ENG 06 Winter 2017 Final Project

Collaboration Policy:

You will form or be assigned to a team of three. The team chooses from three project options. You are only allowed to talk and collaborate with members within your team. Team members are expected to equally participate, and collaboratively work towards the completion of the project. Other than contacting the teaching assistants for clarification, you may not seek the assistance of other persons.

Project Deadlines:

Deadline #1: March 2nd, 9:10 PM One member of each team must sign up during the sign up session. Have a team name, the list of team members including student ID numbers, and a ranked list of projects. Projects will be allocated to teams on a first-come, first-served basis. Use Piazza to find other students to help complete your team. If you cannot find a team, you will be assigned to one.

Deadline #2: March 5th, 11:55 pm: Provide a breakdown of programming tasks, and who will be responsible for what, along with a timeline that will meet the submission deadline. The more specific you can be in defining the programming tasks, what functions should exist, and what each function should accomplish, the better. How the submission will be done will be announced.

Deadline #3: March 18th, 11:55 pm: Submission of all materials required by project. How the submission will be done will be announced.

Grading Criteria:

The projects are open ended. As long as your program can perform the assigned tasks, there will be no correct or incorrect approaches. Certainly there will be more acceptable and attractive solutions, and that will be judged in comparison with competing solutions submitted by your classmates. Projects will be judged against the other projects in the same group that have been submitted. The grading will also take into account that Project 3 is more formulated while Project 1 & 2 are more open ended.

The expectation is that each team member must take responsibility for a specific aspect of the project and grading for each member will be adjusted according to how the project tasks were delegated and who was responsible for what aspects of the project. Each project will allocate at least 10% of the grade to a section that must be included as Appendix A. Appendix A must contain:

- A table with the breakdown of the tasks to complete the project, and who was responsible for what part of the project. The intent here is to determine who did what to implement the project. While it is perfectly reasonable that some tasks can be completed jointly, it is unrealistic to claim that everyone worked together on all aspects of the project equally.

- Each member must provide a brief personal summary of that person's involvement and contributions. Before the project is submitted, the summaries must be provided to all members for review and comment. The end of Appendix A must include the following language: **“All team members have read the task summaries contained in this report and have been given an opportunity to comment. “**

Project Report Requirements:

Each project submission must have a project report that contains any relevant material deemed essential, and it must contain an Appendix A as described previously. The first page of the report must contain the team name and the names of all members. This file must be saved as a PDF document.

The report must provide clear explanations on how your program works, and how problems encountered in the project are solved. Use flowcharts or diagrams to help illustrate the program’s functionality.

The length of your report is what your team feels is required to fully document and explain your team’s work. It is good to keep the report to within a reasonable length, while being concise.

In the event that you have used external resources, you must provide appropriate credit in the project report or in your program code. If it is discovered that you have borrowed or used material from a source that is not credited, it will be considered plagiarism and the case will be turned over to Student Judicial Affairs.

Youtube Video Requirements:

The format of the video is entirely up to your team as long as the following criteria are met:

- Maximum length of the video is 10 minutes
- Each team member must be seen in the video to present their work and contributions
- A clear and easy to follow demonstration that shows the correct functionality of your program (show you program actually working in the video – not screen shots of before and after.)

  Use visual aides to help explain your steps (whiteboard, markers, poster, etc.). The video does not have to be fancy, just effective in relaying the most important information.

Project #1: Video Processing

Introduction

Signal processing is a common task that is performed in MATLAB due to its great ability to deal with and manipulate large sets of data. Signals can come in a variety of types - continuous, discrete, 1D, 2D, and even 3D! For 2D signals (i.e. images and videos), we can use matrices to represent the color information present in each pixel of an image. This is done by stacking three regular 2D arrays, where each array is shows the R,G, or B saturated color intensity. By combining the individual color matrices, we arrive at the final composed image. Similarly, a video feed consists of a stack of these images, and when played back at a certain framerate (the number of frames per second to show the viewer), we can visually observe smooth video playback.

An overview of the project requirements is next and a complete list of requirements is presented below. The core of this project consists of designing a simple video player implemented in a MATLAB constructed GUI. The player basic video player must be designed so that the user can select any valid (and supported) video file to playback. To play the video, your GUI must have the standard array of buttons found on typical digital media players (play/pause, stop, step-forward, step-reverse). Your GUI must also include a scroll bar that (in real-time) scrolls while the video is playing.

