Project Touch-Me-Not

Touch-less Display Interfaces on Edge

Enabling HCI on Edge: Multi-threaded Gesture & Sound Control of kiosks with Intel OpenVINO AI models. Eye Wink & Mouth Aspect with numerical models

Project Overview and Demo

You can **watch the Video Description and Project Demo here:** https://www.youtube.com/watch?v=Wt8VuhWN5Oc

Detailed Project Blog here: https://medium.com/@AnandAI/touch-less-display-interfaces-on-edge-be8dc277c5b8

Problem Definition

Interactive public kiosks are now so widely used viz. Banking (ATMs), Airport (Check-in), Government (e-Governance), Retail (Product Catalogue), Healthcare (Appointment), Schools (Attendance), Corporate (Registration), Events (Information) and the list goes on. **While businesses move towards kiosks to better service delivery, touch-free interaction of all public devices has become an imperative to mitigate the spread of ubiquitous Corona virus.**

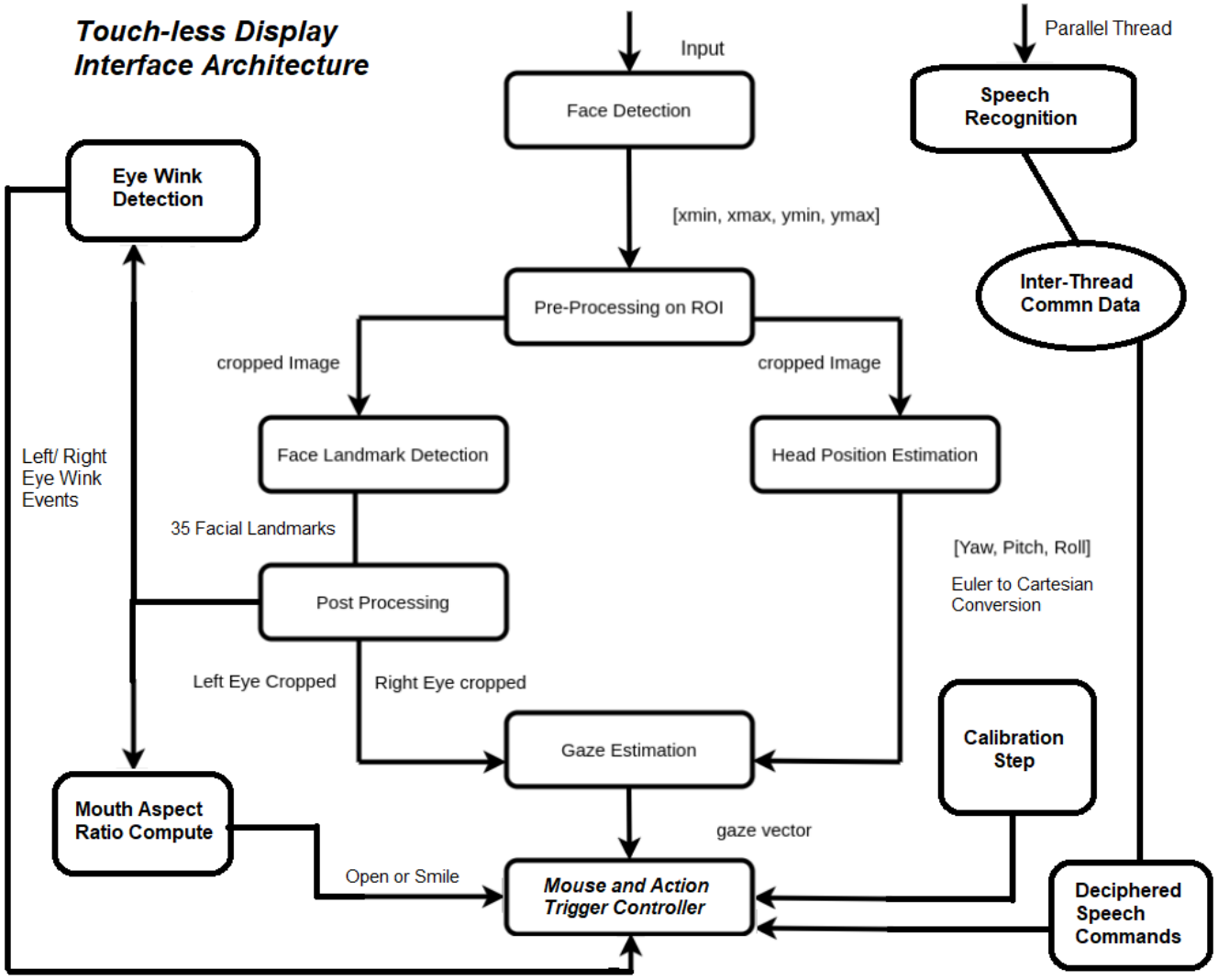
Gesture or Speech Navigation might seem to address the above, but such devices are resource constrained to analyze such inputs. Have you noticed your mobile voice assistant, be it Siri or GAssist, gives up when mobile goes offline? Your voice-enabled car infotainment system fails to respond, while you drive remote roads. Even a conventional computer won't be able to run multiple AI models concurrently.

Ain't it nice to do it all on your device itself? Imagine a bed-side assistant device which can take visual or voice cues from bedridden patients. This is possible with the advent of Intel OpenVINO. It enables and accelerates deep learning inference from the edge, by doing hardware-conscious optimizations. OpenVINO supports CPU, iGPU, VPU, FPGA and GNAs. If you wanna get your hands wet, a Raspberry Pi along with Intel Movidius NCS 2 would be your best bet to toy with.

