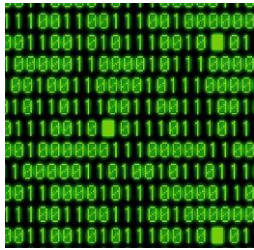


ConnecTied Documentation



Computing World



Physical World

www.connected.com

Contents

1. Problem Statement	2
2. Solution	2
3. Development.....	3
3.1 Marker Detection	3
3.2 Gesture Recognition	4
3.3 Gesture Mapping.....	7
3.4 User Interface.....	7
3.4.1 Modes.....	8
3.4.2 UI Layout	9
Appendix A: Control flow diagram for the implemented system.....	10

1. Problem Statement

With the advent of internet in almost every part of human activity, to access a computing device, one still has to interact with a range of I/O peripherals which are typically not quite in the “human” domain. In other words, the user has to adapt to the limitations of the computing devices in terms of their integration with the everyday objects he/she interacts with. Thus, our world is divided into the physical and computing domains with the I/O devices filling the gap.

2. Solution

The project aims at bridging the gap between the two domains by replacing the traditional ways of interacting with the computer with a more human centric one wherein the user’s environment itself becomes the output medium and user’s gestures becomes the input for the computer.

The team proposed to develop ConnecTied, a portable spatial augmented reality device that would project digital information onto the user’s physical world which can further be controlled using human gestures, objects and environment. Thus, allowing users to interact and control the computing world without any interface barrier of traditional I/O Devices. The simplest implementation would be the user using the wall in front of him/her as a screen, the desk as the keyboard (with keys projected on it), supplemented by voice and gestures to interact with the computing world powered by a wearable processor. The following system level schematic captures the basic implementation details of the proposed project.

