ECE 180D Fall-Winter 2020
Project Proposal

Project Smartify: Smart Music Playlist Suggestor

Team 9
Hyounjun Chang
Karunesh Sachanandani
Gerald Ko
Justin Suh

Abstract
Smartify is an intelligent interactive player that plays the best smacking tunes at the right time in the right place for the users. Smartify can detect the listener's activity, mood, and gestures in order to play the most suitable songs to get the user in the groove.

Project Description

Objectives

- To create music playlist suggestions based upon user inputs (camera, microphone, gestures).
- Create an interactive and engaging way to listen to music.
- Create a "good" playlist (decrease number of songs skipped, increase listening time, analyze emotions while listening to songs).

Target Features

- **Personalization**. Provide a personal touch and specific feel to the music that are suggested based on the user's likes, emotions, and gestures.
- **Collaboration**. Share music suggestions with friends that are subscribed to the same playlist.
- **Voice Activation**. Use voice to control specific actions (e.g. next song, recommend a song similar to the current one, subscribe to a channel).
- **Gesture Control**. Hand gesture in front of the camera to control specific actions (e.g. next song, pause current song, skip ahead 10 seconds).
- **Activity Detection**. Recognize user activities with the IMU used as a remote as an alternative to gesture control.

Potential Features

- Location based music suggestions (Use IP address to estimate approximate location)
- Collaborate with your friends on a playlist
- Plan party music based on mood
- Look up similar music online to play any songs in local playlists

Use Cases

- Control the playlist (skip, pause, play) from a distance with gestures
- Feeling gloomy, create a playlist of happy music to cheer you up!
- Loves classical music, play classical music they have not heard before

Requirements

- IMU
- Microphone

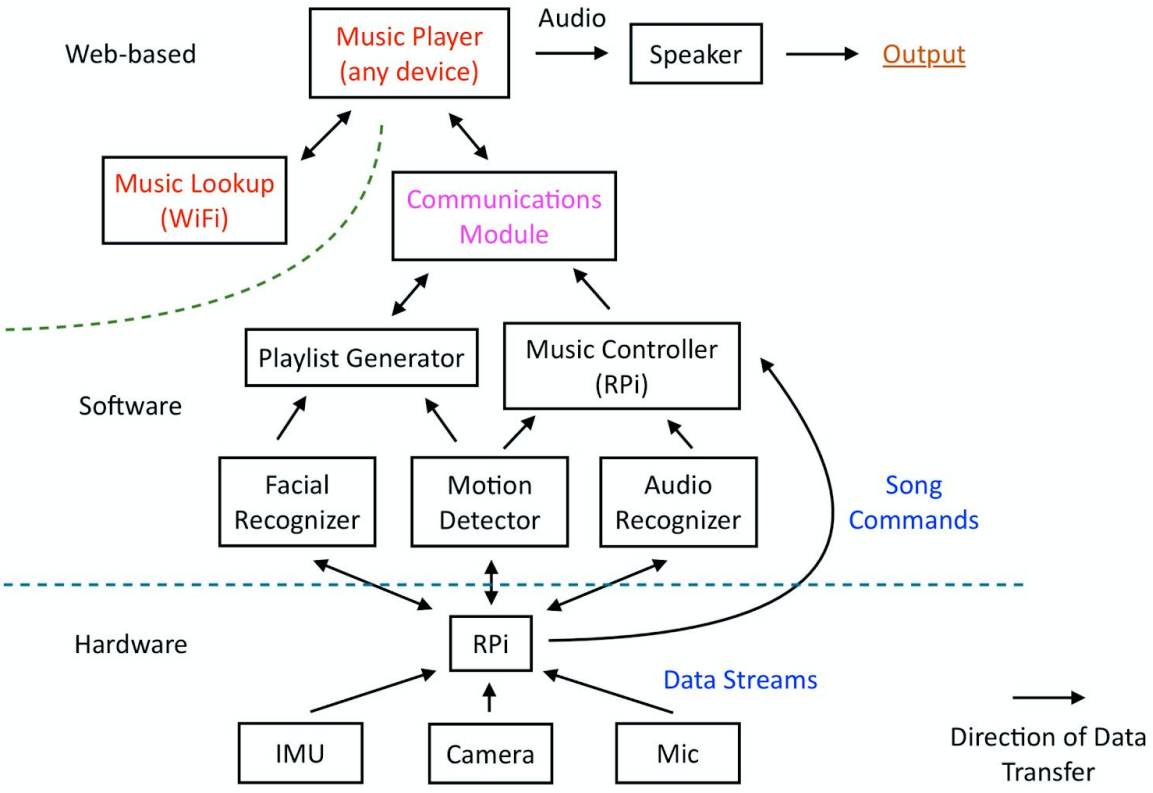
- Camera
- Raspberry Pi
- Music Directory
- Speaker

