Facial Detection

Jianxing Lu
Wenjin Cao

Motivation

- Inspired by Kaggle.com facial detection competition
- The suitability of Actors handling multithreads

Use Cases

- To Detect Faces And Offer Fun Webcam Effects (e.g. to add a pair of glasses on user's face)
- To Analyze Facial Expressions (e.g. to detect smiles potentially)

Tools : OpenCV | JavaCV

- OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision.
  The library has more than 2500 optimized algorithms which can be used to detect and recognize faces, identify objects, etc.

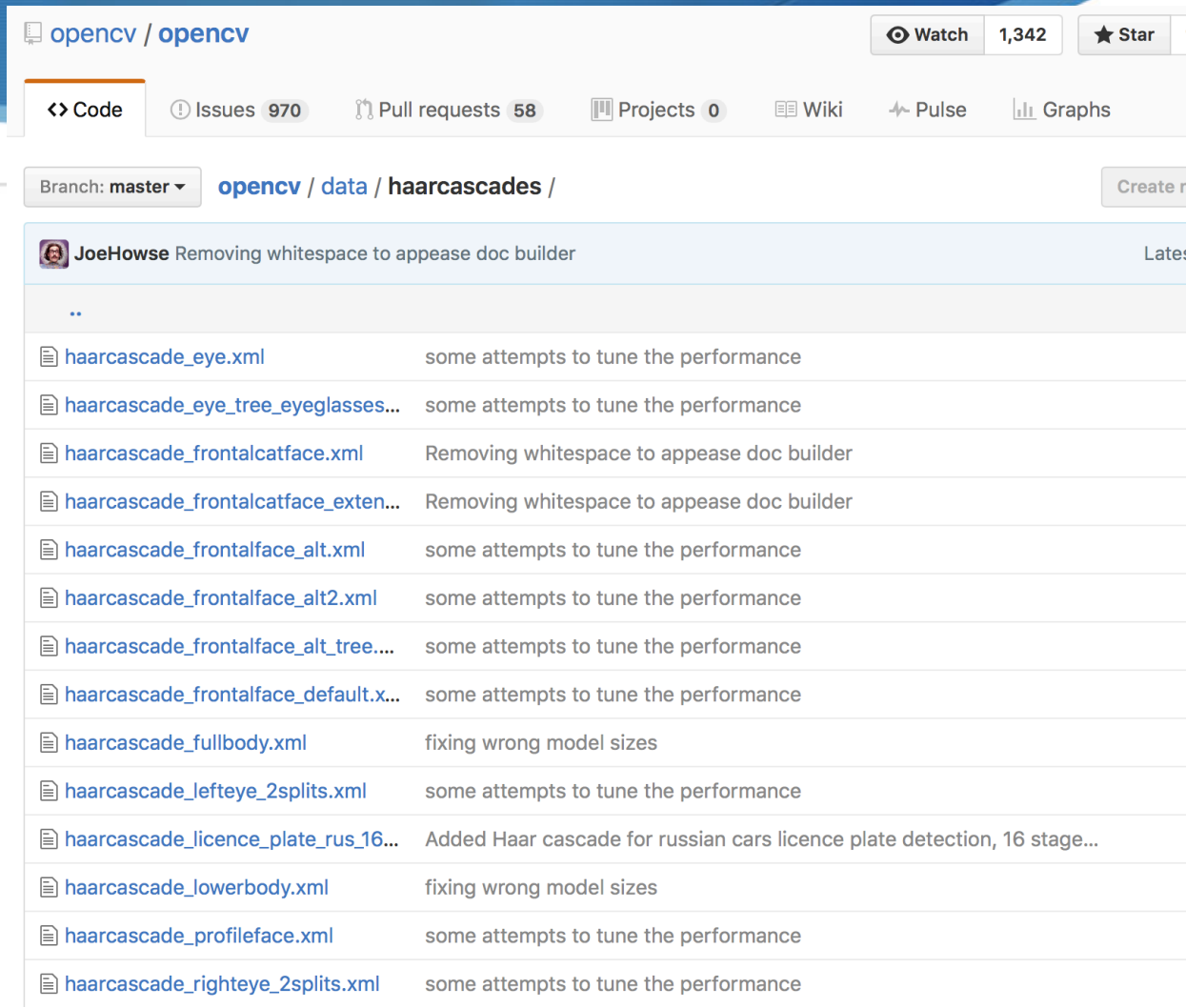
- JavaCV is a wrapper of OpenCV with easy-to-use methods. (Java interface to OpenCV and more.)

