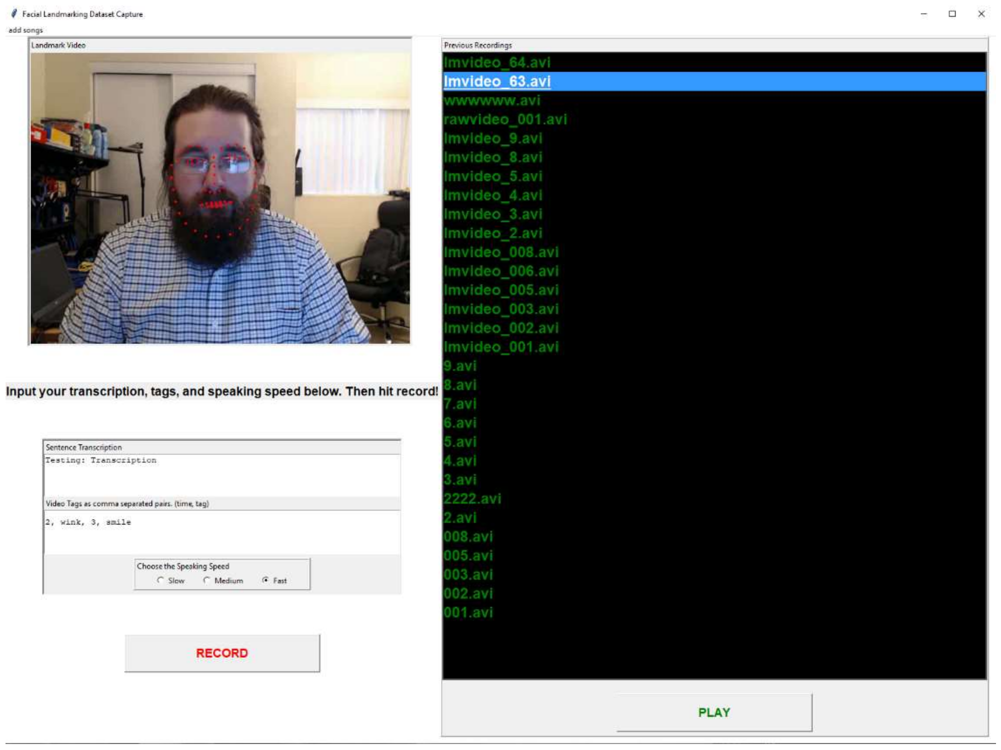
AV Facial Recording & Landmarking Dataset Tool

Final Report

Author: Richard J. Romano III
ECE 544: Robotics III
PSUID: 966849215

Project Summary

For my Robotics III project I have been exploring facial and audio recognition using a landmarking library called DLIB and OpenCV to detect 68 points on a person's face. Currently OpenCV can show these marks on a live camera feed which is cool to see but not particularly useful for data gathering. The overall goal is that eventually this landmarking data is going to be used to help animate an actual physical robot's face such as the Fritz Robotic Head.



Input your transcription, tags, and speaking speed below. Then hit record!

Sentence Transcription

```
Testing: Transcription
```

Video Tags as comma separated pairs. (time, tag)

```
2, wink, 3, smile
```

Choose the Speaking Speed: Slow Medium [✓ Fast]

```
lmvideo_64.avi
lmvideo_63.avi
wwwwww.avi
rawvideo_001.avi
lmvideo_9.avi
lmvideo_8.avi
lmvideo_5.avi
lmvideo_4.avi
lmvideo_3.avi
lmvideo_2.avi
lmvideo_008.avi
lmvideo_006.avi
lmvideo_005.avi
lmvideo_003.avi
lmvideo_002.avi
lmvideo_001.avi
9.avi
8.avi
7.avi
6.avi
5.avi
4.avi
3.avi
2222.avi
2.avi
008.avi
005.avi
003.avi
002.avi
001.avi
```

Figure 1- Complete Recording

The short-term goal is to provide the robot head with a 15 second "record" containing all of the necessary data for the robot to mimic the user's face and speech. Particularly I not only want to provide an easy way to gather these audio/video samples but to also store them in a relational database that can eventually be grown into a possible machine learning training set.