Risks and Concerns

- Difficulty classifying features of music (happy? sad?)
- Technical complexity of capturing “emotions”
- Limited computing ability of Raspberry Pi
- Difficulty in recognizing subtle hand gestures
- Difficulty of testing Machine Learning models

High-Level System

Architecture and Block Diagram



High-Level Task

Design

- **Hardware Level**
  - *Camera, Gyroscope/Accelerometer, Microphone*: Input hardware from user, sends data stream to Raspberry Pi
  - *Raspberry Pi*: Processes all data stream, and redirects it to right software modules
- **Software Level**
  - *Facial Recognizer:* Takes image stream as input, outputs features that may be helpful to identify users.
  - *Motion Detector:* Takes input from Gyro/Accel controller, outputs features that may be helpful to identify users.
  - *Audio Recognizer:* Looks for registered voice commands, and sends them to the command module.
  - *Playlist Generator:* Checks input features to create a playlist.
  - *Music Controller:* Checks results from lower level modules and determines command to send to Communications Module
  - *User Database (Within Playlist Generator and Music Player)*: Stores relevant information regarding users (Favorite song, average time listened, etc.)
  - *Communications Module*: sends data between the controller and music player to send playlist/player commands
  - *Music Player (Within Playlist Generator)*: Plays music from output device, through communicating with Communications Module (with GUI interface)
  - *Music Lookup*: Finds music that is not within the music player's local directory via WiFi, and return an audio stream

Implementation

- **Hardware Level**
  - *Camera, Microphone, Gyro/Accel* - physical hardware to get raw input data from users.
- **Software Level**
  - *Facial Recognizer:* Takes image stream as input, processes it via Computer Vision Library (OpenCV) to detect features, with possible assistance from additional libraries
  - *Motion Detector:* Uses a Threshold Value for Accel/Gyro for certain motions, or create custom gestures through Machine Learning. Outputs command received.
  - *Audio Recognizer:* Use python SpeechRecognition library to parse inputs from users.

- *Playlist Generator:* Uses features selected from input modules to generate a playlist. Model's data will be from User database, and ML training libraries (ie. Scikit-Learn) to train the model to suggest songs.
- *Music Controller:* Polls response from other modules (Motion detector, Audio Recognizer), parses them and output JSON file that will be sent via Communication Module
- *User Database:* Pandas dataframe containing relevant metrics such as (Average time spent per song, user-created playlist, metadata, etc.)
- *Communications Module:* After getting the JSON string from the Music Controller, send it to the music player via MQTT. Parse any response from the Music player (bidirectional)
- *Music Player (Within Playlist Generator)*: Run python-vlc to play a variety of music files/audio streams, given the path/audio stream link. GUI interface will be through PySimpleGUI.
- *Music Lookup*: Send a query to YouTube, get audio stream link via youtube-dl API

Testing

- Make the user look at the camera and make gestures to control play, pause, next, previous one by one.
- Make over-exaggerated facial expressions to test emotion data analysis.
- Possible Test: Play 10 songs for base data and form a playlist from this data and ask if it is to their liking based on the songs they listened to.
- Compare audio recognition for song and artist names of different languages through different APIs. Accuracy is expected to be more noticeable in this cases.
- Test audio command recognition for varying volume levels of background music, including different types of background music (eg: different levels of dynamic changes)

