AUGMENTED REALITY TIC-TAC-TOE

Mitch Larva – mlarva2@uic.edu
Shreyas Pattabiraman – spattt2@uic.edu

ABSTRACT

The project is a simple game of Tic-Tac-Toe in Augmented Reality. The game is played using a piece of paper and a front facing camera attached to a computer. The player plays the first move on the piece of paper and the computer plays its move on the screen after detecting the player's move. The algorithm implements these steps successfully.

INTRODUCTION

Augmented reality has been a prominent field in recent years. The concept of Augmented Reality per Wikipedia, is a live direct or indirect view of a physical, real-world environment whose elements are augmented (or supplemented) by computer-generated sensory input such as sound, video, graphics or GPS data.

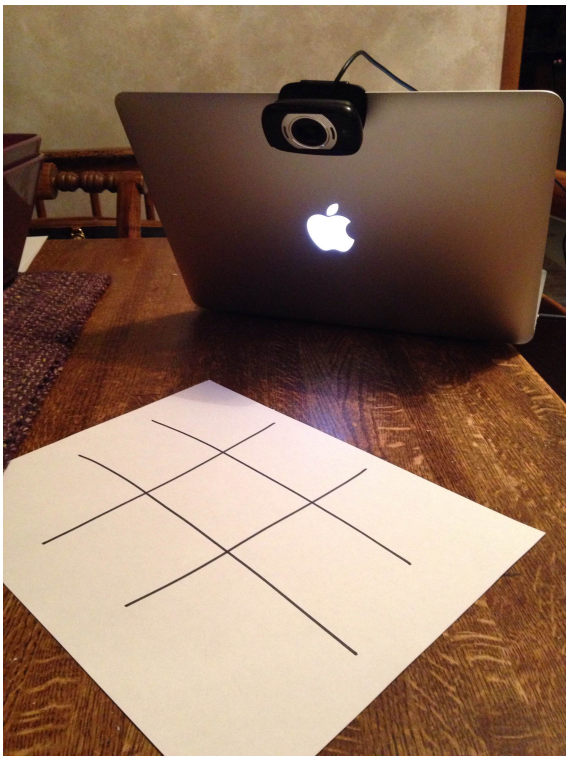
Augmented Reality is a growing field which is being implemented in various domains especially in the field of gaming and other forms of media. Thus, its ability to provide use to a wide range of fields will lead to its popularity in various fields.

In this project, we aim in implementing the game of Tic-Tac-Toe, in Augmented Reality. We voluntary chose an easy game as it would give us the opportunity to concentrate on the Computer Vision part of it rather than the algorithm for the game or the graphics part of it.

We decided that MATLAB is the best form of implementing this project as it has a good set of computer vision and image processing libraries.

SETUP

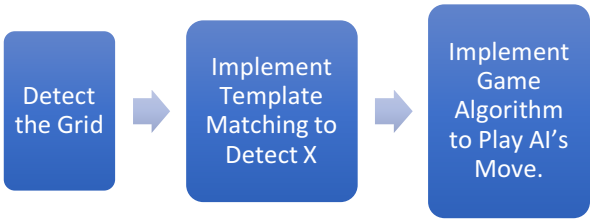
A webcam, a PC, a piece of paper and markers of any variety.



(i) Figure A

Fig. A: setup of our game. The camera is set up in such a way that the grid is visible completely to the camera. The setup helped us to calibrate the camera just to focus the grid completely.

ALGORITHM



1. Detecting the Grid

A snapshot of the move is taken by the camera after being prompted by the player. The image is then converted into grayscale. The grayscaled image is then sharpened and the edges of the sharpened image are detected utilizing the Sobel operator. This process converts the image into black and white.

a) **Hough peaks and Hough lines**: Once edge detection is complete, the image is opened. Opening of an image is done by dilating and then eroding an image. To determine the grid, the houghpeaks and houghlines functions are used on the opened image. The Hough lines are then plotted, 'retracing' the grid lines of the original image.

b) **X and Y Intercept**: Once the grid lines are detected, the next step is to determine the intersection points of the 4 lines. To do this, we use the algebraic formulae to detect the X and Y intercepts of the Hough lines as well as their slopes. A key point to note is that while determining the four obvious intersection points of the grid, we had to keep in mind that the lines are not necessarily parallel and our algorithm ends up detecting six intercepts. We incorporated a process in our algorithm to get rid of those two extreme points.