In this project, we will try to build a Human-Computer Interaction (HCI) module which intelligently orchestrates 5 concurrently-run AI models, one feeding onto another. While the models for face detection, head pose estimation, facial landmarks computation and angle of gaze estimation identify gesture control inputs, another thread is deployed to run

offline speech recognition, which communicates with the parent process to give parallel control commands based on user utterance, to assist and augment gesture control.

Architecture Diagram



How To run:

```
python3 noTouchKiosk.py {command line arguments}

Example: python3 noTouchKiosk.py -f ../models/face-detection-adas-
0001/FP16/face-detection-adas-0001.xml -l ../models/facial-landmarks-35-adas-
0002/FP16/facial-landmarks-35-adas-0002.xml -hp ../models/head-pose-
estimation-adas-0001/FP16/head-pose-estimation-adas-0001.xml -ge
```

```
../models/gaze-estimation-adas-0002/FP16/gaze-estimation-adas-0002.xml -i
../bin/demo.mp4 -it cam -d CPU -vh False -vg True -vf True
```

You can change the model precision and flags given as parameters. vh, -vg and -vf are the visualization debug flags.

The code allows the user to set a flag that can display the outputs of intermediate models. For instance, -vh to visualize head pose results and -vg to visualize gaze model outputs.

Command Line Arguments

```
-f:  Path to .xml file of Face Detection model
-l:  Path to .xml file of Facial Landmark Detection model
-hp: Path to .xml file of Head Pose Estimation model
-ge: Path to .xml file of Gaze Estimation model
-i:  Path to video file or enter cam for webcam
-it: Provide the source of video frames
-d:  Provide the target device: "CPU, GPU, FPGA or MYRIAD is acceptable."
```

IMPORTANT: The output of the project depends on the intial calibration step where the system recognized the extremities of the screen and corresponding gaze angles. Upon execution, the system will direct you to look at the top right corner and then the bottom left corner of your computer screen. Based on the corresponding gaze angles the system is capable to compute the intermediate x, y coordinates using interpolation techniques.

Note:

- **If your mouse pointer behaves improper, then calibration step is the most likely problem.** Please make sure that your face is properly lit and positioned approximately to middle of the screen so that the gaze angles would make sense of left, right, top and bottom.
- The calibration step is optimized for cam input where the user can look at the screen corners. If video is given as input then the mouse will be controlled according to gaze but the direction of mouse pointer can differ. This happens because the person in the video is not looking at the corner of the screen.
- On a different note, if you visualize the output of head pose model then it gives angle of vision but then the orientation of eye balls are not taken into consideration. Instead, when the eyes are cropped and fed into gaze estimation model, the angle of sight is correctly estimated, considering both head pose as well as location of eye ball.

Project Set Up and Installation

Install OpenVino

```
wget http://registrationcenter-
download.intel.com/akdlm/irc_nas/16612/l_openvino_toolkit_p_2020.2.120.tgz
tar -xvf l_openvino_toolkit_p_2020.2.120.tgz
cd l_openvino_toolkit_p_2020.2.120
```

```
sed -i 's/decline/accept/g' silent.cfg
sudo ./install.sh -s silent.cfg
```

Create a Virtual env

```
python3 -m venv edge
source edge/bin/activate
```

Project dependencies

```
pip3 install -r requirements.txt

To download the required models:
python
/opt/intel/openvino/deployment_tools/tools/model_downloader/downloader.py --
name "face-detection-adas-binary-0001"
python
/opt/intel/openvino/deployment_tools/tools/model_downloader/downloader.py --
name "facial-landmarks-35-adas-0002"
python
/opt/intel/openvino/deployment_tools/tools/model_downloader/downloader.py --
name "landmarks-regression-retail-0009"
python
/opt/intel/openvino/deployment_tools/tools/model_downloader/downloader.py --
name "head-pose-estimation-adas-0001"
python
/opt/intel/openvino/deployment_tools/tools/model_downloader/downloader.py --
name "gaze-estimation-adas-0002"
python
/opt/intel/openvino/deployment_tools/tools/model_downloader/downloader.py --
name "face-detection-adas-0001"
```

Benefits & Features

a) To control **public kiosks without touching the device**
b) To control **edge devices with gestures and also sound**
c) To efficiently deploy complex AI solutions at the edge
d) Usage of **efficient Numerical Algorithms along with hardware optimized OpenVINO models**
e) Use visual interaction where touch is not possible or advisable
f) Multiple **facial features are identified and mapped to specific action triggers**
g) **Sound control is done on the edge**, without needing the device to be online.
h) Demonstrates the capability of Intel OpenVINO to handle **multiple Edge AI models in sequence and in parallel.**
i) Many control inputs are also sourced to demonstrate the flexibility.
j) Stickiness feature can be tuned to set the minimum movement of head of gaze to be ignored.
k) Gesture-action mapping can also be modified. Even the control mode can be switched with gestures.