Requirements

MUST

- Take in a 15 second or less input video/audio file.
- Perform dlib/opencv landmarking and store the resulting data.
- Allow user to input their own text transcription.

SHOULD

- Use an appropriate database such as MariaDB, MySQL, InfluxDB, etc.
    - Went with a JSON file for the database to keep it simple and easy to update.
- Allow user to add annotations/tags for things like emotions/speed/etc.
- Play back the recording with the landmarks overlayed on top.

MAY

- Perform speech-to-text and store text.
    - Text is hand-entered by user for now.
- Allow multiple input files (batch processing)
    - You can keep recording files and it adds it to the database, but it definitely could use quite a bit of polish.
- Improved GUI with buttons/playback/input boxes/etc.
    - The GUI is functional but not pretty.

Design Theory

For this project, the design is broken down into four basic areas. The Audio/Video (AV) recording using a computer webcam, facial recognition processing on that recorded video, a user interface to show the results as well as provide inputs for annotations, and a record database to keep track of each recording and its information.

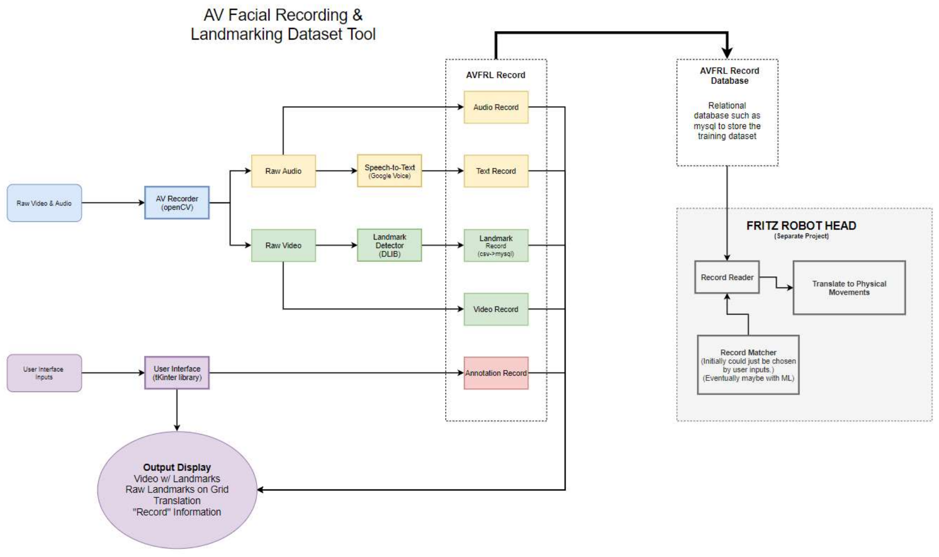


Figure 2- Program Overview

Audio/Video Recording

The AV Recorder was almost completely pulled from a GitHub listed in the appendix. Initially everything was done in OpenCV, which actually has zero capabilities to record sound. Most of the first attempts at getting PyAudio to work did not go very well as syncing the recording was difficult. The av recorder Github is great because it runs the video and audio in individual threads and mux them at the end. I did have to go through and do some minor edits mostly in the file manager so that it renames and moves the individual audio and video files to their associated folders.

Facial Recognition

The detection class was written with some initial code and tutorials from the website pyimagesearch. This gave some useful insight into how to use DLIB with openCV and a training dataset that detects 68 landmarks on the user's face. Most of the time was spent adjusting it to work with the AV recorder encoding as well as to properly output the landmarks in a final CSV file.

