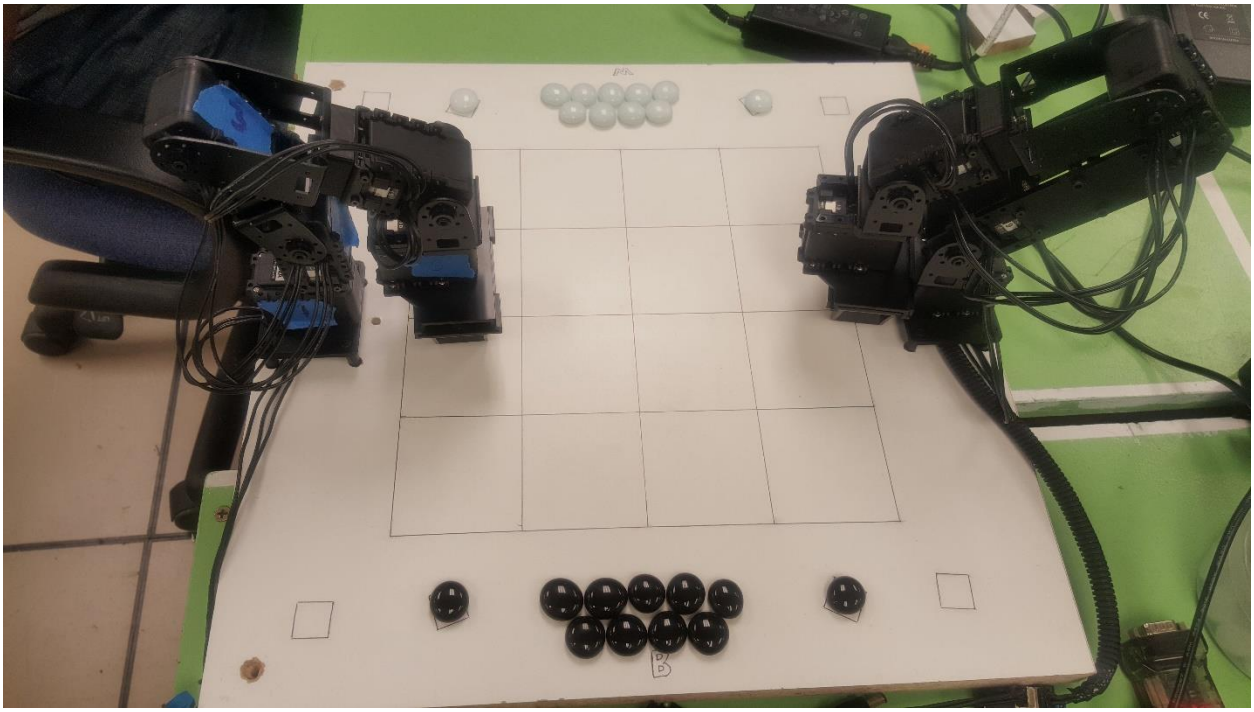
Portland State
UNIVERSITY

Maseeh College of Engineering
and Computer Science

Robotic Arms Play GO Game



By: Mohak Patel

Project for ECE 510

Intelligent Robotics 3

Table of Contents

Introduction:

The goal of the project is to play GO game on real board using robotic arms. Where robotic arm picks and places or remove the stones for human or computer moves.

This project involved the hardware setup for Go game board and two robotics arm to cover the entire workspace of Go game board. The implementation of Go game is done in prolog. Prolog uses the backtracking to find the solutions for players and give the results for captured stones and move for the computer. At the end, prolog will return entire Go game board in form of one list. Python code break that list into the different coordinates to calculate the angle for servos. Each coordinate is taken one at the time and feed to the robotic arms. Which robotic arm do the task will decide based on the coordinate fall into which workspace of the robotic arm. For this project , the board is divided into two half and each half assign to one robotic arms. Based on the coordinate, the robotic arm will pick the stone (black or white according to the player) from outside the board and place that stone at the given coordinate.

The following has been completed:

- Prolog Code implementation for Go game.
- Hardware design for GO game.
- Script for integrating the prolog and python.
- Script for both robotic arms to do different servo movements.