Social impact

This project has got **high social impact index.** Our most pressing concern, at present, is to control the spread of COVID. To facilitate this, **we need to allow touch free interaction of all public devices.** To sanitize hand, right before and after using serums is not a practical solution, in long run. People may fail to adhere. Rather a practical solution is to avoid touch to the extend possible. **This project addresses this problem using multiple AI models and using OpenVINO to deploy them on the edge devices. This will significantly improve the health and safety standards among the public.**

However, the public use of this project is not limited to the spread of Corona vius. To quote another healh care use case, **we can use the same project to take visual commands or help requests from a bed-ridden or a physically challenged patient, be it hospital or home. **

Project Documentation

Control Modes

There are 4 control modes defined in the system, to determine the mode of user input. We can switch between control modes using gestures.

- **Control Mode 0:** No Control Gesture and Sound Navigation is turned off
- **Control Mode 1:** Gaze Angle Control Mouse moves along with angle of eye gaze (faster)
- **Control Mode 2:** Head Pose Control Mouse moves with changing head orientation (slower)
- **Control Mode 3:** Sound Control Mouse slides in 4 directions and type based on user utterance

Calibration Step

To translate the **3D gaze orientation angles to 2D screen dimension, the system has to know the yaw and pitch angles corresponding to opposite corners of the screen.** Given these 2 angles of opposite corners, we can interpolate the (x, y) location in the screen for intermediate (yaw, pitch) angles.

Therefore, the user will be prompted to look at opposite corners of the screen, when the application is initiated. Such a calibration step is needed to map the variation in gaze angles to the size and shape of the screen, in order for the "gaze mode" to function properly.

Without calibration also the system can function, albeit at the expense of generality. To demonstrate, the relative change in head orientation is taken as the metric to move mouse pointer, when the system is in "head pose" mode.

Usage of OpenVINO Toolkit: Improvements or Optimizations

Various benchmarks are done with different model variants to select the optimal model combo for pipeline. The code has also been optimized with the use of Intel VTune.

Benchmark Modes

Used the below parameters for corresponding benchmarks:

```
# To parse the video file given - all FP16 models
 arg = '-f ../models/face-detection-adas-0001/FP16/face-detection-adas-
0001.xml -l ../models/facial-landmarks-35-adas-0002/FP16/facial-landmarks-35-
adas-0002.xml -hp ../models/head-pose-estimation-adas-0001/FP16/head-pose-
estimation-adas-0001.xml -ge ../models/gaze-estimation-adas-0002/FP16/gaze-
estimation-adas-0002.xml -i ../bin/demo.mp4 -it video -d CPU -vh False -vg
True -vf True'.split(' ')

# To take input from the webcam but with FP32 gaze & Landmark detection
models
 arg = '-f ../models/face-detection-adas-0001/FP16/face-detection-adas-
0001.xml -l ../models/facial-landmarks-35-adas-0002/FP32/facial-landmarks-35-
adas-0002.xml -hp ../models/head-pose-estimation-adas-0001/FP16/head-pose-
estimation-adas-0001.xml -ge ../models/gaze-estimation-adas-0002/FP32/gaze-
estimation-adas-0002.xml -i ../bin/demo.mp4 -it cam -d CPU -vh False -vg True
-vf True'.split(' ')


# To take input from webcam but with INT8 Face detection & FP32 gaze &
Landmark detection models
 arg = '-f ../models/face-detection-adas-0001/FP32-INT8/face-detection-adas-
0001.xml -l ../models/facial-landmarks-35-adas-0002/FP32/facial-landmarks-35-
adas-0002.xml -hp ../models/head-pose-estimation-adas-0001/FP16/head-pose-
estimation-adas-0001.xml -ge ../models/gaze-estimation-adas-0002/FP32/gaze-
estimation-adas-0002.xml -i ../bin/demo.mp4 -it cam -d CPU -vh False -vg True
-vf True'.split(' ')
```

Benchmark Results

```
a) Benchmark by taking video input. All models are FP16
FPS = 15.29160590328414
Inference Time of 4 models = 0.08967375755310059

b) Benchmark by taking webcam as input. All models are FP16
Inference Time of 4 models = 0.031467437744140625
FPS = 17.09894984019307

c) Webcam as input. Using FP16 for Face Detection & Head pose and FP32 for
Gaze & Landmark detection models
Inference Time of 4 models = 0.04231882095336914
FPS = 14.50613543612091
```

```
d) Webcam as input. Using INT8 for Face Detection and remaining are FP16
models
Inference Time of 4 models = 0.028314828872680664
FPS = 18.337839492138997

e) Webcam as input. Using INT8 for Face Detection & FP32 for Gaze & Landmark
detection models and FP16 for head pose model.

Inference Time of 4 models = 0.03919792175292969
FPS = 15.08791291804411
```

It is seemingly clear that **increase in number of bits slows down the inference and lower the FPS.** But as with all AI tasks, its always a tradeoff between required accuracy and minimum speed.

For face detection, INT8 model is giving good accuracy but landmark detection and gaze require maximum accuracy for accurate mouse control. Headpose require reasonable accuracy, hence FP16 is used.

FP16 models commonly regarded as the mid-path between accuracy and speed, as compared to FP32 and INT8. But from the above analysis, it seems like Benchmark (e) gives good balance between accuracy and speed for this project.