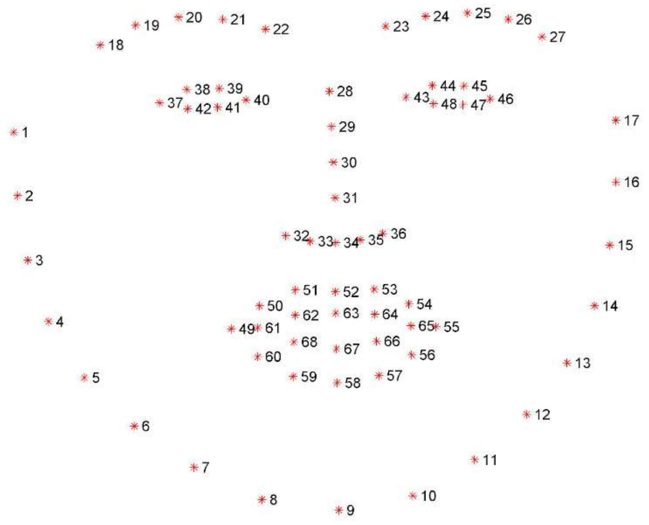


Figure 3 - 68-Point Facial Landmarking

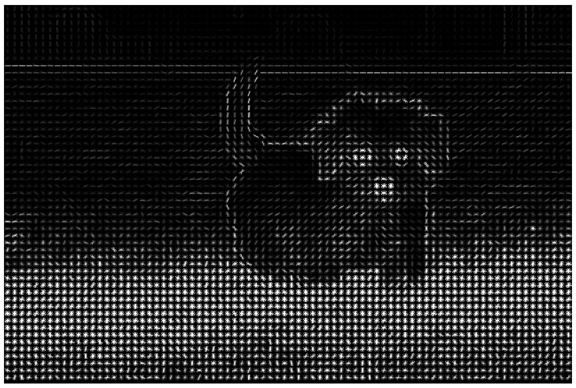


Figure 4- HoG Puppy

The basic way it functions is the detector uses DLIB and predictor.dat file to perform Histogram of Oriented Gradient (HoG) based face detection. By looking at the gradient patterns it can locate similar patterns in the video and set the landmarks accordingly.

Each region of the face is analyzed this way and if it is found to be similar to one of the 68 points then it becomes a positive detection. Each point is aligned with each other and with the HoG gradients to become a rather efficient method of facial detection and landmarking.