Future Work:

- Use machine learning for computer moves.
- Use OpenCV to scan the board so that human can interact with a robotic arm.
- Design hardware to place one stone at pickup position of the robotic arm.

GO game:

Go is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

The game was invented in China more than 2,500 years ago and is believed to be the oldest board game continuously played today. It was considered one of the four essential arts of the cultured aristocratic Chinese scholars in antiquity. The earliest written reference to the game is generally recognized as the historical annal Zuo Zhuan .

Rules of GO:

A game of Go starts with an empty board. Each player has an effectively unlimited supply of pieces (called stones), one taking the black stones, the other taking white. The main object of the game is to use your stones to form territories by surrounding vacant areas of the board. It is also possible to capture your opponent's stones by completely surrounding them.

Players take turns, placing one of their stones on a vacant point at each turn, with Black playing first. Note that stones are placed on the intersections of the lines rather than in the squares and once

played stones are not moved. However, they may be captured, in which case they are removed from the board, and kept by the capturing player as prisoners.

At the end of the game, the players count one point for each vacant point inside their own territory, and one point for every stone they have captured. The player with the larger total of territory plus prisoners is the winner.

Steps:

1. The board is empty at the onset of the game (unless players agree to place a handicap).
2. Black makes the first move, after which White and Black alternate.
3. A move consists of placing one stone of one's own color on an empty intersection on the board.
4. A player may pass their turn at any time.
5. A stone or solidly connected group of stones of one color is captured and removed from the board when all the intersections directly adjacent to it are occupied by the enemy. (Capture of the enemy takes precedence over self-capture.)
6. No stone may be played so as to recreate a former board position.
7. Two consecutive passes end the game.
8. A player's area consists of all the points the player has either occupied or surrounded.
9. The player with more area wins.

Prolog implantation for Go game:

This prolog code (Attached with email) show the implantation of the 5x5 Go game. Start the game by typing (play.). To enter the position of the move, add the first row number (0,1,2,3,4) to the first column number (0,5,10,15,20).

Example: Here player (x) place at position 22 (row(2)+column(20))

```
Player 1 Move? (x)
|: 22.

    0 1 2 3 4

0   - o - - x
5   o x - - -
10  - o - - -
15  - - - - -
20  - - x - x
```

For starting game:

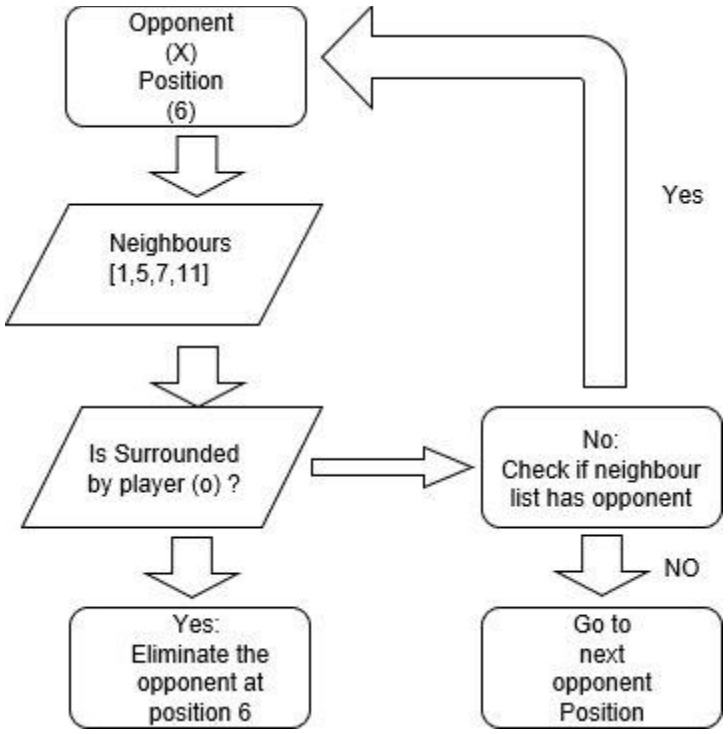
1. Play.
2. Choose single player (Human Vs Computer) or Multiplayer (Human Vs Human)
3. Black Player will start the game.

```
SWI-Prolog (AMD64, Multi-threaded, version 7.7.11)
File  Edit  Settings  Run  Debug  Help
?- play.

**********************
* Prolog GO game *
********5x5 Board************


Single player or Multiplayer? (s or m)
|: s.


Player 1? (o or x)
|: x.


    0 1 2 3 4

0   - - - - -
5   - - - - -
10  - - - - -
15  - - - - -
20  - - - - -

Player 1 Move? (x)
|:
```