In addition to the base video playback function, your GUI must contain 2 special features that enhance the capabilities of the project. The special features can be decided by your team and they do not have to relate to each other. For example, you can choose to enable sound playback on your GUI by including a functional “mute” button. You can also choose to analyze each frame of the video and compute the number of saturated red, green, or blue pixels in the image. Such information will of course be valuable to a film maker.

An example GUI with two special features is shown later in this document. Note that this GUI has all of the fundamental features that are required for the core of this project. Moreover, this GUI also has two notable “special” features - one that shows a live histogram of the RGB elements of each frame, and another that shows a sharpened version of the original video stream by using a sharpening filter.

Requirements for the Project Core

1. Your GUI must have a button that will allow the user to load any valid video file for playback. Note that you must be able to load videos outside of the working folder in MATLAB. One required file format is (.mp4) but you are free to add others if you so desire.

2. Upon successful loading of a video file, you must show the following bits of information about your video stream:
    a. The name of the video file (e.g. test.mp4).
    b. The resolution of the video file (e.g. 1920x1080).

    c. The framerate of the video file (e.g. 25fps).
    d. The total duration of the video file (e.g. 1h 14m 08s).

   If there is an issue loading the video file, then an error must be displayed in the form of a message box to the user.

3. Once the video file is loaded and the required bits of information are displayed to the user, the video should not start playing on its own. To play the video, the user will be required to hit one of the appropriate play button options. The following buttons are required for the core of this project and must be implemented with the exact functionality mentioned below.
    a. Play/Pause button. This button should be able to play and pause the video at any point! If Play is pressed after the video has been stopped, then the video will be played back from the first frame. However, if Play is pressed after the video has been paused, then the video will continue playing from that frame. In addition, the text or graphic of the Play/Pause button must change appropriately from "Play" to "Pause" and vice-versa depending on the current available transition of that button. For example, if the video is currently paused, then the Play/Pause button must show "Play".
    b. Stop button. This button will stop the video stream. If the video is currently stopped then the text for the Play/Pause button must show "Play". When the Play button is pressed, the video must play back from the first frame!
    c. Step-Forward button. This button pauses the video stream and advances to the next frame. Note that this causes the Play/Pause button to show "Play" since that is the appropriate available action for the Play/Pause button. If this button is pressed at the last frame of the video, then nothing should happen!
    d. Stop-Back button. This button pauses the video stream and steps back one frame. If this button is pressed at the first frame, then nothing should happen!

4. It's quite boring to have the video stream controllable solely from a few buttons, so it makes sense to include a more interactive way of "searching" through a video stream using a Scroll Bar. Some requirements for the scroll bar:
    a. The Scroll Bar must be working in real-time and must move as the video file is played. If the video is paused, then the scroll bar must stop moving and remain fixed in place. Similarly, if the video is stopped, then the scroll bar must relocate to the beginning.
    b. If the Scroll Bar is moved (regardless of the video being played back, paused, or stopped) the video stream must pause and advance to show the proper frame. In addition, the Play/Pause button will show "Play" as the next available state. Note that the video screen must show the current paused frame as selected by the Scroll Bar. Lastly, all of your "special" features must update along with the movement of the Scroll Bar (if applicable).

5. Your GUI must obviously have a video playback screen. If a video file with an extremely large or small resolution is loaded, then your GUI must be able to resize the video's dimensions to fit the screen in the GUI. Note that this resizing may cause some aliasing in certain video files. You are free, though not required, to alleviate this issue by adding a basic "anti-aliasing filter" as a special feature.

6. The core GUI must be designed only to playback the video portion of a video file, no sound is required. In addition, you must try to playback the video at the intended framerate in the video file.

Requirements for the Special Features

Your program is required to have at least two special features. Once again, these special features are for you to increase the feature set of your program. You may choose any two special features from this list but you are also encouraged to come up with a new idea! If you do wish to propose your own feature(s), be sure to obtain approval from Saneel (snkhatri@ucdavis.edu) before you write any code for the feature you wish to implement.

Keep in mind that some special features are much more difficult to implement than others, but those more difficult to implement will of course do much more interesting operations to the video stream. As an added bonus, attempting to implement a difficult special feature may earn you between 5-10% extra credit for the project grade. Again, you are free to propose your own complex special feature or you can choose one from the list below (marked with ***).

