Augmented Reality Tic-Tac-Toe with Raspberry Pi 3.0

1- Summary

The original idea for this board game was create a 3D Augmented Reality Tic-Tac-Toe (ARTTT) in a Video See-Through (VST) Head-Mounted Display (HMD) using two Raspberry Pi 3.0s to capture and render each eye separately to increase performance. However, Pi underwhelming power and low frame rates, and project timeline constraints resulted in a stripped down version of the game.

Currently, the game only supports one Raspberry Pi device and can render and play tic-tac-toe board game in 3D at an average of 5 frames per second. The board is a marker board from ARUCO, and can be placed on wall, desk, or any other flat surface. The game keeps score and renders “X” and “O” objects, as well as the board in 3D.

2- Key Algorithms

The game has been written with C++ with use of public open source libraries like OpenCV and ARUCO. The tic-tac-toe engine uses MinMax algorithm that is borrowed Matthew Steel work from 2009 and modified to fit this ARTTT game.

i- OpenCV 3.0 Capture and Calibration

These algorithms and API function calls are the input part of this application. OpenCV captures a frame from the camera, and passes it to ARUCO engine to decode the markers and board position.
OpenCV also has been used to calibrate the camera, and generate camera parameters matrix needed for real-time ARUCO markers detection. ARUCO requires a camera matrix to be generated using a checkerboard or other calibration patterns.

ii- **ARUCO 1.3.0**

ARUCO is an easy to use marker generation and detection open source package for OpenCV that supports marker boards. Marker boards consist several markers position in an array laid on the board, and they are more robust than single markers because if several markers are occluded, the other markers can estimate the board position and orientation. ARUCO outputs both position of the markers with respect to camera, and also Rotation and Translation vectors to project 3D points back into camera space.

![ARUCO Markers](image)

After markers and ARTTT board is detected, above picture is the actual board, the 3D objects are projected back into the camera space and rendered via OpenCV built-in functions like fillPoly(), line(), and putText().

iii- **Tic-Tac-Toe Engine**

After designing simple data structures to keep the game state and score, the Artificial Intelligence engine is borrowed 2009’s Matthew Steel MinMax work. In this algorithm, the AI performs a full exhaustive tree search in the tic-tac-toe game space, and always picks the winning move. It is impossible to win the game against computer.

3- **Limiting Factors**

Considering that this project was due in less than six weeks, time was a limiting factor. In addition, Raspberry Pi’s underwhelming power was a major drawback in this project.

a. **OpenCV and Capturing Problems**
Raspberry Pi requires the user to compile OpenCV from scratch. The problem raised when the default parameters, and many online documentation, only install the GTK and GTK 2.0 libraries with OpenCV. Unfortunately, GTK performs poorly and only can capture at 3 to 6 frames per second. It was even before adding the complexity of AR routines to find markers and just for capturing a single frame from camera and showing it on screen.

After much exploration, it was observed that if OpenCV is installed with QT library, it can capture frames at 20 frames per second for up to 1280x900 pixels on Raspberry Pi 3.0s. Any resolution above is a performance hit even after increasing the Pi’s shared graphic memory to 256MB.

b. OpenGL Driver Problems
Raspberry Pi’s OpenGL driver is notoriously slow. Many different OpenGL libraries like SDL and GLUT have been explored, but the performance has been dropped significantly after using them. Therefore, it was chosen to only use built-in OpenCV 3.0 functions.

c. ARUCO Performance
After using ARUCO markers and board detection routines, the performance has been dropped by 15 frames per second. Now, the game only renders at 5 frames per second.

d. Time Constrains
As explained earlier, the design has been simplified to fit the project timeline. The current game needs the player to input the position of her move with a keyboard which later can be replaced with additional markers. It also only supports one machine, and later it can be expanded to two Pi’s communicating in parallel.

4- Design Perspective and Challenges

The goal of this project was to render both eyes on a VST HMD Augmented Reality settings with two separate Raspberry Pi 3.0 machines. In the original design idea, the Raspberry Pi’s would communicate over the Ethernet cable to share the state of the Tic-Tac-Toe game and next moves. Since each Pi was directly attached to a different camera, the stereo 3D was maintained, and they could decouple each other from syncing that information. Each machine could separately render the scene based on their camera parameters and projection matrix.

In the stripped down version, the game is still playable and functional. A lot of designed idea has been invested to make sure the game is presentable.

a. Using a 3D Table for Tic-Tac-Toe
To make the game more fun to look at, the Tic-Tac-Toe table is rendered in 3D using the camera projection matrix. Moving the camera/eye over the board changes the perspective of the camera, and this table is rendered accordingly creating a nice 3D visual effect.

b. Using 3D Labels for X’s and O’s

In addition to the 3D board, both “X” and “O” labels are also generated and rendered in 3D with respect to camera view. This was a challenge because every object had to be projected back to camera space in order to be rendered in accurate position.

c. Different Colors: Last Move, Player, Invalid, and Winning

Each player has different colors for XO labels. Player is blue and computer is green. However, computer “O” labels have a slightly lighter green color for the last move, so it can easily be identified in a game play. If the player chose an invalid pre-occupied location, that location label turns red. If one player wins the game, the whole row, column, or diagonal corresponding to the winning state will turn red.

d. Displaying Messages and Score

regarding how to play and other alerts are displayed on the bottom of the screen.

e. Operating Keys

Pressing “R” will restart the game and zeroes the score. Pressing Space key will replay another round. Pressing “L” key will cycle through camera view, board view, and game view. Pressing 1-9 corresponds to location on the Tic-Tac-Toe table.