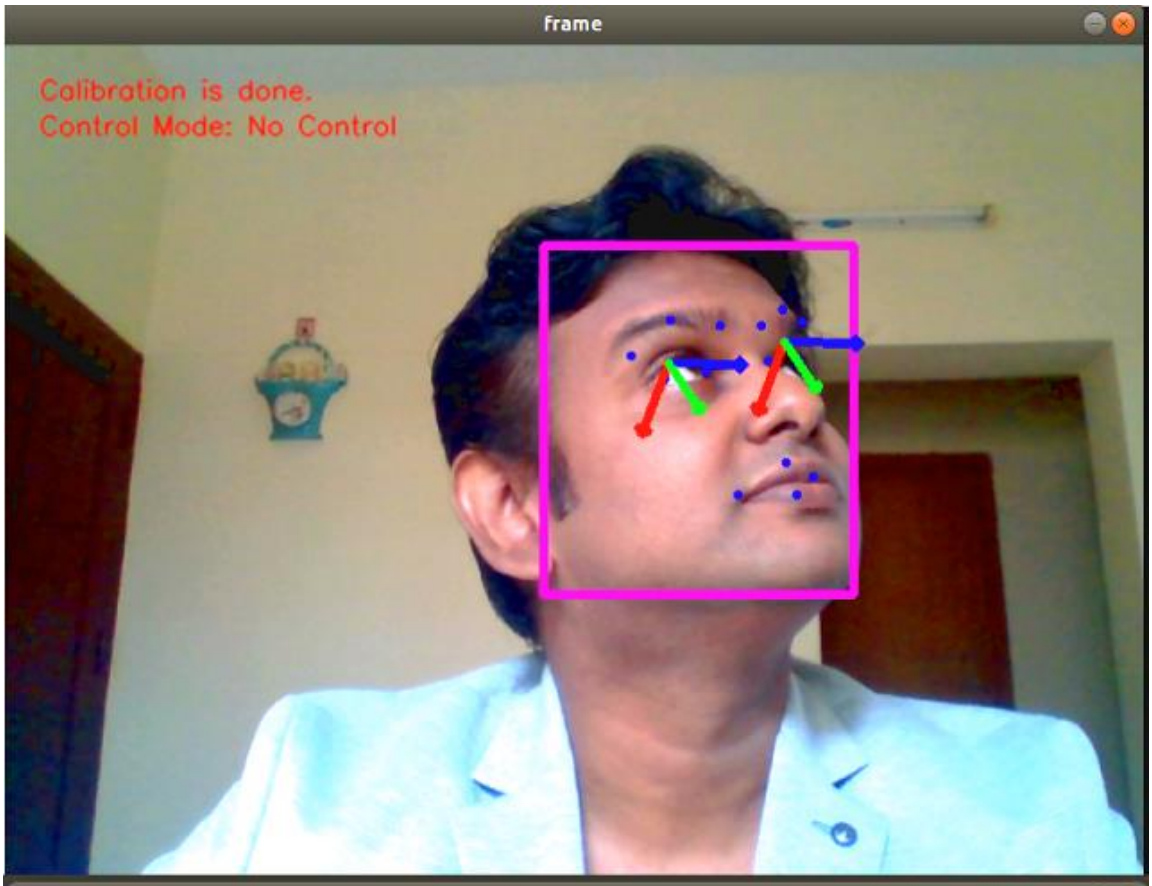
Finally, **Intel VTune profiler is used to find hotspots and optimize the application code.** A shell script vtune_script.sh is fed into the VTune GUI which initiates the project with suitable arguments. The lines in the code which takes more time are identified and possible ones are optimized. For instance, the curve_fit algo seemed to take much time and iterations with 'dogbox' optimize function, which was then replaced with 'lm' method. This has improved efficiency significantly.

