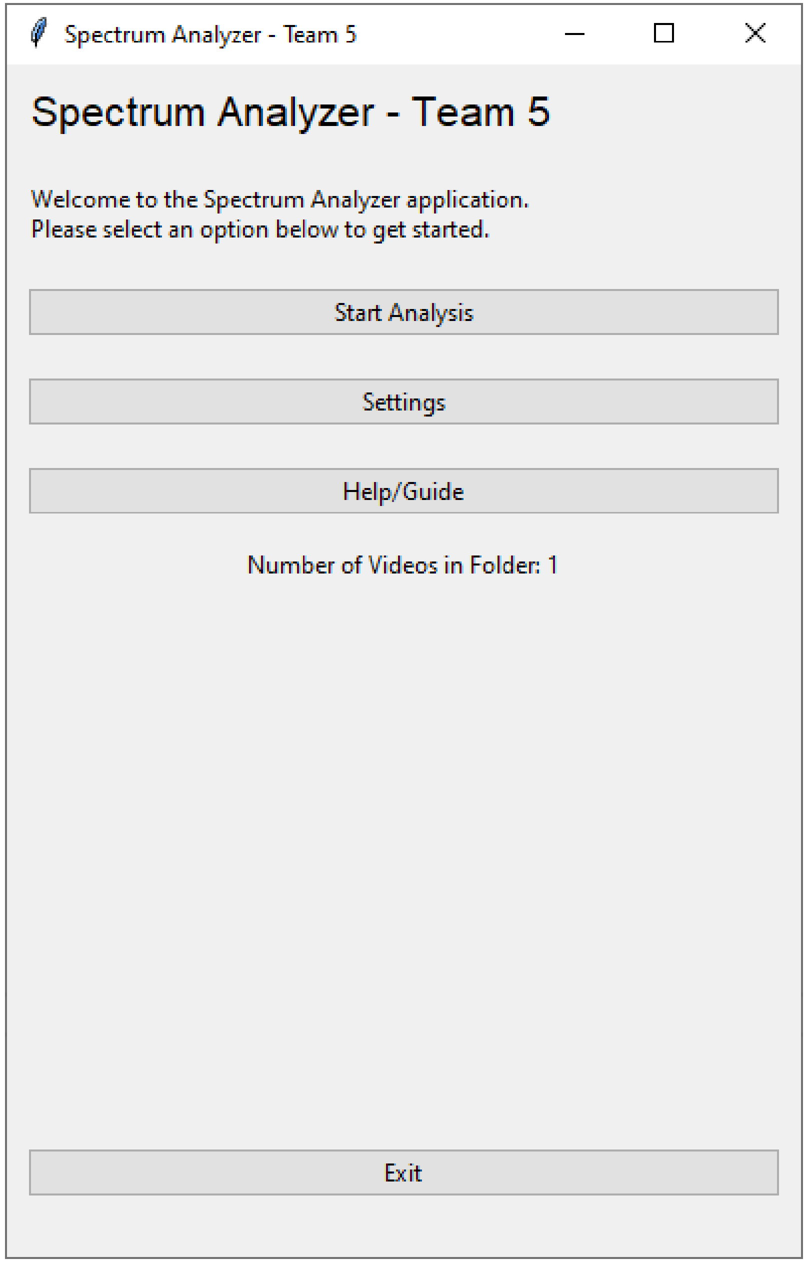
Project Overview

Team 5

Team members: Masood Afzali, Ashly Altman, Brooke Ebetino, Tyler Haley, Joey Thompson

Goal: Analyze spectrum data from videos using OpenCV and Python



Spectrum Analyzer - Team 5

Welcome to the Spectrum Analyzer application.
Please select an option below to get started.

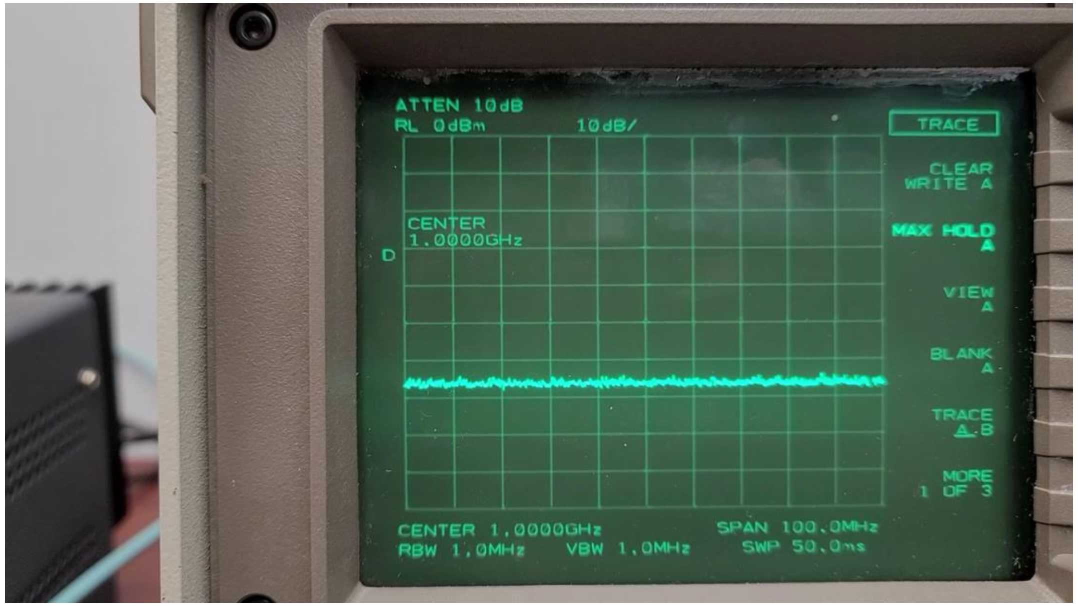
Number of Videos in Folder: 1

Problem Statement

Robins AFB has videos containing a spectrum analyzer screen with a signal being displayed requiring analysis.

