Control for Resonant Microbeam Vibrotactile Haptic Displays

By

Kendra Lee-Ann Kim

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Angela Sodemann, Chair
John Robertson
Ajay Bansal

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

The world's population is currently 9% visually impaired. Medical sciences do not have a biological fix that can cure this visual impairment. Visually impaired people are currently being assisted with biological fixes or assistive devices. The current assistive devices are limited in size as well as resolution. This thesis presents the development and experimental validation of a control system for a new vibrotactile haptic display that is currently in development. In order to allow the vibrotactile haptic display to be used to represent motion, the control system must be able to change the image displayed at a rate of at least 30 frames/second. In order to achieve this, this thesis introduces and investigates the use of three improvements: threading, change filtering, and wave libraries. Through these methods, it is determined that an average of 40 frames/second can be achieved.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

1. Introduction

There are 19% of people in the world that must live with disabilities. These impairments have a substantial effect on each person's life. Although this is unfortunate, there are assistive technologies both in use and being developed world wide to help overcome these disabilities. One of the more common disabilities is visual impairment. Visual impairment includes partially sighted, legally blind, and totally blind individuals. Sensory disabilities which include visual impairments are currently being assisted with biological fixes or assistive devices.

Biological fixes are the utilization of the human abilities to aid the visually impaired through other bodily functions. Biological fixes can provide aid with a range from allowing the patient to function at a high level to increase visual capability. Echolocation is an example of a biological fix that allows the patient to function at a high level. Echolocation is the ability to locate objects with the reflected sound on those objects shown in Figure 1. The primary visual cortex drives a remapping phenomenon, neuroplasticity to echolocate objects [1].

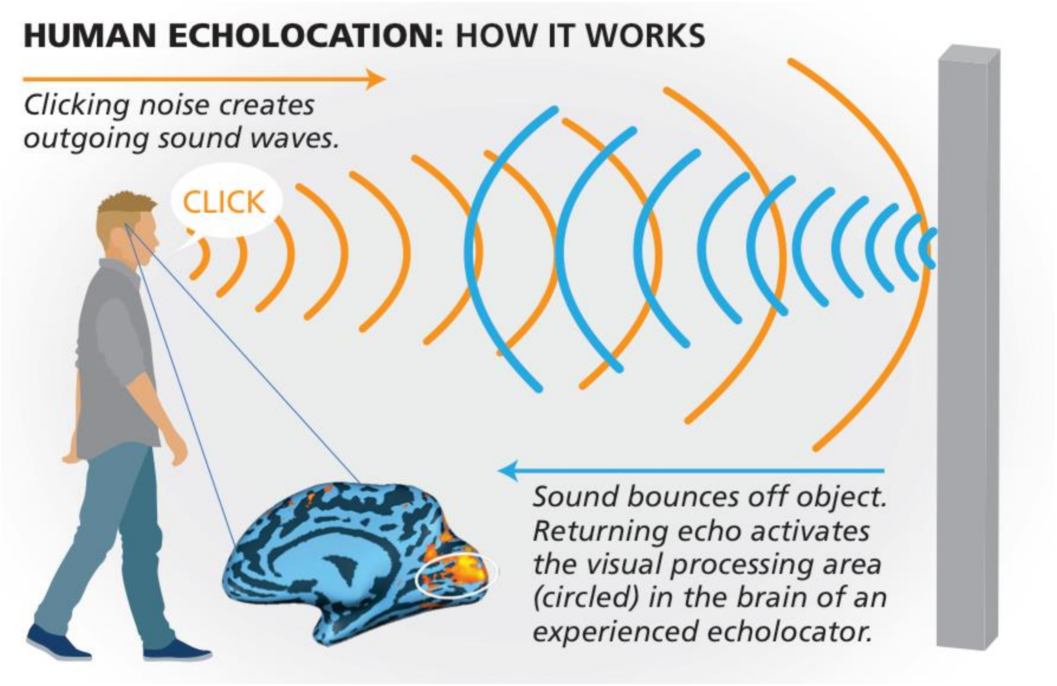

HUMAN ECHOLOCATION: HOW IT WORKS
Clicking noise creates outgoing sound waves.
CLICK
Sound bounces off object. Returning echo activates the visual processing area (circled) in the brain of an experienced echolocator.

Figure 1: Representation of Human Echolocation

Another biological fix is the retina implant. The restoration of sight to people that are blind by retinal degeneration is done with retinal prostheses. Retinal implants utilize an external camera to convert an image to an electrical signal. The pattern of this electrical signal is used for a improved visual ability of the patient shown in Figure 2. The retina implant has 1500 microphotodiodes which is roughly 38 x 38 resolution [2]–[4]. Although this is a significant increase in vision, the resolution is restricted by photodiodes sizes and input/out ratio of the signal.

