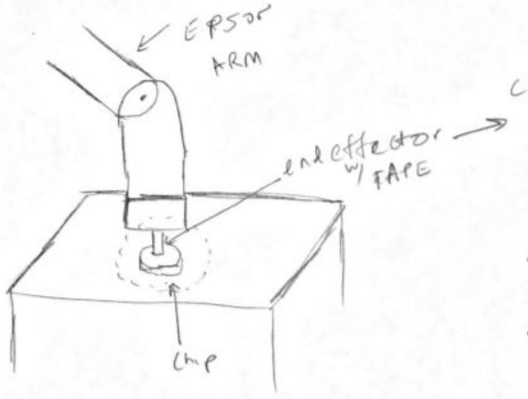
[2] “Time Complexity.” Cs.odu.edu, www.cs.odu.edu/~toida/nerzic/390teched/computability/complexity.html.

[3] Čežek, Dušan, and Nikola Markovic. “How to Build You Own Robot.” Behance, 2011, www.behance.net/gallery/2966457/How-to-build-you-own-robot.

End Effector Brainstorming Sketches Photos

BRAINSTORM

ADHESIVE



close up
mechanical extension of EPSON
TAPE TO CONNECT TO CHIP

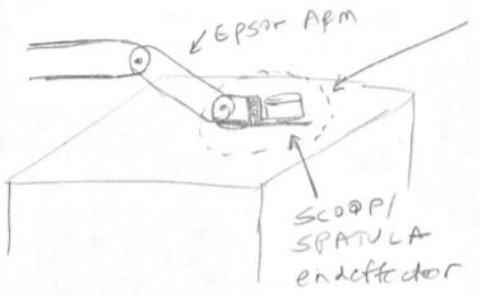
CONS

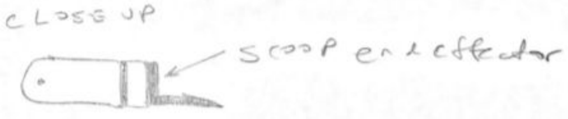
- TAPE WILL LOSE adhesion over plays.
- not repeatable
- imprecise
- will need to use a device to "disconnect" chip from tape to drop the chip

PRO

- easy to implement

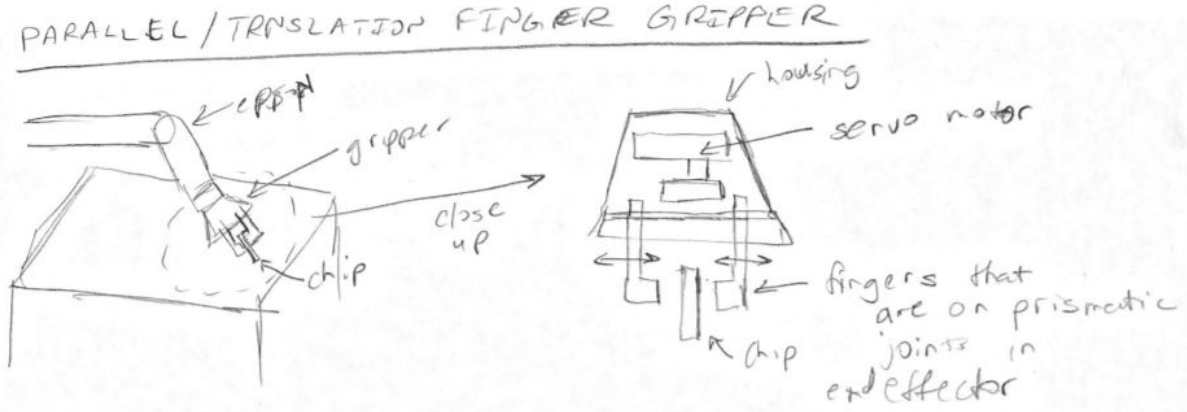
"SPATULA" /SCOOP





- PROS
  - easy to implement
- CONS
  - NOT AS COMPLIANT
  - will need somepiece to provide chip resistance to move when "scooping" piece

PARALLEL / TRANSLATION FINGER GRIPPER



PROS

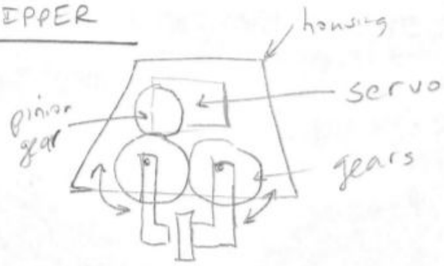
- pure translation motion WRT chip makes dropping chip precise.
- repeatable