Develop a program to record signal data instead of a person having to manually watch a video by hand

Record signal attributes for analysis by a person to see if results fall outside of acceptable ranges

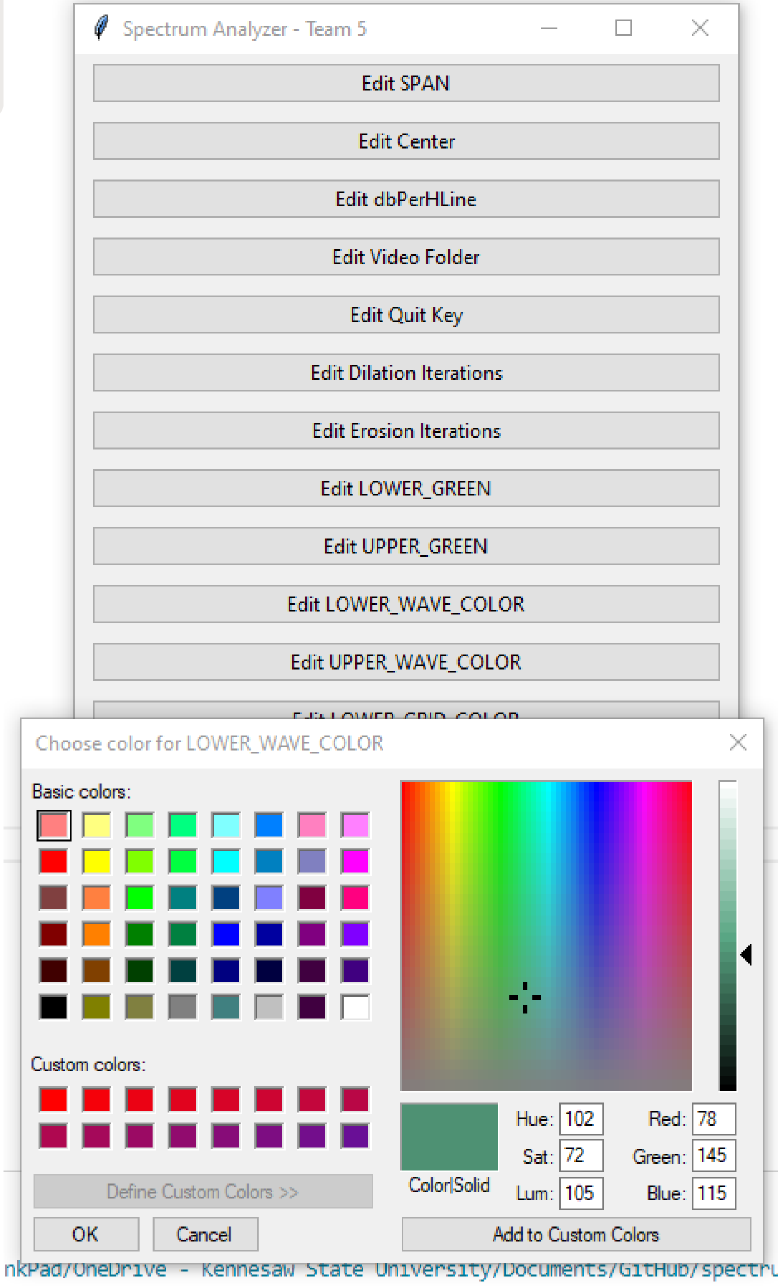


Technologies

- Python for Project
  - Multiprocessing for running simultaneous videos
  - DateTime for naming CSV files
  - Webbrowser for opening Video folder after processing
- Tkinter for GUI
- OpenCV for Image Processing
- NumPy for Numeric Operations
- SciPy for finding line of best fit

OpenCV for Image Processing

- Captures and processes video frames
- Applies color filters to isolate wave
- Detects contours to find wave shape



Spectrum Analyzer - Team 5

Choose color for LOWER_WAVE_COLOR

Hue: 102
Sat: 72
Lum: 105
Red: 78
Green: 145
Blue: 115

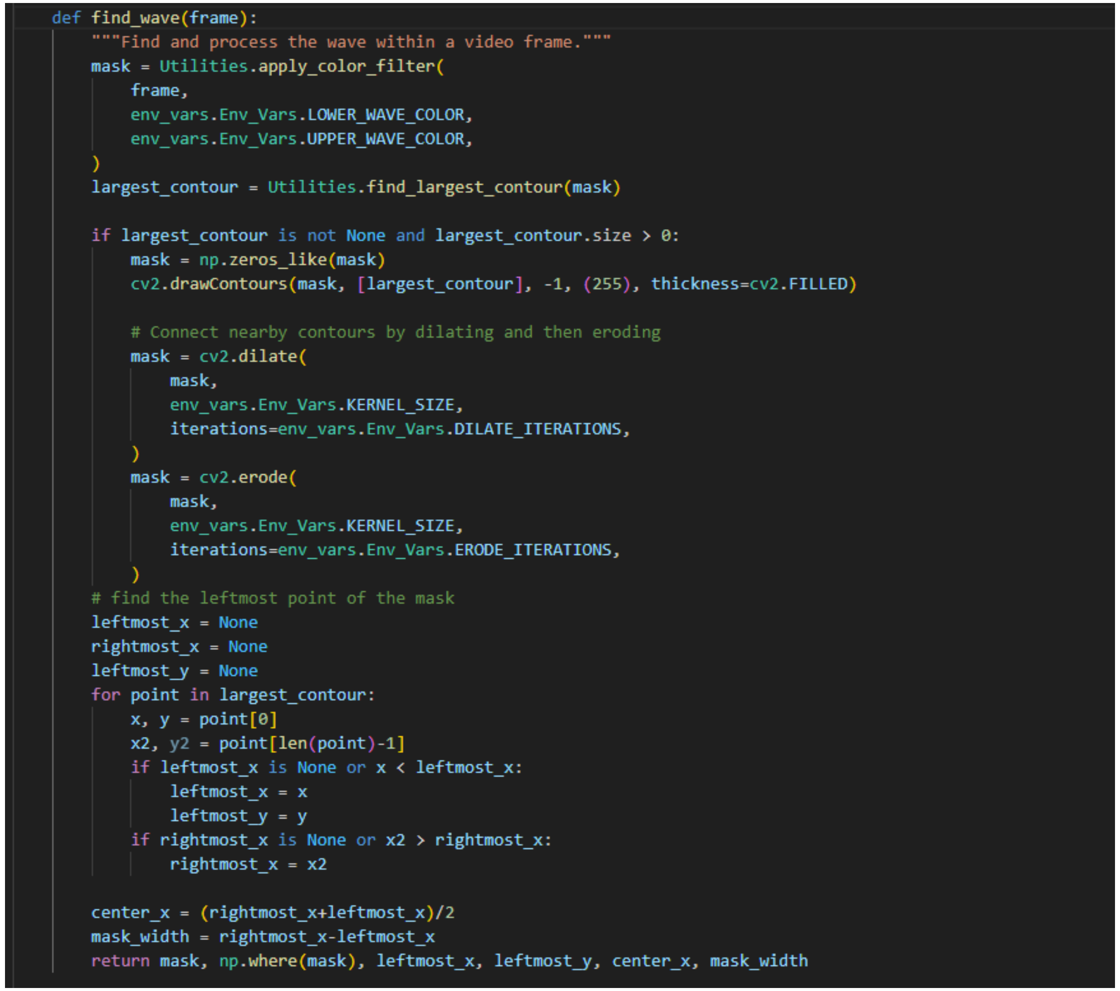
nkPad/OneDrive - Kennesaw State University/Documents/GitHub/spectru