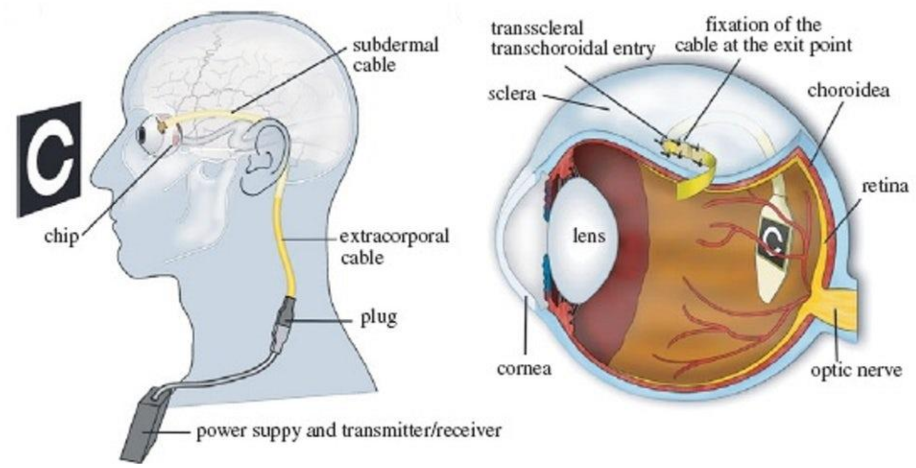

C
subdermal cable
chip
extracorporal cable
plug
power suppy and transmitter/receiver
transscleral transchoroidal entry
fixation of the cable at the exit point
sclera
choroidea
retina
lens
C
cornea
optic nerve

Figure 2: Representation of Retina Implant

In addition to the resolution limitation, the implants are invasive, requiring surgery and the destruction of the patient’s existing visual mechanism. Also, the implants are expensive with a cost of $150,000. Unfortunately, the only patients that have this solution available to them are individuals who lost their photoreceptors due to retinal diseases and can afford the surgery [2]. Because of these limitations, the retinal implants are not the most ideal solution for majority of the visually disabled.

In addition to biological fixes, there are also assistive devices that improve the life of the visually disabled person. One of the simpler assistances available is the guide dog shown in Figure 3. These assistive animals are trained to go around obstacles and safely lead their blind owner. Another simple assistive device available is the cane. There are two different kinds of commonly used canes. The first is a support cane that not only provides support but helps identify the user as an individual with low vision. The second is a probing cane; which assists in locating obstacles.

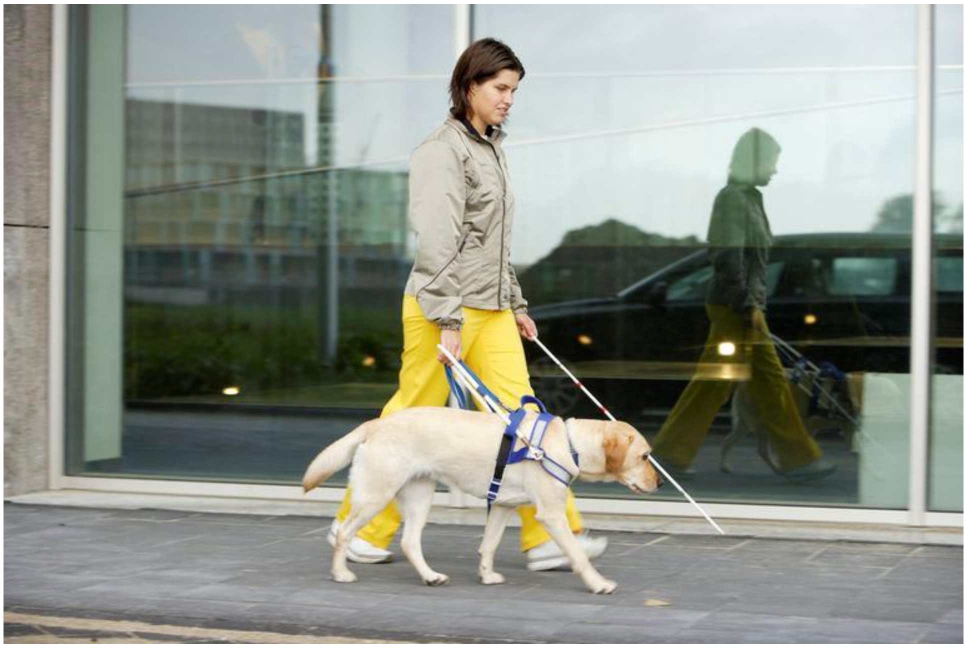


Figure 3: Photo of Blind Seeing Eye Dog and Cane in Use

Although these devices are beneficial to ensure the safety of the patient, they do not enable the patient to understand what obstacles they are encountering. An alternate type of assistive device which does allow understanding of the encountered obstacles is the audio device. Audio devices are a type of technical assistance that provides either a description or an echolocation support for the visually disabled. Audio device assistive technologies are helpful because visual disability is independent of the patient's sensory development [5]–[7]. Due to the sensory development being independent of the disability, some visually impaired were able to detect and classify different objects in complex scenes with echolocation [1].