CONS

- application vs complexity ration is not good.
- implementation of building gripper is unecessary for chip application.
- active gripper required control from computer

ANGULAR GRIPPER

similar to paraller



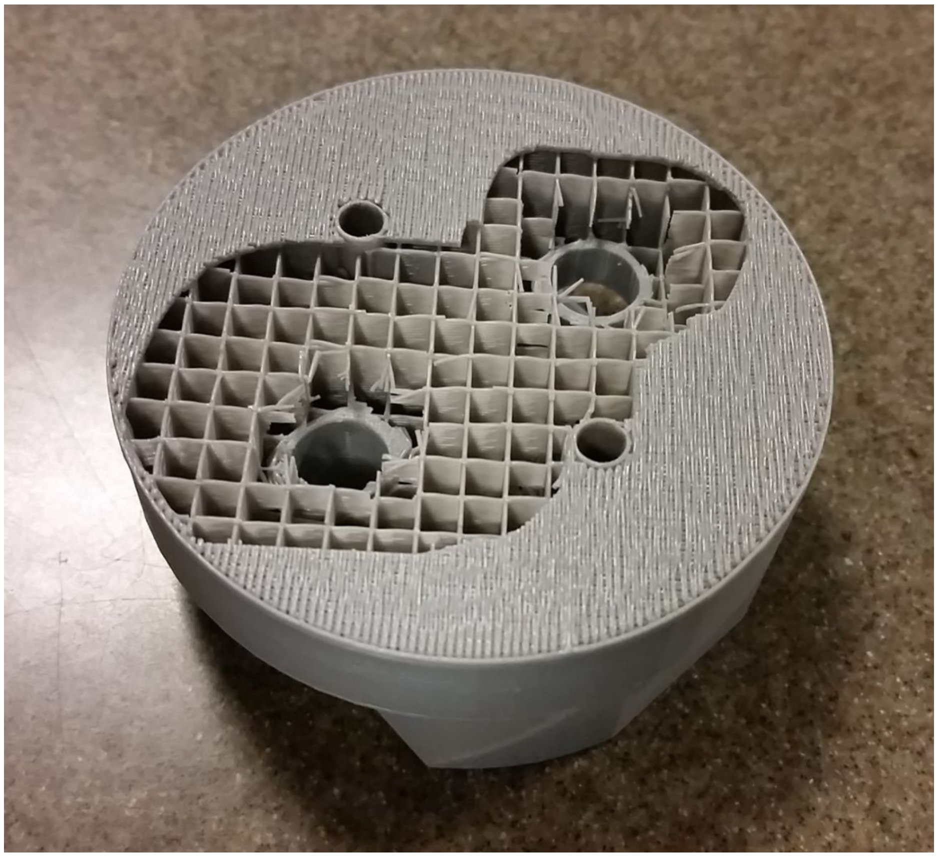
Pros

- precise
- repeatable
- low build complexity vs. parallel design

con

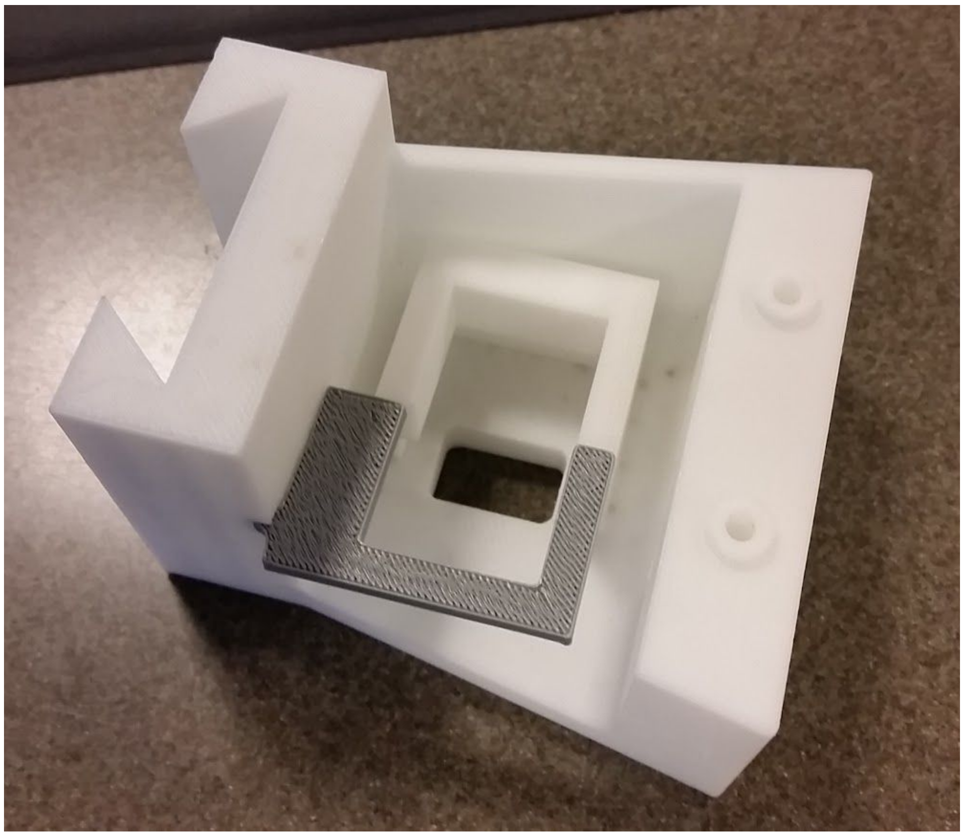
active method will required control from computer to open / close