Numpy for Data Analysis

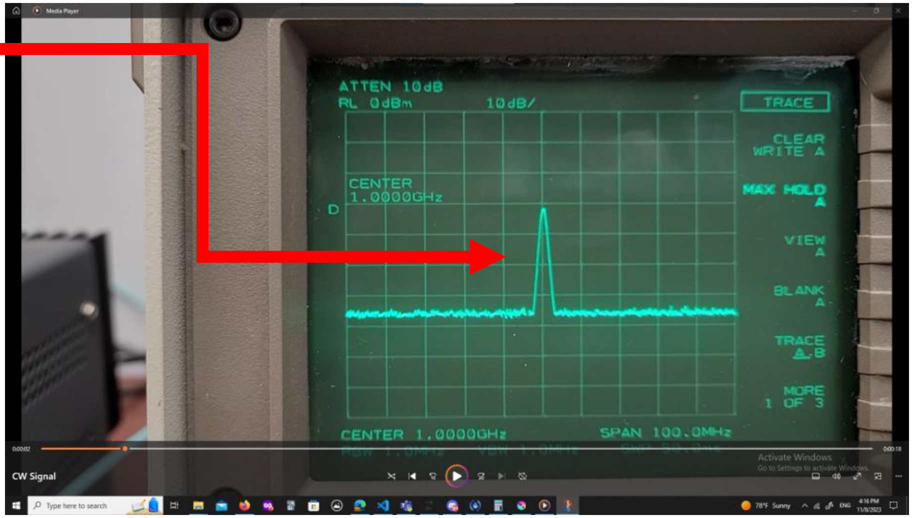
- Stores pixel data as arrays
- Fits parabola curve to wave
- Calculates frequency from curve vertex

