The Computational Image
Nabil, Anis, Will, Jonathan

Multi-touch table

Final Project Write-up

1) Project Idea:

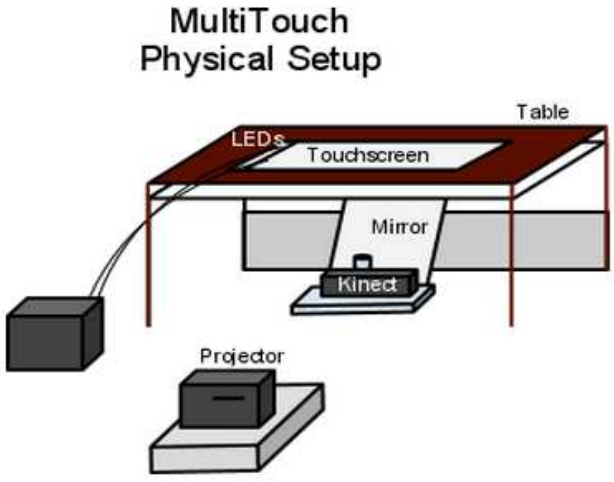
As our final project we are going to build a multi-touch surface using processing and as a demo we will have a multi-player game. Some of the major challenges that we have to overcome are: building the physical surface and syncing the camera and projector, tracking multiple objects and finally coding the game. We are going to use IR LEDs and IR camera to detect fingers touching the screen. Our next goal is to move objects in the game based on the finger movements.

Inspired by one of apple's games named, “Osmosis” we thought it would be great to have that particular application for our multi-touch surface. Our game will have features where several people can play the game at the same time; in addition to that the physics implementation will add some augmented reality part to our platform

2) Table prototype:

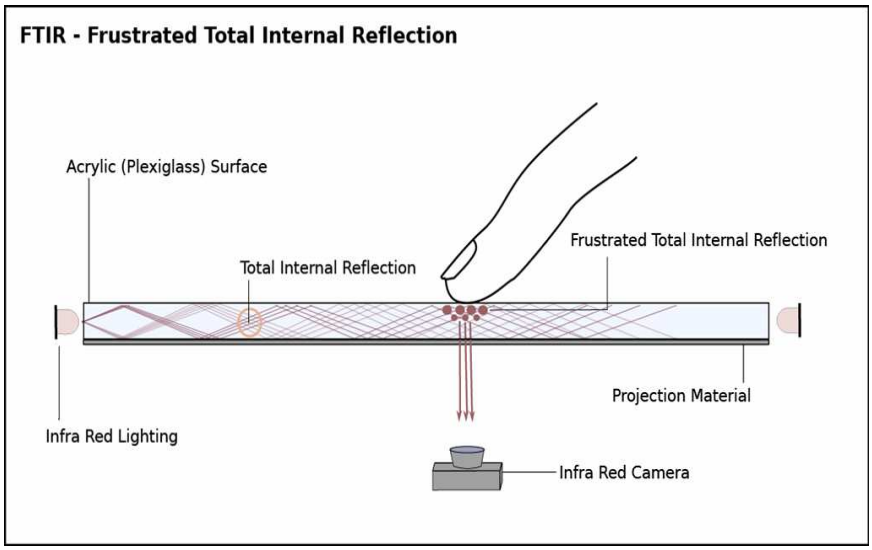
The platform consists of

I. A table made from scratch
II. 12V power supply
III. 12V Infra-red Led strips
IV. Microsoft Kinect
V. Two 18x24 acrylic glass sheet, frosted(1/4″) and clear(1/8”)
VI. 18x24 white piece of paper
VII. A projector
VIII. Mirror
IX. Laptop



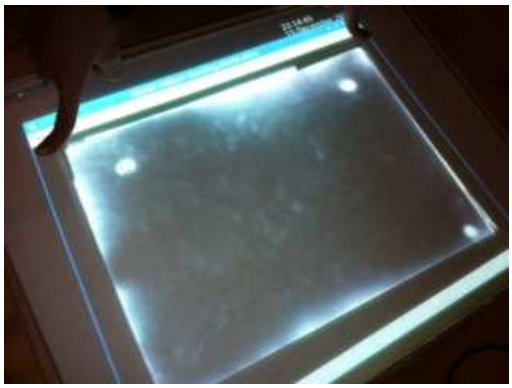
3) Sensing technique:

The IR Led enters the acrylic glass at a small angle and total internal reflection occurs inside the glass. When the acrylic is touched, the light rays are no longer trapped inside. The rays refract, due to a change in medium, and escape through the opposite side, but only over the point of contact. So whenever the surface is touched, IR is emitted and the Infrared camera underneath detects this light and thus we detect the finger points as brightly lit white blobs.



4) Homography and Finger detection:

Our first step is to detect the blobs from the IR camera and filter out and threshold the image so that we get only what we want to consider as fingers. But before we even look for blobs we do back ground subtracting which lets us to avoid any noises around the corners which might appear as false blobs. We hardcode some threshold values for blob size and treat this value as “ideal” for our project in the detection code.



Before we even start the tracking any object, it is imperative to make sure that the IR camera sees what we want it to see in the correct position. Our first problem is to sync the camera coordinate system with the projector coordinate system. This is done in order to avoid any form of offset between the camera and the projector. The algorithm for syncing the camera and the projector gives us a matrix called the homography matrix. The algorithm takes a point (*x,y*) from the projected image surface and creates a homography matrix that allows us to find where exactly the point (*x, y)* is located on

the IR camera image. Since we know the size of our sketch window (i.e. width(), height()), we know the four corners of our projection window. We then select the corresponding four corners in the camera coordinate. Once we have these eight corners, we compute the homography matrix which we save as a matrix. This homography matrix is later used to translate every tracked point. Every time we detect a potential blob which could be a finger point, we multiply it with the homography matrix that tells us where exactly it is located in the projector coordinate.