Task Allocation

Hyounjun Chang

- Module Integrations/ Integration Tester
- Shell/Pipelining
- ML model research

Karunesh Sachanandani

- Audio Recognition Module
- MQTT communication between users
- Web API research

Gerald Ko

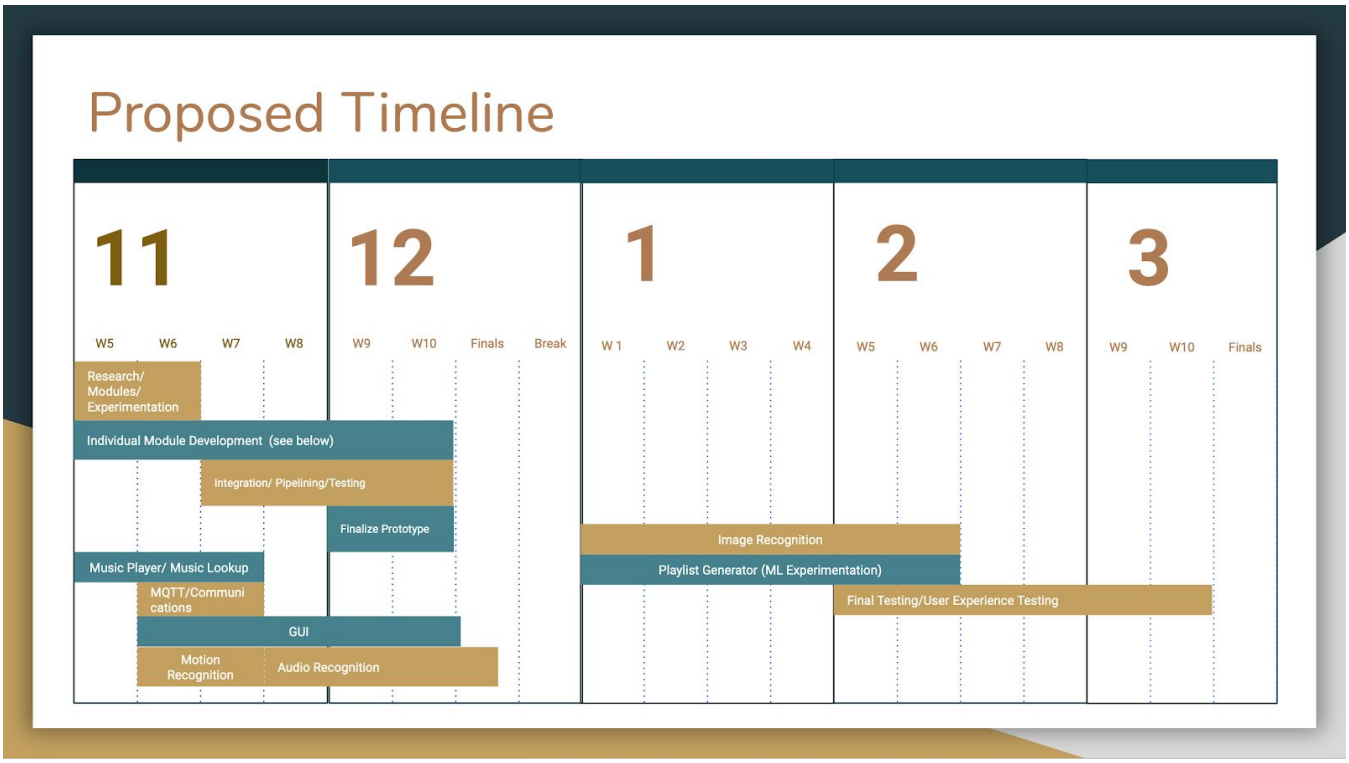
- GUI (album cover image viewer, MP3 player, camera window)
- UI/UX
- Computer Vision (face, mood, and gesture detection)
- Testing/QA

Justin Suh

- Motion Detection (IMU)
- Image Detection/Computer Vision
- Side: ML for image processing for emotion/hand gestures

Project Schedule

- Scrum sprint cycles (1-2 weeks) until ready
    - Incremental feature development
    - Create priority of features to implement in each cycle
    - Unit testing
    - Integration testing
- Create a working prototype by end of quarter (with core features)
    - Wireless commands
    - Web Query (1st quarter)
    - Smart playlist generation with GUI design (2nd quarter)



Research & Progress

Audio Recognition (Karunesh)

As a first step, we are using a laptop's microphone to accept voice commands from the user. This is because the RPi does not have a built-in microphone. Ultimately, of course, having a device (RPi) that can work as a music player autonomously, while implementing all the "smart" features of Smartify will be an important characteristic of our final product.