Tools : HAAR Classifiers

- **HAAR FEATURE-BASED CASCADE CLASSIFIERS**
- -- Supported directly by OpenCV to **detect objects in images**
- -- A machine learning based approach with **pre-trained** classifiers
- -- “CASCADE” : Instead of applying all the features at once, group the features into different stages of classifiers and apply one-by-one. Only if the previous stage is passed, the next stage is applied and the process is continued. (An area/object is considered to test positive (be identified) if all features in all stages of the definition return positive.)

Tools : HAAR Classifiers

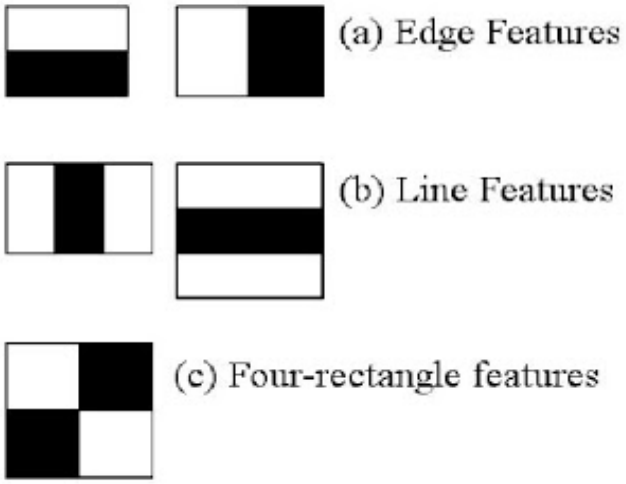
- **OpenCV has many pre-trained HAAR classifiers for frontal/profile face, eyes, etc.**
- We got those XML files from Github:
- It's also possible to self-train a classifier, though we used these pre-trained ones for time's sake.

