Introduction

Human Computer Interfaces, or HCIs provide a method for a computer user to manipulate data dynamically as the computer runs. Early HCI's are keyboards, and later mice. These two interfaces only allow a limited range in reading human movement, though they are still useful enough to control the modern computer.

New applications for computing have pushed the need for HCI's that gather more data about human movement. The Wiimote from Nintendo for instance, gathers information about 3d location and orientation. Modern cell phones even have built in accelerometers that can also track movement in 3d. There are also applications in development, such as Microsoft's Project Natal, that gathers information about the movement of the entire human body.

In this project, we seek to demonstrate a HCI that allows tracking of the human head and movement of the face, which is then translated to movement of the onscreen cursor.

Section I : Detecting faces with the Eigenface method.

   I used the The MIT-CBCL face recognition database to generate the Eigenfaces used for face detection. This is done by using the SVD command in matlab on the set of normalized images loaded from the database. The following shows a few eigenfaces generated.

1. Gather input video.
   I used OpenCV's video capture abilities to capture the video stream used for processing.
   The frames captured will be put through the following steps. Here is the first frame of the video capture:
1. **Sweep through the input image and detect faces.**

I chose to select a reasonable area in the center of the image for initial detection of the face. The function sweeps through the reasonable area with a patch the same size as the database images, then reconstructs the image in face space. Then I selected the patch that best represented a face based on its Euclidean distance to the images in the database, and found the centroid for that location. Here is a reconstruction of the face found in the first frame of the video:

Section II: Lucas Kanade optical flow to track face pose movement.

1. **Use centroid input from face detector to bound box for calculating Optical Flow**

I used a simple fixed size bounding box in this situation, as the size of the face tends not to change throughout the video capture. Below is a the first face frame of the video stream:

1. **Calculate LK Optical Flow for two adjacent frames and find the average flow in the image.**

I used Sohaib Khan’s HierarchalLK.m code here to calculate LK optical flow between two face frames using 3 pyramid levels and a window size of 3 pixels. The following shows the optical flow from frame 2 to 3, and the average flow:
1. Apply the average flow vector to onscreen cursor movement.

   Cursor movement can be applied in Matlab by manipulating the PointerLocation parameter of the root handle. Unfortunately, I wasn’t able to test this as it didn’t work in the OSX environment.

Section III: Project Observations and pitfalls.

This project has major practical applications in the real world. Unfortunately, I encountered many issues that limited the success of my method.

The first major issue was in the speed of processing in Matlab. My algorithms for detecting the face and applying the LK optical flow were either not optimized, or Matlab didn’t process them fast enough, which didn’t allow me to perform the process in real time. I ended up using a prerecorded video.

The second issue is with face detection in the case of rotation of the face pose. I had to default to an arbitrary location when the face wasn’t detected, such as when my face was pointing almost completely upwards. This can be rectified by using a specialized database of input faces that includes rotated face images.

Thirdly, the results of the optical flow in part 2 needed to be smoothed out. This way, the cursor movement wouldn’t be affected by jerky direction vectors due to some unpredictability with the image.

Conclusion

This project is definitely a proof of concept. In order to enable real time processing of the video stream, this project would probably have to be written in a more optimized language, and with better algorithms. Also, there are improvements, or different approaches that may produce better results. For instance, face detection may be switched to known real time methods such as the Viola and Jones approach. I am content with this project as a proof of concept, but would like to develop the idea further.

This project requires OpenCV and OpenCV libraries for Matlab.