Smart Switch using Eye Tracking and IOT

Premanand Kumar | p8kumar@ucsd.edu | A53225915
Prashant Singh | prs032@ucsd.edu | A53222484

Introduction:
The main underlying idea behind the project is that the visual representation of the real world could take Man-machine interaction to next level, which can be further exploited in creating user friendly IOT applications. This project aims to train the raspberry pi to understand the difference between 'seeing it' and 'not-seeing it'. When it is 'seen', the raspberry pi will take it as a visual cue to start an actuation. The project also makes sure that the device is ‘personalized' by making the actuation at predefined ‘states’ based on what it has learnt from previous instances, which it accomplishes using a Neural Network capability. Thus, we explore the possibility to deploy perception capabilities on a resource constrained embedded hardware platforms which are closer to humans in their daily life.

Relevant work and commercial Products:
As part of Machine Learning Research, many companies have come up with their own applications of implementing this concept to fit into their model of business. This has helped them getting users attention as well as leverage it as a feature to their consumers.

For example, Facebook has developed an uncanny ability to recognize our friends in photographs. Earlier, Facebook used to make us to tag our friends in photos by clicking on them and typing in their name. Now as soon as we upload a photo, Facebook tags everyone for us:

Facebook’s face recognition algorithms are able to recognize our friends’ faces after they have been tagged only a few times. It’s pretty amazing technology—Facebook can recognize faces with 98% accuracy which is pretty much as good as humans can do!

The more complex part of their algorithms, was to distinguish between very similar faces, which have minute facial differences. This problem in Machine Learning area was resolved with a method invented in 2005 called Histogram of Oriented Gradients algorithm (HOG).

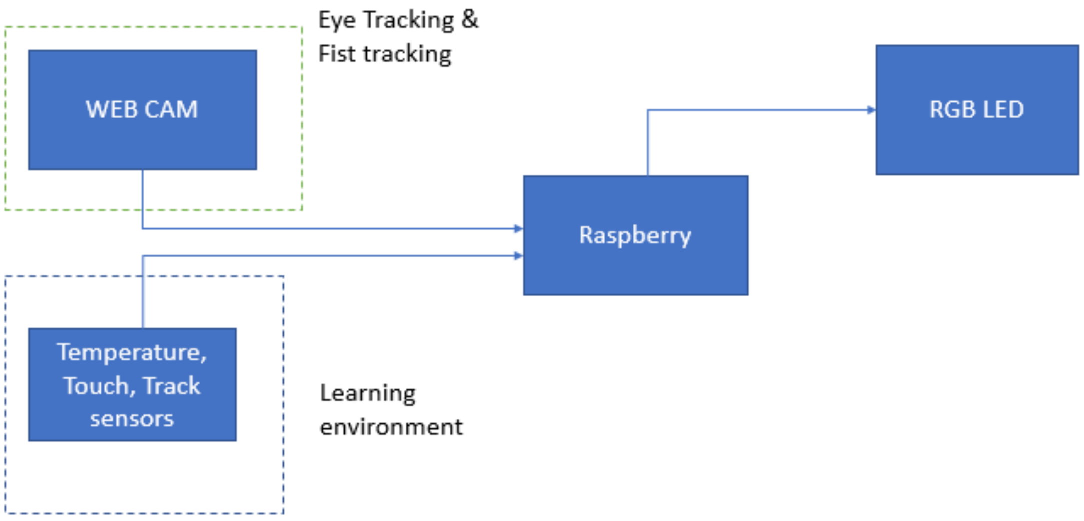
This is an example of how the algorithms can be optimized/written in a way to solve more challenging areas of machine learning.