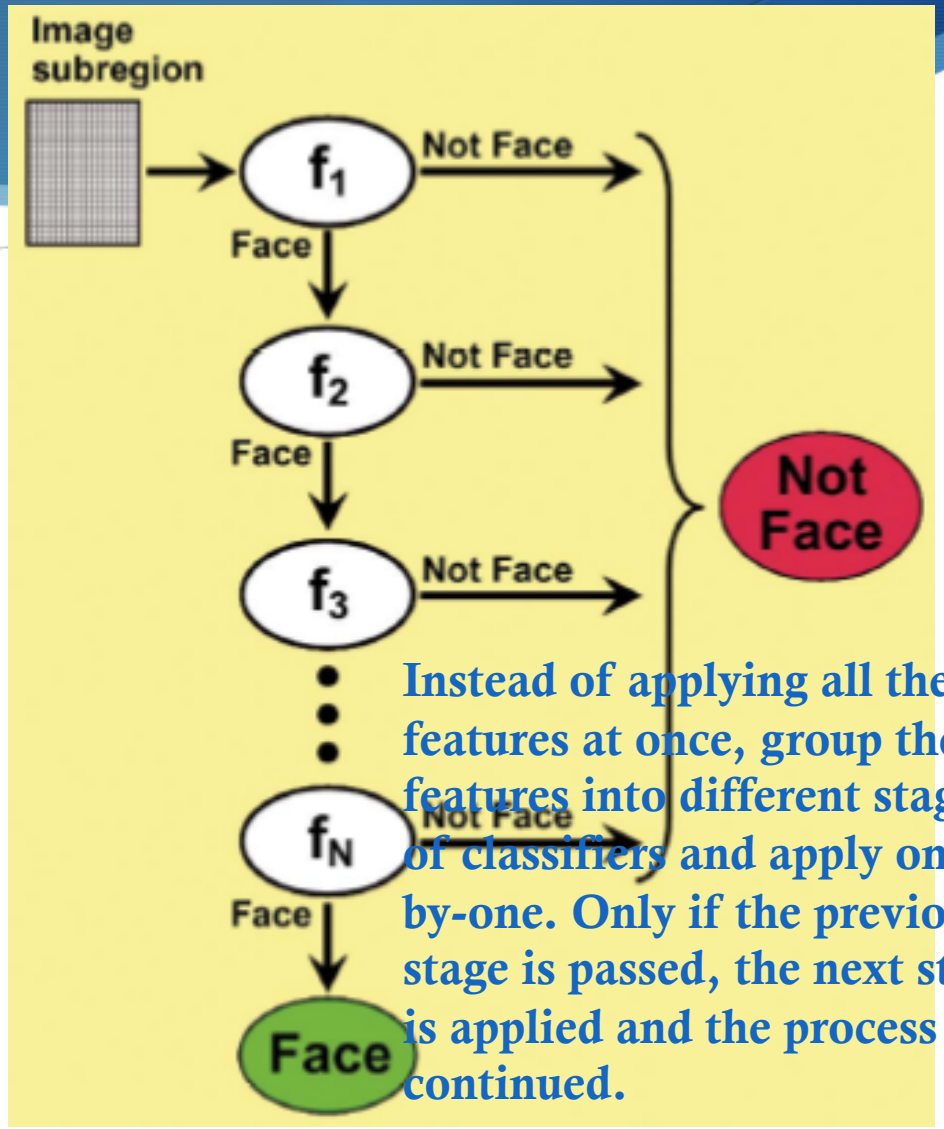


Tools : HAAR Classifiers

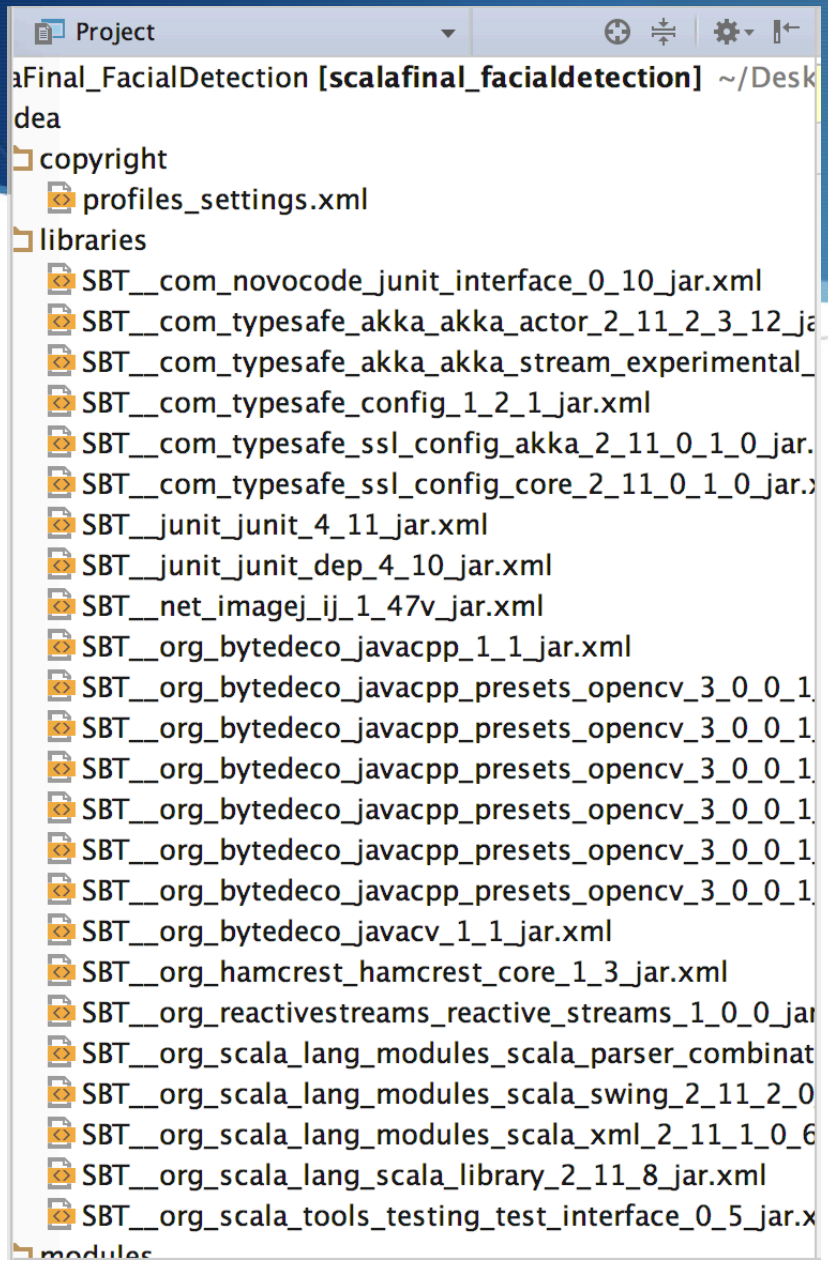
- **"CASCADE"**