1. Real-time RGB Histogram - implement a histogram that shows the color content of each frame. This should be a single figure with three colored plots showing the pixel count vs intensity for each of the red, green, and blue color channels. An example of this feature can be found in example GUI shown below. From this information, you should be able to detect and display when a particular frame is over-saturated (i.e. consists of a lot of 255's for uint8).

2. Color/Saturation Correction - You can implement a series of small slider to apply small gains to the RGB pixels to change the color of the image. If you do choose to implement this, your sliders should work in real-time.

3. Filtering*** - You can implement any of the filters below or propose an entirely new filter!
    a. Anti-Aliasing Filter - this filter removes sharp "pixelated" edges in images by averaging neighboring pixels.
    b. Sharpening Filter - this filter performs element-by-element multiplication with a small matrix (known as a kernel) to highlight and sharpen the image.
    c. Smoothing Filter - this filter does the exact opposite of the sharpening filter to smooth out overly-sharp images.

    Note that if you choose to implement a filter, you will be required to construct your own filtering algorithm, not one already available within MATLAB.

4. Frame-Rate Conversion*** - This would involve implementing your own 2D interpolator to mix sequential frames together. The idea behind this is to create new frames to "upsample" the video stream. Suppose you had a 10fps video stream of a bouncing ball, clearly the animation would be extremely choppy as you would only see about 10 snapshots of the ball every second. Since most humans consider video playback to be smooth upwards of 30fps, your goal would be two create

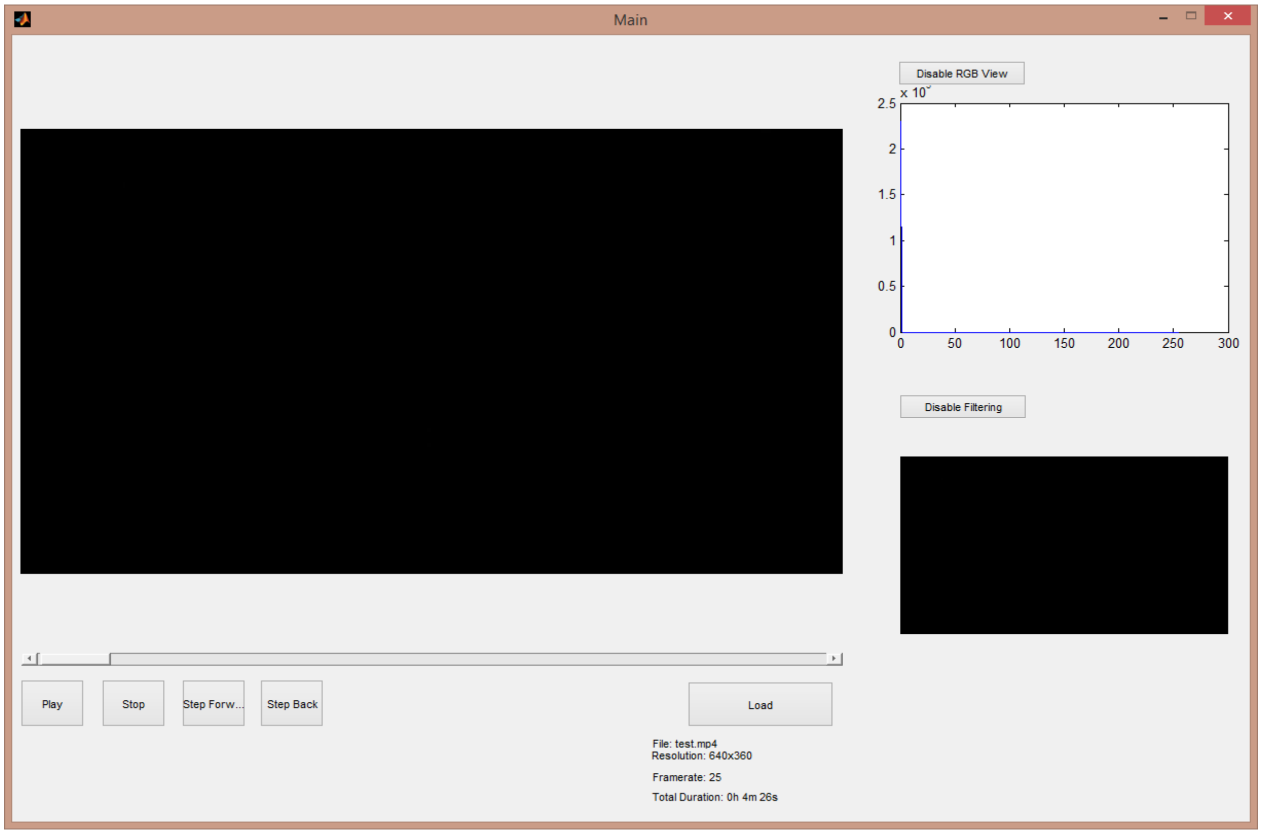
and inject two new "artificial" frames between a pair of "real" frames to make the video appear smoother (thus converting the framerate from 10fps to 30fps). If you choose this feature, be sure to pick a starting framerate and convert only to 30fps. A couple of viable options would be:
   a. 15fps to 30fps.
   b. 20fps to 30fps.
   c. 24fps to 30fps (Known as 3:2 pulldown in the film industry).

