Man v/s Bot : Software Final Report

Team Members:
Dhanesh Pradhan - A53097299
Yashank Sakhardande - A53090928

Abstract:
Boardgames form part of a multi million industry. Player to player interaction is the essence of increasing demand for board games. However, in the recent years, this industry has lacked innovation and novelty. In this digitally emerging world, technology has led to wide expanse of innovations. However, research[1] shows that it has also led to an alarming increase in the loneliness which in turn has led to increasing introverts and unmotivated human beings. We introduce a novel low-cost, portable system called 'man vs bot' to improve user interaction that can be used between a human and a robotic arm to play a 2 player board game. The assembly involves a convolution of hardware and software where a smart Artificial Intelligence algorithm will have the ability to predict its moves based on the situation on-board and make a move with the help of the robotic arm.

Introduction:
The current age although considered as an age of increasing innovation and technological advancements, it also has a flip side to it; the side that we generally turn a blind eye towards. This age is also known as the “age of loneliness”. Researchers have done a number of experiments that portray that the land of opportunities often turns out to be leading to a land of loneliness. As a result, cities like San Francisco, Dublin, London, Sydney feature in the top 10 cities in WHO’s world loneliness index[2]. With problem comes solutions as we humans strive to fight against our problems. Also, it can be noted that this problem can occur to a small child engrossed most of the times in mobile phones and can range to elders who crave for companionship but no one's around to meet their needs. There have been a number of ways[3] to curb this problem which range from practising physical activities to helping self and others. Many success stories have been written by following such golden steps worldwide. Here, in our project, we propose Man v/s Bot - a multipurpose solution to this global problem that can act as a friend, a companion as well as an entertainer.

System Architecture:

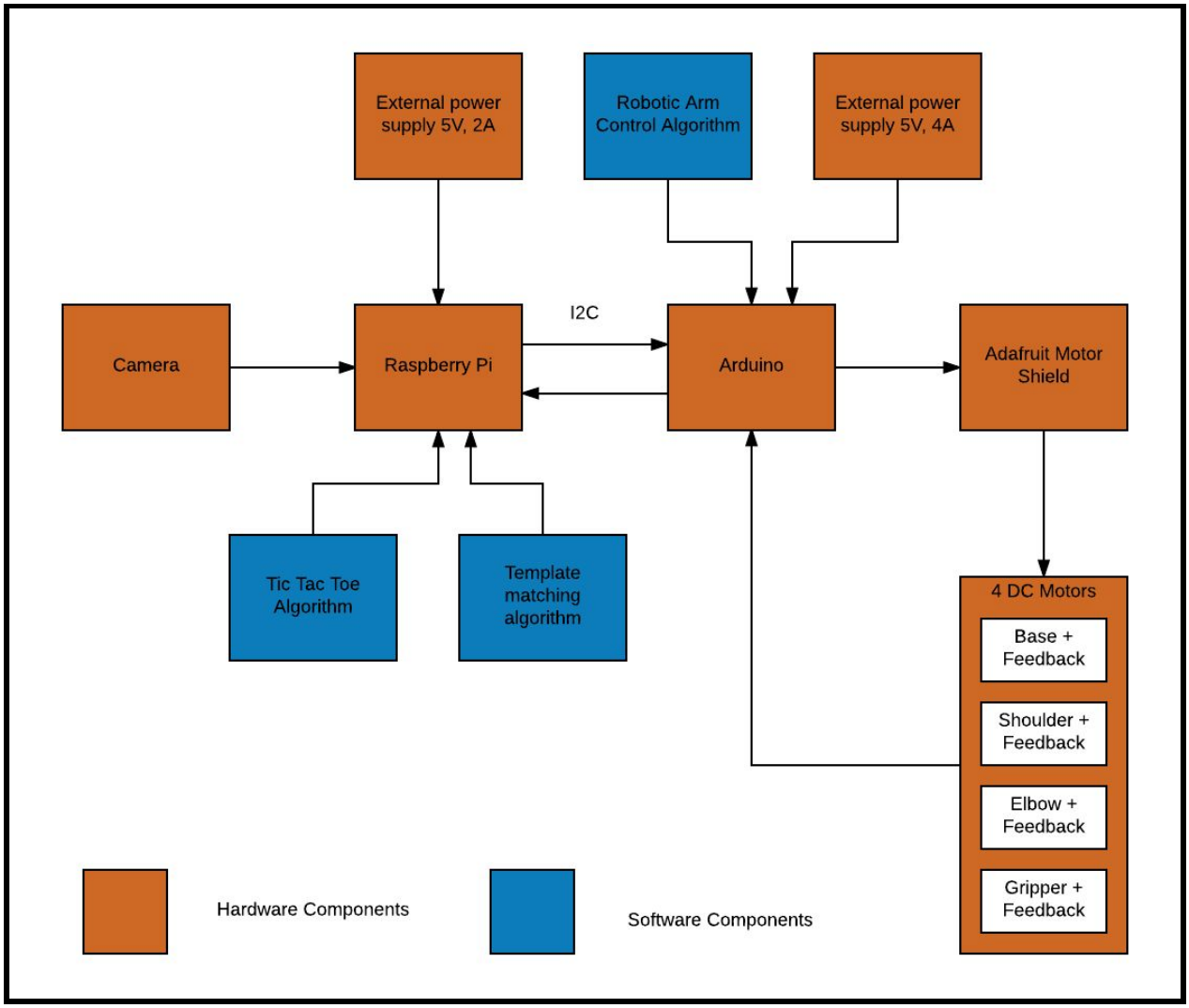


Fig 1. System Block Diagram

Software tasks

Tic-Tac-Toe:

For implementing TicTacToe, we wrote minmax algorithm. The Algorithm looks for all the possible moves that the computer can play and choses the best possible move to go forward. To demonstrate this to user, we have two successful implementation, one displaying the position with letters and other using a GUI. The first implementation analyses the pieces on board and makes a predictive move using the algorithm mentioned above. It is represented as: ‘B’ for Blank, ‘O’ for zero and ‘X’ for Cross. Since, we used Python to implement it, we place the grid positions into a List and display it. For eg. assuming a blank grid, it represents [‘B’,‘B’,‘B’,‘B’,‘B’,‘B’,‘B’,‘B’,‘B’,]. Based on the user’s move it will update the grid and play its

move. Let's say after the user's move it is ['X','B','X','B','O','B','B','B','B',], this means there is an 'X' in position 1 and 3, it will play its move in position 2.
For better realisation, we also implemented a GUI. The GUI mirrors the image of the state of the board. In the GUI, we have provided the user an interface to play the move by clicking on the empty box. After user plays its turn, the GUI has a "Next move" button which should be pushed to proceed with the game. Once the button is pressed, the algorithm runs in the background and the next move is played by the computer. We have also provided the restart button in which case the computer takes a snapshot of the initial stage of the game and proceeds with the next move.