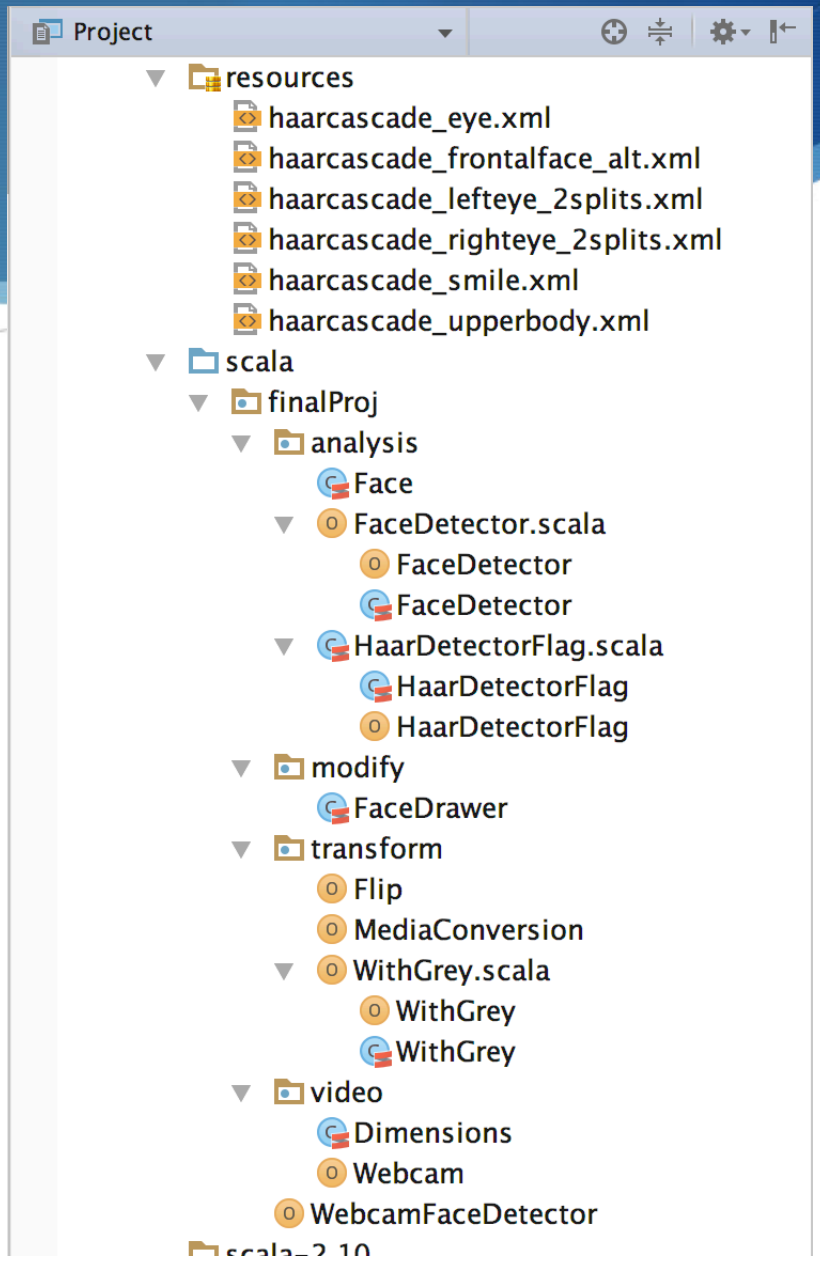
  A 'cascade' is a series of 'Haar-like features' (digital image features used in object recognition) to form a classifier.



