Assignment-1 Report CS7034: Augmented Reality

Design and implement a real-time vision system that uses a Wiimote and a camera as input to a game

Submitted to:

Peter Redmond (predmond@tcd.ie), Dr.
School of Computer Science and Statistics,
University of Dublin,
Trinity College

Prepared by:

Pankaj Gupta (guptapg@tcd.ie), 11263065
MSc. Interactive Entertainment Technology,
University of Dublin,
Trinity College

07th March 2012

Abstract: This report covers my first assignment to design and implement a real-time vision system that uses camera and Wii Remote (referred as Wiimote henceforth) as input to a game-play. Assignment mandates use of a camera and an inertial input device (like Wiimote). Instead of using an existing game (and plugging-in the inputs from my vision system) I preferred writing my own game because I wanted to handle the interaction in game-play. Vision system takes live stream from camera, orientation and button click from Wiimote. Camera live stream is handled using Opensource OpenCV library to detect a human face using Haar classifier cascade features and the live stream forms the background of the game screen. Orientation and button click input from Wiimote is handled using Opensource WiiYourself! Library and represents orientation of a play-tool (a cannon in the game) and fire event of the cannon respectively.

1.0 Introduction:

The game is a two player 3D game (2D was minimum required as per assignment requirements) with one player(A) using Wiimote to fire cannon on the face of other player(B). So the game represents "Fire and Dodge" behavior. Player A is supposed to fire the cannon on face of player B and aim of player B is to avoid the cannon falling on his/her face. 3D perspective of the game view changes according to the head position of player B and a red flash appears when the cannon fire hits the face.

Live video stream is captured using a normal webcam and OpenCV library is used to detect the face (if any) in the scene. The face detection is accomplished using Haar classifier cascade features of a human face, which means the face detection is scale-invariant and tackles changes in lighting conditions. The Haar classifier cascade file "haarcascade_frontalface_alt.xml" comes with the OpenCV library distribution. We obtain the bounding rectangle of the detected face in the scene which we will be using to manipulate the view perspective and frustum. We will also utilize bounding rectangle information to detect the collision of cannon-fire with the face.

To augment the real world scene captured above, I have used OpenGL and Glut libraries to model the 3D world between the player and the camera positions. Live stream captured using OpenCV is passed to OpenGL for display in the game screen.

2.0 Implementation

Every frame of live stream from camera is checked for presence of human face using OpenCV function `cvHaarDetectObjects()`. Using Haar classifier cascade features gives us very robust face recognition implementation. It is a scale-invariant method which is resilient to variation in lighting conditions. This method is arguably faster or modern machines which makes it to be used in real-time on commonly available devices.

After detecting a face, we calculate the bounding rectangle of the detected face, its centre and then use the frame as a texture in OpenGL. The only way of displaying video in OpenGL is by displaying them as texture. Following OpenGL function is key in achieving this –

```
gluBuild2DMipmaps( GL_TEXTURE_2D, 3, image->width, image->height,
                   GL_BGR_EXT, GL_UNSIGNED_BYTE, image->imageData );
```

We adjust the frustum culling and perspective depending on the head-position. The head position returned by OpenCV functions does not represent the head-position in OpenGL window. One major problem is to coordinate-system mapping between OpenCV and OpenGL.

It took me substantial time to figure-out that in OpenCV, the view direction is towards me but in OpenGL, view direction is away from me. Further, due to frustum culling and applying the OpenCV frame as a texture, it's hard to map coordinates between the two systems.

With much trial and error, I came-up with some conversion figures those were working as expected (refer source code for more detail).

We are calculating center of face for each frame and adjusting the frustum and perspective for that centre location. Real-time movements are jittery and hence give a jerky view. To solve this, I use Kalman Filter to approximate the location of center of head. By using Kalman Filter, the view became many times smoother.

To give the player 3D perception, I have added red, concentric circles as targets which change their location in space as player moves.

For our game, we have cannon at the bottom-centre of the window, which rotates as the Wiimote is rotated and fires towards the face in Wiimote direction when button B (refer Wiimote user guide) is pressed. Wiimote orientation data is good way to determine orientation input but is unstable at times(good enough for a game-play).

Instead of general approach of polling for the Wiimote status, I use a callback function which gets invoked whenever there is a change in Wiimote state. This saves some computations for us and a lot of Wiimote battery. (Refer source code for detailed information about my implementation of Wiimote use).