Broken Tool Interface Part



This piece was damaged during the first move of the epson when the End effector was installed. There was a collision of the end effector with the wall, causing this part to fail at the bolts connecting to the Epson. See Learning Experiences section for details.

Gripper Housing Design, Initial design



This was the initial design for the housing. It was later redesigned and built, once the manufacturing tolerances of the 3-D printer were empirically discovered so that there was better fit with the mating components.

Images for Section 2.5

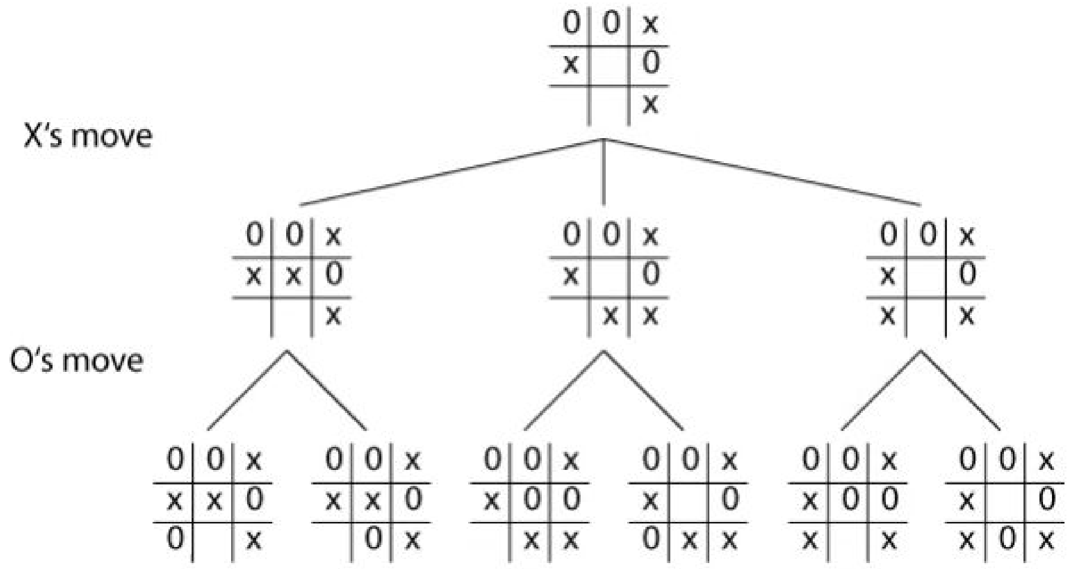


Figure 2.5-1: Minimax Implementation for Tic-Tac-Toe [1]