Screenshot of the Tictactoe GUI

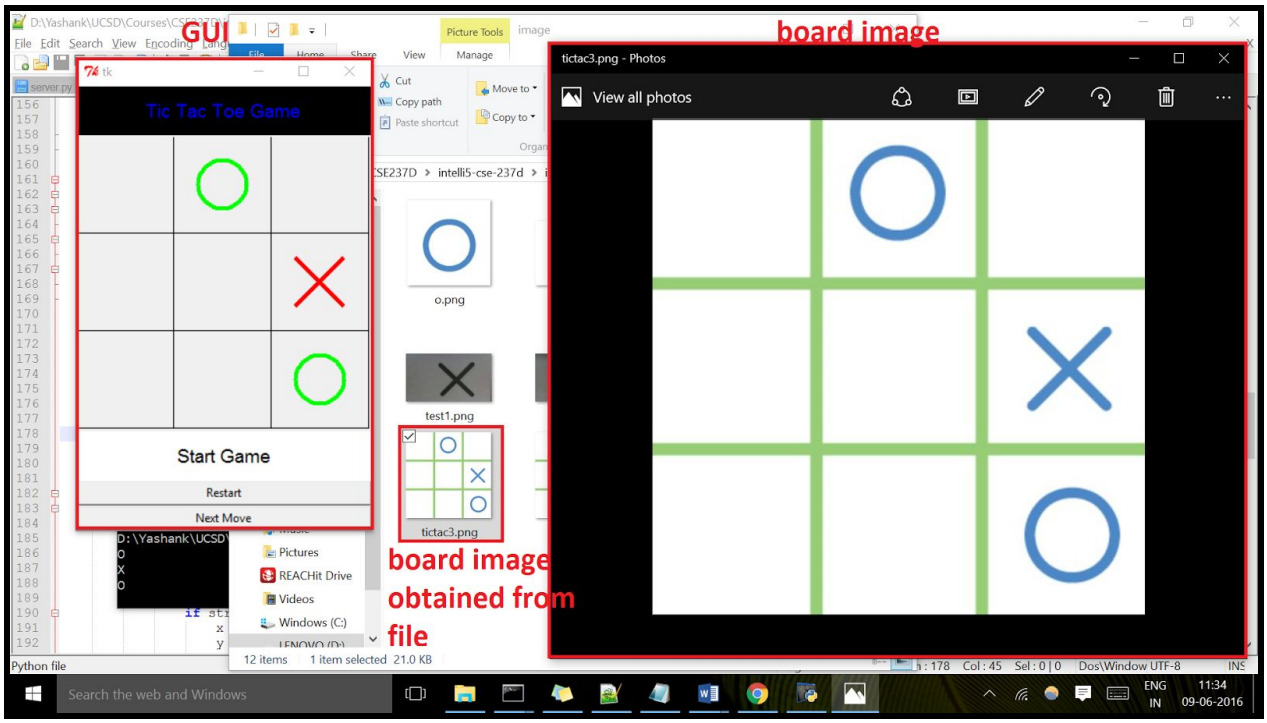
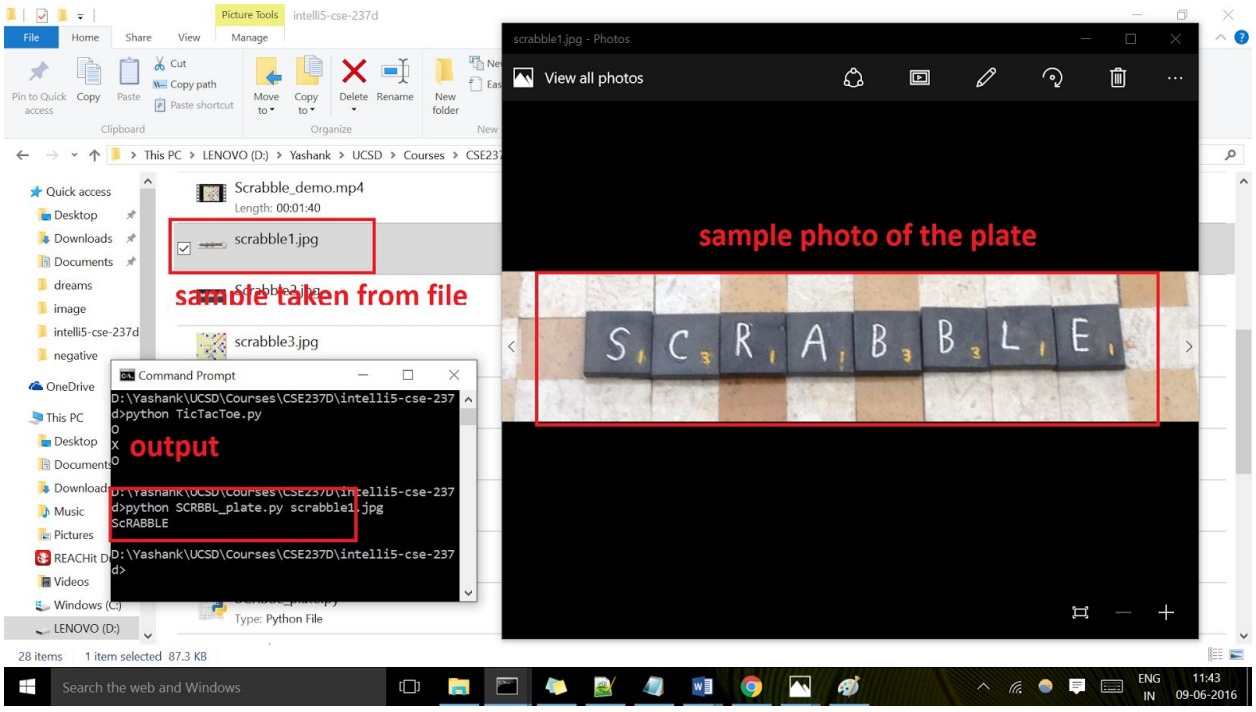