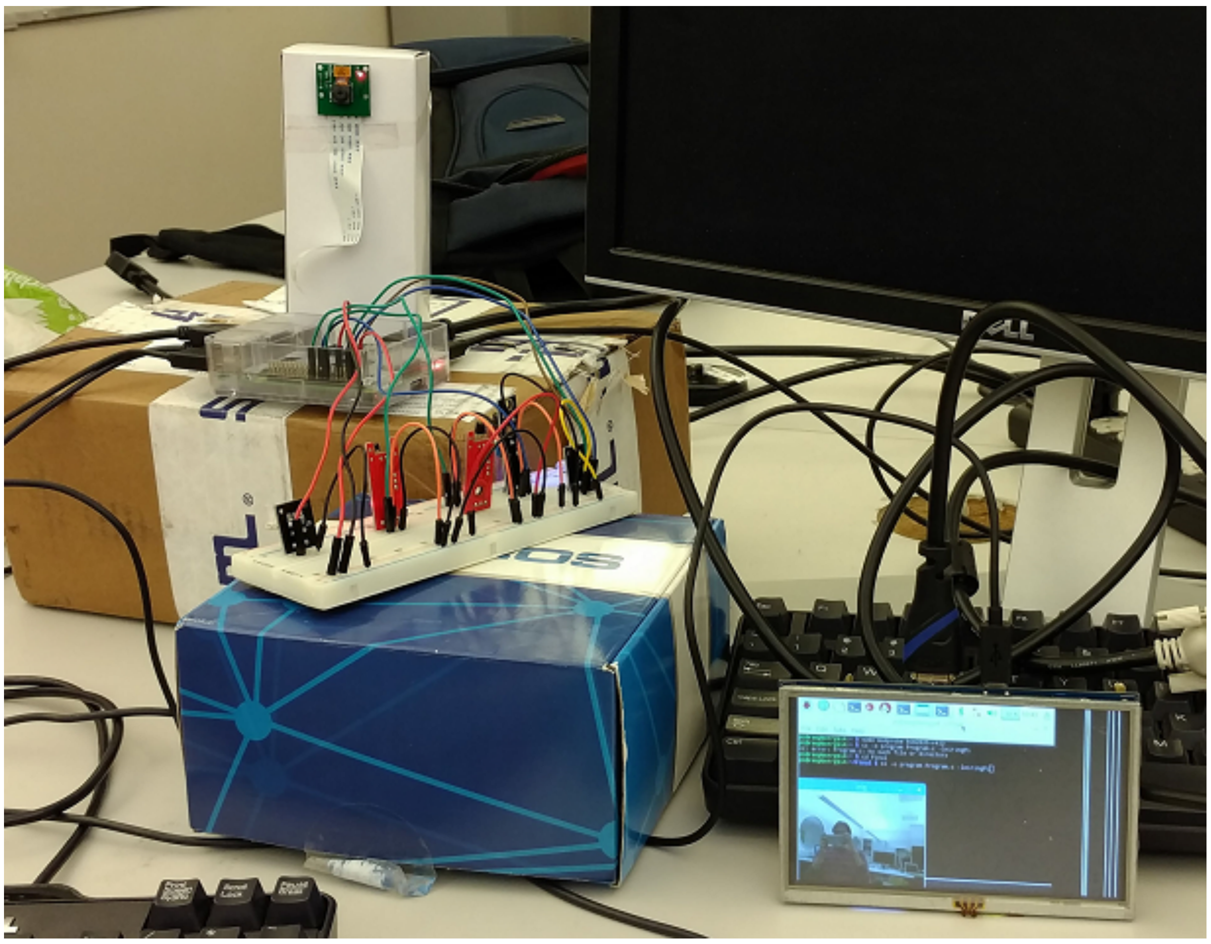
Following are various ML implementations which has made IOT as their application areas:

- Facial Recognition Home Security: This project trains the algorithm to make face recognition and opens the door automatically for the legit person. This project was part of Microsoft’s Hack the Home initiative.
- Image Search & Processing: This is very much the same how Facebook implements its ML algorithms to recognize a face. Moreover, Google took it further by implementing it as an image search engine, where the algorithms analyse the image, learn its features, and then crawls through the web to find similar images for the user.
- Bringing eyes to IOT (Image Understanding): The next level of research underway is to interpret the visuals of eye on runtime & thus making sense of what is being viewed, thereby extracting the “relevant” information and saving it as data. Bringing image understanding into digital devices will solve one of the huge problems: the data deluge—that is, how to transmit, store, and analyse all the photos and videos being recorded by people and things.