5- How to Compile and Run:

Please refer to APPENDIX A for how to install OpenCV and other libraries on Raspberry Pi 3.0. Assuming all the libraries are already installed:

$ g++ `pkg-config --cflags --libs opencv` -laruco arttt.cpp -o arttt -std=c++11

$ ./arttt

To Calibrate the camera and get camera.yml file:

$ cd opencv-3.1.0/samples/cpp
$ ./calibration -w=7 -h=5 -s=0.02 -pt=chessboard -o=camera.yml -op -oe
To generate the board:

$ cd aruco-1.3.0/
$ cd build/utils
$ ./aruco_create_board 5:2 board.png board.yml

6- Future Improvements

Other than adding a second Raspberry Pi into the mix, several ideas can be explored:

a. Player’s Markers Object

With using extra marker objects, player can put her moves physically onto the board. For example, making 5 markers to represent player’s “X” labels, the player can put these markers on the board by her hand.

b. 3D Object with Textures

Another improvement could be using photo-realistic 3D objects and texture maps.

c. Different Difficulty Levels

With altering with the AI, and putting a depth limit on the search, different difficulty levels can be achieved. At this point, computer never loses which makes this game very boring to play.

7- References:

OpenCV http://www.OpenCV.com
ARUCO http://www.uco.es/investiga/grupos/ava/node/26
Matthew Steel’s MinMax https://gist.github.com/MatthewSteel/3158579
Raspberry Pi and NOOBS https://www.raspberry.pi.org/
APPENDIX A:

1- Installing NOOBS on Raspberry Pi 3.0
   a. Using SDCardFormatter program copy everything
   b. Install NOOBS

2- Configuration
   a. Inside Raspbian, go to MENU > Preferences > Raspberry Pi Configuration
   b. Change Graphics memory to 128MB
   c. Expand File System if necessary
   d. System will ask to restart, say yes.

3- Adding Static IP if needed:
   a. Terminal:
      
      ```
      sudo nano /etc/network/interfaces
      ```
      
      Change “iface eth0 inet manual” to static

      ```
      address 192.168.1.100
      netmask 255.255.255.0
      gateway 192.168.1.1
      dns-nameservers 4.4.4.4 8.8.8.8
      ```
   b. `sudo /etc/dhcpcd.conf`
      add:
      ```
      interface eth0
      static ip_address=192.168.1.100
      static routers=192.168.1.1
      static domain_name_servers=4.4.4.4 8.8.8.8
      ```
   c. `sudo systemctl daemon-reload`
   d. `sudo service networking restart`
   e. `sudo reboot`

4- Updating:
   a. Open Terminal, run:
      ```
      sudo rpi-update
      ```
   b. Reboot
   c. Execute:
      ```
      sudo apt-get update
      ```
      ```
      sudo apt-get upgrade
      ```
      ```
      sudo apt-get dist-upgrade
      ```
      ```
      sudo apt-get autoremove
      ```
      ```
      sudo remove
      ```

5- Installing Dependencies:
   a. `sudo apt-get -y install build-essential cmake cmake-curses-gui pkg-config`
   b. `sudo apt-get -y install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libeigen3-dev libxvidcore-dev libx264-

dev libgtk2-devs libgtk-3-dev

wget https://github.com/Itseez/opencv/archive/3.1.0.zip -o opencv-3.1.0.zip

unzip opencv-3.1.0.zip
cd opencv-3.1.0
mkdir release
cd release
cmake ../
ccmake ../

Edit Options to your Preferences ***

make -j4
sudo make install
sudo ldconfig -v

6- Test:
   a. Try compiling OpenCV samples:

      cd ..
cd samples/cpp
g++ `pkg-config --cflags --libs opencv` facedetect.cpp -o facedetect
./facedetect

7- Installing QT:

      sudo apt-get update
      sudo apt-get install qtcreator
      sudo apt-get install qt5-default qt5-qmake qtbase5-dev-tools

8- Installing ArUco:

      sudo apt-get update & & sudo apt-get upgrade

      sudo apt-get libicu-dev freeglut3 freeglut3-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libxine2-dev

      wget http://sourceforge.net/projects/aruco/files/1.3.0/aruco-1.3.0.tgz
tar zxvf aruco-1.3.0.tgz
cd aruco-1.3.0
mkdir build
cd build
cmake ../
make

some of the example files need this line at the top: #include <opencv2/imgproc/imgproc.hpp>

sudo make install
sudo ldconfig -v

create a board:
cd aruco-1.3.0/
cd build/utils
./aruco_create_board 5:2 board.png board.yml

Calibrate Camera:
cd opencv-3.1.0/samples/cpp
g++ `pkg-config --cflags --libs opencv` calibration.cpp -o calibration
./calibration -w=7 -h=5 -s=0.03 -o camera.yml -op -oe

copy camera.yml to aruco-1.3.0/build/utils/

Test Board:
cd aruco-1.3.0/
cd build/utils
./aruco_test_board live board.yml camera.yml 0.025