Capturing Algorithm:

For capturing the opponents and to detect the captured opponents, I have used tree expansion and searching the opponents/player at nodes of the tree. It uses the prolog backtracking to find all solutions. For example, as shown in below figure, the player (x) is almost surrounded by the player (o). So in the next turn of player (o), it will place at position 7 so that it can eliminate the one stone of player (x) placed at position 6.

Flowchart:



Human Vs Computer (AI):

I have used a same capturing algorithm to predict the computer move by assuming that opponent is almost surrounded by the player and only one move is left to capture the opponent. The first computer will play one prediction move and use capturing algorithm. If prediction move gives the output yes of capturing algorithm then it will be the best move for the computer to play. And if it will give output no then the algorithm will first find the opponent who has minimum neighbors so that computer can capture that opponent by playing minimum moves and then it will play at any neighbor position of that opponent.

For example, in below figure 1, the player played at position (0). So computer played at position (5) to capture the opponent. In figure 2, the player played at position (6). So computer played at position (1) to capture the opponent. Figure 3 shows the result of that

```
Player 1 Move? (x)
|: 0.

    0 1 2 3 4

0   x - - - -
5   - - - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
Player 2 Move? Computer (o)
5

    0 1 2 3 4

0   x - - - -
5   o - - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
```

Figure 1

```
Player 1 Move? (x)
|: 6.

    0 1 2 3 4

0   x - - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
Player 2 Move? Computer (o)
1

    0 1 2 3 4

0   x o - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
```

Figure 2

```
Player 2 Move? Computer (o)
1

    0 1 2 3 4

0   x o - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0

    0 1 2 3 4

0   - o - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :1
Player 1 Move? (x)
|: 0.
```