Project Set Up:

The project was set up in following way as depicted in the flow diagram as well as the snapshot from the demo:

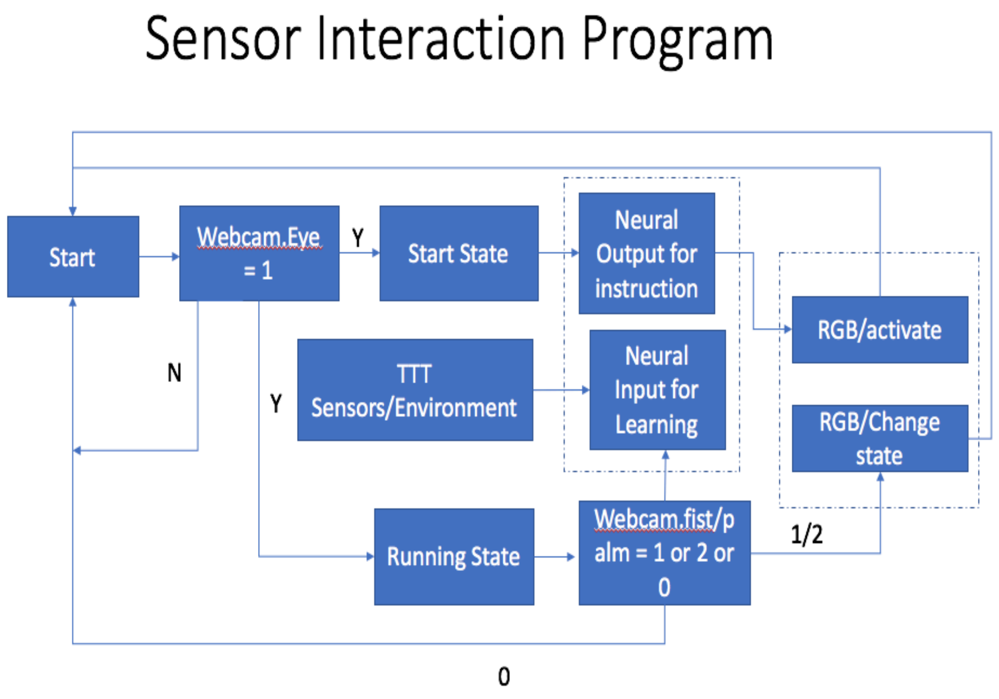




Sensor Interaction Program:

The sensor interaction program runs three different programs in the background

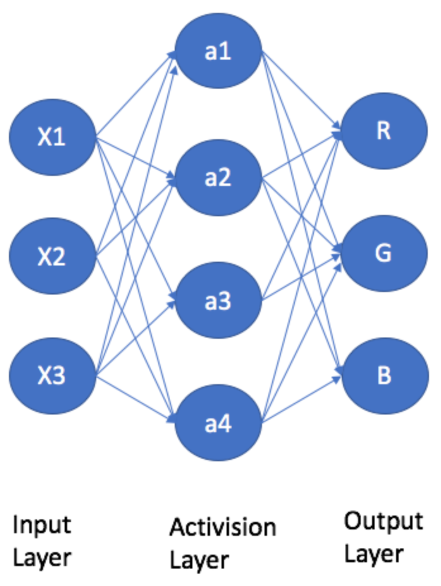
1. Webcam Switch – To detect eye and fist.
2. Neural Network - which reads the current state of the environment (as output by the main program) and learns the user preference (again, as output by the main program) to predict what the state of RGB should be for the current state of the environment.
3. Main Program – Interacts with Webcam Switch and Neural network to start the RGB, Change state of RGB. It outputs the data for Neural Network machine learning and current state of sensors for prediction.


Sensor Interaction Program
Start
Webcam.Eye = 1
Y
Start State
Neural Output for instruction
RGB/activate
N
Y
TTT Sensors/Environment
Neural Input for Learning
RGB/Change state
Running State
Webcam.fist/palm = 1 or 2 or 0
1/2
0

Neural Network:

When the webcam signals actuation by detecting the eyes, the neural network decides the state at which the RGB LED has to start based on the current state of the environment and the previous user decisions.