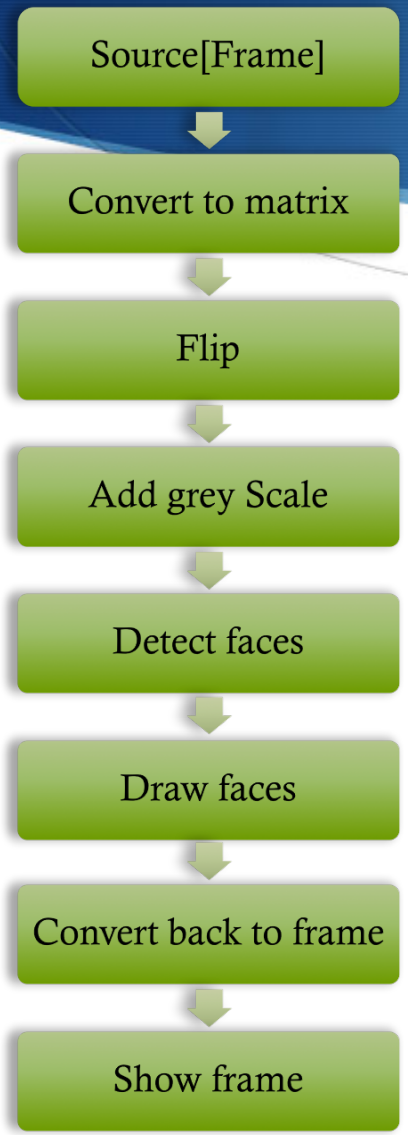


SBT-based Scala Project





Flow Chart



- **Conversion to grey scale :**
  Many image analysis tools need to be run on greyscale images, both for simplicity and efficiency.

- **To detect faces :**
  Instantiating a **CascadeClassifier**
  Instantiating an instance of **RectVector** (a wrapper for a native vector of rectangles to denote where our objects are in the image matrix).
  Pass the RectVector to the CascadeClassifier along with a greyscale image and some other options

- **A Swing app :**
  1st frame: to load the Haar cascade classifier file that's packaged in resources folder
  2nd frame: to shows our feed along with rectangles around detected objects in the CanvasFrame

Implementation of Flow Chart

```
val flow = webcamSource
  .map(MediaConversion.toMat) // most OpenCV manipulations require a Matrix
  .map(Flip.horizontal)
  .map(WithGrey.build)
  .map(faceDetector.detect)
  .map((faceDrawer.drawFaces _).tupled)
  .map(MediaConversion.toFrame) // convert back to a frame
  .map(canvas.showImage)
  .to(Sink.ignore)

flow.run()
```

AKKA for Webcam Frame

-- Reactive Stream

- **IMPLEMENTATION of AKKA:**
- Mainly used in Webcam class by generated an ActorSystem
- Utilized FrameGrabber to avoid bad things happens like video not be synchronized
- Designed a lazy WebcamFramePublisher to work with grabber

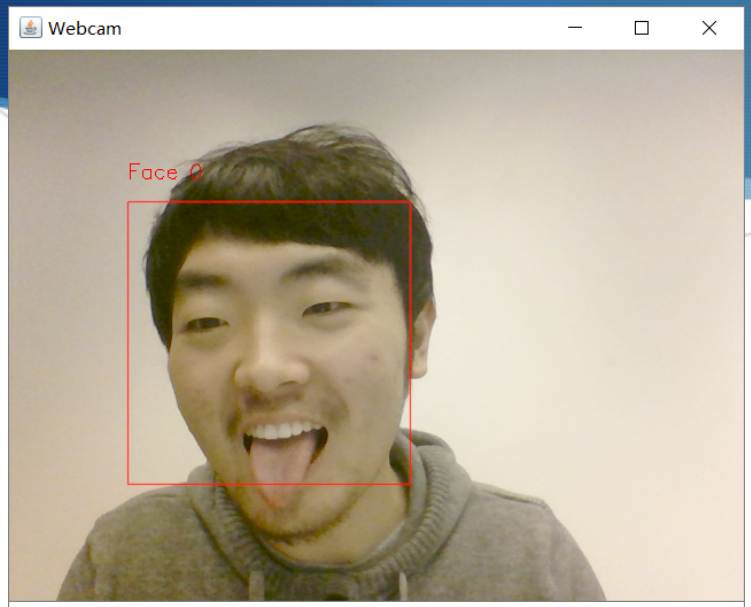
- akka.stream.scaladsl.**Source<Out,Mat>** -- Source[Frame, Unit]. A set of stream processing steps that has one open output(JavaCV Frame).
- akka.actor.Props -- To configure the creation of the actor
- akka.stream.actor.{ActorPublisher, ActorMaterializer}
  - To make the actor a stream publisher that keeps track of the subscription life cycle and requested elements.
  - Materialization turns a Source into a Reactive Streams Publisher
- akka.stream.scaladsl.Sink -- A Sink is a set of stream processing steps that has one open input and an attached output.