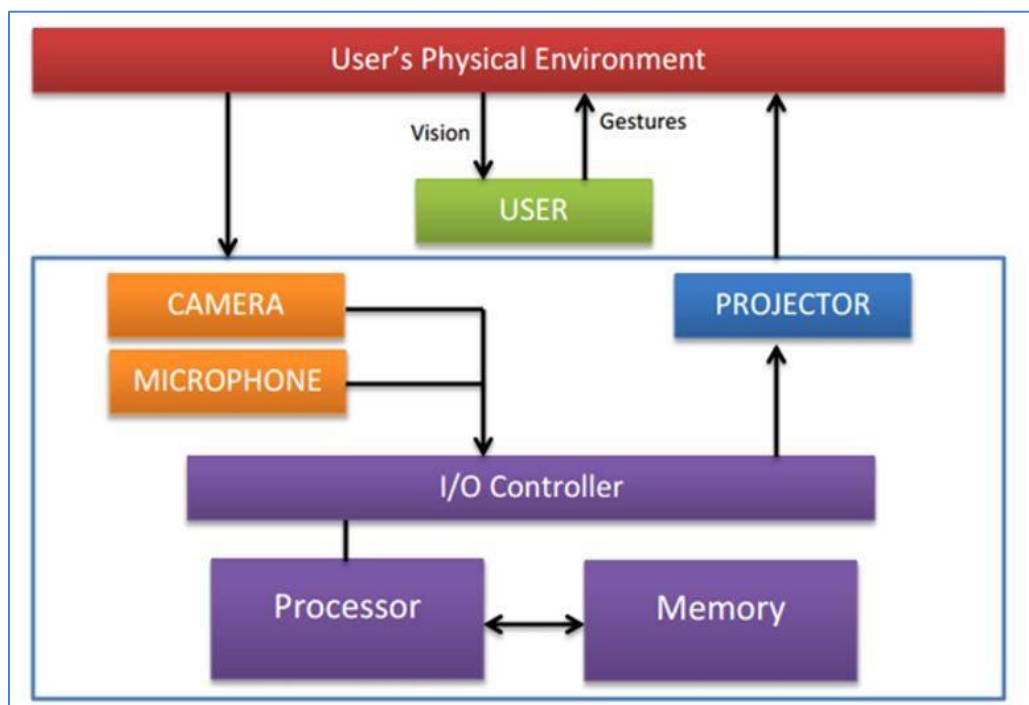


Figure 1. Block diagram of sub components

3. Development

During the semester, the team developed a computer vision system capable of recognizing human gestures using colored marker caps on the fingers. Each of these gestures were mapped to an action to be performed such as moving the mouse pointer, clicking, taking a picture etc. These actions constitute one of the ways the user would interact with the proposed system. The other way in which user can interact with the system is by using objects in the physical environment. The implemented system can use the image of the object in the user's surrounding and use it as a keyword to do online search using Google Image Search. The system is also capable of reading and decoding of barcodes in the image. In terms of scalability, the implemented system is a standalone application package which can be used on any windows laptop with a webcam without the requirement of any additional hardware.

To tackle the projection problem, Kinect sensor was used for depth-sensing to detect planar surfaces suitable for use as a screen, combined with a small mobile computing device and a pico projector to project a screen onto the detected plane. The colored camera on a Kinect could be also be used to recognize gestures instead of a webcam.

Appendix A consists of the control flow diagram for the implemented system. The following sections reiterate the journey of development, along with challenges faced and implementation details of each of the major component of the project.

3.1 Marker Detection

The basis of gesture recognition is the marker points on the user's fingers. The implementation takes in the input from the camera feed and processed in a series of steps to obtain accurate values for the markers in the video using the following series of steps.

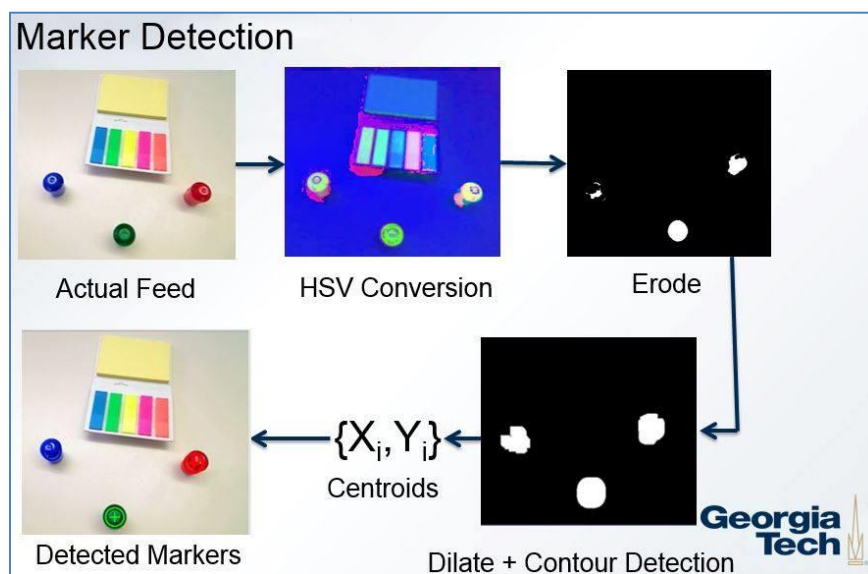


Figure 2. Marker Detection Process Flow

The live image feed is converted to HSV color scale and pre-determined thresholds are applied to get the binary image with all the colored markers. The binary image is then subjected to erosion to get rid of the noise. The eroded image is the dilated and passed through openCV's contour detection method to get the centroids of all the detected markers. These values are then added to the buffer with the marker points from the last 30 frames.

One of the major drawbacks of using HSV values is that the system is dependent on the lighting conditions of the environment. Various methods like background reduction and exposure manipulation were tried to limit this phenomenon. One of the future goals would be to further increase the robustness of the system.

