Augmented Reality Tic-Tac-Toe

Joe Maguire, David Saltzman
Department of Electrical Engineering
jmaguire@stanford.edu, dsaltz@stanford.edu

***Abstract*: This project implements an augmented reality version of Tic-Tac-Toe. In this game, the user draws an X on a paper board. Using a webcam and digital image processing techniques, the computer then determines the board state. Using this information, the computer chooses a move and displays the move in the appropriate place on the screen. Our finalized algorithm successfully completes these steps with high repeatability.**

Introduction

As mobile processing power increases, mobile handsets become more able to carry out standard image processing techniques. This enables smartphones to carry out image processing tasks formally prohibitive on a mobile platform.

One of the tasks that has gained prominence in recent years on mobile platforms is augmented reality. Augmented reality seeks to project details onto images captured by the handset. This ability has tremendous potential in a wide range of fields from advertising to gaming. The vast applicability of augmented reality will surely increase its popularity as it becomes more feasible on mobile platforms (Carmigniani).

Our project sought to delve into the technical details of augmented reality to develop our own system using image processing building blocks.

For the purpose of a demonstration, we sought to implement a basic game, namely, Tic-Tac-Toe. We choose a simple game so our main efforts would focus on the image processing components, not the graphical components or the AI agent for the computer player.

We also opted to implement the system on a computer rather than on a mobile platform. This decision was, once again, motivated by our desire to focus on the image processing not the issues of using a mobile framework and testing on a mobile platform. Our decision also does not preclude the possibility of porting our code to a mobile platform. This is because we use OpenCV, a software package with Android support.

Setup

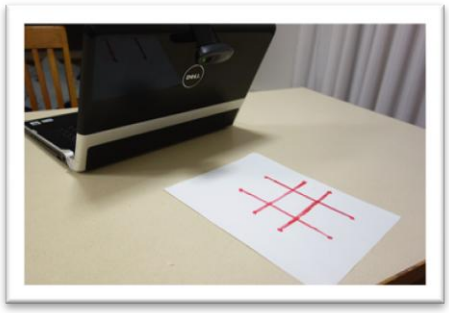


Figure 1

Our setup, shown in Figure 1 uses a computer, a webcam, a piece of paper, and a marker. A tic-tac-toe grid is drawn on the paper, and ticks on the end of each of the grid line are added to help track features. The webcam then starts oriented the same way as the grid, but can be moved and rotated freely after initialization. The user then draws X's in the spaces on the paper, and the computer processes the image, generates a move, and displays it on the screen in the appropriate spot on the board.

Algorithm

Our Algorithm can be broken into three parts:

1. Acquiring the homography
2. Detecting if the player moved
3. Updating the projected board

Part I: Acquire Homography

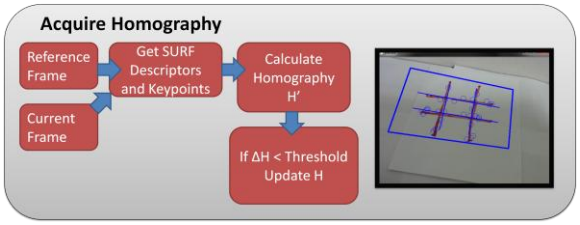


Figure 2

Our algorithm first acquires a reference frame of the board. This is done by simply taking a photo of the board with a webcam attached to the computer. If necessary, the built in webcam can be used.

After the reference frame is captured, our program begins to capture live frames of the board. In Figure 2, the live frame is represented as *Current Frame*. As shown by the block diagram, our algorithm computes SURF keypoints and descriptors for both the *Current Frame* and *Reference Frame* and then uses those keypoints to calculate a homography (Bay).

We originally designed a line homography algorithm under the direction of Dr. Roland Angst from the EE department to take advantage of the fact that the board is formed by four lines. This algorithm worked under ideal conditions but ultimately proved too susceptible to noise. Specifically, as the camera's orientation changed and as the camera moved away from the board, our algorithm's implementation of Hough-Lines detected extraneous lines that threw the homography off. This led us to settle on the more robust but computationally expensive SURF detector to calculate keypoints for the homography.

The SURF detector, used to calculate SURF keypoints as well as descriptors, is a robust scale and rotation invariant keypoint detector. Its principle limitation lies in the inherent nature of the Tic-Tac-Toe board: the board is composed of edges, something detectors are designed to ignore. However, boards drawn with markers proved feature rich enough to be compatible with a SURF implementation.

To discard false matches, we process the matched keypoints with a standard implementation of RANSAC in OpenCV. RANSAC, RANdom SAmple Consensus, iteratively computes a model matching two sets of points only considering the inliers. This is done with a threshold of 6 pixels to establish whether rectified points and matched points are inliers or not.