So far, we have been working to implement the Python library of Speech Recognition in order to accept voice commands, parse the audio input from the user in order to extract specific actions, and output them to the music player. The first iteration of the music player was visioned to accept a certain song name and artist name from the user through the laptop's microphone and play the top search result on youtube. And so, for the voice recognition module, the goal was to accept a command of the form "Play (song name) by (artist name)", recognize the command accurately, extract the user fields, and send the command over to the music player. In order to develop the voice recognition independently of the other modules, I have assumed for now that the standard output of this program is a text file containing the song name and the artist name of the song to be played.

I have used google's web search API for now, due to its accuracy, but I will be testing the CMU Sphinx API as well, and comparing the two for command recognition accuracy. The latter has the advantage of being able to operate offline, which would be a significant advantage as it is important for the final product to be able to work on a controller (RPi) without an internet connection.

Here, ambient noise is filtered out by a python function. A more impressive module may need to be designed in order to filter out music playing in the background as this function only considers ambient noise for a brief period (1 second for example) and uses that as the base level.

Here is the code that performs the above described task:

```
import speech_recognition as sr
#print(sr.__version__)
#3.8.1
rec = sr.Recognizer()
mic = sr.Microphone()
with mic as source:
        rec.adjust_for_ambient_noise(source)
        #wait a second for the above function to adjust for ambient noise before inputting a voice command
        audio = rec.listen(source)
        strinput = rec.recognize_google(audio)
        wordsinput = strinput.split(" ")
        #convert string input to array of words
        playind = -1
        byind = -1
        #find indices of 'play' and 'by' in order to pull out the song and artist names
        for i in range(0, len(wordsinput)):
                if playind == -1 and wordsinput[i] == 'play':
                        playind = i
                if byind == -1 and wordsinput[i] == 'by':
                        byind = i
        if byind > playind + 1 and len(wordsinput) > byind + 1 and playind > -1:
                songname = ""
                artistname = ""
                for i in range(playind+1, byind):
                        songname = songname + wordsinput[i] + " "
                for i in range(byind+1, len(wordsinput)):
                        artistname = artistname + wordsinput[i] + " "
                songname = songname[:-1]
                artistname = artistname[:-1]
                #write to a text file
                speechfile = open("speechInput.txt", "w")
                speechfile.write(songname)
                speechfile.write('\n')
                speechfile.write(artistname)
                speechfile.close()
```

MQTT Communication (Karunesh)

The initial target for the MQTT module was as follows: Assuming a json file with song information (song and artist name for now) is present in the controller (RPi), we wish to send the song information over to all subscribers in the user's channel (a Laptop for example). Then the subscribers need to be able to parse that data and extract the fields, sending them to a text file to be accepted by the music player. This is what has been implemented.

Moving forward, the metadata will certainly get more complicated as we design a full music player capable of reading from a large library, but the basic framework for json parsing and communications through MQTT has been laid out.

Here is the code that accepts the following json file and performs the above described task:

```
def publish(client):
    msg_count = 0
    while True:
        time.sleep(1)
        with open('musicInfo.json') as musicfile:
            musicjson = json.load(musicfile)
            musicstr = json.dumps(musicjson)
        msg = musicstr
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            print("Send " + msg + " to topic " + topic)
        else:
            print("Failed to send message to topic " + topic)
        msg_count += 1
```

```
def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
        musicjsonsub = json.loads(msg.payload.decode())
        musicfilesub = open("musicInput.txt", "w")
        musicfilesub.write(musicjsonsub['songName'])
        musicfilesub.write('\n')
        musicfilesub.write(musicjsonsub['artistName'])
        musicfilesub.close()

    client.subscribe(topic)
    client.on_message = on_message
```

In order to test the initial Audio Recognition and MQTT modules described above the following script was developed to take in the song and artist names from the text file returned by either script, and play the first result on youtube corresponding to those fields.

```
import pafy
import sys
import  vlc
import time
import json
from youtube_search import YoutubeSearch
print(sys.argv[1])
if sys.argv[1] == "audio":
        filename = 'speechInput.txt'
elif sys.argv[1] == "json":
        filename = 'musicInput.txt'
else:
        print("enter type of input")
        exit()
inputfile = open(filename, "r")
searchstr = inputfile.readline()
searchstr = searchstr + inputfile.readline()
inputfile.close()
result = YoutubeSearch(searchstr, max_results=1).to_dict()
link = "https://youtube.com/watch?v=" + result[0]['id']
print(link)
video = pafy.new(link)
audio = video.getbestaudio()
player = vlc.MediaPlayer(audio.url)
player.play()
time.sleep(300)
player.pause()
```