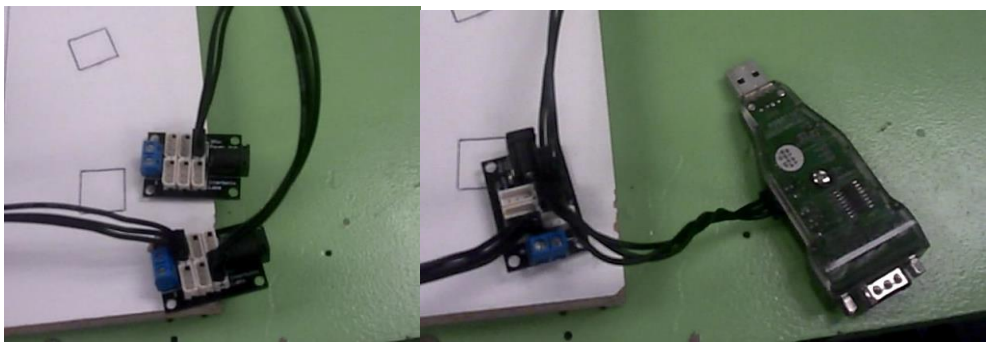
Figure 3

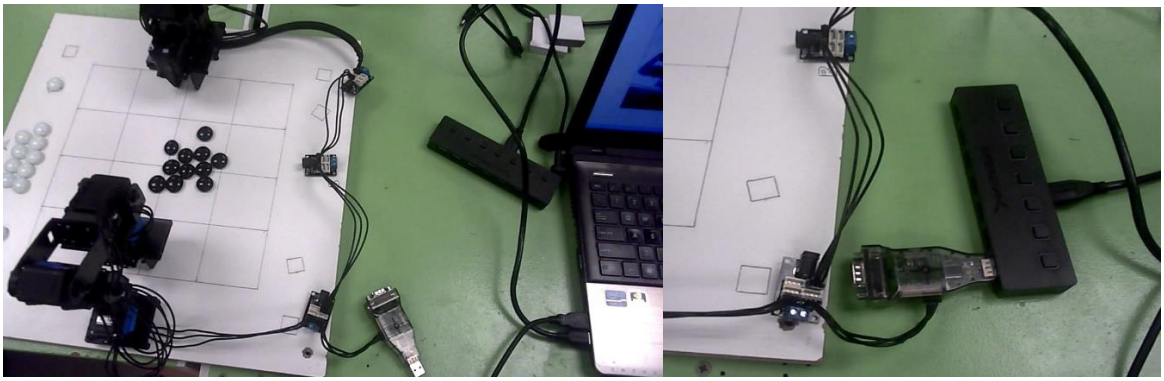
Figure 3 shows the result of scored player. It will be updated at each iteration of player.

Setup for Robotic Arms:

Materials used in pictures:

Robotis USB2Dynamixel Adapter, USB2 Hub, 6 Port AX/MX Power Hub, 12v 10A Power Supply, 10 - Dynamixel AX-12A servos





Python Implementation:

The python script is divided into three sections, each performing a different task while also cooperating with each other. The first section implements connection with prolog file and read the output of the prolog and store that output in one array to determine the coordinate, while the other is strictly for feeding movements into the robot arms.

Description of files:

1. Robotics.py: This file read the prolog output and determine the coordinate based on output and tell servos to go to the particular coordinate.
2. Ir3_base.py: This file is for opening the computer USB port for serial communication with servos, set the speed for the servo, tell the servo to go to particular position.
3. Positions.py: This file includes all the 25 board positions, 4 pickup positions, 4 positions for eliminated stones, 4 initial positions for both robotic arms.

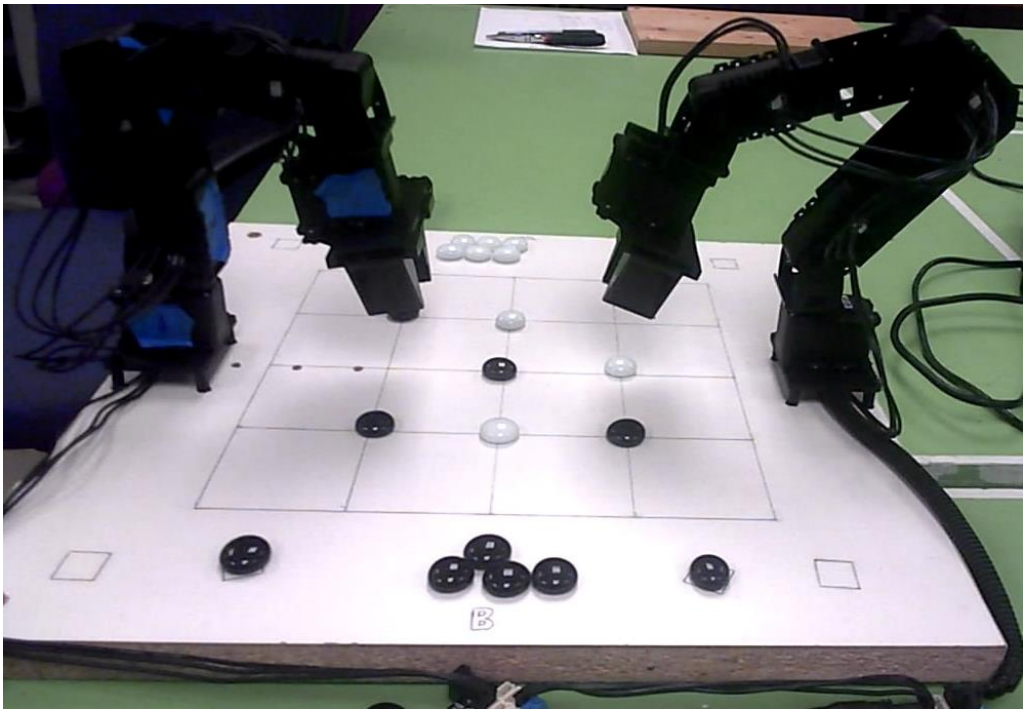
Troubleshooting:

- The main problem to interface the prolog with python. Python 2.7 is compatible with prolog but python 3+ have some bugs that are still not fixed. Because of that prolog code is not giving the expected output. I think it is because pyswip python library has some bugs for windows 10. It is more compatible with linux operating system.
- Prolog is not working properly with python 3. So I have to figure out the other way to interface the prolog and python because the dynamixel AX-12A servos library supports python 3+.
  Solution: we can use file handling in both python and prolog. So I have to write the output of the prolog code to one text file and read that file in python in while loop and then do the task according to the output of prolog.
- One the main advantage of prolog is backtracking but this is 450 lines of prolog code and sometimes prolog is giving unnecessary output so it is very difficult to find the place from where I can stop the backtracking.

- It is very difficult to do smooth movement dynamixel AX-12A servos using their library. I have to find particular speed (for this project 90-130) so that it can do fast and smooth movement.
- It is hard to grab the Go game stones with robotic arm because stones are not identical in size so that it sometimes slips from robotic arm gripper.
- There are 25 board positions, 4 pickup positions, 4 positions for eliminated stones, 4 initial positions for both robotic arms so it is time-consuming find a particular angle for 10 servos.

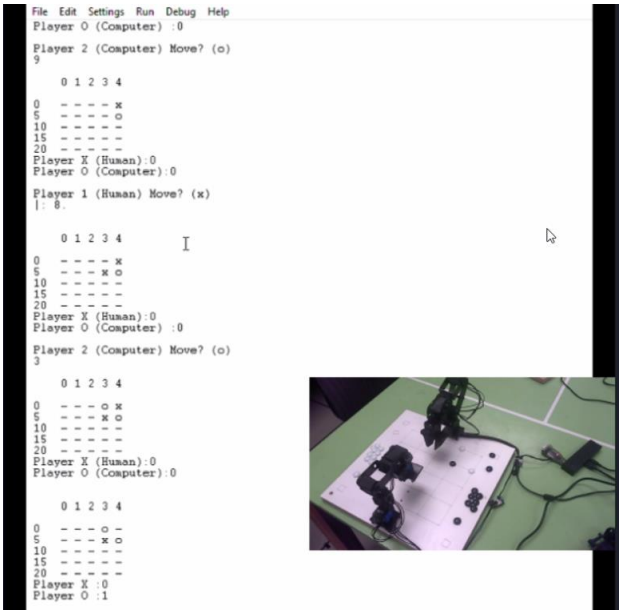
Conclusion and Results:

GO game capturing algorithm is working same as original GO game. The main advantage of prolog is the backtracking. In this go game, there is an infinite number of possible moves and solution so prolog is very useful in this case to find all the solution using backtracking. It is very difficult to do backtracking by any other scripting language (python, C++) that I have used in this prolog GO game. The robotic arms pick and place very precisely at one coordinate. It is difficult to work with both robotic arm but it gives me the good results as expected. Below figure shows that two robotic arms are playing GO game with a human.



Video Description:

In the bottom right-hand side of the video, you will see a recording of what the robot arm is doing in response to what the human or computer moves in the left of the video. The left side is the SWI-prolog console where human play (type) move and in respect to that computer play its move. You can compare the Go game board in SWI-Prolog and original Go game board on which robotic arms play its move.



In SWI-prolog console, the Go game board and score of human and computer is shown.

Video Link: https://drive.google.com/open?id=1hhxofOdt0jlm2T5jnkicdTNMWkr0TuAU

References:

1) www.montefiore.ulg.ac.be/~lens/prolog/tutorials/tictactoe.pl
2) https://senseis.xmp.net/?ComputerGoProgramming
3) https://en.wikipedia.org/wiki/Go_(game)
4) https://github.com/jeremiedecock/pyax12