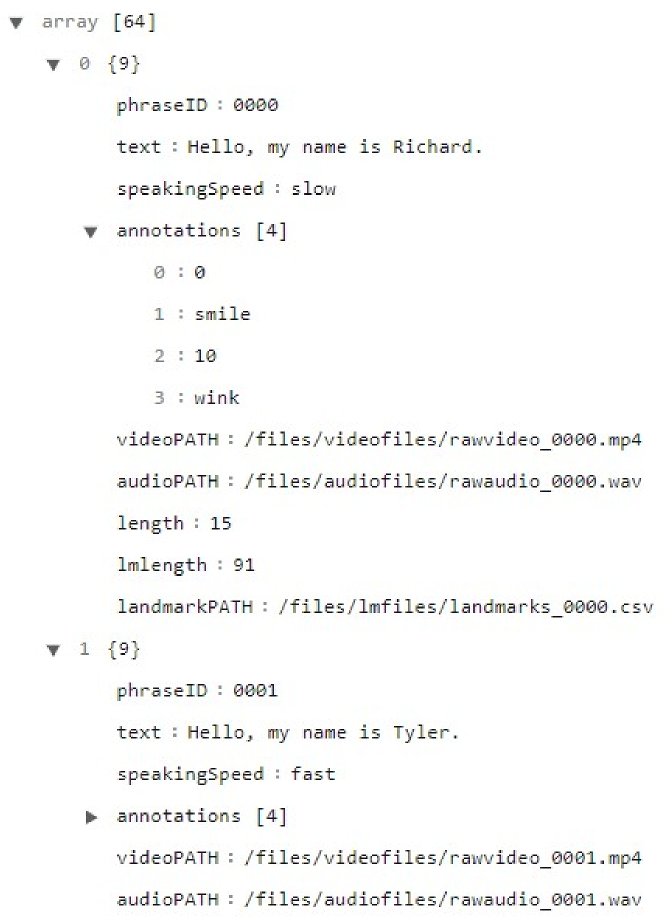
Record Database

Each recording comes with a small amount of information that will be useful for the robot as well as the user. It includes the actual spoken phrase, speaking speed, comma-separated annotations, as well as the various paths and lengths of the required files.

Each record is stored in a list and serialized into a JSON file named phraseDB.json. At the beginning of the program the user is asked for this json file. It is then loaded into a list of dictionaries which determines where to load the next record.

Each time the user presses record it takes a snapshot of the current record, appends it to the database and dumps it to the json file. It then increments the record id allowing for another recording to be added.

The majority of time was probably spent on this area of the project. It was difficult to come up with a database scheme that covered all of the requirements but was not overly complex. I originally looked at using a full database system such as MySQL or MariaDB. Although these would work great for a large database it would be difficult to implement for such a small use-case. Another interesting choice would have been to use TensorFlow Records (TFRecord.) Youtube uses TFRecords for a wide variety of applications including some of their extremely large training datasets. If this landmarking tool was made professionally, TFRecords would probably be the route I would go as you could possibly leverage Youtube's analytics and machine learning algorithms they already employ on large video recorded datasets.



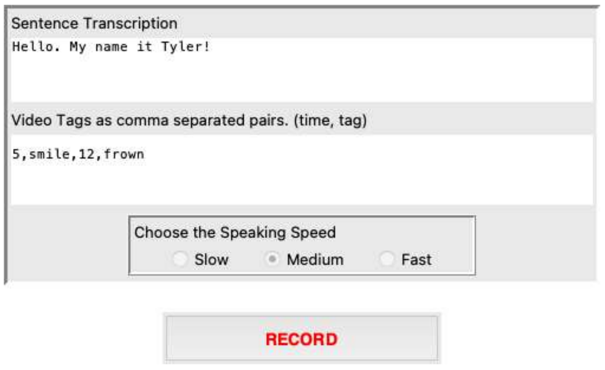
```
▼ array [64]
  ▼ 0 {9}
      phraseID : 0000
      text : Hello, my name is Richard.
      speakingSpeed : slow
    ▼ annotations [4]
        0 : 0
        1 : smile
        2 : 10
        3 : wink
      videoPATH : /files/videofiles/rawvideo_0000.mp4
      audioPATH : /files/audiofiles/rawaudio_0000.wav
      length : 15
      lmlength : 91
      landmarkPATH : /files/lmfiles/landmarks_0000.csv
  ▼ 1 {9}
      phraseID : 0001
      text : Hello, my name is Tyler.
      speakingSpeed : fast
    ▶ annotations [4]
      videoPATH : /files/videofiles/rawvideo_0001.mp4
      audioPATH : /files/audiofiles/rawaudio_0001.wav
```

Figure 5 - JSON Database

User Interface

For the graphical user interface (GUI,) I received some help from Tyler Hull who was also in my 508 Python Workshop. I did not have any experience with GUI creation in python and he was quite a great resource for getting going on it. The two user interfaces that were evaluated for use were pyQT and tkinter for this project. The advantage of pyQT is that it has a graphical editor and probably would have been a bit easier to set up in the beginning. There seemed to be less tutorials on it and overall the consensus seemed to be that if you were doing odd stuff to use the more advanced and flexible tkinter library. Since our program would not only have to record but also playback the video and audio sources we thought it best to dig into tkinter. The user interface went through several iterations as it moved towards a final version of the app. There are a few versions of the code in the "Test Code" folder that contain different iterations, tools, and attempts at completing the GUI.