5. Sound Playback - Add the ability to have sound playback within your GUI (this would be the sound in the video file). If you choose to look into the possibility of adding sound playback, make sure that the sound playback behaves appropriately to the core GUI (Play/Pause button, Stop button, etc). You must also include a "Mute" button on your GUI.

6. Video Stabilization*** - Essentially creating a way to remove the "shaking" that is present when people record video without a tripod or without digital/optical image stabilization on the video camera. This would involve looking at multiple frames and computing several points of interest between those frames to determine how much change there is between one frame and another. Note that if you choose this feature, you must provide a video stream that clearly shows that your algorithm "anti-shakes" the original.

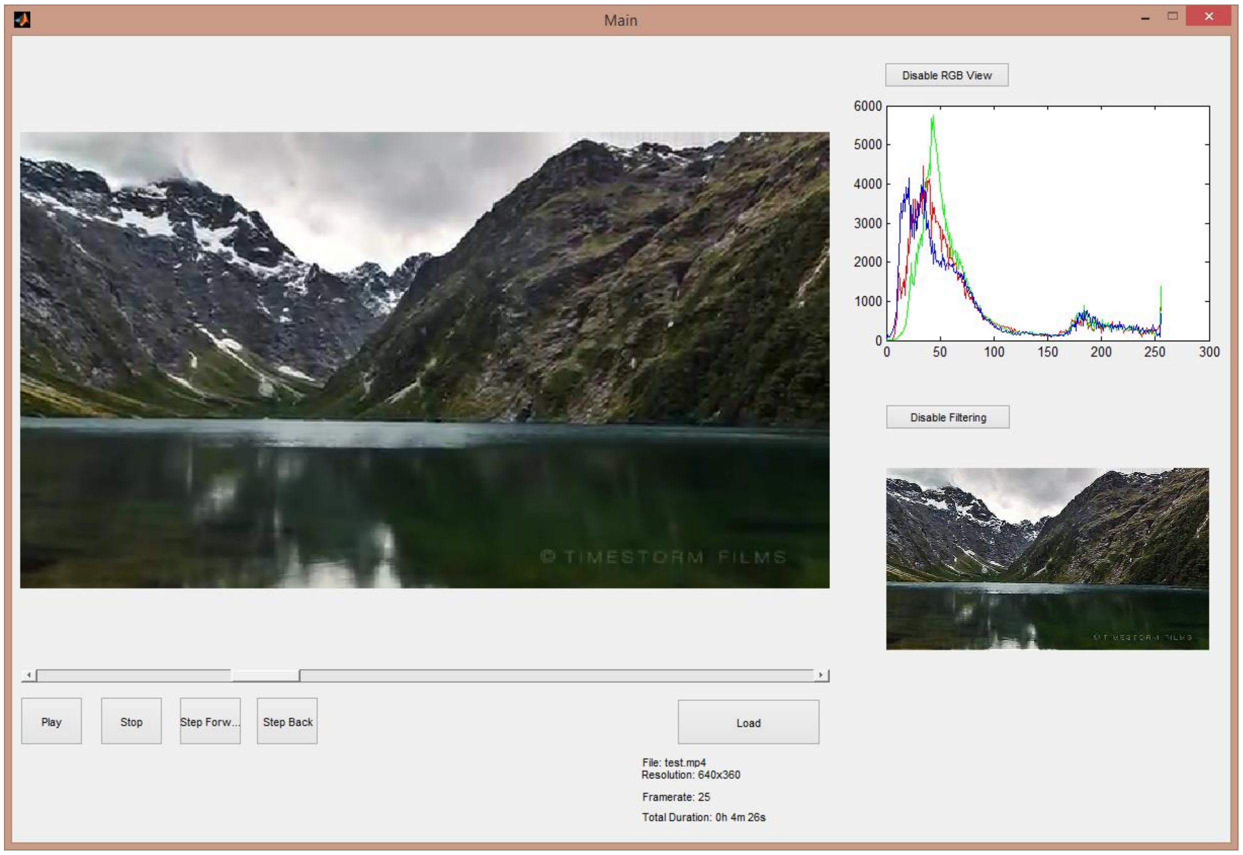
7. Video Compression - Take the original video file and compress the video stream to reduce the overall filesize. This is commonly done for mobile applications, so there is still a lot of work going into this technology. So, you will have to come up with a smart way of compressing the data (elements in each RGB array) so that there is less information that is contained in each frame. For this task you should aim to compress in real-time instead of writing the compressed frames to a new video file. You can do this by simply adding a push-button that enables and disables compression to show the user the artifacting introduced by compression.