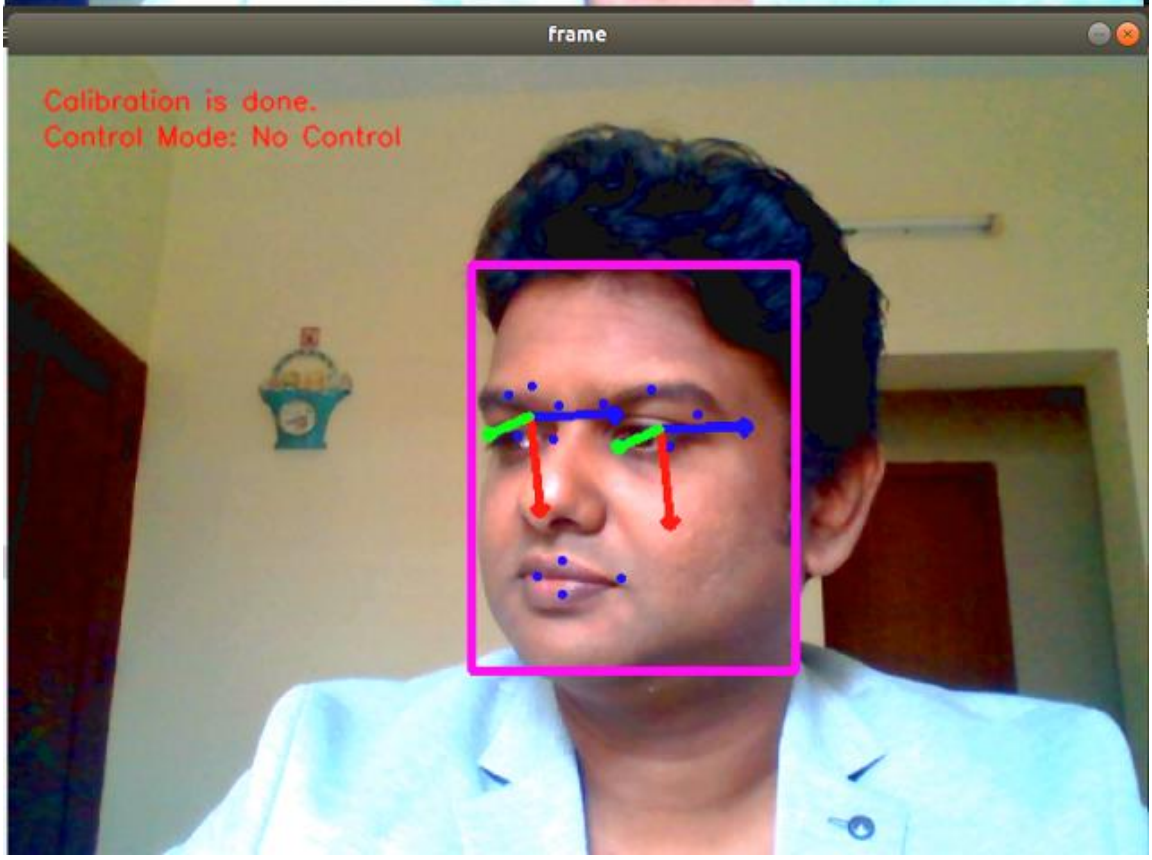
Models Used

Gesture Detection Pipeline Models

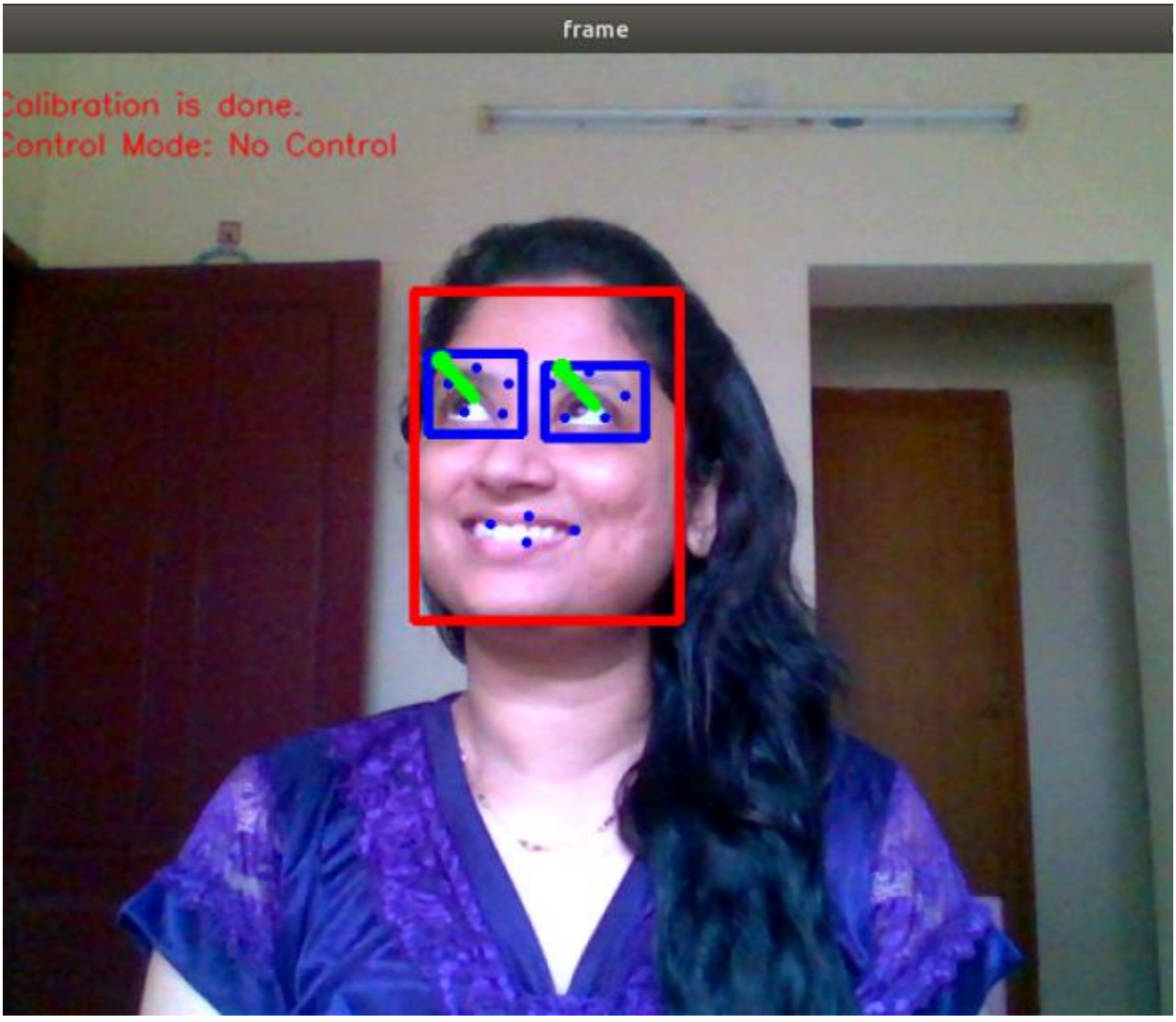
Four Pre-trained OpenVINO models are executed on the input video stream, one feeding onto another, to detect a) Face Location b) Head Pose c) Facial Landmarks and d) Gaze Angles.