3.2 Gesture Recognition

Once the marker coordinates were determined they were used to provide input to the gesture recognition system. The initial version of this program was developed in MATLAB using existing libraries and Hidden Markov models. The next step uses Hidden Markov models to classify gestures. It is an algorithm that, given a sequence of results, helps to find a sequence of states or events leading to the result. The HMM first needs to be trained to recognize various gestures so that it can compare the input feed with the known gestures and try to find the best match. It then compares the live input feed against the trained gestures and computes a result. If the result is greater than the set threshold, the gesture is recognized. The following figures show the training and testing dataset of the two basic gestures.

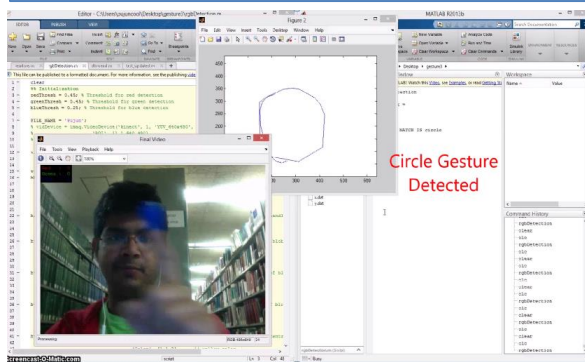
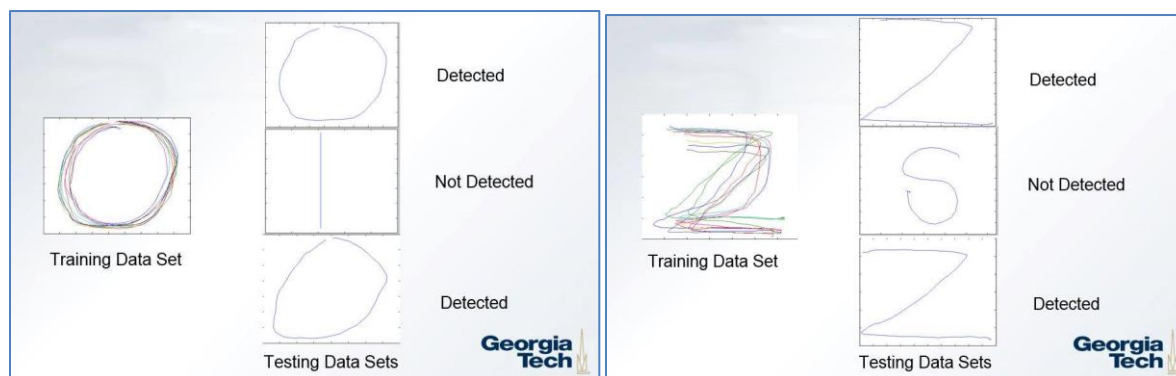


Figure 3. HMM working on 'O' gesture.

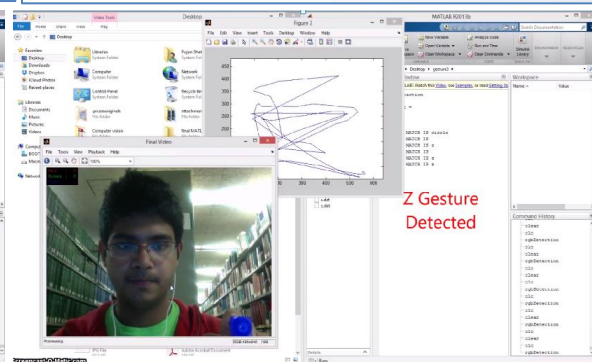


Figure 4. HMM working on 'Z' gesture.

However, as the team further advanced in the project it was realized that the MATLAB version would be unsuitable for the small, embedded system intended as an end-platform, as it would require either a running instance of MATLAB or a system executable compiled from MATLAB along with the MATLAB runtime libraries. The team initially attempted to compile the code from MATLAB, but found compile times to be prohibitively large for fast code iteration (~45 mins - 1 hr).