Sample GUI



File: test.mp4
Resolution: 640x360
Framerate: 25
Total Duration: 0h 4m 26s

This is an example of the GUI you will be making. The core components of the base GUI are shown to the left of the “Disable RGB View” and “Disable Filtering” buttons. As you can see, the “Load” button allows the user to select a video file. Once a proper file has been selected, the File, Resolution, Framerate, and Total Duration fields will automatically update. The two figures on the right are the two special features chosen for this implementation. By default, the GUI starts with both features enables (and thus both are visible). Clicking on the “Disable RGB VIew” and “Disable Filtering” buttons disables the respective features, hides the associated figures and data, and finally changes the text of the button to read “Enable RGB View” and “Enable Filtering”.



File: test.mp4
Resolution: 640x360
Framerate: 25
Total Duration: 0h 4m 26s

The image above shows the program in action with a live view of the video being played back along with the two special features. The two special features work in real-time with the video stream and even respond properly to the movements in the Scroll Bar as mentioned earlier in the project description.

Project #2: Audio Sampler

In this project, your group will be required to program a basic audio sampler in GUI form. It will model some of today's commercial hardware based audio samplers from companies like Akai, Native Instruments, Roland, etc.

As described in Mathworks Documentation:

"The audio signal in a file represents a series of *samples* that capture the amplitude of the sound over time. The *sample rate* is the number of discrete samples taken per second and given in hertz. The precision of the samples, measured by the *bit depth* (number of bits per sample), depends on the available audio hardware. MATLAB® audio functions read and store single-channel (mono) audio data in an *m*-by-1 column vector, and stereo data in an *m*-by-2 matrix. In either case, *m* is the number of samples. For stereo data, the first column contains the left channel, and the second column contains the right channel."

Given this information, we can manipulate audio easily since it is contained in arrays. We can choose the sample size, sample rate and manually chop up the audio section by section according to user defined input. This forms the basis of our audio sampler.

Here are examples of commercial audio samplers (hardware and software):

SONiVOX Sampla Software Based Sampler
Native Instruments Maschine Hardware/Software Sampler
Akai MPD32 Hardware Based Sampler

Keep in mind these examples and others can be used as a reference for what features you would like to add to your sampler in order to be more creative. You will not be expected to produce a commercially ready product, however, you will be expected to implement some of the core functionality that these types of samplers include.

Functionality Requirements

1.) Basic Audio File Input

- The user will be able to load various samples, or .wav files into the program for playback. The user must be able to load the samples into the program and associate the samples with buttons that, when pressed, provide audio playback.
- A minimum of 9 buttons (3 x 3 grid) must be implemented. If your group is feeling more ambitious, there is no limit to the grid sizing you can use (4 x 4, 5 x 5, 6 x 6 etc.)
- The loaded samples will play back when the individual grid sample buttons are pushed, NOT a single play button for all the samples. The point is to be able to play each loaded sample by its button in real time, very much like an electronic drum or instrument rack/pads.

2.) Effects and Sample Modification

- The user should have the option of editing the individual samples that have been loaded through use of an interactive menu.
- The user should be able to add audio effects to each sample. These audio effects can be implemented by altering the sound data arrays. For example, in order to reverse the sample, one would simply flip the array backwards.