1. Face Detection: A pruned MobileNet backbone with efficient depth-wise convolutions is used. The model outputs (x, y) coordinates of the face in the image, which is fed as input to steps (b) and (c)
2. Head Pose Estimation: Model outputs Yaw, Pitch and Roll angles of head, taking face image as input from step (a)


frame
Calibration is done.
Control Mode: No Control


frame
Calibration is done.
Control Mode: No Control

3. Facial Landmarks: a custom CNN used to estimate 35 facial landmarks. This model takes cropped face image from step (a) as input and computes facial landmarks, as above. Such a detailed map is required to identify facial gestures, though it is double as heavy in compute demand (0.042 vs 0.021 GFlops), compared to the Landmark Regression model, which gives just 5 facial landmarks.
4. Gaze Estimation: custom VGG-like CNN for gaze direction estimation.
   The network takes 3 inputs: left eye image, right eye image, and three head pose angles - (yaw, pitch, and roll) - and outputs 3-D gaze vector in Cartesian coordinate system.



Speech Recognition Models

To decode sound waves, we use OpenVINO Feature Extraction & Decoder Library which takes in and transcribe the audio coming from the microphone. We have used the speech library as mentioned in OpenVINO toolkit to run speech recognition on the edge, without going online.

Post-Processing Model Outputs

To feed one model output to another model as input, the return values of each model need to be decoded and post-processed. For instance, to determine gaze angle, the head orientation need to be numerically combined with the vector output from gaze model.

Similarly, the facial landmarks model returns ratio of input image size. Hence, we need to multiply output by image width and height to compute (x, y) coordinates of 35 landmarks.

While output of facial landmark and gaze estimation models can be easily post-processed as above, the output of head pose estimation model has to converted from Euler angles to rotation matrices.

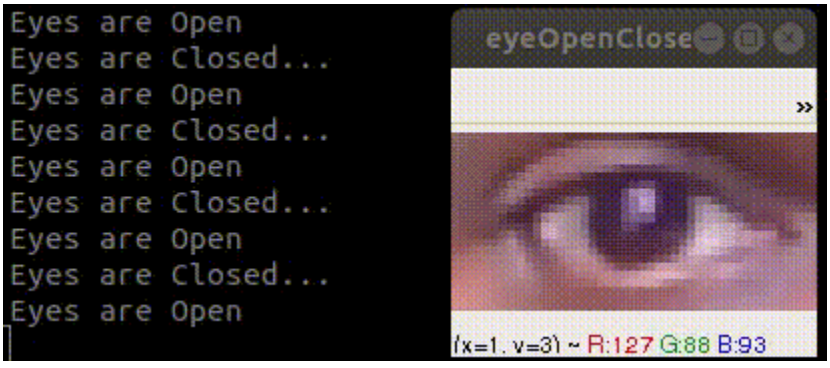
$$T_{0,3} = T_{0,1}T_{1,2}T_{2,3} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} =$$

$$\begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) \end{bmatrix}$$

Numerical Models

Eye Wink Detection

To use a kiosk, you also need to trigger events, such as 'Left Click', 'Right Click', 'Scroll', 'Drag' etc. In order to do so, **a set of pre-defined gestures need to be mapped to each event, and be recognized from the visual input.** Two events can be mapped to 'wink' event of left and right eye, but they need to be identified as 'wink'.



You can easily notice that the number of white pixels will suddenly increase when the eyes are open, and decrease when closed. We can just count the white pixels to differentiate open vs closed eye.

But in real world, above logic is not reliable because white pixel value itself can range. We can always use Deep Learning or ML techniques to classify but its advisable to use a numerical solution, in the interest of efficiency, especially when you code for edge devices.

Lets see how to numerically detect winks using signals in 4 steps!

1. Calculate frequency of pixels in range 0–255 (histogram)
2. Compute spread of non-zero pixels in the histogram. When an eye is closed, the spread will take a sudden dip and vice-versa.
3. Try to fit a inverse sigmoid curve at the tail-end of the above signal.
4. If successful fit is found, then confirm the 'step down' shape of fitted curve and declare it as 'wink' event. (no curve fit = eye is not winking)

Algorithm Explanation:
If above steps are not clear, then see how the histogram spread graph falls, when an open eye is closed.

You can imagine that the curve would take shape of 'S' when the eye is opened for a few seconds. This can be mathematically parameterized using a sigmoid function. But since we need to detect 'wink' event shown above, the shape of the curve will take the form of an inverse sigmoid function. To flip the sigmoid function about the x-axis, find f(-x).

Thus, if any similar shape is found by parametric curve fit algorithm, at the tail end of the histogram spread curve, then we can call it a 'wink'. The curve fit algo tries to solve a nonlinear least-squares problem.

Note: An efficient way to compute the above can be, i) Consider strip of 'n' recent values in Histogram Spread. ii) Compute the median & std of 'k' values in the front and tail end of strip. iii) If difference in median > threshold and both std < threshold, then detect eye wink event, as it's most likely an inverse sigmoid shape.

Alternatively, we can also use the below algo to find eye winks.

a) Take the **first differential of Histogram Spread values**
b) **Find the peak in the first differential values to find sudden spike**
c) **Find reflection of the signal and find peak to find sudden dip**
d) If peak is found in both the above steps, then its just a blink
e) If peak is found only in reflection, then its a wink event.

The above method is more efficient than curve fitting, but can lead to many false positives, as peak detection is not always reliable, especially at low light. Middle of the road approach would be to use median and standard deviation to estimate the shape of the curve.