User Record Input



Sentence Transcription

```
Hello. My name it Tyler!
```

Video Tags as comma separated pairs. (time, tag)

```
5,smile,12,frown
```

Choose the Speaking Speed

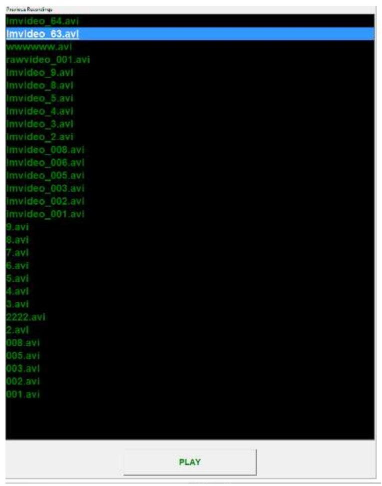
Slow [✓ Medium] Fast

Figure 6 - User Interface

The user is able to enter information about the recording on the lower left-hand side of the GUI. Labels were used to title each text box, and default example values were placed in each box on start. The user also needs to choose a speaking speed from the radio buttons and the slow speed is chosen by default. When the user hits a record, the recording functions are called, a recording is made, and the user's inputs are used to create a record that is added to the database.

Video Selection, Playback, and Recording

In the screenshot to the right, you can see the user interface after a video selection has been made and the play button has been pressed. Since the GUI loads the user's videos from the 'video' folder on start, we have a few testing videos that do not follow the automated naming conventions at the bottom of the list.



Previous Recordings

```
lmvideo_64.avi
lmvideo_63.avi
wwwwww.avi
rawvideo_001.avi
lmvideo_9.avi
lmvideo_8.avi
lmvideo_5.avi
lmvideo_4.avi
lmvideo_3.avi
lmvideo_2.avi
lmvideo_008.avi
lmvideo_006.avi
lmvideo_005.avi
lmvideo_003.avi
lmvideo_002.avi
lmvideo_001.avi
9.avi
8.avi
7.avi
6.avi
5.avi
4.avi
3.avi
2222.avi
2.avi
008.avi
005.avi
003.avi
002.avi
001.avi
```

Figure 7 - Playback Menu

There are two main flaws I would have liked to have more time to fix for this project.

1. When you click the record or play button the GUI freezes while it completes this procedure. This is mostly a threading issue. The program is a bit complex as it has a video, audio, playback, and display thread all running and talking to each other at the same time. I believe with a bit of cleanup and better communication between the threads the overall program will run a lot smoother.
2. The video selection is based on whatever is in the folder and will even try to play non-video files. This is mostly just a matter of modifying the selection library code to only pull files that have names matching the database.

Landmark Translation

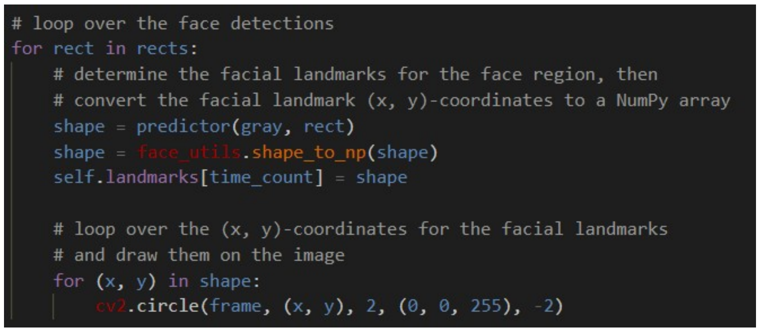
The next part of this project was to take the landmark.csv file that was produced and translate it to corresponding Fritz robot head movements. There were two main areas that Abhiraj and I identified as problematic for implementing this part of the project which I will discuss in more detail below.