- **Environment** – Touch, Track, Temperature – ON/OFF
- **States for RGB** – Red Green Blue



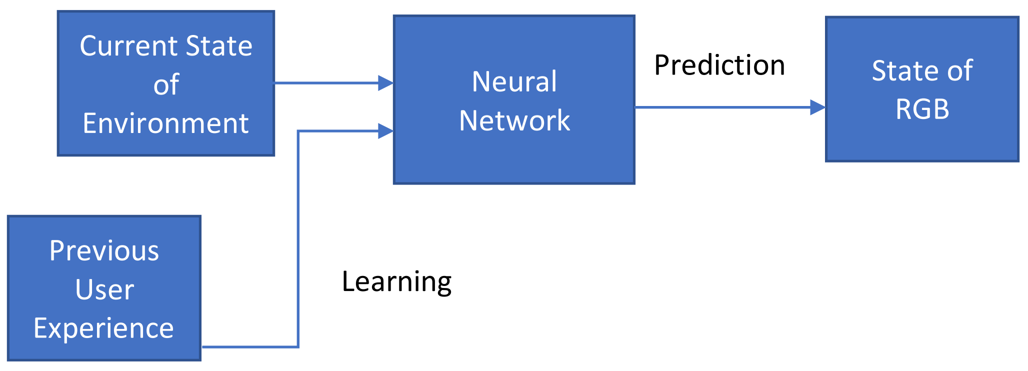
Input Layer
X1, X2, X3 – Touch, Track, Temperature

a1, a2, a3, a4 – Activation nodes

R, G, B – Red, Green, Blue of RGB state

Bias nodes are not shown.

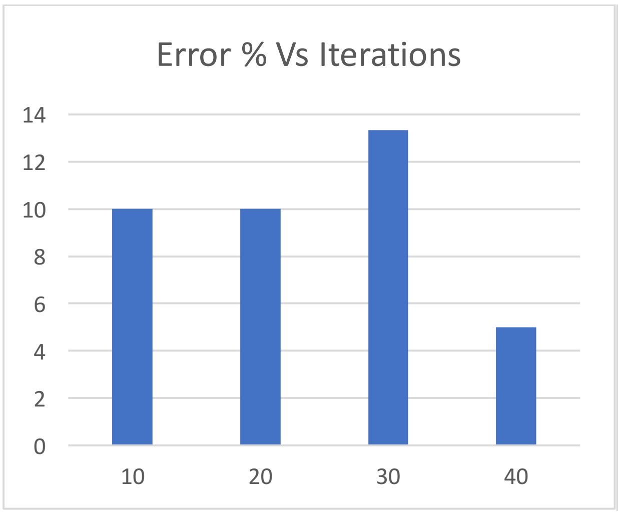
When called upon, the neural network does the following,

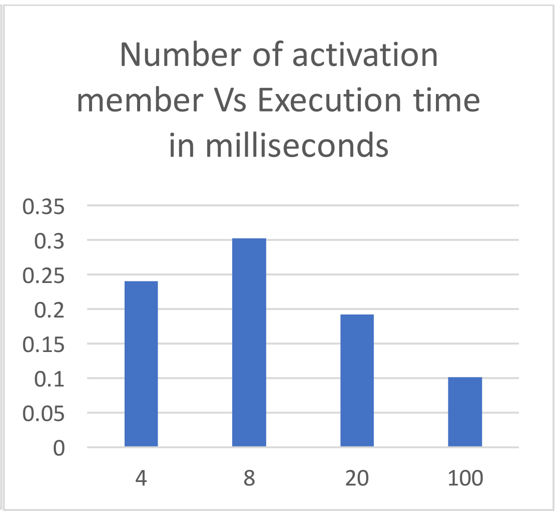


For our experiment, the neural network was trained with eight predefined input set which instructs the network what is to be done when the Webcam signals the start of RGB.

State of Touch	State of Track	State of Temperature	State of RGB
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	2
0	0	1	2
0	0	0	2

The neural network was trained with the above data and the prediction was checked for accuracy. It is observed that the neural network, because of random initialization of the weights, goes to minimum other than the desired and had the tendency to produce wrong result for the trained values. However, the error due to initialization is completely random and it does not depend on the number of iteration.