Mouth Aspect Ratio (MAR)

Eye Aspect Ratio (EAR) is computed in this classic facial landmark paper to determine eye blinks.

Inspired by EAR, **we can compute MAR based on the available 4 mouth landmarks **obtained from OpenVINO model.

$$\frac{|p_9^x - p_8^x|}{|p_{11}^y - p_{10}^y|}$$

Two gesture events can be identified using MAR:

1. **if MAR > threshold**, then person is smiling
2. **if MAR < threshold**, then mouth is wide open

We have liberty to attach 2 commands corresponding to these two gestures.

Threading and Process-Thread Communication

To enhance control, we can enable sound based navigation also, along with gesture control. However, system then needs to continuously monitor user utterances to identify commands while it is analyzes image frames from input video stream.

Naturally therefore, **it is prudent to run the speech recognition model in a different thread and let the child-thread communicate with the parent process. The child thread will recognize vocal commands to move the mouse or to write on the screen and pass it on to the parent using Queue data structure in Python**.

The parent process will run all the above AI models and the computation required for gesture recognition, to enable head and gaze control modes. Thus, it is possible to take gesture and sound control commands in parallel, but for the sake of usability, in this project we chose to take sound commands separately in Control Mode 3.

Speech Recognition

To decode sound waves, we use OpenVINO Feature Extraction & Decoder Library which takes in and transcribe the audio coming from the microphone. We have used the speech library as mentioned here to run speech recognition on the edge, without going online.

As the recognition model is optimized at the expense the accuracy, some tweaks are required to identify spoken command. Firstly, we limit the command vocabulary to say, 'up', 'down', 'left' &

'right' only. Secondly, similar sounding synonyms of command words are stored in a dictionary to find the best match. For instance, 'right' command could be recognized as 'write'.

The function is so written that commands and also synonyms can easily be extended. To enable user entry, speech to write function is also enabled. This has enabled to user to type in alphabets and numbers. **Eg:** PNR number.

Stickiness Feature

The gaze of an eye or pose of a head will continuously change at least a bit, even if unintended. Such natural motions should not be considered as a command, otherwise the mouse pointer will become jittery. Hence, **we introduced a 'stickiness' parameter within which the motion is ignored. This has greatly increased the stability and usability of gesture control.**

Complete BOM

This is a **software project** though the models used and the code written can be deployed on the edge, given the device support Intel Architecture as specified in OpenVINO Documentation.

Here is the **complete list of Software, Models and Tools used:**

1. Python 3.6 and its libraries, espcially PyAutoGUI for navigation.
2. Intel OpenVINO 2020
3. These are the OpenVINO Models Used:
    - **Detailed Facial Landmark Detection**: "facial-landmarks-35-adas-0002"
    - **Head Pose Estimation**: "head-pose-estimation-adas-0001"
    - **Gaze Estimation**: "gaze-estimation-adas-0002"
    - **Face Detection**: "face-detection-adas-0001"
    - **Speech Recognition**:
    - OpenVINO Inference Engine plugin
    - OpenVINO Feature Extraction Library
    - OpenVINO Decoder Library
4. Numerical Models
    - **Inverse Simoid Curve Fitting using Non-Linear Least Squares**
    - **Mouth Aspect Ratio derived from EAR concept from a Research Paper [3]**
    - **Peak Finding Algorithm**
    - **Statistical Analysis**
5. Tools Used:

    - Intel VTune
    - Shell Script
    - http://fooplot.com as Math Visualization Tool
    - Mobile as Light Source

6. Laptop with Intel CPU and Webcam.

Creative Elements (20 points)

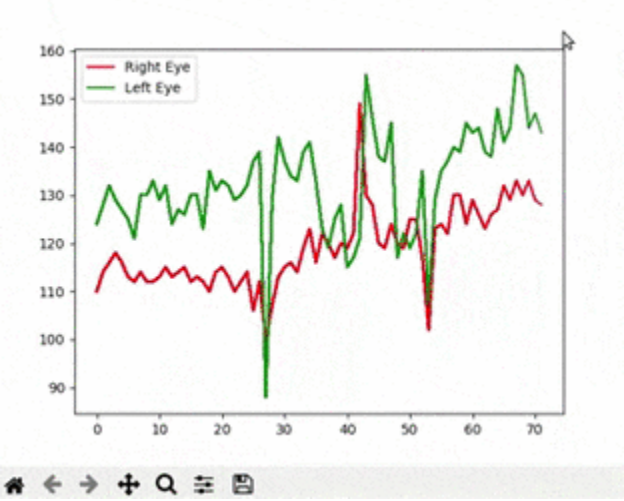
1. **Innovative use of Numerical Algorithms** What makes the project unique is the innovative use of numerical algorithms **to replace AI models**, in line with the advocacy in my blog here: https://towardsdatascience.com/the-power-of-mathematical-ingenuity-49c7b6cfe05e

   This idea is especially important when OpenVINO optimized models are deployed on the edge. The models are already optimized to the extend possible and the computation overhead is with the remaining code. Here, we need to use efficient statistical analysis or numerical algorithms to save the compute. Why to use a sledgehammer to crack a nut?

   a) **Eye Wink Detection**

   - **Inverse Simoid Curve Fitting** - Imaginately solved the problem using Non-Linear Least Squares
   - **Peak Detection in first differential signals** - self written algorithm

   b) **Statistical Analysis of non-zero Histogram Spread** - innovative way to efficiently detect and differentiate blink/wink. c) **Mouth Aspect Ratio to detect smile and yawn** (idea derived EAR concept as found in Research Paper [3])