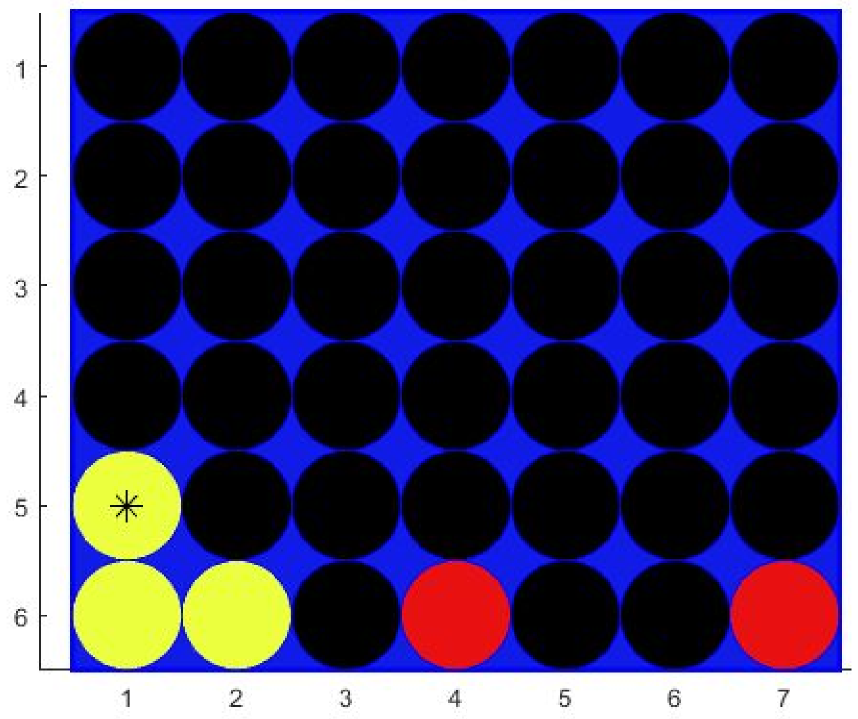


Figure 2.5-2: Optimal Play Position based on testAlgorithm Function

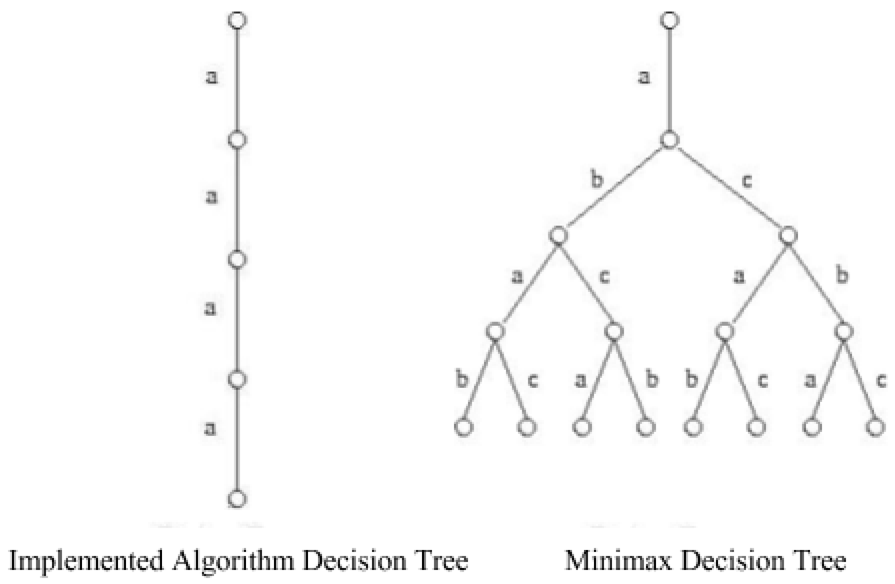


Figure 2.5-3: Decision Tree Comparison

Images for Section 2.6

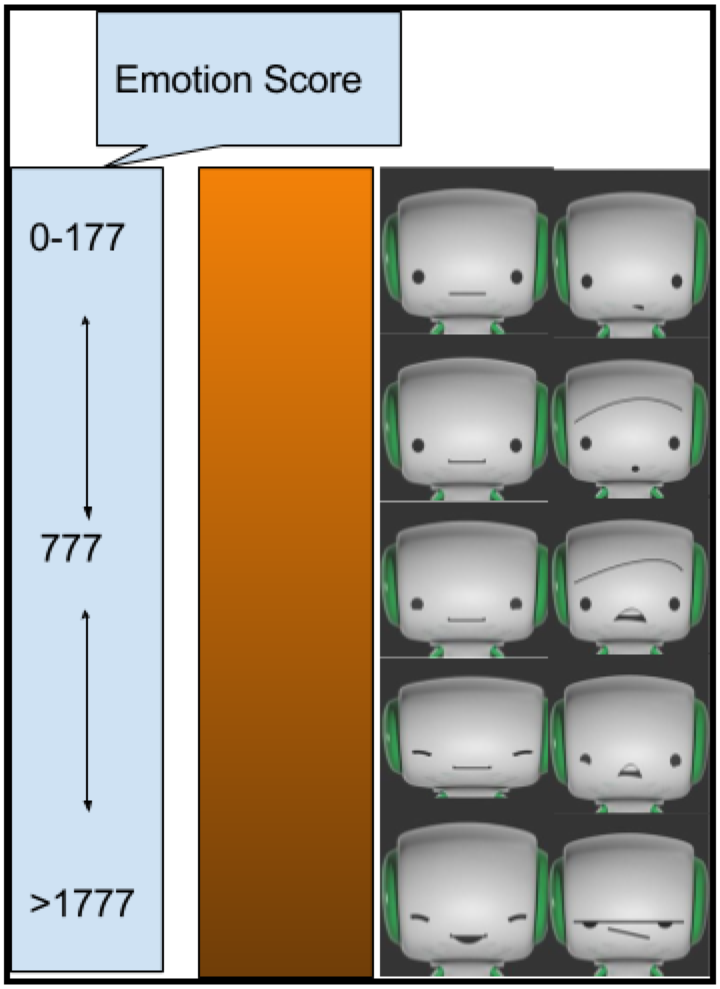


Figure 2.6-1: Emotion Score - Expression Relationship [3]

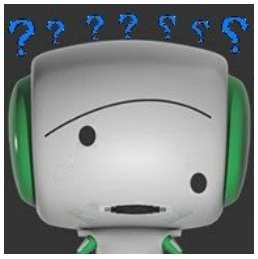


Figure 2.6-2: Bewildered Expression for Invalid Camera Images

Table 2: Emotion Score Conversion Criteria

Play Position Point Evaluation (Maximum # of consecutive chips)	**Emotion Score**