IMU Motion Detection (Justin)

The IMU will be used to recognize command gestures to control the mp3 player. There will be commands such as play, pause, next, previous, and maybe even suggest a song(depending on their emotion). The IMU is most likely to be placed on the palm of the hand for now.

The specific gestures are not 100% decided yet as I will be performing some analysis on what gesture is most intuitive to people. This will be done by asking family, friends, classmates, etc. However, as a placeholder for now, we have decided to use a horizontal hand tilt to the right and left for the next song and previous song respectively and vertical tilt up and down to play and to pause respectively. The play and pause commands will most likely be changed as it is very uncomfortable to command in such a way.

The figure below is only the tilt to the right to play the next song output. This result was created through taking the difference between the read GYRx value and its previous value. If the difference exceeded 1000 it was considered as a command for the next song. With this logic it is fairly simple to create the previous song command where the difference is -1000, but this will be implemented in the very near future.

```
-50
-217
-139
-133
-108
-251
1634
-162
Next Song
-1559
Next Song
385
3
-165
2178
-1103
Next Song
517
-2172
Next Song
-2538
66
48
```

Bird's eye view of the hand. The arrow represents the movement from initial hand position

GUI (Gerald)

For the GUI part of Project Smartify, we have three main sub-modules:

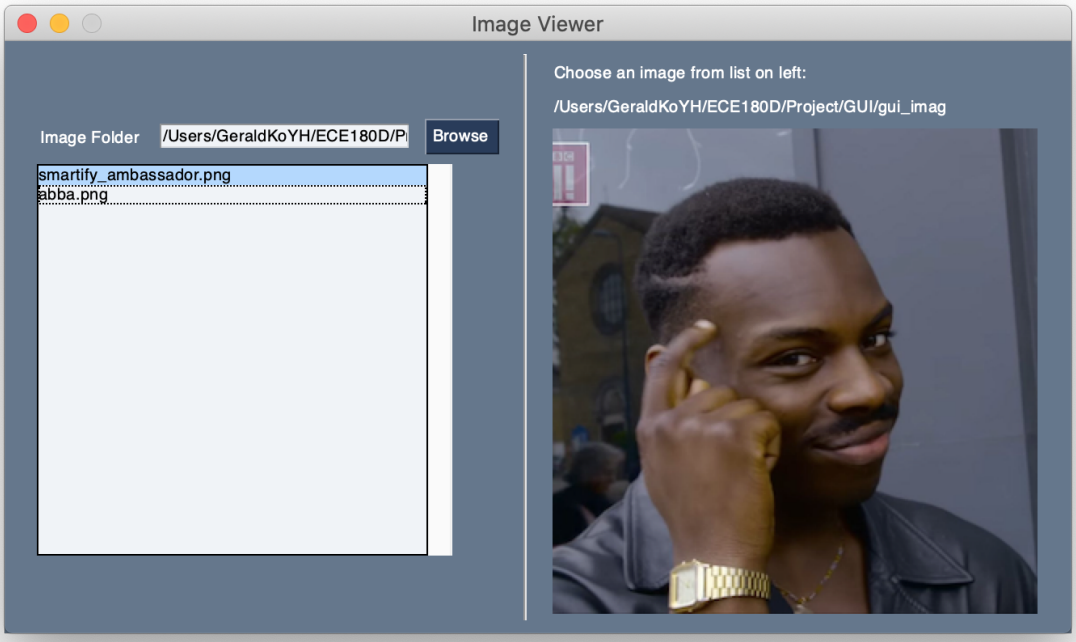
1. MP3 Player GUI
2. Album Cover Image Viewer GUI
3. Camera Input Visualizer for Computer Vision GUI

On a high-level, we are using the below packages/modules for our project. I have also included the rationale behind why we decided on them.