Fig 2. TicTacToe software implementation and visual GUI

Scrabble:
The initial phases of this implementation consisted of searching for efficient image to text algorithms. To start with, we implemented image processing algorithm like template matching to convert an image to text. But this involved lots of interference from the texture of the board. As a result, we implemented this algorithm using Pytesseract. Pytesseract is an efficient image processing algorithm that involves some background computations to convert the text on an image to a string. We tested this algorithm on some sample images to extract the text from a car number plate.
Finally we integrated this feature on existing scrabble images and it could successfully extract the text out of it. Besides, image to text conversion, the next step involved developing an AI for

the scrabble game. We implemented this using python where a set of letters are specified and based on the given letters, we were able to develop meaningful dictionary based words.
After extracting all the possible combinations we check for available slots that needs to be filled and can be filled. Based on that, we can predict the next word that can be attempted. The scrabble AI involves implementing a heuristic approach to predict the best possible word (longest in case of scrabble generally leads to more points). It has been programmed in a heuristic approach where it tries to find a meaningful longest word and drop a letter if it doesn't make sense at all. The same is repeated to achieve better accuracy.
Again to make things interesting, we implemented a smart GUI that can extract the precompiled image and find words out of it. The algorithm traverses from all rows and moves in each column and processes the image to text and presents it to the user in the form of GUI. We did not integrate this software with the hardware as we did not have the robotic arm with proper granularity.
Support for hardware or communication between Raspberry Pi and Arduino wasn't developed in this case as we realised that the arm cannot have that level of granularity.

Screenshot of the Scrabble GUI



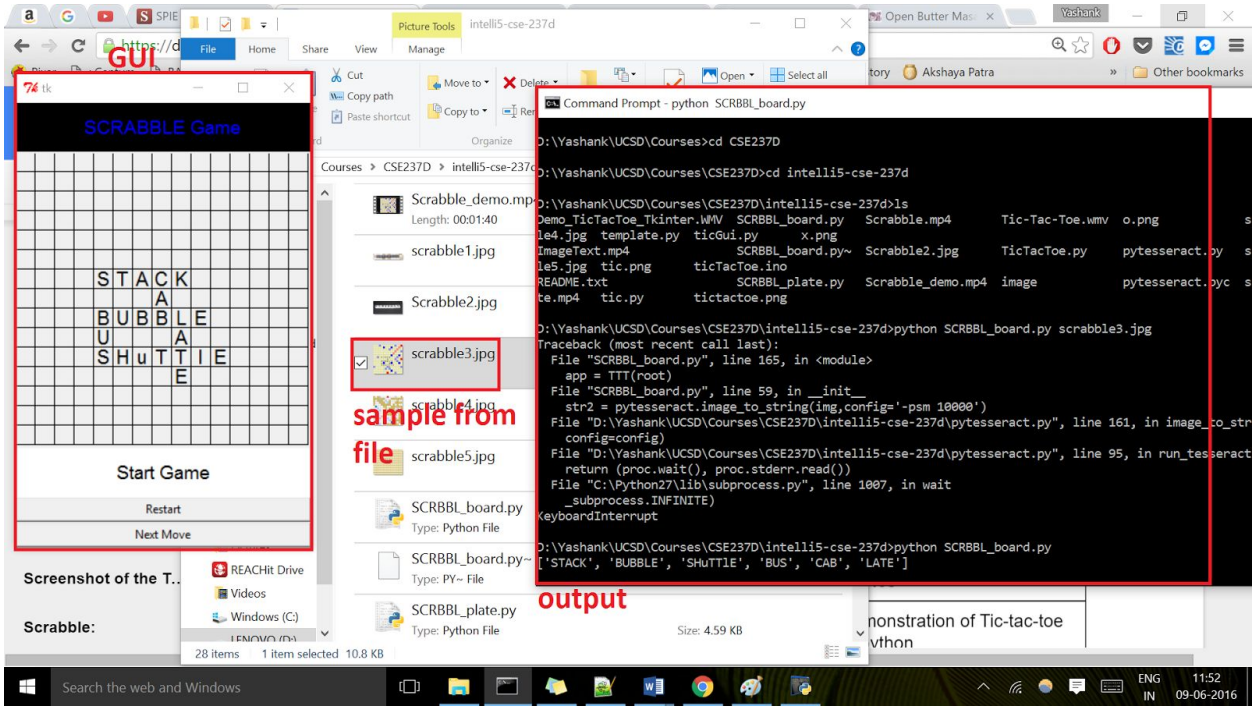


Fig 3. Scrabble implementation using Pytesseract

Hardware Integration:

For the final integration with the hardware team, we had a choice to implement TictacToe with either Python Pytesseract or Template Matching algorithm. Although Python Pytesseract has an exceptional ability to convert image to text on selected high quality images, it failed on real-time implementation when the lighting conditions weren't pretty good and image quality wasn't not high enough. Thus, we had to resort to template matching where the first frame of a blank board will be considered as a reference and then the blocks will be detected. We even had a predefined template of 'X' and 'O' saved that gets checked every time before implementation as shown below.