Saving the homography to a text file is not the best option for us because every time we turn off the projector we have to readjust the mirror and keystone the projector window. Thus we end up physically moving the setup, which ultimately will change the correspondence between the camera and the projector coordinate system.

5) Tracking fingers:

First we filter fingerprints that qualify as legitimate track points. This means checking if the detected fingerprints are similar to a certain shape; we describe legitimate track points as those which are elliptical, not too small in area, and not too big at the same time.

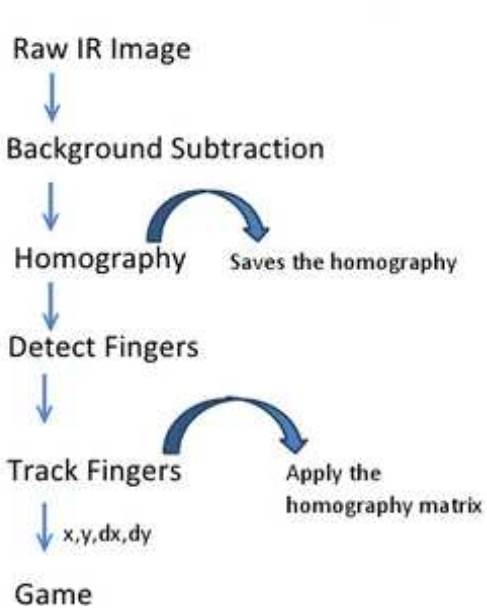
We use an arraylist of floating point arrays to store information about the detected track points. We store the x,y coordinates of each fingerprint center and the dx,dy values corresponding to the horizontal and vertical displacements of the current tracked point center between the current frame and the last frame. This gives us information about the velocity of the fingerprint in case the finger has been swiped over the screen.

The execution time of the fingerprint tracking algorithm is $O(n^2)$, where n is the total number of legitimate fingerprints tracked.

The fingerprint tracking algorithm is shown below:

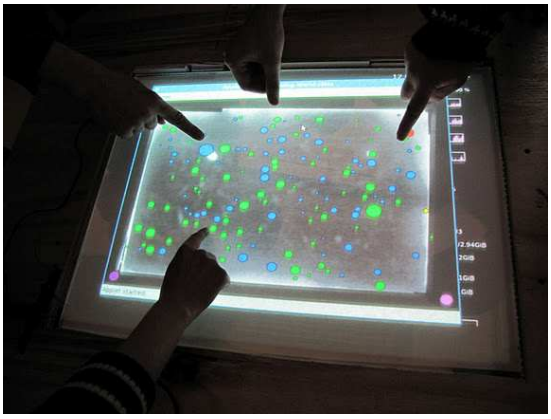
```
ArrayList <float []> fingers;        / / Arraylist to hold legitimate fingerprint data
while keepGoing:
    prints = Track();        / / Track() detects & filters the legitimate fingerprints and
                             / / saves them in an array represented by x,y coordinates
    for each p in prints:
        for each f in fingers:
            pt[f][p] = dist (f,p)      / / dist() calculates the magnitude of the
                                       / / straight line distance between existing
                                       / / and new track points; pt holds this data
/ * Next step:
    * calculate minimum distance from all existing track points to new track points. If this
    * distance  is smaller than a threshold, then update the existing track point with the
    * corresponding new track point data. Also calculate dx, dy. Otherwise delete existing track
    * point. Finally, create track points with all the remaining new track points that didn't have
    * any correspondence with existing track points.
*/
    while i < fingers.length():
        j = minDist (pt, i)          / / where minDist(pt,i)= pt[i][j]
        if (minDistance < threshold)
            fingers(i).x = prints[j].x
            fingers(i).y = prints[j].y
            fingers(i).dx = prints[j].x – fingers(i).x
            fingers(i).dy = prints[j].y – fingers(i).y
            i++
            / / we also mark j so that its not being compared with other existing track points
        else
            fingers.remove(i)
    endwhile
    for each j in prints.length:
    if (doesCorrespond(prints(j), fingers) == False)
        fingers.add(prints(j))
endWhile                                  / / go back restart the algorithm
```

6) Flow Diagram:



How the game is updated using the tracked points:

The game uses the tracked fingers to allow the users to interact with the balls. Tracked fingers give the x,y coordinates of the finger, along with dx and dy. These values are used to update the game board by adding the dx to the x coordinate and dy to the y coordinate of the particular bubble located at the x,y coordinates of the finger.



7) Team Members and Contributions:

Anis: Decide which parts to buy. Designed and built the physical platform. Worked on blob detection and the camera projector homography section. Helped Nabil as much as I could with the finger tracking. Helped in debugging the code, especially the homography part (with Will).

Nabil: Built the multiple fingerprint tracker. Helped Anis in building the platform. Worked in merging the entire code and debugging them.

Will: Programming application which the multi-track will be used on. Helping to build the multi-touch structure. Helped Anis in debugging the homography part that made the game a lot faster.

Jonathan: Helped Anis in buying parts and in building the platform.

8) Timeline:

- By 11/12- Building complete (pending some part orders)
- By 11/19- Entire Physical Setup Complete with projector to plexiglass projection working correctly
- By 11/23- Progress Report due
- By 11/26- Finish touch recognition
- By 12/3- Work on App and Finish loose ends (combining app and touch recognition)
- By 12/12- App complete, project set up and done