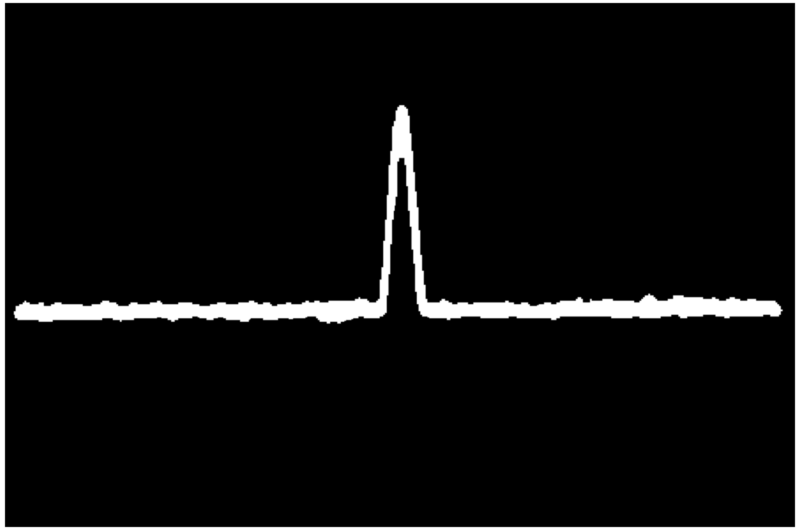
Extracting Data from Video

- Detect the wave and generate wave mask



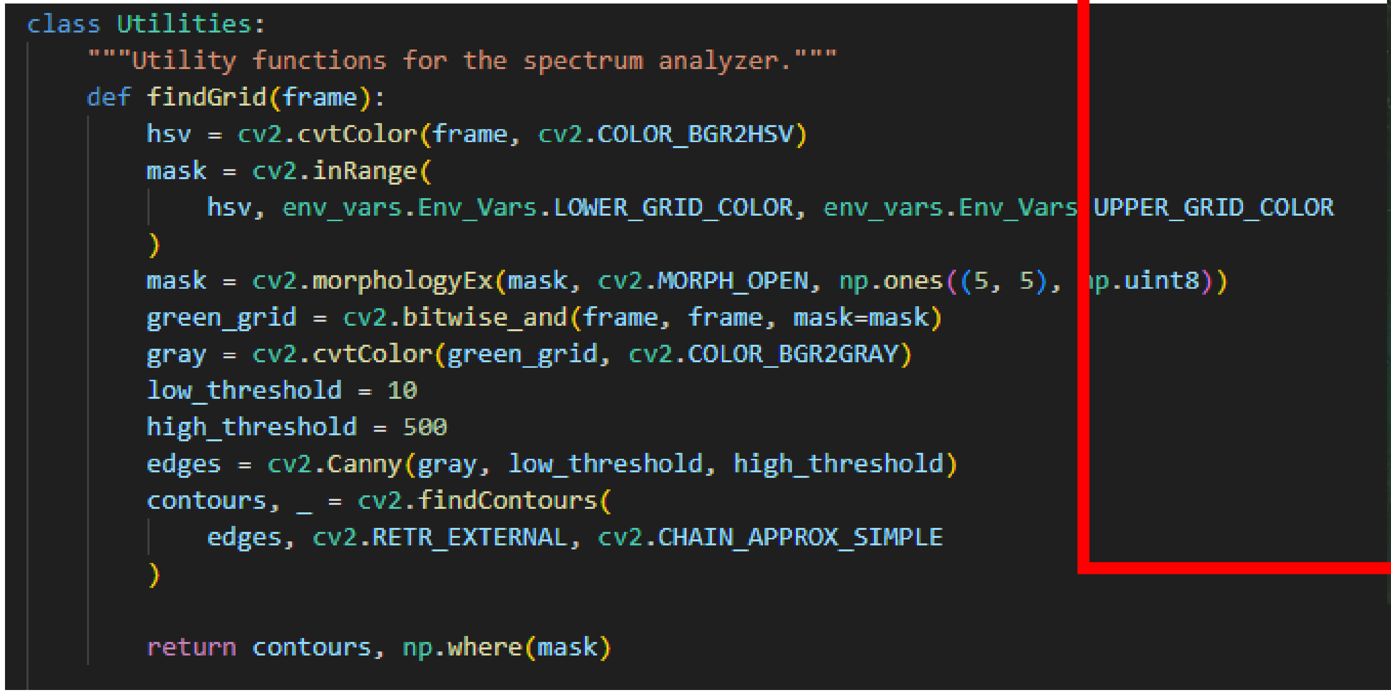
```python
def find_wave(frame):
    """Find and process the wave within a video frame."""
    mask = Utilities.apply_color_filter(
        frame,
        env_vars.Env_Vars.LOWER_WAVE_COLOR,
        env_vars.Env_Vars.UPPER_WAVE_COLOR,
    )
    largest_contour = Utilities.find_largest_contour(mask)

    if largest_contour is not None and largest_contour.size > 0:
        mask = np.zeros_like(mask)
        cv2.drawContours(mask, [largest_contour], -1, (255), thickness=cv2.FILLED)

        # Connect nearby contours by dilating and then eroding
        mask = cv2.dilate(
            mask,
            env_vars.Env_Vars.KERNEL_SIZE,
            iterations=env_vars.Env_Vars.DILATE_ITERATIONS,
        )
        mask = cv2.erode(
            mask,
            env_vars.Env_Vars.KERNEL_SIZE,
            iterations=env_vars.Env_Vars.ERODE_ITERATIONS,
        )
    # find the leftmost point of the mask
    leftmost_x = None
    rightmost_x = None
    leftmost_y = None
    for point in largest_contour:
        x, y = point[0]
        x2, y2 = point[len(point)-1]
        if leftmost_x is None or x < leftmost_x:
            leftmost_x = x
            leftmost_y = y
        if rightmost_x is None or x2 > rightmost_x:
            rightmost_x = x2

    center_x = (rightmost_x+leftmost_x)/2
    mask_width = rightmost_x-leftmost_x
    return mask, np.where(mask), leftmost_x, leftmost_y, center_x, mask_width
```



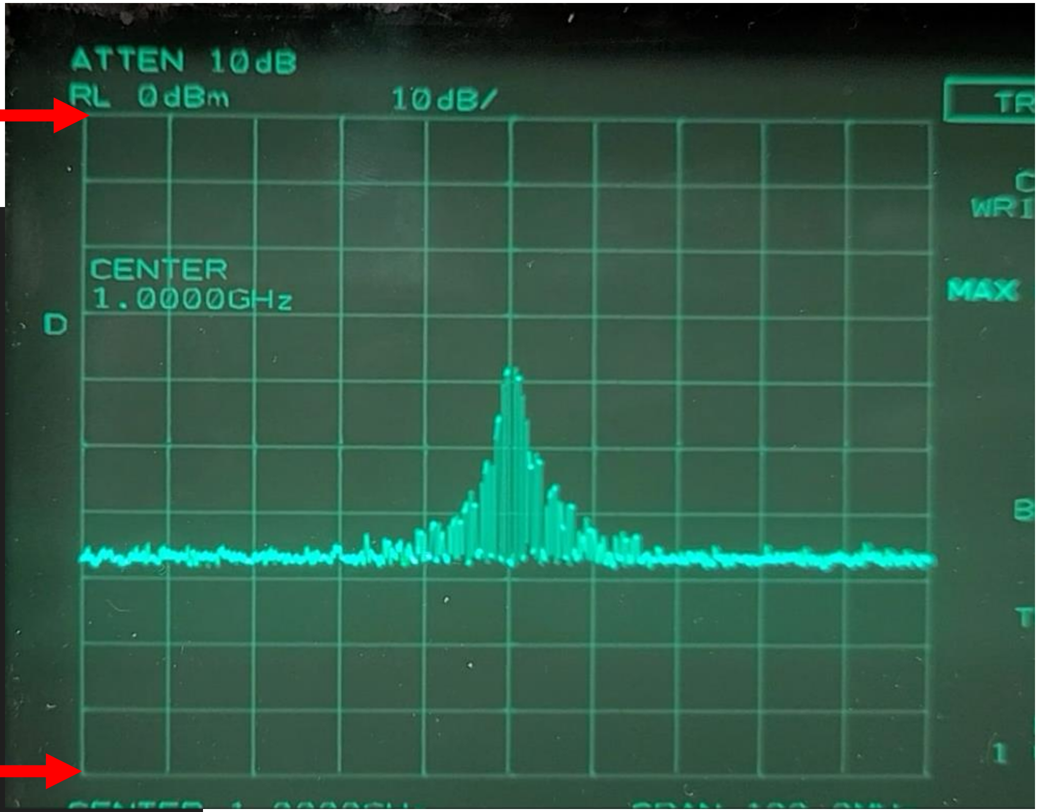


Extracting Data from Video

- Detect the grid
- Generate grid mask



```
class Utilities:
    """Utility functions for the spectrum analyzer."""
    def findGrid(frame):
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(
            hsv, env_vars.Env_Vars.LOWER_GRID_COLOR, env_vars.Env_Vars.UPPER_GRID_COLOR
        )
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, np.ones((5, 5), np.uint8))
        green_grid = cv2.bitwise_and(frame, frame, mask=mask)
        gray = cv2.cvtColor(green_grid, cv2.COLOR_BGR2GRAY)
        low_threshold = 10
        high_threshold = 500
        edges = cv2.Canny(gray, low_threshold, high_threshold)
        contours, _ = cv2.findContours(
            edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
        )

        return contours, np.where(mask)
```