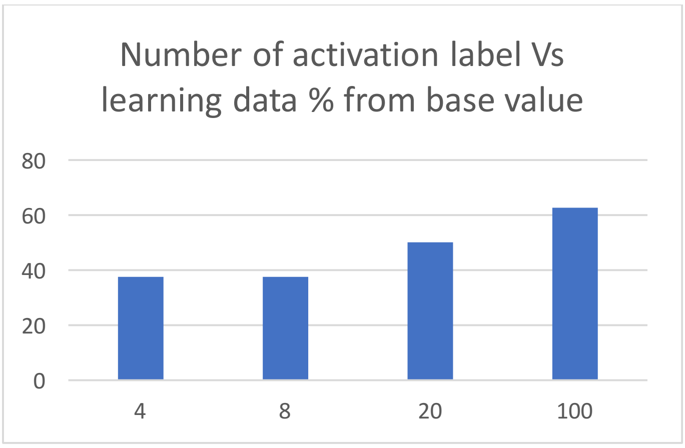




The neural network was chosen with the purpose that it has to produce reasonable result at acceptable execution speeds. So, it was checked while the number of layers were kept constant, the number of activation nodes were increased to see its effect on execution time and accuracy.
It is observed that,

a. Since the initial weights are randomized, the execution time for each iteration (for same number of activation nodes) is different. However, average execution time is plotted against various number activation nodes and was found that the execution time was not proportional to the number of activation nodes due to randomization.
b. On the other hand, it is observed that the error occurrence increases with increase in the activation. We attribute it to the fact that activation nodes are much greater as compared to the decision states. Hence, without an additional layer which could give proper weightage to the decision by the activation nodes, it is counter advantageous to add number of activation number to improve the accuracy.

As we could have observed in the training set, the data favors the use of state 1 and state 2 (1 –Red and 2 – Green) for any state of the environment condition. Suppose the user after starting the RGB (which is guaranteed to start with Red or Green), decides to change the state to Blue, the user preference is appended into the training set. Experiments are run to observe when the neural network decides that the RGB should start at blue instead of the initial instruction of red and green. It is observed that as the number of activation nodes increases, the neural network takes time to learn. It is similar to the error trend observed earlier.



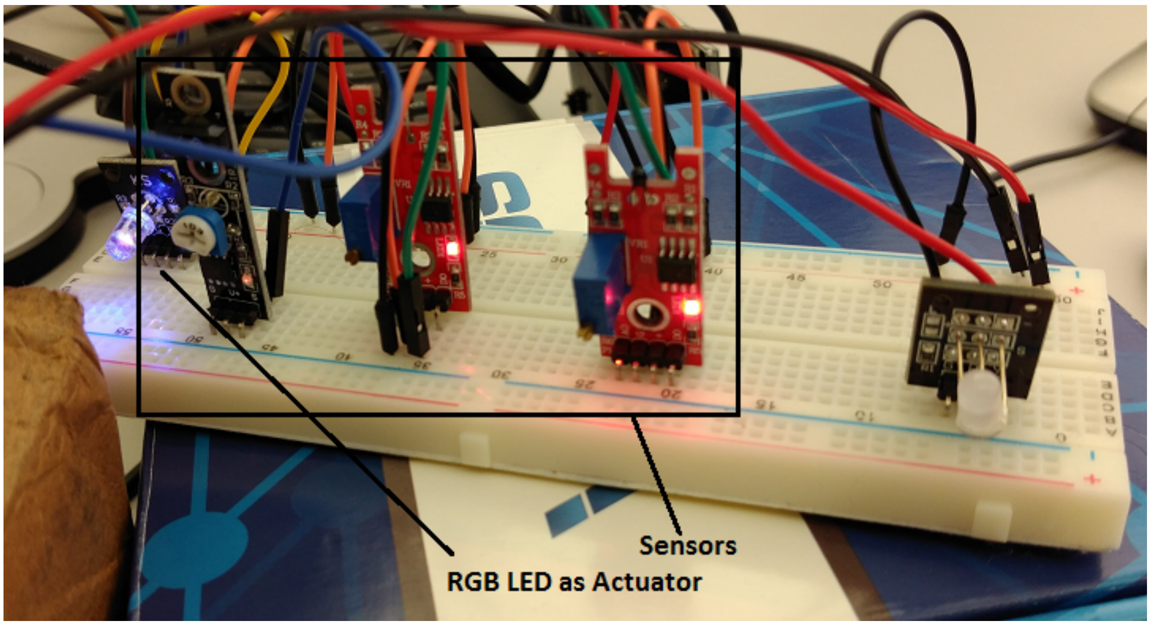
Thus, it is decided to go ahead with the chosen number of activation nodes.