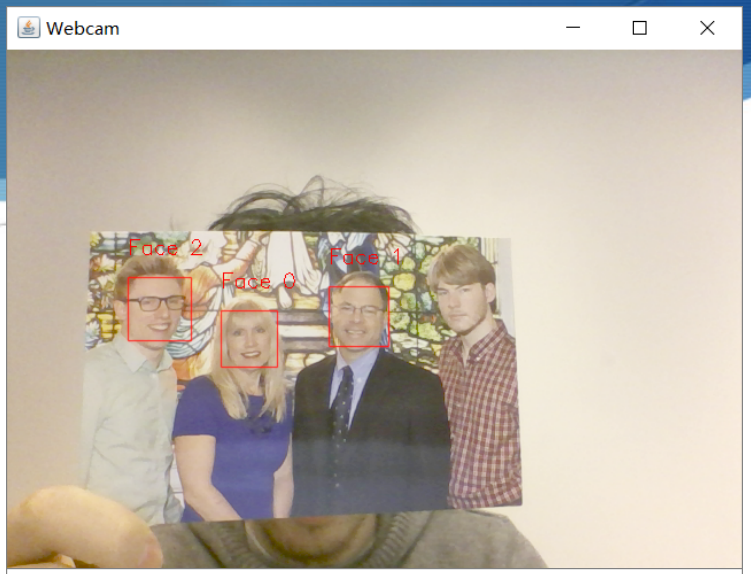
```scala
def source(
  deviceId: Int,
  dimensions: Dimensions,
  bitsPerPixel: Int = CV_8U,
  imageMode: ImageMode = ImageMode.COLOR
)(implicit system: ActorSystem): Source[Frame, Unit] = {
  //Create actor
  val props = Props(
    new WebcamFramePublisher(
      deviceId = deviceId,
      imageWidth = dimensions.width,
      imageHeight = dimensions.height,
      bitsPerPixel = bitsPerPixel,
      imageMode = imageMode
    )
  )
  val webcamActorRef = system.actorOf(props)
    //Keep Tracking
  val webcamActorPublisher = ActorPublisher[Frame](webcamActorRef)
  //Open Output
  Source.fromPublisher(webcamActorPublisher)
}
```

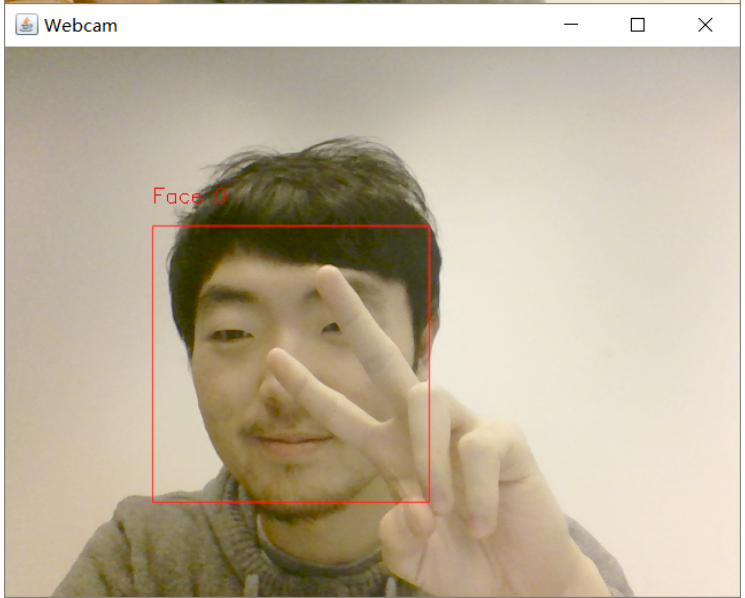
Acceptance Criteria!