2. TEMPLATE MATCHING

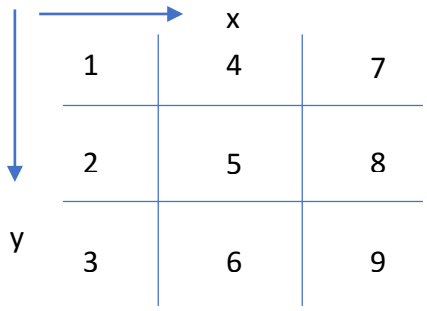
The template matching function we used is a CPU efficient function which calculates matching score images between template and a 2D image. It calculates:

1) The sum of squared differences (SSD Block Matching), robust template matching.
2) The normalized cross correlation(NCC), is independent of illumination, and is only dependent on texture.

Both matching methods are implemented using FFT based correlation.
In our case, the template image is Fig. 2.

Figure 2

The input of this function is a snapshot from the camera in the double format. This is run through the functions along with the template image and gives the x and y coordinates, i.e the location of the X in the input image. The coordinates are marked with the red spot in the Fig. 3. The board is split into 9 quadrants and we defined that the quadrants are numbered 1 through 9 like below.



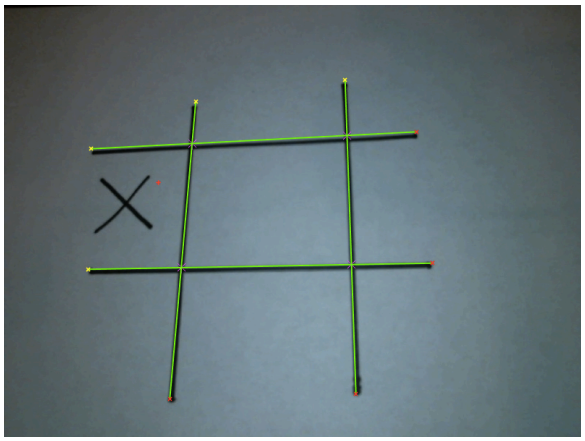


Fig. 3

3. IMPLEMENT GAME ALGORITHM TO PLAY AI'S MOVE

We first initialized a 3x3 board state matrix, consisting entirely of the value -1. After determining the location of the player's move, the board state was updated corresponding to the position of the grid. The quadrant was given a value of 1. Our game algorithm then decided where the AI would move given the remaining open spaces. After the first move was made, the value of state becomes 0 at that position.

One problem we faced was when the second move was made. When we tried running the template matching function for two outputs, however, the function showed only the better matching X.

To prevent this from happening, we decided to implement our own method of 'whitening' that quadrant.

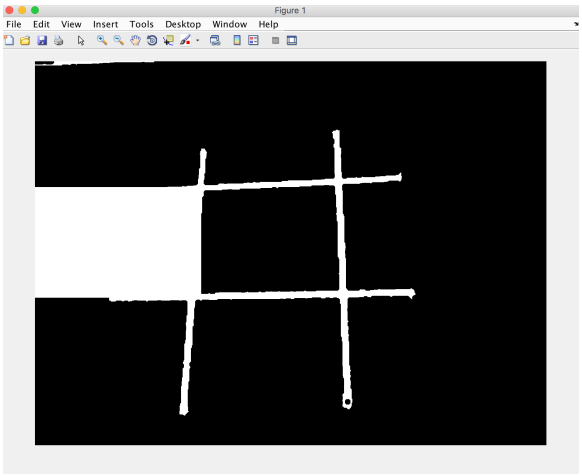


Fig. 4

Figure 4 shows the quadrant in which X is present whitened out such that the next X being played by the player will be guaranteed to be the only X detected by the template matching function.

4. RESULTS

We determined that the camera angle should range from 60 to 120 degrees, if the table is flat. The best results correlate when the camera angle is right above the paper at 90 degrees. With 90 degrees, the success rate of detecting the X was 90%. One other observation we made was that X need not be black and the paper need not be white, any set of colors which are contrasting to each other worked. The distance between the camera and the paper was not a constraint as long as there aren't any detectable lines other than the grid.

The success rate for detection of the Hough lines in the image with our coded parameters and optimal camera angle and distance was determined to be 90%.

5. Future Work

In the future we hope to make our algorithm more robust. We would like to detect the grid at any orientation and be able to change the orientation throughout the gameplay. We would also like to implement a more intelligent AI.
Finally, we would want to be able to determine the player's move through a live video, instead of prompting the player.

6. References

1) Dieter Schmalstieg, Tobias Hollerer. Trends in mobile Augmented Reality. 2012 IEEE Conference 12 April 2012.

2) Fast/Robust Template Matching (https://www.mathworks.com/matlabcentral/fileexchange/24925-fast-robust-template-matching)

3) MATLAB WEBSITE www.mathworks.com