The Interactive Environment:

In our environment, we have 3 sensors:

- Track
- Temperature
- Touch

For the Switch, we use Flash LED. To actuate the results, we use RGB LED. Please find below the snapshot of the environment:



Capturing the human interaction with the environment:

To have user provide the input to the sensors as a smart IOT application, we use image processing to provide input to our machine learning algorithm. This is achieved by using an open source image processing library called OpenCV.

Following are the main points related to OpenCV:

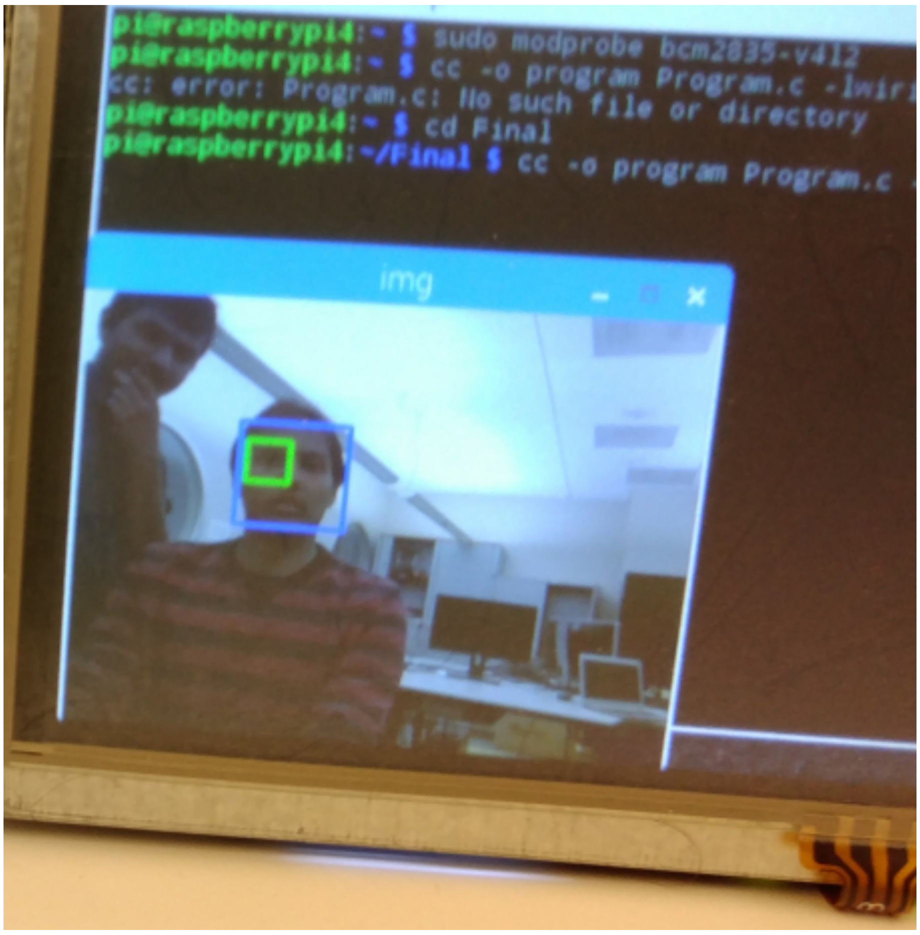
- To incorporate human computer interaction involved with this smart IOT application, we use the computer vision to capture the events creating an interactive environment.
- The computer vision library used is OpenCV, which is now a highly sophisticated open source API for working with computer graphics.
- Since its launch in 1999 by Intel Research, it has now various application areas including 2D & 3D features.
- Though OpenCV involves numerous object/human recognition features, we use two of them – face/eyes & hand cascades.