Landmark Origin Translation

The first issue we encountered was that the landmark coordinates depended on where the user's face was in the camera image. So, if they moved their head too much or adjusted their camera to a different angle their landmark coordinates would change dramatically. Since the Fritz Robot cannot actually move its head, these landmark movements were extraneous, and we wished to eliminate that source of error.

The plan was to have me adjust the "origin" of the landmarks. Referring to figure 3 above, my first goal was to separate out the 1st and 20th landmark point as they represent the furthest west and north data point. My next goal was to translate or shift each landmark by the x distance of the 1st landmark and the y distance of the 20th landmark. This would move the origin to (0,0) and make Abhiraj's work easier.

Sadly, I ran into a ton of issues actually implementing this process. The DLIB library is doing a lot of alterations to the shapes before they turn into actual (X, Y) coordinates and meddling with the process in the middle turned out to be challenging. I think if I dug in a little further into how face_utils worked and what data types it was actually using I could have gotten this part implemented.



```
# loop over the face detections
for rect in rects:
    # determine the facial landmarks for the face region, then
    # convert the facial landmark (x, y)-coordinates to a NumPy array
    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)
    self.landmarks[time_count] = shape

    # loop over the (x, y)-coordinates for the facial landmarks
    # and draw them on the image
    for (x, y) in shape:
        cv2.circle(frame, (x, y), 2, (0, 0, 255), -2)
```