Required Effects Include:

a.) Sample reversal
b.) Delay
c.) Tone Control (Filtering)
d.) Speed up
e.) Voice removal

Here is a reference to help you get started on basic audio file manipulation:
http://homepages.udayton.edu/~hardierc/ece203/sound.htm

3.) Chopping

- The user should have the ability to choose a loaded sample, and chop/edit the length of the sample.
- They should be able to pick the new start and stop times of the editable sample. This can be accomplished by editing the size of the array.

4.) Basic Tone Generation (Synthesizer)

- User should be able to load into a sample box a pure tone generated through a mathematical function.
- Creativity points will be assigned to students that can generate more than just a basic sine wave, like a square wave, triangle wave, amplitude modulation, etc.
- For the basic sine tone, the user must be able to select the frequency or pitch of the tone. This can be done using an interactive keyboard or switchboard/drop down menu.

5.) Loop Recording

- The user should be able to record sample buttons pressed in real time into a basic loop. The user will need to specify some basic properties of the loop in order to record

<u>Loop Properties:</u>

1) **Beats per minute (BPM):** The number of beats per minute, or speed of quarter notes
   The user will need to be able to enter this as a number.

   Quarter notes align directly with each beat per minute.
   One eighth note is half a quarter note
   One sixteenth note is half an eighth note

2) **Time Signature:** Denotes the time scale of the notes, and number of those notes per bar

The notation is as follows:

(# of notes per bar) / (type of note: quarter, eighth, sixteenth)

For example: A 4/4 time signature means each bar 4 quarter notes
A 3/4 time signature means each bar is 3 quarter notes
A 4/8 time signature means each bar is 4 eighth notes

Implement the time signature as basic 4/4, that adjusts itself to the user specified BPM

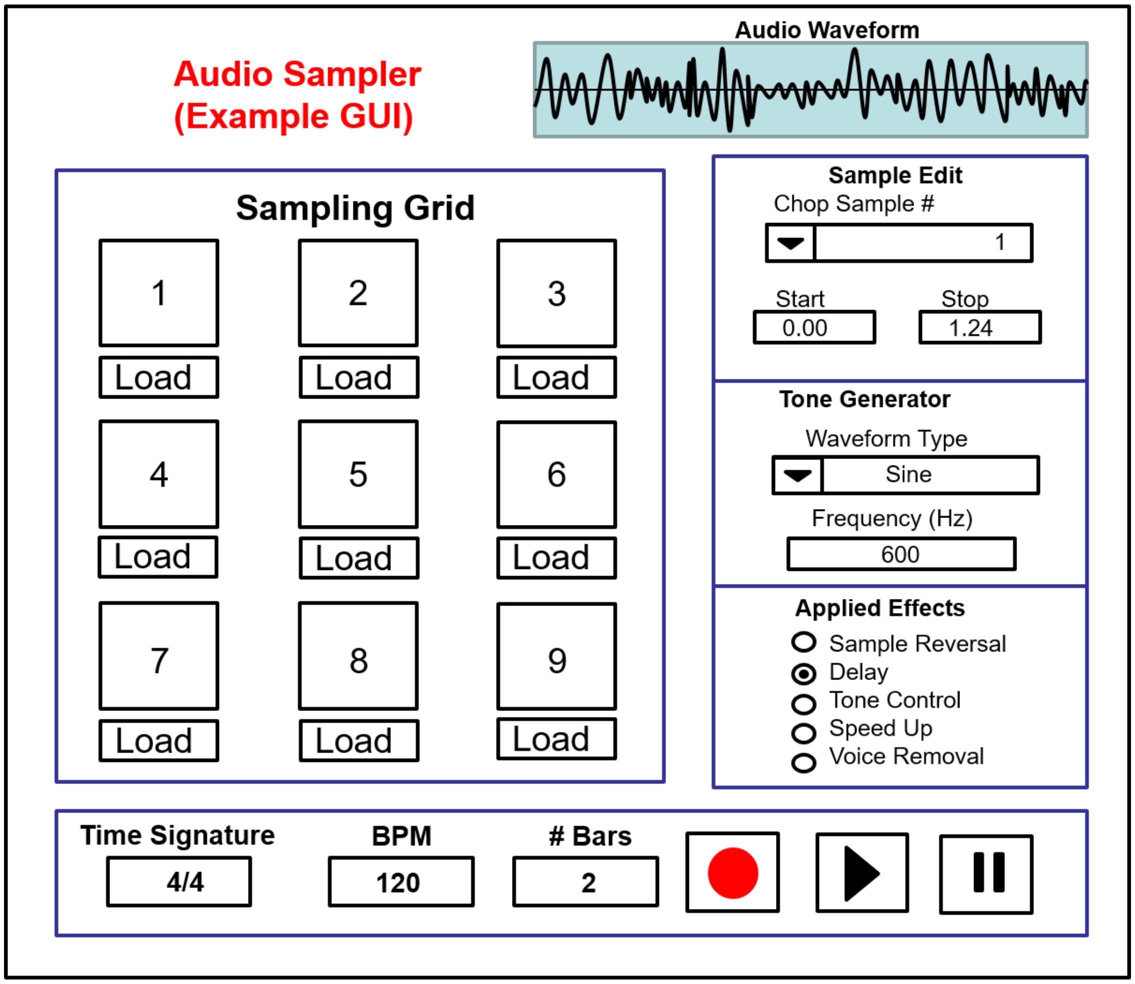
3) **# of Bars:** User should be able to define the number of bars in a loop
A bar is a subdivision of time that depends on the defined time signature.
If the user defines 4 bars, they should be able to record for 4 bars w/ a 4/4 time signature. Once the loop hits the end of the 4th bar, recording comes back to the 1st bar, and any new clicks of sample buttons overwrite the original recording.