1. Eye(Pupil)/Fist/Palm Tracking:

- The two *haar cascades* used are – frontal face/eyes & fist.
- We use cv2 class to capture the video frames from the webcam.
- Using the *face cascade*, we capture the face area which is further used to capture the eyes using the *eyes cascade*.
- We use specific dimensions to mark the rectangle recognition areas around face, eyes & fists.

- More challenging task of “eye pupil tracking” was accomplished using C++ support of the library, however couldn’t be promoted to our demo project due to compatibility issues with raspberry pi’s raspbian OS.
- The output of the object recognition is saved in an output file with corresponding identity values, as expected by the machine learning code to achieve supervised learning & respective actuation of sensors.

Following snapshot shows the successful face & eye recognition:



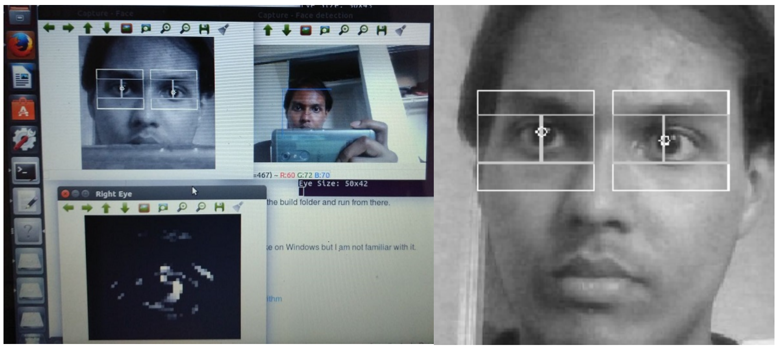
```
pi@raspberrypi4:~ $ sudo modprobe bcm2835-v4l2
pi@raspberrypi4:~ $ cc -o program Program.c -lwir
cc: error: Program.c: No such file or directory
pi@raspberrypi4:~ $ cd Final
pi@raspberrypi4:~/Final $ cc -o program Program.c
```

Table: Detection Time Vs Resolution

S. No.	Resolution	Face detection - Time Taken for detection (milliseconds)	Face detection - Maximum distance of detection in meters	Eye detection - Maximum distance of detection (meters)
1	320X240	210	1.5	1.1
2	640X480	570	2	1.5
3	800X600	1135	>2	2
4	1024X600	2545	>2	2
5	1024X768	3327	>2	2.0-2.2
6	1280X720	3460	>2	2.2-2.5

Pupil Tracking:
Along with eye tracking, we went ahead to accomplish further complex task of tracking eye ball i.e eye pupil which can be another way of interacting with the sensors. This was accomplished using the C++ version of the OpenCV API. Following are some points related to this development:

- This used the OpenCV supported with C++ libraries.
- The code was worked upon Fabia Timm's algorithm which marks specific image recognition parameters with which the high accuracy of pupil's movement is obtained.
- The algorithm: "Accurate eye centre localisation by means of gradients," – The Paper.
- Here are few screenshots from development on Intel architecture & Ubuntu environment:



Challenges encountered:
While working on this project we encountered many problems. Among these many were resolved, however few issues persisted and remain to be dealt with. Among the significant ones, we have following points to be highlighted:

- The first major challenge which we came across was the eye's pupil tracking, which involved highly complex OpenCV libraries & their use along with their compatibility with C++.
- This complexity coupled with library code's strong binding with Intel's architecture was one obstacle which forced us to go with just eye tracking which we accomplished using OpenCV's support of Python.
- Example of this complexity include the use of custom data types like "GLfloat", "GLdouble" etc which are not supported by the ARM architecture and has to be replaced with the native types. (As suggested by Prof. Rosing, we will further explore this by using Neon library to resolve this issue)
- The other challenge faced was distinguishing fist recognition with the palm.

References:

1. **https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/**
2. **http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf**
3. **https://developer.microsoft.com/en-us/windows/iot/docs/HackTheHome.htm**
4. **http://spectrum.ieee.org/view-from-the-valley/computing/embedded-systems/bringing-eyes-to-the-internet-of-things**
5. https://www.coursera.org/learn/machine-learning Andrew NG – Machine learning neural network for hand writing recognition.
6. Sensor interaction program from Individual project.