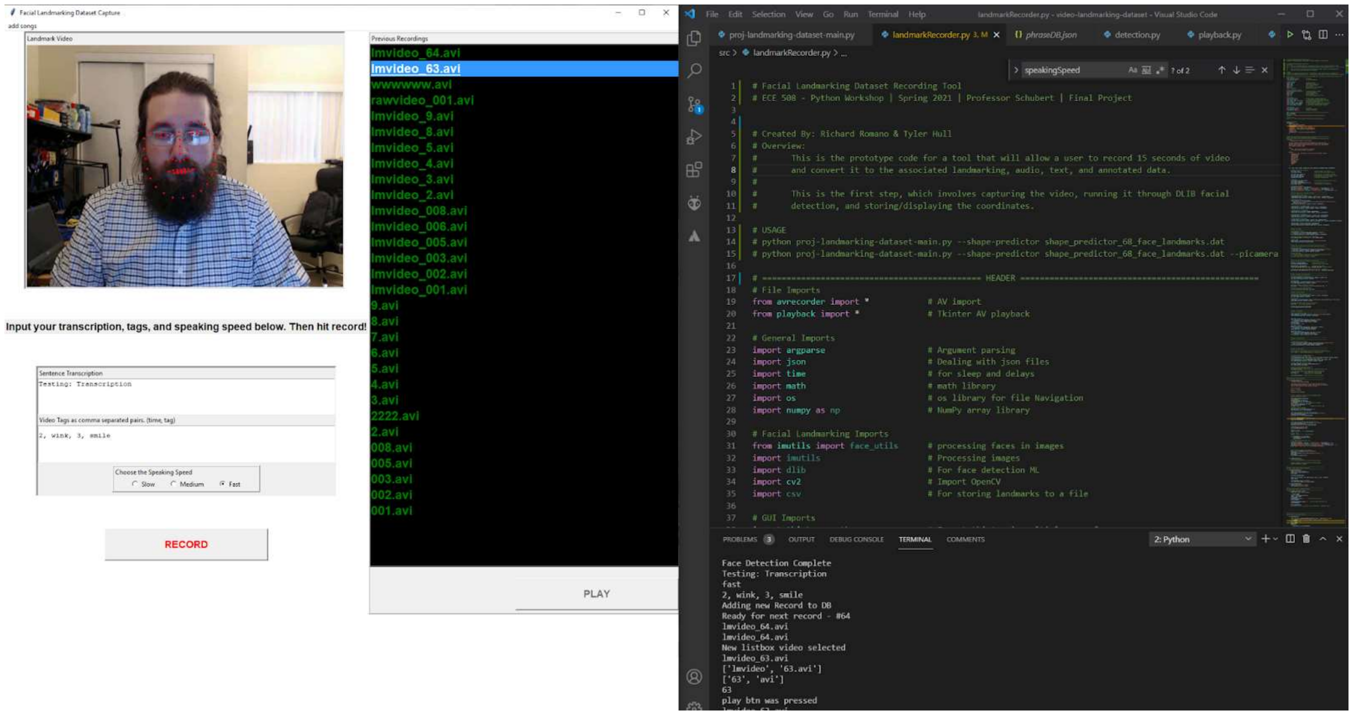
Figure 8 - Detector Code Section

Matching Landmarks to Motors

The next area of issue was that although we had 68 landmarks to work with, many of them didn't actually line up to the few motors within the Fritz robot. This area was mostly solved by Abhiraj only choosing a few landmarks such as 52 and 58 to determine if the mouth was open and close. He used the distance between them to create an equivalent distance that the Fritz robot should open its mouth. I would definitely be interested in trying these landmarks against a robot that has significantly more options for facial movement. It would be amazing to see all 68 points in actual use.

Outcome

This project got surprisingly far in some areas and hung up pretty bad in others. Overall, I was quite happy that the program already does all of the core requirements. It is definitely still rather buggy and has a lot of issues with file paths that need to be cleaned up. The program also needs to better match the needs of the actual robot so quite a few changes can be made in that regard.


Facial Landmarking Dataset Capture
add songs
Landmark Video
Previous Recordings
RECORD
PLAY
File Edit Selection View Go Run Terminal Help
landmarkRecorder.py - video-landmarking-dataset - Visual Studio Code
proj-landmarking-dataset-main.py
landmarkRecorder.py 3, M
phraseDB.json
detection.py
playback.py
src > landmarkRecorder.py > ...
speakingSpeed
? of 2
PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
COMMENTS
2: Python

Input your transcription, tags, and speaking speed below. Then hit record!

Sentence Transcription

```
Testing: Transcription
```

Video Tags as comma separated pairs. (time, tag)

```
2, wink, 3, smile
```

Choose the Speaking Speed: Slow Medium [✓ Fast]

```
lmvideo_64.avi
lmvideo_63.avi
wwwwww.avi
rawvideo_001.avi
lmvideo_9.avi
lmvideo_8.avi
lmvideo_5.avi
lmvideo_4.avi
lmvideo_3.avi
lmvideo_2.avi
lmvideo_008.avi
lmvideo_006.avi
lmvideo_005.avi
lmvideo_003.avi
lmvideo_002.avi
lmvideo_001.avi
9.avi
8.avi
7.avi
6.avi
5.avi
4.avi
3.avi
2222.avi
2.avi
008.avi
005.avi
003.avi
002.avi
001.avi
```

```python
# Facial Landmarking Dataset Recording Tool
# ECE 508 - Python Workshop | Spring 2021 | Professor Schubert | Final Project


# Created By: Richard Romano & Tyler Hull
# Overview:
#       This is the prototype code for a tool that will allow a user to record 15 seconds of video
#       and convert it to the associated landmarking, audio, text, and annotated data.
#
#       This is the first step, which involves capturing the video, running it through DLIB facial
#       detection, and storing/displaying the coordinates.

# USAGE
# python proj-landmarking-dataset-main.py --shape-predictor shape_predictor_68_face_landmarks.dat
# python proj-landmarking-dataset-main.py --shape-predictor shape_predictor_68_face_landmarks.dat --picamera

# ============================================== HEADER ==============================================
# File Imports
from avrecorder import *            # AV import
from playback import *              # Tkinter AV playback

# General Imports
import argparse                     # Argument parsing
import json                         # Dealing with json files
import time                         # for sleep and delays
import math                         # math library
import os                           # os library for file Navigation
import numpy as np                  # NumPy array library

# Facial Landmarking Imports
from imutils import face_utils      # processing faces in images
import imutils                      # Processing images
import dlib                         # For face detection ML
import cv2                          # Import OpenCV
import csv                          # For storing landmarks to a file

# GUI Imports
```

```
Face Detection Complete
Testing: Transcription
fast
2, wink, 3, smile
Adding new Record to DB
Ready for next record - #64
lmvideo_64.avi
lmvideo_64.avi
New listbox video selected
lmvideo_63.avi
['lmvideo', '63.avi']
['63', 'avi']
63
play btn was pressed
```