Once a proof-of-concept was working in MATLAB, the team began the transition into C++. The team opted to use OpenCV, which, handily enough, comes with an installer and has its own (very poorly documented) built-in matrix framework. Machine learning libraries such as Shogun or MLpack were initially considered, but these ultimately proved too time-consuming to install and use. There was limited documentation available and most of them were not cross compatible. Finally, CVHMM library was used and after sleepless nights, HMMs were implemented on C++. But it was soon realized that HMM training was taking too much time for all the gestures to be determined within a reasonable range of error. Therefore, it was decided that we would adopt a custom approach that was called ‘Vector Modelling’.

The vector modelling approach divides the image feed into a grid of bins and analyzes the coordinates of various markers over the last few frames. It further calculates the movement vectors for each of the markers. Then it tries to do basic curve fitting to find the gesture being performed based on a set of pre-defined gesture model. The following flow chart explains the overview of the process.

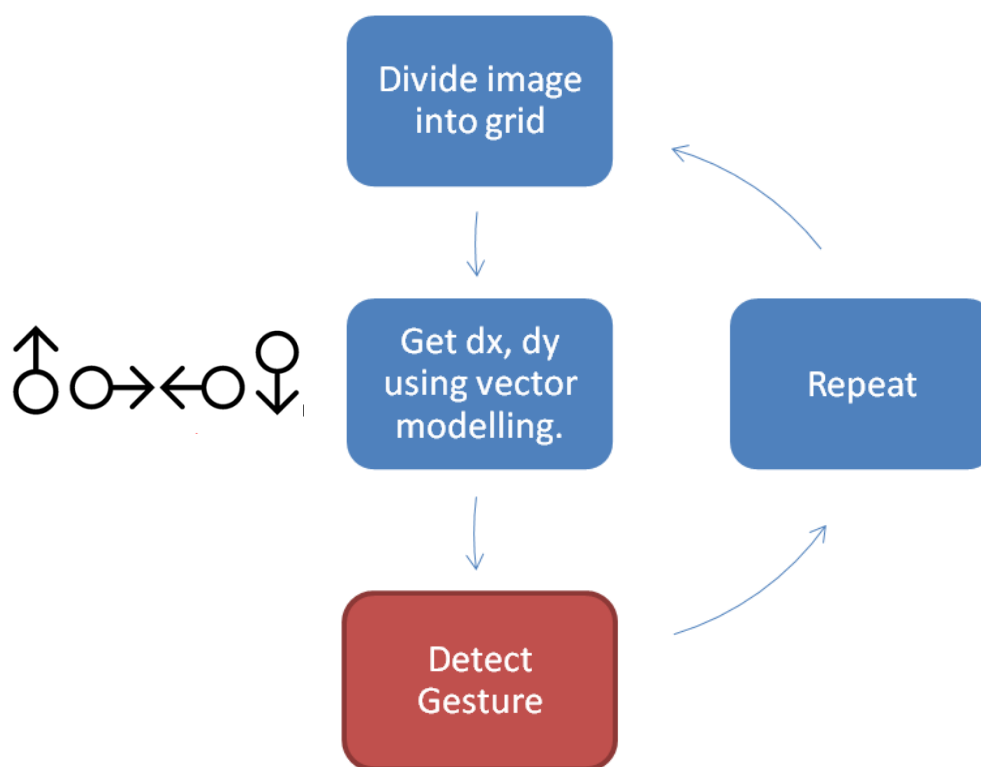


Figure 5. Gesture Detection process flow.