0 or 1	1
2	10
3	100
4 or greater	1000

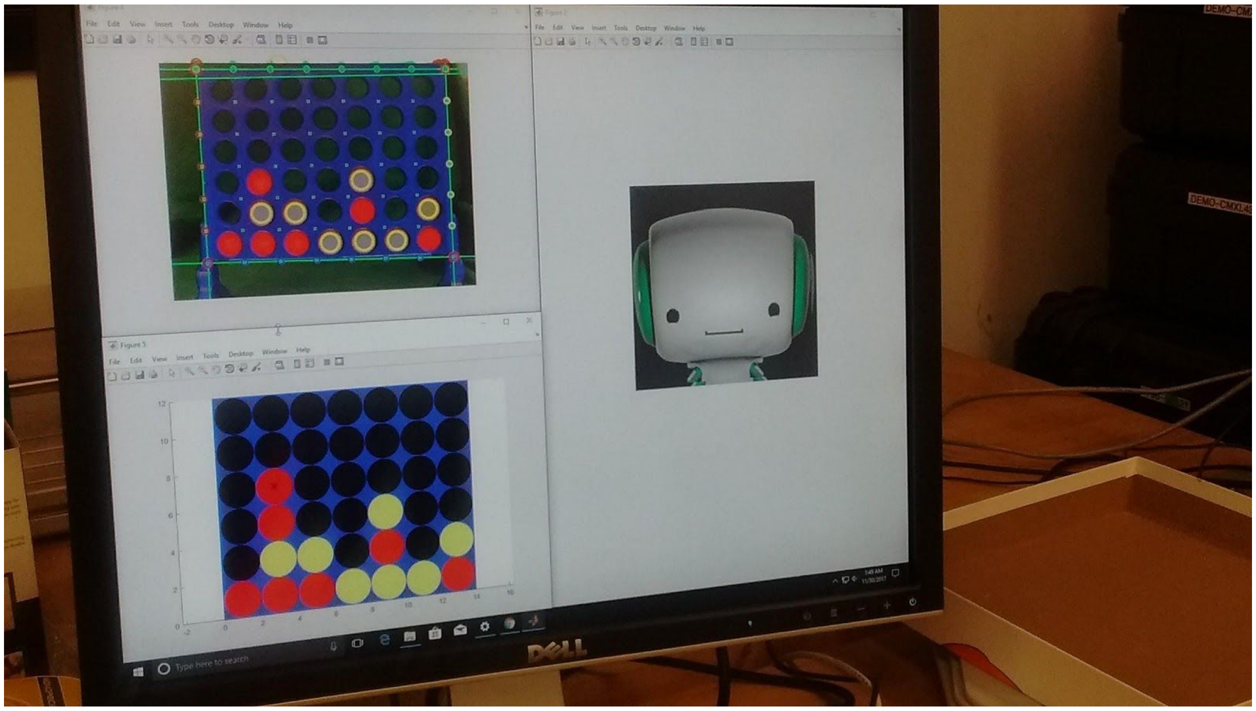


Figure 2.6-1: Computer Interface
(Top Left: Machine Vision Live Preview; Bottom Left: Game Animation ; Right: Facial Expression)

Code:

The code was implemented in Matlab to run the algorithms, control the epson and control the gripper. The code has been explained via discussing the respective algorithm in the Design and Implementation Section.

The full code can be found on the group member's GitHub page at this link:

https://github.com/naudzghebre/connect4Bot