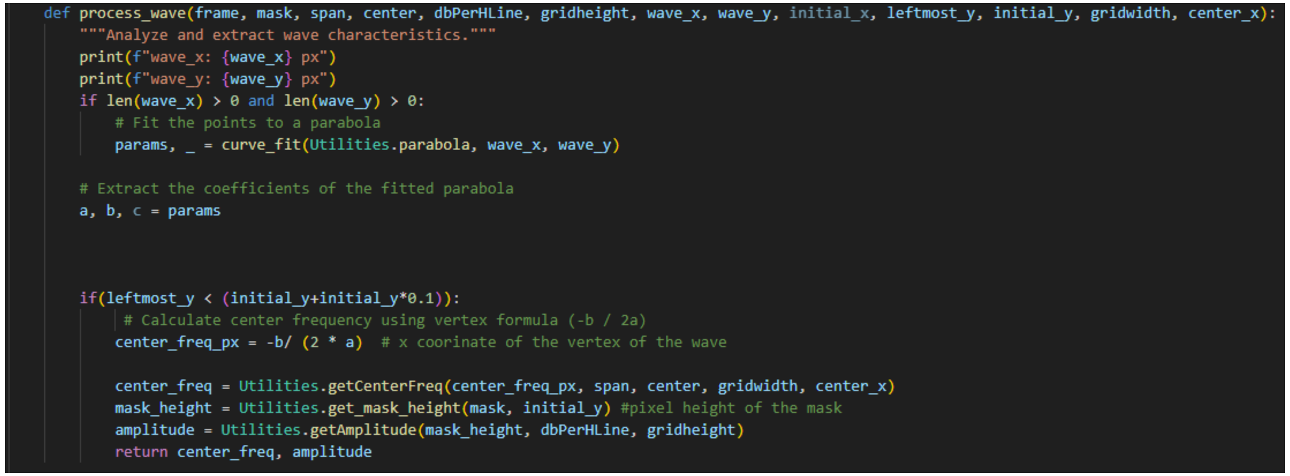


Extracting Data from Video

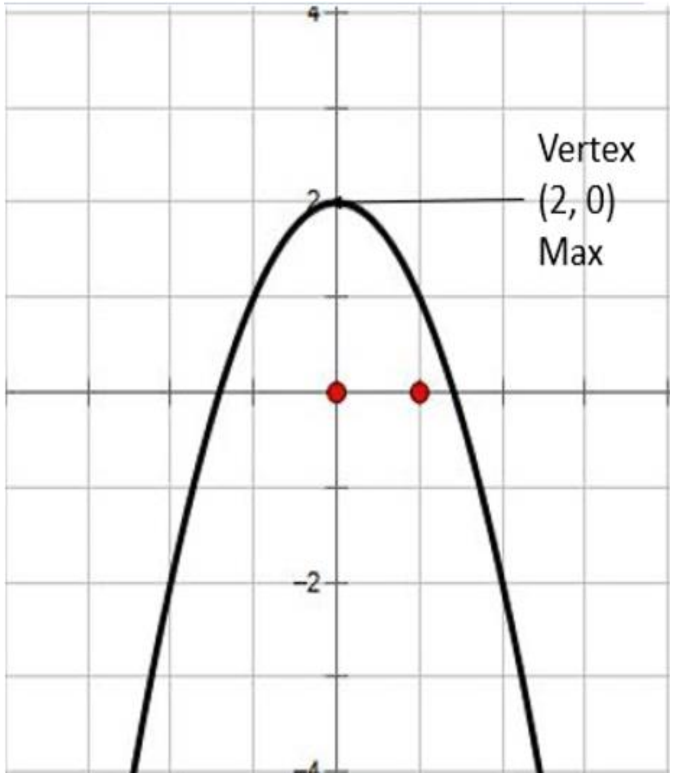
- Extract the coefficients of the wave using python's curve_fit() function which fits the x,y coordinates of the wave to the defined curve, parabola
- Find the center frequency of the wave using the vertex formula:

  $Y = ax^2 + bx + c$, x coordinate of vertex = $-b/2a$

- Use center frequency, mask height, and amplitude functions to calculate the desired data from the wave



```python
def process_wave(frame, mask, span, center, dbPerHLine, gridheight, wave_x, wave_y, initial_x, leftmost_y, initial_y, gridwidth, center_x):
    """Analyze and extract wave characteristics."""
    print(f"wave_x: {wave_x} px")
    print(f"wave_y: {wave_y} px")
    if len(wave_x) > 0 and len(wave_y) > 0:
        # Fit the points to a parabola
        params, _ = curve_fit(Utilities.parabola, wave_x, wave_y)

    # Extract the coefficients of the fitted parabola
    a, b, c = params



    if(leftmost_y < (initial_y+initial_y*0.1)):
        # Calculate center frequency using vertex formula (-b / 2a)
        center_freq_px = -b/ (2 * a)  # x coorinate of the vertex of the wave

        center_freq = Utilities.getCenterFreq(center_freq_px, span, center, gridwidth, center_x)
        mask_height = Utilities.get_mask_height(mask, initial_y) #pixel height of the mask
        amplitude = Utilities.getAmplitude(mask_height, dbPerHLine, gridheight)
        return center_freq, amplitude
```