Basic Operational Flow:

- When the user pushes the record button, a loop should start recording in real time any click of the sample buttons from the user.
- The time signature specifies how many beats per bar, and the number of bars specifies the amount of bars per loop cycle.
- At the end of a loop cycle, the sample clicks are recorded over the previous loop cycle's recording.
- Once the user is done recording, they hit stop or end, and the recorded loop can be played back.

Sample GUI Layout:

In addition to implementing the core functionality, your group must create a
GUI for the user to navigate around and select different sampling options. A very simple sample GUI is presented below.



The more extra features your group adds to the GUI, the more points you will get for creativity. You may want to use the provided examples of commercial samplers in order to figure out how to lay out your GUI.

Minimum Required Features of the GUI:

1. Sampling pads (3x3 or 4x4) with load sample buttons.
2. Sample chopping menu and start/stop selection while chopping.
3. Effects menu or buttons with the ability to select which loaded sample to apply the effect to.
4. Tone generation drop-down menu or keyboard

Your group may implement different features as menus, or individual buttons. The level of intuitiveness of the GUI will add to creativity points.

Project #3: Sustainable Urban Planning

For this project, you will be making a program that helps engineers and architects to develop and deploy renewable energy for a small area, in particular solar and wind.

Getting started:
Select one of the cities from the database that was used in Project 1. Then go onto Google maps.
Go to that city, find an "appropriately sized" (see details below on choosing the area) area, and take a snapshot of the aerial image of the area. Save the snapshot as an image.

Selecting an "appropriately sized" area:

❑ The area you select should be appropriately sized such that solar panels or wind turbines can be realistically placed in a box on the grid. The size of the area you choose also impacts the grid size: too large an area results in very small boxes on the grid while too small an area results in large boxes on the grid.

❑ Each box on the grid can represent one solar panel, or multiple. It is up to your team. But should choose the dimension of the box considering the panel/turbine dimensions.

❑ Figure below is a good example. It is part of downtown Davis and the street block labeled is approx. 400 feet. The length of each box is around 20 feet (there are about 20 boxes). Each 20ft x 20ft box in this case contains approx. 16 to 25 solar panels (ignoring the dimensions shown in table below).

❑ Experiment with different area/box sizes! However, large area results in large grid with too many small boxes, which can slow down your program and potentially exhaust the available memory.

Tasks: You should include a GUI in your design. Your GUI should meet the following criteria:

❖ City selection and placing a solar panel or wind turbine:

1. Allow a user to select from three different area/cities to work with. After the user has selected a city, you should display the image of that city and draw a grid of appropriate size on top of the image.

2. You should also display a list of solar panels and wind turbines. All types of solar panels, wind turbines and their properties are listed in the two tables below.

3. The user should be able to select the type of solar panel or wind turbine to place on the grid on top of the city image. Each item/type should have a color code. (Assign the color/shading at your own choice).

   - The user first select an item (a type of solar panel or wind turbine), then clicks anywhere inside the box on the grid to place the item. Your program should automatically fill in the box with the respective color. (Example is shown below).