After calculating the homography, we compare the homography matrix to the homography matrix used in the last iteration. If the L2-Norm of the difference between these two matrices exceeds an established threshold, the new homography is rejected and the old homography is used instead. The threshold is established by sampling a user defined number of frames. During this time, the webcam is assumed to be moving in a representative fashion of during the game. Without this step, the homography is sometimes prone to changing even when the camera orientation remains the same or when an object like a hand blocks the significant portions of the board.

The homography is passed on to the later sections of the algorithm to detect player moves and update the projected board.

Part II: Detecting if the player moved

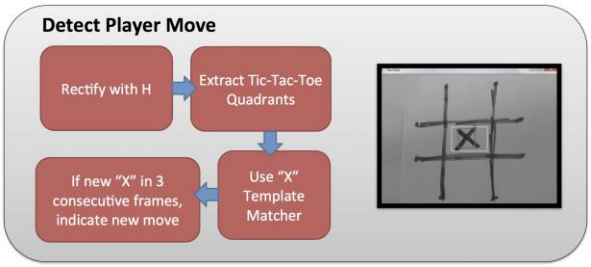


Figure 3

A rectified image is produced by applying the inverse of the calculated homography to the original image. A sub-image for each of the nine sectors of the board is extracted; because the rectified image has known size and orthogonal lines, this can be done by taking a fixed range in the rectified image. For each sub-image, it is then calculated whether or not an X has been drawn in that sector.

To detect X's, first template matching is performed on the sub-image using OpenCV's matchTemplate function. An image created by superposing an image of an X, one of the X rotated 7 degrees clockwise, and one of the X rotated 7 degrees counterclockwise (Figure 4) is used as the template. Of the methods OpenCV's template matcher can use, the best for this application is the correlation coefficient method, which computes the correlation

Figure 4

between the mean-subtracted template and the mean-subtracted sector image. If the normalized correlation is above a threshold of 0.4, then it is decided that it looks like there is an X there. However, there is some random noise causing sectors to occasionally spontaneously show X's, so this noise is filtered out by requiring X's to be present in any sector for three consecutive frames before it is counted. Whenever an X is detected this way in a sector that had been empty, it is treated as a new player move, and the board is passed on to the AI.

Updating the projected board

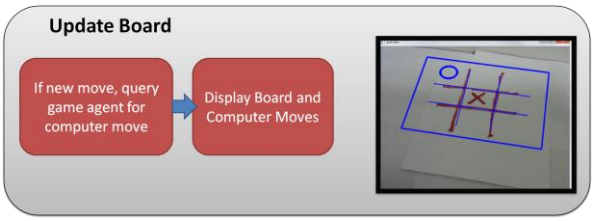


Figure 5

At the conclusion of the second part of the algorithm, the code passes on information regarding whether a new move was detected and, if so, where. The final part of the algorithm uses this move information and a Tic-Tac-Toe game player to calculate the computer's move. The general outline is shown in Figure 5.

The Tic-Tac-Toe game player is largely beyond the scope of this class so it will be described briefly. This player represents the game of Tic-Tac-Toe as a state containing the current board and the current role (X or O). It is accompanied by a MinMax game agent to computer moves.

A MinMax game agent, where the computer is defined to be the agent, assumes that the human player, the opponent, will choose moves that minimize the score while the computer will choose moves to maximize the score (this is the 'min max assumption'). Since the computer is always the agent and the human is always the opponent, all scores are defined as relative to the computer (+100 for computer win, -100 for human win). The MinMax game agent starts by generating a tree whose first nodes are possible computer moves. At the next level of the tree, the human chooses a move. The tree terminates when an end state is reached. The values are percolated up according to the min-max assumption (that the path taken is one where the computer picks the maximal reward moves and the player chooses the minimal reward moves). Since the game space of Tic-Tac-Toe is small, the optimal move for the computer, the one with the highest reward, is always the globally optimal move.[1]

Once the game player calculates the computer's move it returns the updated piece locations. A standard circle is drawn at each place where the computer has moved. Then, the homography is used to project these circles onto the current captured webcam frame. The circles and rectified game board are then drawn over the current frame which is displayed in real-time to the user.

Results

The board was consistently detected at rates of 5-10 fps when the distance between the camera and the board was less than 3ft. As the board moved within that distance, the rates dropped to 2-5 fps but detection remained consistent. Further than 3feet away, the homography encountered intermittent errors.

Our detection of X's also proved robust. Any X matching Figure 4 was detected with near 100% precision while the board was within 3 feet of the camera. At further distances, false positives began to occur at a rate of 1 per 3 minutes. This occurs when the homography briefly errs leading to false X like shapes.