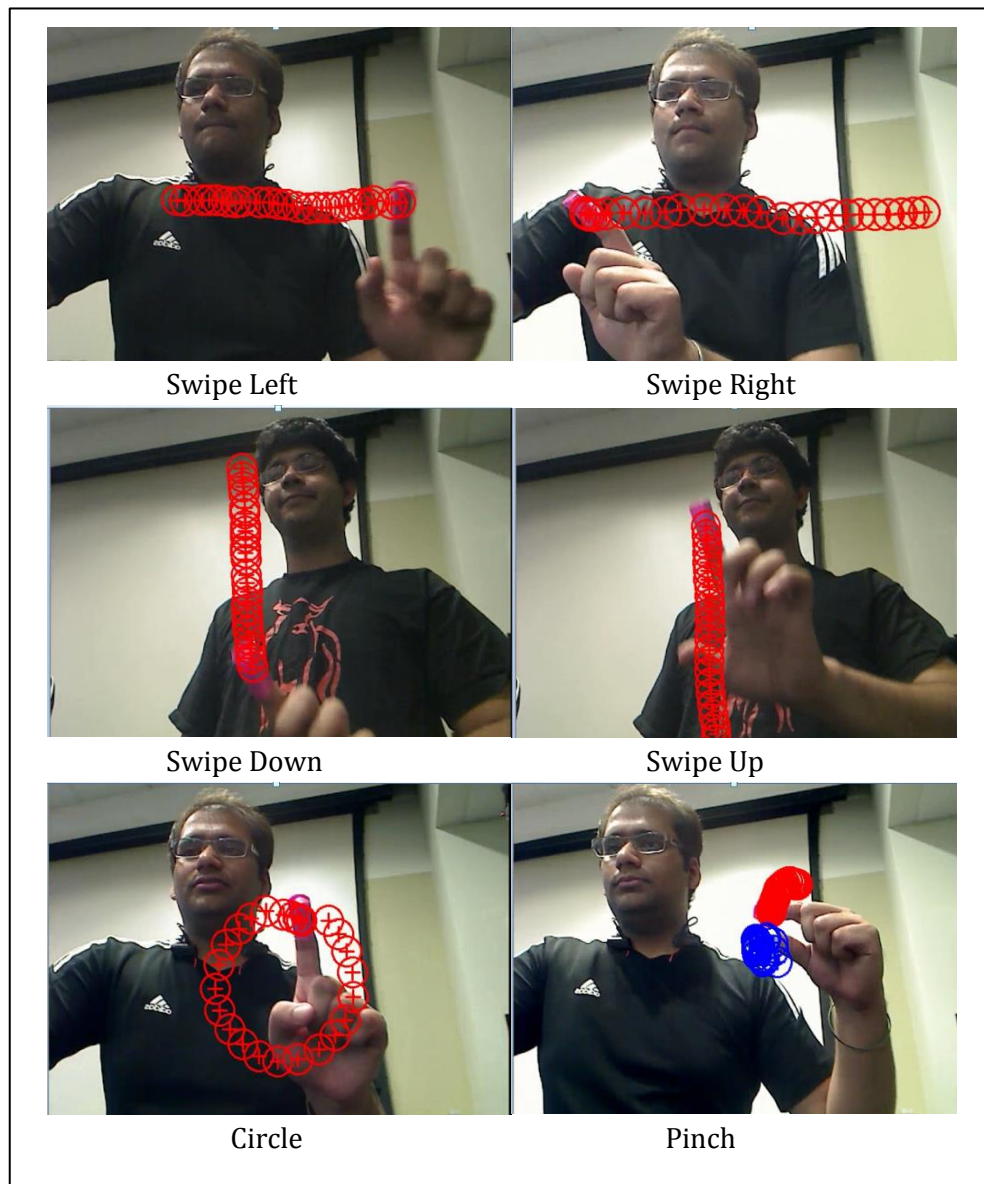


Figure 6. Some of the implemented gestures.

3.3 Gesture Mapping

The next step implemented was to tie the detected gestures to ActionListeners, so that as soon as the gesture is recognized the associated action is performed. All the other gestures were tied to different actions using windows API depending on the state of the application. For ex: swipe left in picture mode takes a screenshot. The basic action of moving the mouse pointer and clicking by pinching was kept universal throughout the application. The following figure describes the various actions linked to each gesture.