3.0 Features

These are some of the salient features of my assignment work-

- 3D game-play
- Two player support, extensible for more players by adding multiple Wiimotes.
- Perspective exploration by moving head.

4.0 Algorithms

Following algorithms that I have used are worth mentioning in this section-

4.1 Haar Classifier Cascade: Haar classifier uses Haar like features of an object for classification which is a cascade of features those are invariant to the scale of an object and are even remain constant over the variation in light conditions. Feature detection is always a tradeoff between robustness of algorithm and computational resources required. Face detection using Haar classifier is a balanced approach between the two.

Other computationally viable option like template matching is not scale-invariant and suffers a lot under different lighting conditions. Also, provided previous restrictions, it will only work for the face whose template has been provided.

4.2 Kalman Filter: When we are dealing with fluctuating data causing jittery behavior, we prefer filtering the data to attain smoothness. One way is to overage the current data with some of the previous data but it is does not give impressive results. Kalman filter comes to rescue here as it also takes into consideration the uncertainties in its estimate. It works in two phases, first phase produces estimates along with their uncertainties and in second phase, outcome of the next measurement is observed, the estimates are updated using weighted average.

5.0 Decision choices

For accomplishing this assignment, I made some decisions and learned with them. This section is dedicated to mention some of them along with reason to choose them

5.1 Framework for Vision System: I choose OpenCV framework for my vision system as it is pose minimal hardware requirements and is accepted widely in academics. Other option were Microsoft Kinect SDK and OpenNI frameworks.

5.2 Suitable inertial sensor and its library: For choice of inertial sensor, I did not had any option other than using Wiimote which is available to us in the lab. For a suitable library for Wiimote, I choose WiiYourself library. The main reason for choosing this library is being written in C++ and have minimal dependencies among other libraries written in C++ programming language.

5.3 My own game-play: An important question after implementing my Vision system was- if I should plug-in the inputs from Vision System to some readily available game or should I write one of my own. In the first case, I would get readily available rich game interface and game-mechanisms and in second case, I can implement my own game interaction. The whole idea of assignment being learning, I choose the second option of implementing my own game interaction.

Writing my own game-play posed one main challenge of coordinate-system mapping from OpenCV system to OpenGL system as I wanted 3D objects to interact with my body. It was apparent after trying for few days that it is not possible to map all points from OpenGL to OpenCV system. Hence I decided to limit the coordinate-system mapping between the two systems and now I map OpenGL coordinates to detected face in OpenCV system.

5.4 2D game or 3D: Although the minimum requirements of the assignment could be fulfilled using only OpenCV, I decided to augment that using rendering features of OpenGL.

6.0 Demonstration Video

Demonstration of the game can be seen online at youtube.com – http://www.youtube.com/watch?v=N2iq5OwW8dM

7.0 Acquisition and use

The implementation source code is mainly in five files-

`Painter.cpp`	This is the main file containing OpenCV and OpenGL initialization, program entry point and Keyboard handling routines.
`cvfaces.h`	This file contains face detection function and Kalman filtering functions (for predicting the centre of head position)
`glscene.h`	This file contains functions to adjust projection, frustum and drawing augmented scene in the game.
`texture.h`	File contains routines to load textures.
`wii.h`	Routines to handle Wiimote.

The source code can be build using Microsoft VC++ compiler or GCC on Microsoft Windows platform.

Minimum Hardware requirements: An OpenNI compliant camera, a Wiimote, 256MB RAM, 2.6 GHz processor.

Minimum Software requirements: Microsoft Windows XP, OpenCV2.3, OpenGL2.0

8.0 Tools

Associated work has been done on Microsoft Windows7 Operating System using *OpenCV, WiiYourself, OpenGL, GLUT and a general purpose Math library for vector and matrix operations.*

9.0 Future scope

There are couple of things to be worked-on in future-

- Mapping complete OpenCV coordinates to OpenGL coordinate system.
- Multiple cannon firing for multiple Wiimotes connected to the game.
- Player score management.
- Player registration by face.

10.0 Acknowledgements

- Many thanks to Dr. Peter Redmond for providing camera that I used for this assignment.
- Thanks to Fabrizio Bellicano for discussions about the game and problems faced.