Extracting Data from Video

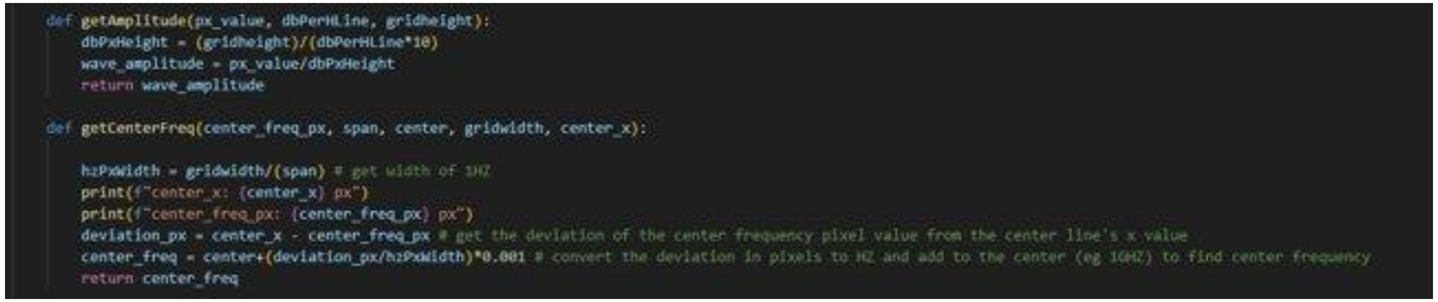
Use pixel coordinate data of the masks

with the user entered parameters

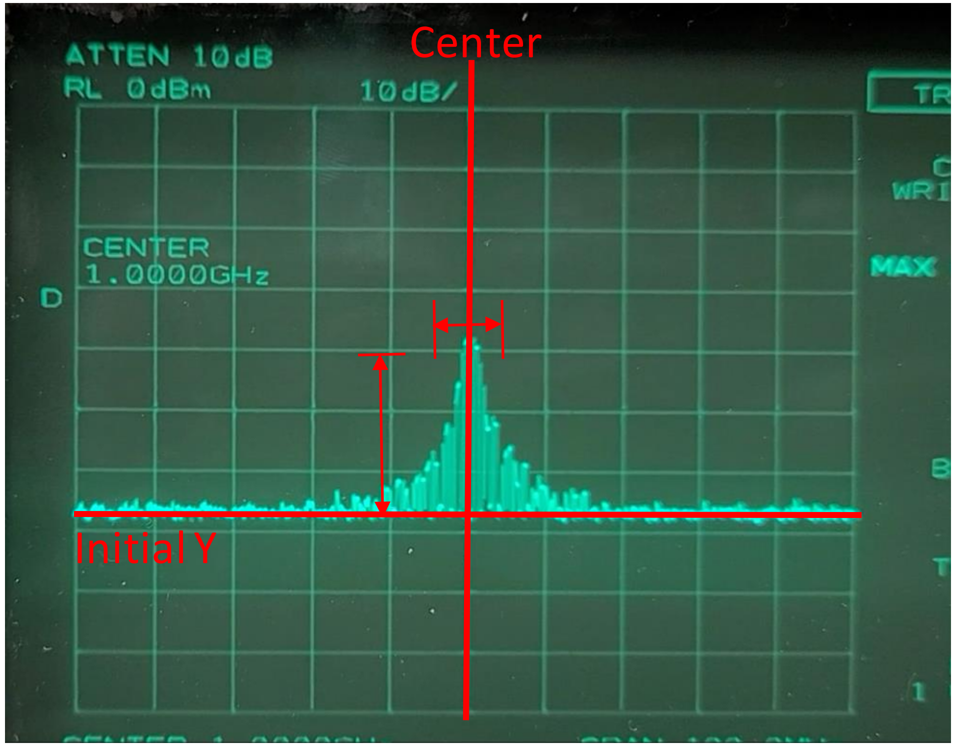
to calculate Center Frequency

and Amplitude of the signal

- Find the center frequency using the deviation from the center
- Find the amplitude using the height of the wave in relation to the initial Y coordinate of the wave (ignore data when y coordinate of mask is below initial y, i.e. when screen is being cleared)
- Convert pixel dimension to HZ and dB by finding the pixel dimensions of 1HZ and 1dB



```
def getAmplitude(px_value, dbPerHLine, gridheight):
    dbPxHeight = (gridheight)/(dbPerHLine*10)
    wave_amplitude = px_value/dbPxHeight
    return wave_amplitude

def getCenterFreq(center_freq_px, span, center, gridwidth, center_x):

    hzPxWidth = gridwidth/(span) # get width of 1HZ
    print(f"center_x: {center_x} px")
    print(f"center_freq_px: {center_freq_px} px")
    deviation_px = center_x - center_freq_px # get the deviation of the center frequency pixel value from the center line's x value
    center_freq = center+(deviation_px/hzPxWidth)*0.001 # convert the deviation in pixels to HZ and add to the center (eg 1GHZ) to find center frequency
    return center_freq
```