2. Pipeline of OpenVINO models to solve the dire need of **distancing and safety during COVID**. The same solution can be used in health care as well for physically challenged or bed ridden or elderly.
3. **Threading and Process-Thread Communication**
   To continuously monitor user utterance, I have designed **a parallel thread to listen to microphone.** But since it is a optimized edge model, the accuracy of OpenVINO pre-trained model was not great. To solve this problem I have **introduced some custom tweaks to identify commands. Once the command is identified, it is put into a shared queue, from where the parent process will collect and execute.**

4. **Sound Tweak** The **similar sounding synonyms of command words are stored in a dictionary to find the best match.** For instance, 'right' command could be recognized as 'write'

   The function is so written that commands and also synonyms can easily be extended. To enable user entry, speech to write function is also enabled. Even the numbers and alphabets are converted in typing mode.

5. **Calibration Step**
   It was found that the position and size of the interface as well as the location and angle of user with the screen impacts the gaze vector, a lot. Hence, **a calibration step was introduced to ask the user to look at the opposite corners of the screen to get the corresponding yaw and pitch vectors.** Based on the input from these 2 corners, the gaze vectors corresponding to all the 4 corners are calculated. This information is **used to interpolate (x, y) mouse location when the user is looking at an intermediate gaze vector location based on angles.** It was a fun to code this algorithm.
6. **Intricate Math involved**

   There is good amount of math involved not only in the numerical models or calibration, but also in post-processing of all models, especially head pose model.

   The head pose model returns only the attitude, i.e. Yaw, Pitch and Roll angles of the head. To obtain the corresponding direction vector, we need to compute the rotation matrix, using attitude.

   We can place a **3D body in any orientation, by rotating along 3 axes, one after the other. Hence, to compute the direction vector, you need to multiply the 3 rotation matrices, derived from Euler angles.**

   Rotation matrix for Euler-Cartesian conversion is coded in python to find the Pose Vector:

   | cos(yaw)cos(pitch) -cos(yaw)sin(pitch)sin(roll)-sin(yaw)cos(roll) -cos(yaw)sin(pitch)cos(roll)+sin(yaw)sin(roll)|
   | sin(yaw)cos(pitch) -sin(yaw)sin(pitch)sin(roll)+cos(yaw)cos(roll) -sin(yaw)sin(pitch)cos(roll)-cos(yaw)sin(roll)|
   | sin(pitch) cos(pitch)sin(roll) cos(pitch)sin(roll) |

   To process gaze vector also, we need to do some math.

```
 def preprocess_output(self, output, head_position):
 '''
 Before feeding the output of this model to the next model, preprocess
the output.
 '''
 roll = head_position[2]
 gaze_vector = output / cv2.norm(output)
```

```
cosValue = math.cos(roll * math.pi / 180.0)
sinValue = math.sin(roll * math.pi / 180.0)

x = gaze_vector[0] * cosValue + gaze_vector[1] * sinValue
y = gaze_vector[0] * sinValue + gaze_vector[1] * cosValue
return (-x*10, y*10)
```

7. **Stickiness Feature**

   Even after doing all the tweaks, I was not able to stabilize the jittery mouse pointer. Then, I **introduced a min value called stickiness to ignore minor eye ball or head movements, to be recognized as input.**

   When the stickiness parameter is set right, then only conscious and significant movements are taken as input. **This idea has greatly stabilized the system**, both in gaze control mode and head pose control mode.

8. **Choice of Gestures**

   The choice of gestures were done in accordance with the metric value we calculate. For instance, as we compute the change in spread of eye histogram, it was natural to choose "looking up" gesture because this will trigger maximum hike in spread. Similarly, yawn was found to be most accurate to measure and hence "mouse left click" event was associated to yawn gesture.

Gesture	Action
Yawn	Left Click
Looking Up	Right Click
Right Wink	Increment Control Mode
Left Wink	Unassigned
Smile	Unassigned
Vocal 'Up'	Move Mouse 30 pixels up
Vocal 'Down'	Move Mouse 30 pixels down
Vocal 'Left'	Move Mouse 30 pixels left
Vocal 'Right'	Move Mouse 30 pixels right
Vocal: Alphabets	Type the alphabet
Vocal: Numerals	Conver to number and type

10. This is an **individual submission**. Hence, all the above ideas are entirely mine and not output of a discussion or team work.

Conclusion

The project demonstrates the **capability of Intel OpenVINO to handle multiple Edge AI models in sequence and in parallel.** Many control inputs are also sourced to demonstrate flexibility. But to deploy custom solution you can choose controls, as you deem fit.

For instance, Gaze control may be ideal for big screen while head pose control for laptop screen. Either way, Sound Control can help to accept custom form entries or vocal commands. Gesture-action mapping can also be modified. Yet the point you can drive home is **the possibility to chain multiple hardware optimized AI models on the Edge, coupled with efficient numerical computing to solve interesting problems.**

References

[1] **Intel OpenVINO Official Docs**: https://docs.openvinotoolkit.org
[2] **Intel® Edge AI for IoT Nanodegree** by Udacity. Idea inspired from Final Course Project. https://classroom.udacity.com/nanodegrees/nd131
[3] **Real-Time Eye Blink Detection using Facial Landmarks** by Tereza Soukupova and Jan Cech, Faculty of E.E., Czech Technical University in Prague.