- For real-time frontal face detection:
  70% of the expected objects get detected correctly.
- For real-time eye detection:
  50% of the expected objects get detected correctly.

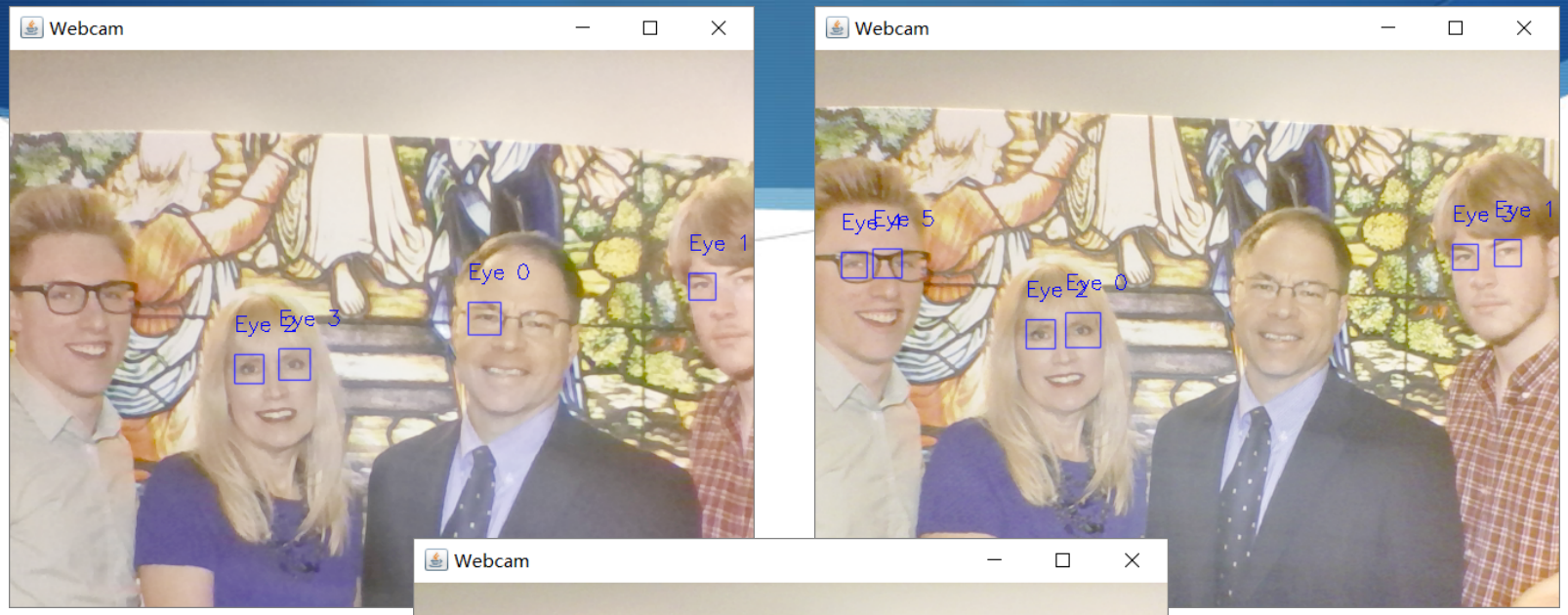
Frontal Face



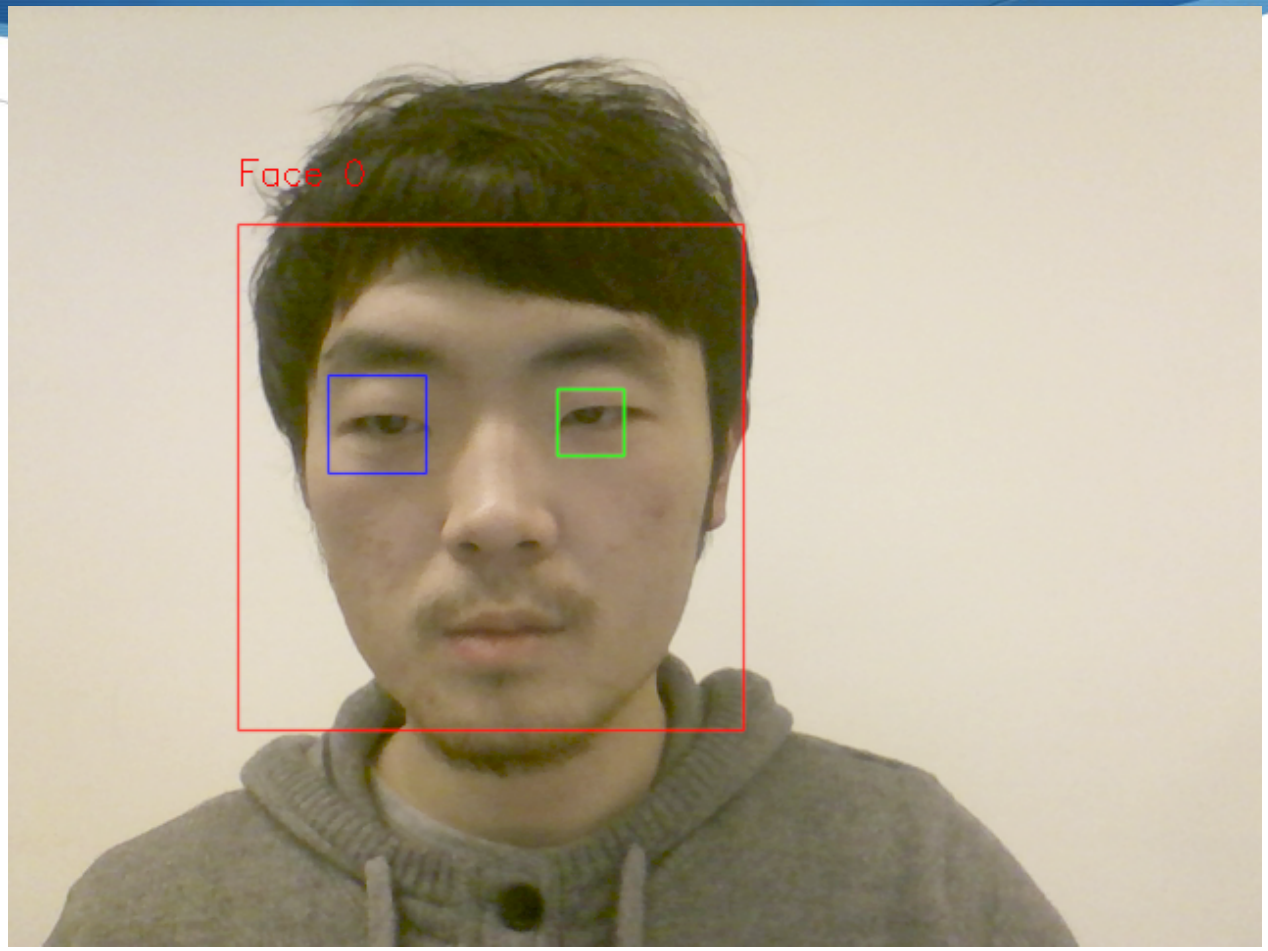




Eye


Webcam
Eye 0
Eye 1
Eye 2
Eye 3
Webcam
Eye 0
Eye 1
Eye 2
Eye 3
Eye 4
Eye 5
Webcam
Eye 0
Eye 2
Eye 3
Eye 4

Face-LeftEye-RightEye



Demo Time

- Function 1: Face only detection
- Function 2: Eyes detection
- Function 3 (combined function): Facial detection of number of faces with left and right eye

Challenges

- For blocking and heap overflow issues
- 1. Using one specific ActorSystem to handle those operations
- 2. Using Future to complete the block and set the limit of those kind of blocks
- 3. Defining one Actor to deal with a group of resource which could be blocked.