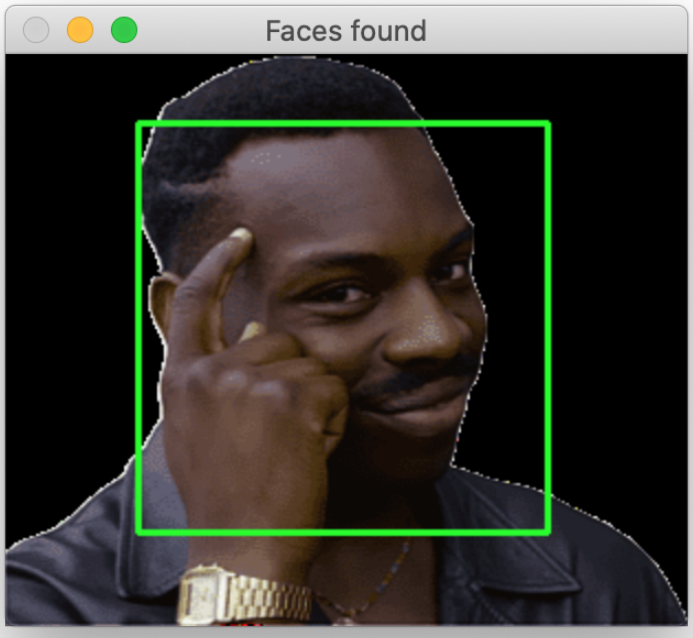
1. PySimpleGUI
    a. Very established with a lot of reference material.
    b. Strong library of various demos.
2. Tkinter and Mutagen
    a. This is mainly for the MP3 player.
    b. There are good reference sources for various implementations.
    c. Universal and stays consistent across platforms and devices.
3. Pillow
    a. Additional package to help support the image viewer to be able to read more than just png and gif.
4. OpenCV
    a. This is used for the computer vision aspect of our product, such as face detection and gesture recognition.
    b. For facial detection, we are using the Cascade approach so we can have an appreciable accuracy and high response rate for real time use.
    c. We are still exploring different alternatives for more advanced recognitions such as mood detection.
5. Pygame (tbd)
    a. This is used for our MP3 player implementation but this has dependency issues with VLC.
    b. We are still exploring fixes and alternatives to this.

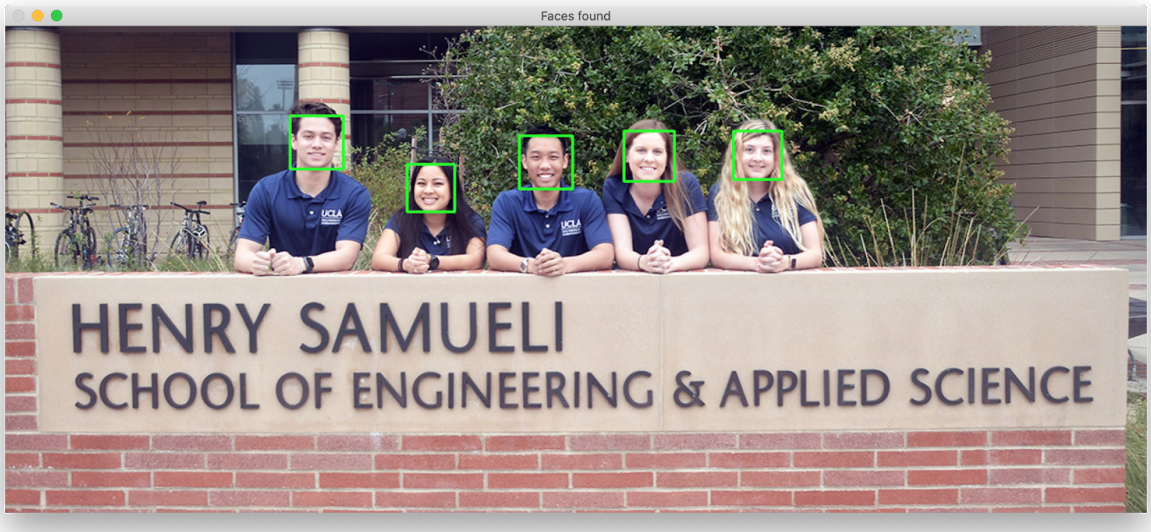
Here are some screenshots:

Image Viewer



Face Detection





Webcam Face Detection