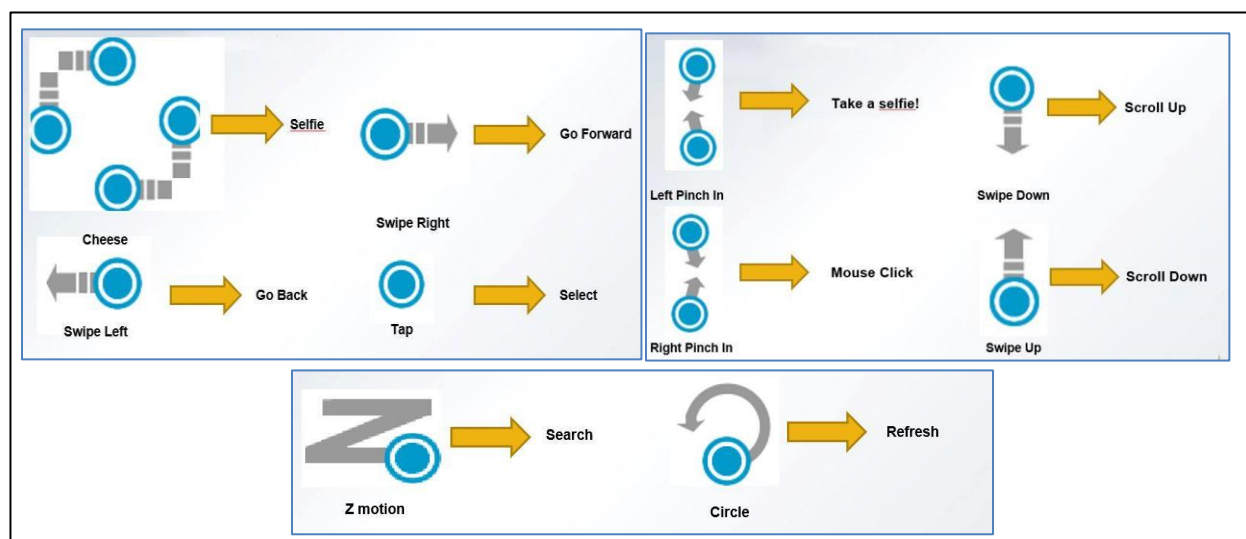


Figure 7. Gesture Mapping to actions.

3.4 User Interface

The team also developed a user interface in VB.net. This served as a central hub that users could use to enter different “modes,” supporting different sets of gestures. For instance, the camera mode supports a gesture to take and save a picture, and the Google search mode supports a gesture to take a picture and perform a reverse image search on it. The same gesture may perform different actions in different modes. Users may return to the hub at any time from a mode by just clicking the back button.

Mouse control was also an important part of the application. The user may use the **red** marker cap to move the mouse, and pinch red and blue fingers together to left-click. This easily enables other actions that normally use the mouse, such as drawing or drag-and-drop.

3.4.1 Modes

Currently the following five modes have been implemented successfully:


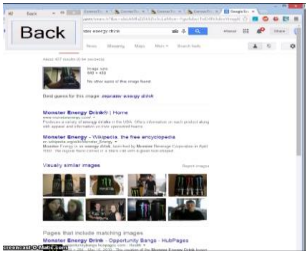




	<p style="text-align: center;"><u>Shopping Mode</u></p> <p>User Input: Image from the camera Action: Google Image Search</p>	
	<p style="text-align: center;"><u>Paint Mode</u></p> <p>User Input: Mouse Move, Pinch and click Action: Draw on custom paint application</p>	
	<p style="text-align: center;"><u>Barcode Mode</u></p> <p>User Input: Image from the camera with barcode Action: Decode barcode and display information</p>	
	<p style="text-align: center;"><u>Web Mode</u></p> <p>User Input: Mouse Move, Pinch and click Action: Browse Web</p>	
	<p style="text-align: center;"><u>Photo Mode</u></p> <p>User Input: Photo Gesture Action: Take a self shot</p>	