Observations/Issues

Successfully implementing an augmented reality system introduced us to problems unforeseen at the onset of the project. The principle problem came from using a reference image composed of edges. This mistake occurred because we designed our project before covering Keypoint Detection in

[1] A more in depth description can be found: http://arrogant.stanford.edu/ggp/chapters/chapter_06.html.

lecture. The mistake showed us what is now obvious; relying on certain reference images for augmented reality presents more difficulty then other reference images. This explains why many commercial systems like games or advertisements use known, feature rich templates for their augmented reality implementations. The card shown in Figure 6, used for Nintendo's augmented reality system exemplifies this approach.

Figure 6

Thankfully, many arbitrary references, like paintings and structures, are also feature rich. So, in the end, our difficulties were most likely not a general concern in augmented reality but rather a consequence of a uniquely poor reference image.

Future Work

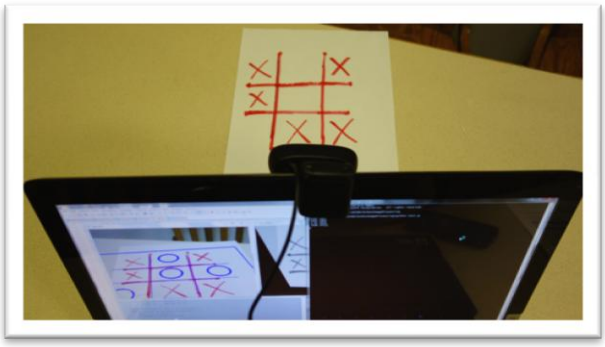
As we worked with and researched more Image Processing techniques we discovered opportunities for improvement. One of the most promising was recent advances in keypoint detection. Algorithms like FREAK and BRISK offer performance increases in speed ranging from several factors to an order of magnitude (Leutenegger and Alahi). However, neither of these algorithms is offered in OpenCV's current python wrapper. Once they are offered, it would be fascinating to compare detection times and accuracies with the standard SURF detector.

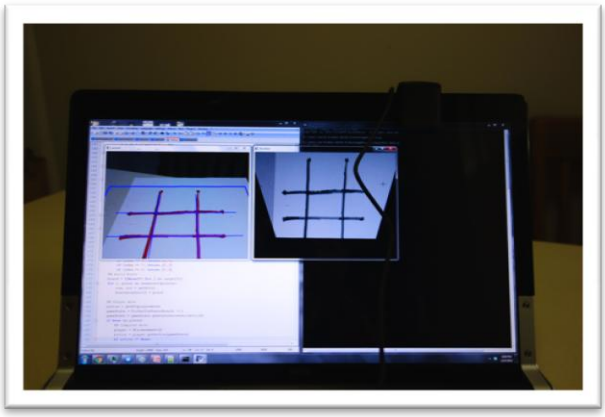
Continuing the idea of augmented reality playing an ever more important role in mobile applications, could port our system on to a mobile platform. An immediately obvious issue is that certain detectors may or may not be available in the OpenCV android implementation.

Conclusion

Our project successfully demonstrated an interactive augmented reality system. Due to the generalized approach, it can be easily adapted to different augmented reality tasks as well as different platforms like Android. Also, due to the modularity of our design, we can easily insert a faster detection algorithm when the python implementation comes online. This, coupled with a better target image would make our system viable for real world tasks like mobile gaming.

Media





Video Link:
http://www.youtube.com/watch?v=whxE7URhJAI

Acknowledgements

We owe a tremendous amount to the Course Staff of EE 368 as well as our advisor, Dr. Roland Angst. They provided constant guidance and recommendations. Thank you

Appendix

Work breakdown:

Joe: Algorithm Part I (Acquiring Homography) and AI related aspects (Tic Tac Toe game state and min-max player)

David: Algorithm Part II (Detecting moves and sending them to the game agent)

Both: Algorithm Part III (Displaying the board) and extensive debugging

References

Alahi, A.; Ortiz, R.; Vandergheynst, P. *FREAK: Fast Retina Keypoint*. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference, vol., no., pp.510,517, 16-21 June 2012. infoscience.epfl.ch/record/175537/files/2069.pdf

Bay Herbert, Tinne Tuytelaars, and Luc Van Gool. *SURF: Speeded Up Robust Features.* Computer Vision – ECCV 2006. pp 404-417. http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

Carmigniani, Julie, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, Misa Ivkovic. *Augmented reality technologies, systems and applications.* Multimedia Tools and Applications January 2011, Volume 51, Issue 1, pp 341-377

Leutenegger, S.; Chli, M.; Siegwart, R.Y. *BRISK: Binary Robust invariant scalable keypoints.* Computer Vision (ICCV), 2011 IEEE International Conference on , vol., no., pp.2548,2555, 6-13 Nov. 2011. http://www.asl.ethz.ch/people/lestefan/personal/iccv2011.pdf

The EE368 Lecture Notes and reference Matlab code

Code Resources:

OpenCV library; http://code.opencv.org.

Numpy library: http://www.numpy.org/