We also had a calibration mechanism that determines the position of first block and based on that all the grid are located. This helps in locating the position of blocks in the grid and based on their template, the AI will determine the next move.

Fig 4. Sample templates of X and O used for template matching

Implementation Flow/ Project Milestones:

Serial Number	Description	Deliverables
1	Tic-Tac-Toe Game Development	Video demonstration of Tic-tac-toe game in python

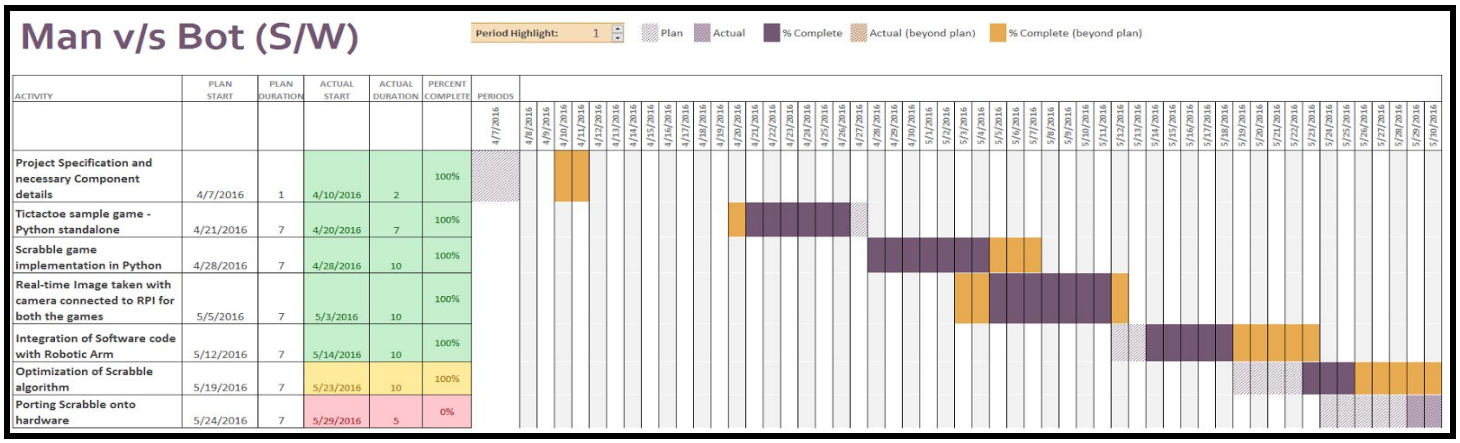
2	Scrabble Game Development	Video demonstration of Scrabble game in python
3	Image to text Conversion	Video demonstration of python code which has input as image and outputs string of text in it
4	Test on Rpi	Video demonstration of camera attached to Rpi implementing Image to text module
5	Integrate software with hardware	Video demonstration of Robotic arm playing the game of Tic-Tac-Toe

For the deliverables, please check out our [Bitbucket Repository](#) .

Technical Challenges:

- Template matching has its own limitations. The template needs to have the same lighting conditions, orientation, camera alignment and the aspect ratio as that of the original image against which it is matched. Also, template matching when implemented on board games like scrabble would slow down the algorithm significantly due to a large number of templates to be matched.
- The Pytesseract library works best when the image contains text in black color with a white background. This is not the case in scrabble and thus the image to text conversion may lead to few errors. This can be possibly removed using template matching similar to that of Tic Tac Toe implementation. However, here the number of templates to be used is 26 (one for each letter) which makes the algorithm slow.

Conclusion & Future Scope:



We were able to achieve the game development algorithm for both TicTacToe and Scrabble. It involved developing one step at a time and improving on the base AI. We could achieve the required results and also portray the same using GUI for better visual effects. Some scope for future work would involve communication with the arduino to coordinate the pick and place algorithm. This would require high amount of precision to enable movement of robotic arm efficiently to pick a small tile and again place it back. Also our current template matching algorithm implies that blocks are placed in a horizontal position. However, our arm didn't have the support to provide a rotatory motion to place in exactly that fashion. Better robotic arms with higher degrees of freedom can help tackle this issue.

References:

[1]http://www.independent.co.uk/life-style/health-and-families/features/the-loneliness-epidemic-more-connected-than-ever-but-feeling-more-alone-10143206.html
[2]http://www.theregister.co.uk/2006/05/15/loneliness_index/
[3]http://www.huffingtonpost.com/margaret-paul-phd/7-ways-to-avoid-lonelines_b_4999225.html
[4]http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html
[5]https://pypi.python.org/pypi/pytesseract/0.1