Multiprocessing

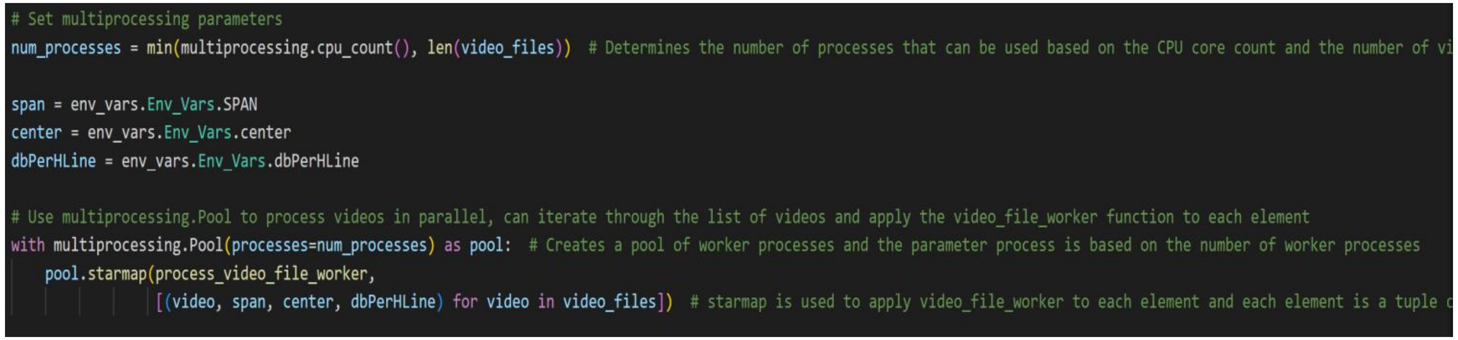
PROCESSES VIDEOS IN PARALLEL

DRAMATICALLY FASTER THAN SEQUENTIAL

USES MULTIPROCESSING LIBRARY

Multiprocessing

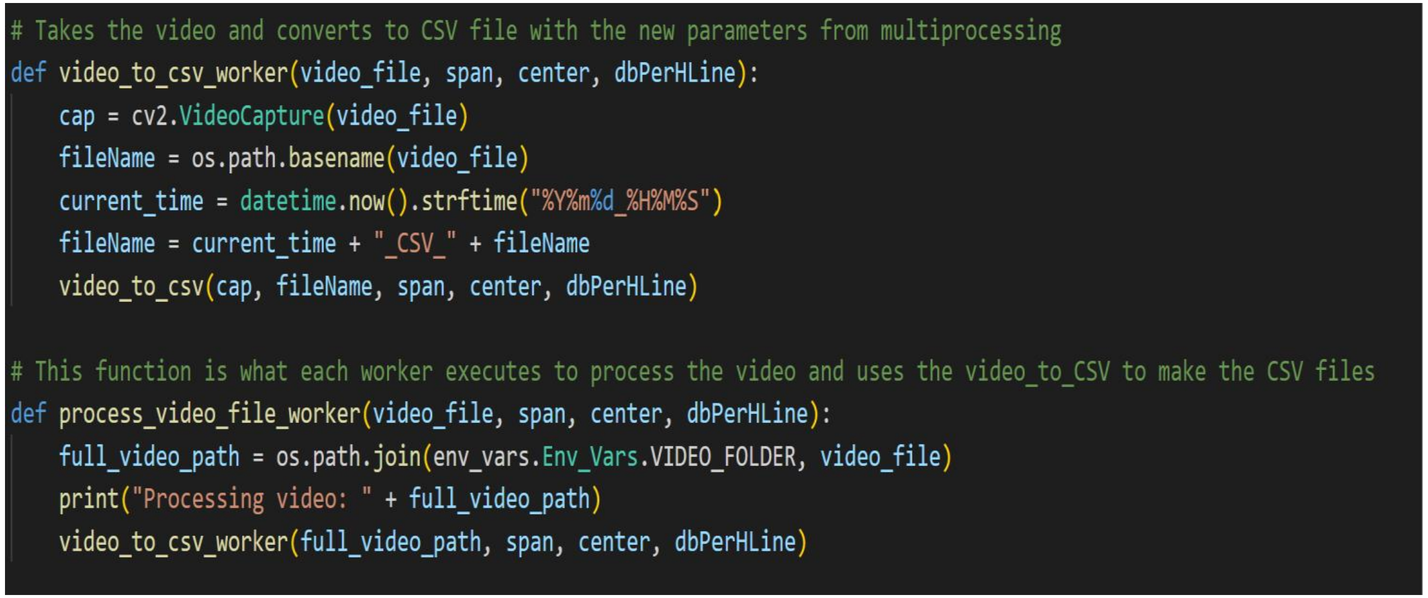
- Imports the multiprocessing feature from python.
- The number of processes is determined based on the CPU core count and number of videos.
- Makes a pool of worker processes where the parameter process is based on the number of processes.
- Then makes a starmap to apply video_file_worker to each element, where each is a tuple containing the video file name and new parameters.



```python
# Set multiprocessing parameters
num_processes = min(multiprocessing.cpu_count(), len(video_files))  # Determines the number of processes that can be used based on the CPU core count and the number of vi

span = env_vars.Env_Vars.SPAN
center = env_vars.Env_Vars.center
dbPerHLine = env_vars.Env_Vars.dbPerHLine

# Use multiprocessing.Pool to process videos in parallel, can iterate through the list of videos and apply the video_file_worker function to each element
with multiprocessing.Pool(processes=num_processes) as pool:  # Creates a pool of worker processes and the parameter process is based on the number of worker processes
    pool.starmap(process_video_file_worker,
                 [(video, span, center, dbPerHLine) for video in video_files])  # starmap is used to apply video_file_worker to each element and each element is a tuple c
```