4. Continue to perform step 3 until the user stops or reaches a limit.

- A user can at most click 10 boxes at a time. There should be a prompt to the user when he/she reaches the limit. Also, there should be a clear way for the user to tell the program that he/she has finished.

❖ Save the work and create a bill of materials:

5. When the user finishes, your program should allow the user to:

   a. Save the picture with the user modifications (a .png picture of the result).

   b. Save as text output (an N-by-M text file containing the content of each box in the grid, N and M are dimensions of your grid). You decide the text file format.

   c. Create a BOM, "Bill of Materials" containing the quantity and types of panels, turbines, and total cost and save it as a text file. When the user requests for a BOM, it should open the text file.

❖ Calculate the estimate:

6. Solar Energy Estimate:
   - Using the total area occupied by the solar panels from the table below and the grid size find the number of solar panels placed by the user (be careful about the units).
   - Assume solar panels in table below are capable of producing the amount supplied by the sun (calculate the average solar insolation of the city from Project 1 database). Solar data is in **kW per square meter**.
   - Together with database from Project 1 and the table, obtain an estimate for the total amount of potential solar power available. Display the estimate at the user request in the GUI or update the estimate in GUI every time the user clicks a box.

7. Wind Energy Estimate:
   - Using the total area occupied by the wind turbines from the table below and the grid size find the number of wind turbines placed by the user (be careful about the units).
   - Assume for every 1mph of wind, 600 kW of power is produced (calculate the average wind speed of the city from Project 1 database).
   - Together with database from Project 1 and the table, obtain an estimate for the total amount of potential wind power available. Display the estimate at the user request in the GUI or update the estimate in GUI every time the user clicks a box.

❖ Load saved data and continue working:

8. Add a feature into your program that allows the text file saved in 5b to be loaded into your program (with the proper image, of course). In a way, this feature allows users to save their work and continue working on it later.

9. The user should be able to switch between different city images and continue their work.

Creativity points: The GUI should be user-friendly and the images should be loaded and saved in an efficient manner. You can prompt the user with useful messages to guide them through the process. You can include buttons, drop-down menus, text output, display output, text box, secondary GUIs, etc. How you design your GUI is up to you. Make sure it looks clean and user-friendly and performs the functionalities efficiently. You can add extra features like reset, remove, etc.

The tables are given below:

Table of Solar Panels. (Data from www.amazon.com)

Brand	Model	Type	Power (watts)	Dimension (inch x inch x inch)	Weight (lbs)	Price ($)
Sunforce	50048	Amorphous Silicon	15	42.5 x 1.5 x 16	11	279.95
Sunforce	39810	Polycrystalline	80	21 x 48 x 2	22	499.95
Instapark	SPCC-5W	Mono-crystalline	5	11 x 8 x1	2.8	34.95
Instapark	SP-100W	Mono-crystalline	100	45 x 1.5 x 26	21	319.99
Instapark	SPCC-30W	Mono-crystalline	30	21.5 x 1.1 x 17.2	7.2	114.70
Instapark	SP-10W	Mono-crystalline	10	14 x 11 x 1	2.8	39.95
Ramsond	100SP	Mono-crystalline	100	47 x 1.5 x 21.8	12	245.99
Epcom	WK50-12	Polycrystalline	50	32 x 22 x 1.4	12	99.99
Sun Power	E18	Mono-crystalline	400	41.18 x 81.36 x 2.13	56	249.5
Sun Power	T5	Mono-crystalline	320	43.06 x 75.13 x 8.37	47	199.99

Table of Wind Generators/ Turbines. (Data from www.amazon.com)

Brand	Model	Type	Power (watts)	Diameter (feet)	Price ($)
Windmax	HY 1000-5	Wind Generator	1000	15	999.99
Windmax	HY 400	Wind Generator	500	13	686.40
GudCraft	WG400	Wind Generator	400	13	399.00
GudCraft	WG700	Wind Generator	700	13	449.00
All Power America	APWT400A		400	10	476.93
Sunforce	45444	Wind Turbine	600	10	749.99
Sunforce	44444	Wind Generator	400	10	474.34
WindyNation	WCK-750	Wind Turbine	750	15	999.98

Figure of Davis zoomed in, overlay with grid:
* Filled boxes indicate solar panels. Colors indicate solar/panel/wind generator type.