Table 1. UI Modes

3.4.2 UI Layout

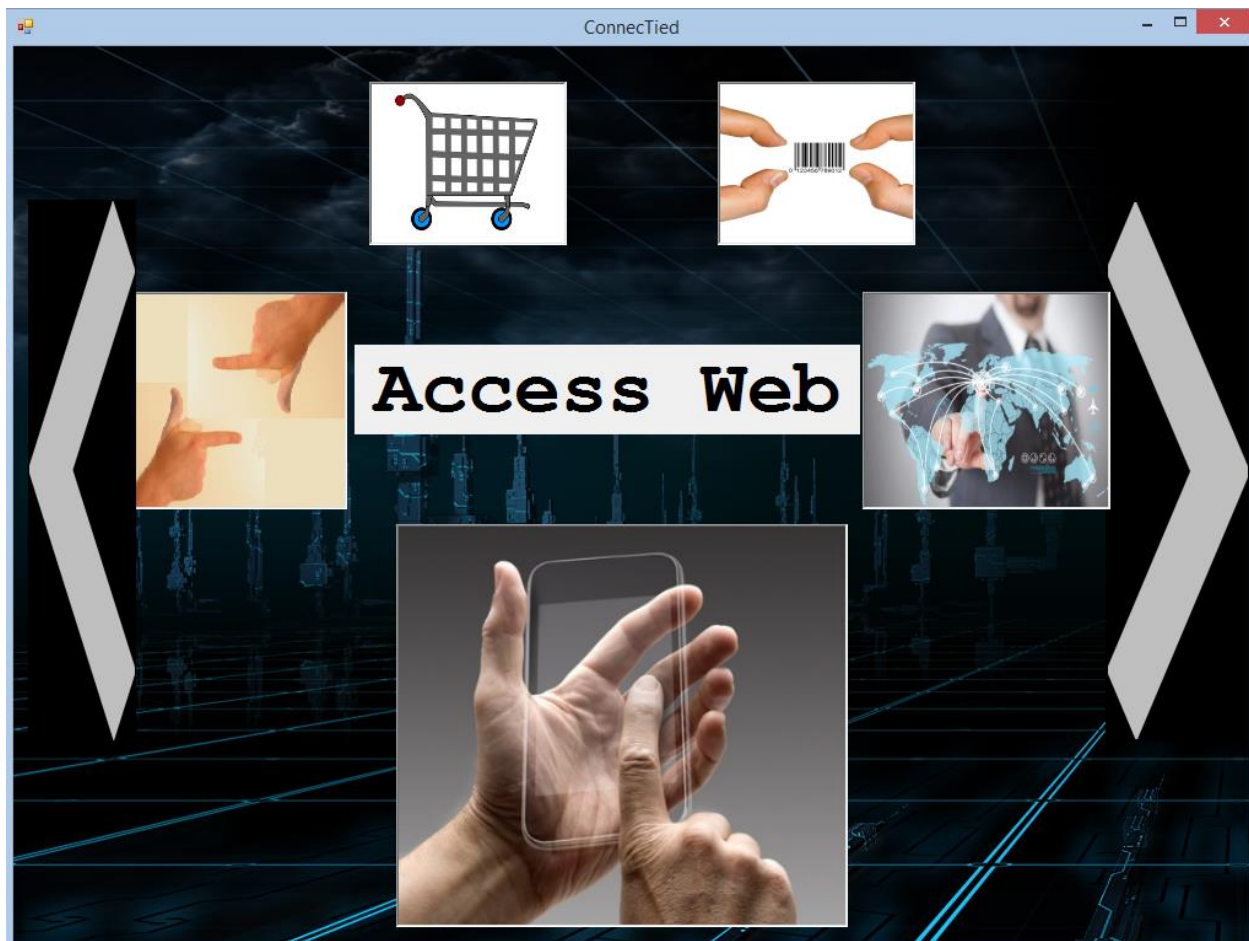


Figure 8. The User Interface Layout

Appendix A: Control flow diagram for the implemented system