Figure 9 - Final Outcome

Next Steps

I would really like to clean up the GUI some more and fix some of the challenges mentioned previously. I would also like to play with TensorFlow Records and to see what interesting results could come from that. I would also like to see this program adapted to matching the user's movements and facial gestures to the robot in real time. Real time mimicking of a person by a robot would be really interesting. In the manufacturing plants I have worked at most of the robots are trained by an operator with a training wand. If a similar method could be used to train a robot's face, I think it could be quite interesting.

Conclusion

The project still has a decent amount of effort to go, but overall, it is functional and provides us with a tool to record videos and landmarking data. It turned out to be a rather interesting project as it required me to dig into all sorts of areas of python from GUIs, threading, JSON, computer vision, and all sorts of other topics.

Appendix

Setup/Run Instructions

Unzip video-landmarking-dataset.zip in a working directory.
The program runs using an Anaconda Virtual environment, below is the following list of dependencies to be installed:

```
pip install numpy scipy matplotlib scikit-learn jupyter
pip install cmake
pip install opencv
pip install imutils
pip install dlib
pip install pygame
pip install tk
pip install tkvideo
conda install ffmpeg
conda install pyaudio
```

NOTE: If your USB camera/audio is on a different index than default. You will need to adjust them in avrecorder.py; there is provided a Test Code > deviceFinder.py to determine your index.

NOTE: Sadly, some of the process is still hard coded to a path.
You will need to update paths inside landmarkRecorder.py, change line 44 and 340 accordingly.

If running in vscode update settings.json path; it should run from there.
If using the command line, type: python3 ./src/landmarkRecorder.py

You should be prompted for a database json file. One is provided at db/phraseDB.json
It does not have good error handling for incorrect or empty databases yet.

The GUI should pop up next.
You can enter in a phrase, annotations, and set the speaking speed.
When ready, hit the record button.
Note: Currently the GUI freezes for the 15 second record window.
You can see start/stop messages in the terminal though.
This is due to the way the threading is done currently.

After recording the new video should automatically be added to the playlist.
You can hit play and watch the new lmvideo_XX.avi or the raw 15s video XX.avi
The landmarked video is sped up 5x due to some issues with the landmarking.
It is compressing the 90 frames into 3s instead of spreading them out over 15s.

When done you can exit the GUI and the program should terminate.

References

1. Facial landmarks with dlib, OpenCV, and Python
   https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

2. Dlib landmarking library
   http://dlib.net/

3. OpenCV Computer Vision Library
   https://opencv.org/

4. iBUG 300W 68-Point facial database (Chosen Training Set)
   https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

5. 5 Million faces (Free image datasets)
   https://lionbridge.ai/datasets/5-million-faces-top-15-free-image-datasets-for-facial-recognition/

6. Smile Capturer (Euclidean distance between facial points using scipy.spatial)
   https://www.freecodecamp.org/news/smilfie-auto-capture-selfies-by-detecting-a-smile-using-opencv-and-python-8c5cfb6ec197/

7. Display Videos with Tkinter
   https://www.develog.net/2018/09/19/open-and-show-videos-in-tkinter/

8. Google Cloud Speech-to-Text
   https://cloud.google.com/speech-to-text

9. Python to SQL Database
   https://www.freecodecamp.org/news/connect-python-with-sql/

10. Fritz Robotic Head (Used in future projects)
    https://kerkits.com/products/fritz-the-robotic-head