Multiprocessing

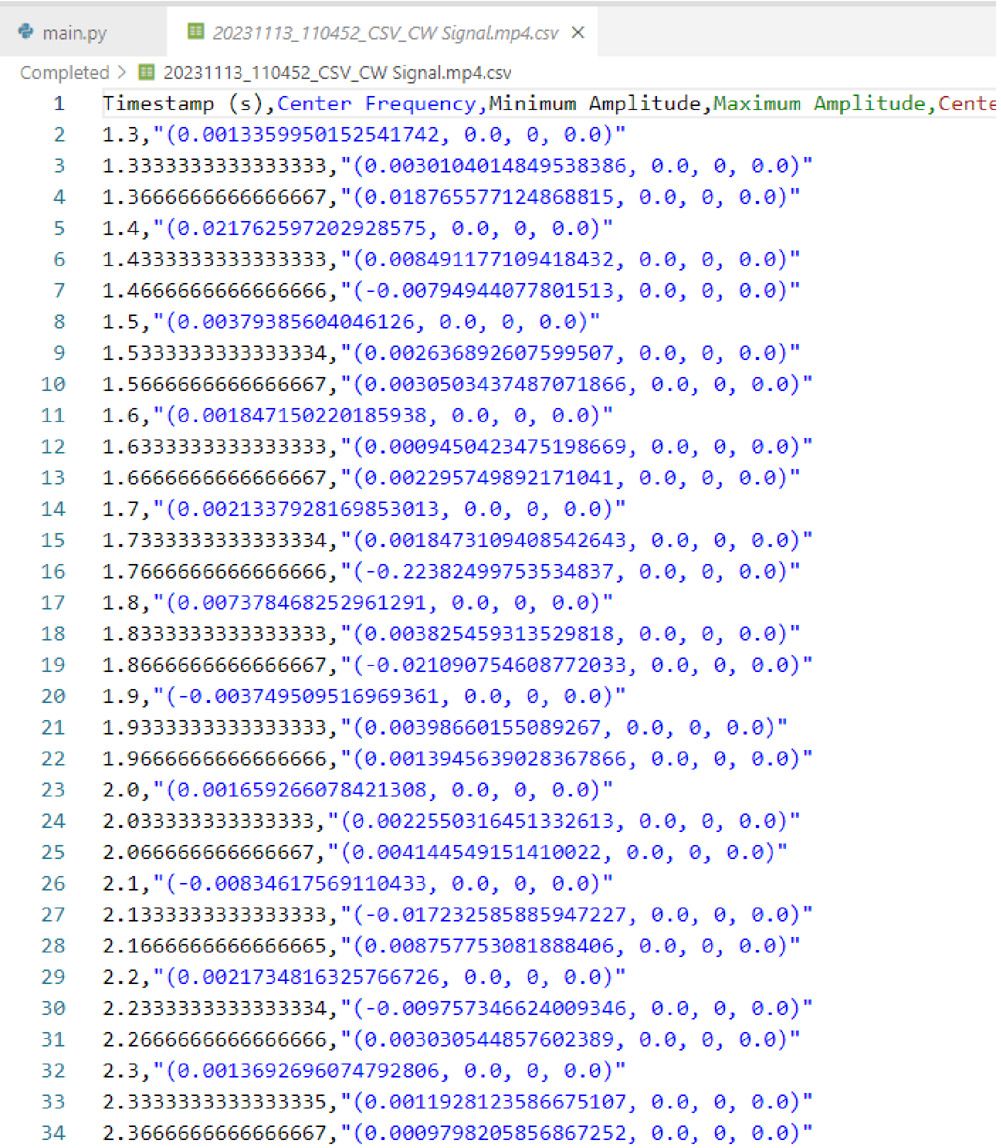
- The video_to_csv_worker takes in the video file with its parameters and converts it to a CSV file with current date and time.
- The process_video_file_worker takes in the video file and parameters and then calls the video_to_csv_worker once it is processing through the video. The data is then put to the CSV where it can be looked at.



```
# Takes the video and converts to CSV file with the new parameters from multiprocessing
def video_to_csv_worker(video_file, span, center, dbPerHLine):
    cap = cv2.VideoCapture(video_file)
    fileName = os.path.basename(video_file)
    current_time = datetime.now().strftime("%Y%m%d_%H%M%S")
    fileName = current_time + "_CSV_" + fileName
    video_to_csv(cap, fileName, span, center, dbPerHLine)

# This function is what each worker executes to process the video and uses the video_to_CSV to make the CSV files
def process_video_file_worker(video_file, span, center, dbPerHLine):
    full_video_path = os.path.join(env_vars.Env_Vars.VIDEO_FOLDER, video_file)
    print("Processing video: " + full_video_path)
    video_to_csv_worker(full_video_path, span, center, dbPerHLine)
```

Output CSV File

- Timestamped results for each frame
- Frequency and amplitude data
- Plot spectrum over time from CSV


main.py
20231113_110452_CSV_CW Signal.mp4.csv
Completed > 20231113_110452_CSV_CW Signal.mp4.csv

```
Timestamp (s),Center Frequency,Minimum Amplitude,Maximum Amplitude,Cente
1.3,"(0.0013359950152541742, 0.0, 0, 0.0)"
1.3333333333333333,"(0.0030104014849538386, 0.0, 0, 0.0)"
1.3666666666666667,"(0.018765577124868815, 0.0, 0, 0.0)"
1.4,"(0.021762597202928575, 0.0, 0, 0.0)"
1.4333333333333333,"(0.008491177109418432, 0.0, 0, 0.0)"
1.4666666666666666,"(-0.00794944077801513, 0.0, 0, 0.0)"
1.5,"(0.00379385604046126, 0.0, 0, 0.0)"
1.5333333333333334,"(0.002636892607599507, 0.0, 0, 0.0)"
1.5666666666666667,"(0.0030503437487071866, 0.0, 0, 0.0)"
1.6,"(0.001847150220185938, 0.0, 0, 0.0)"
1.6333333333333333,"(0.0009450423475198669, 0.0, 0, 0.0)"
1.6666666666666667,"(0.002295749892171041, 0.0, 0, 0.0)"
1.7,"(0.0021337928169853013, 0.0, 0, 0.0)"
1.7333333333333334,"(0.0018473109408542643, 0.0, 0, 0.0)"
1.7666666666666666,"(-0.22382499753534837, 0.0, 0, 0.0)"
1.8,"(0.007378468252961291, 0.0, 0, 0.0)"
1.8333333333333333,"(0.003825459313529818, 0.0, 0, 0.0)"
1.8666666666666667,"(-0.021090754608772033, 0.0, 0, 0.0)"
1.9,"(-0.003749509516969361, 0.0, 0, 0.0)"
1.9333333333333333,"(0.00398660155089267, 0.0, 0, 0.0)"
1.9666666666666666,"(0.0013945639028367866, 0.0, 0, 0.0)"
2.0,"(0.001659266078421308, 0.0, 0, 0.0)"
2.033333333333333,"(0.0022550316451332613, 0.0, 0, 0.0)"
2.066666666666667,"(0.004144549151410022, 0.0, 0, 0.0)"
2.1,"(-0.00834617569110433, 0.0, 0, 0.0)"
2.1333333333333333,"(-0.017232585885947227, 0.0, 0, 0.0)"
2.1666666666666665,"(0.008757753081888406, 0.0, 0, 0.0)"
2.2,"(0.0021734816325766726, 0.0, 0, 0.0)"
2.2333333333333334,"(-0.009757346624009346, 0.0, 0, 0.0)"
2.2666666666666666,"(0.003030544857602389, 0.0, 0, 0.0)"
2.3,"(0.0013692696074792806, 0.0, 0, 0.0)"
2.3333333333333335,"(0.0011928123586675107, 0.0, 0, 0.0)"
2.3666666666666667,"(0.0009798205856867252, 0.0, 0, 0.0)"
```

Live Demo

Questions?

Thank you!