There are also audio devices that produce words to describe a scene that have proven to be very helpful to the patients. However, these audio devices are still limiting. One of the main concerns for those that are visually disabled is their safety. There is an assistive technology that maps the environment of the patient and alarms them of traffic and pedestrian signals in real time [8]. Audio solutions have not been considered successful because of the limitations in the information that can be sent to the user. The echolocation solution previously discussed in the introduction allows the patients to determine that there is an object ahead, however, it does not give a depiction of what that object ahead is. With the technologic assistive devices, the patients can hear an audio voice to describe the environment however, this is limiting the available information to the patient. For example, the assistive device may state that you are walking in a park, but may not include the detail of the fall season and describe the color changes in the leaves. Due to these limitations in audio solutions, they are to be considered not as effective in the lifestyle improvement of the user as other assistive devices, such as the haptic devices.

In addition to audio devices, there are haptic devices that also assist the visually disabled. There are two main types of haptic display devices. The first type of display are the electro tactile displays that are represented in the braille and tongue placed solutions. The second main type of display is the vibration display. A revolutionary assistive device to help the visually impaired is the electro tactile tongue placed display. The electro-tactile tongue placed display consists of an array of small electrodes placed on a flat surface. The electrode array is connected to a cable which connects it to a camera in the user’s glasses. As illustrated in Figure 4, a grayscale image is captured from the camera, then the charge on each electrode is varied based on the grayscale level read from a corresponding pixel in the image.

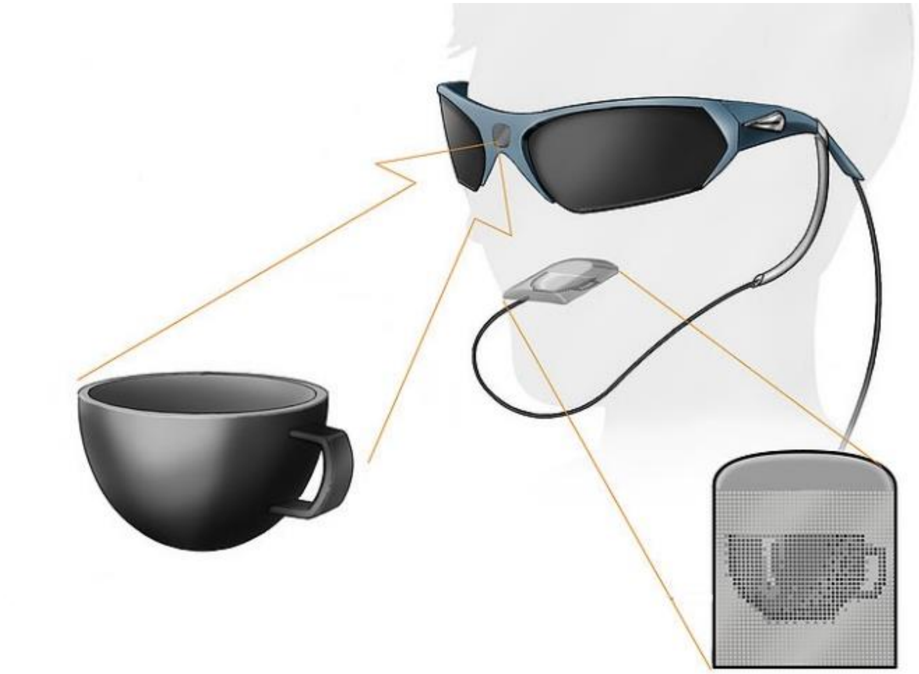


Figure 4: Electro tactile tongue display

The electrode array is placed on the user's tongue. Thus, the user perceives an image through a 'tingling' sensation that varies with the captured image. The electro-tactile tongue display has successfully resulted in an improvement in the patient's quality of life with the ability to sense objects within the view of the user. In one case, a patient was able to participate in a tic tac toe game for the first time with his daughter [9]. Unfortunately, the electrode array gives a maximum resolution of 32 x 32. This low resolution is due to the limited surface available on the tongue as well as the number of electrodes required in the output of the device.

In addition to the electro tactile tongue display, the electro tactile braille display has also been developed. The electro tactile braille display converts the letters of text into braille that the user is able to feel on the finger sleeve, as shown in Figure 5 [10]–[13].

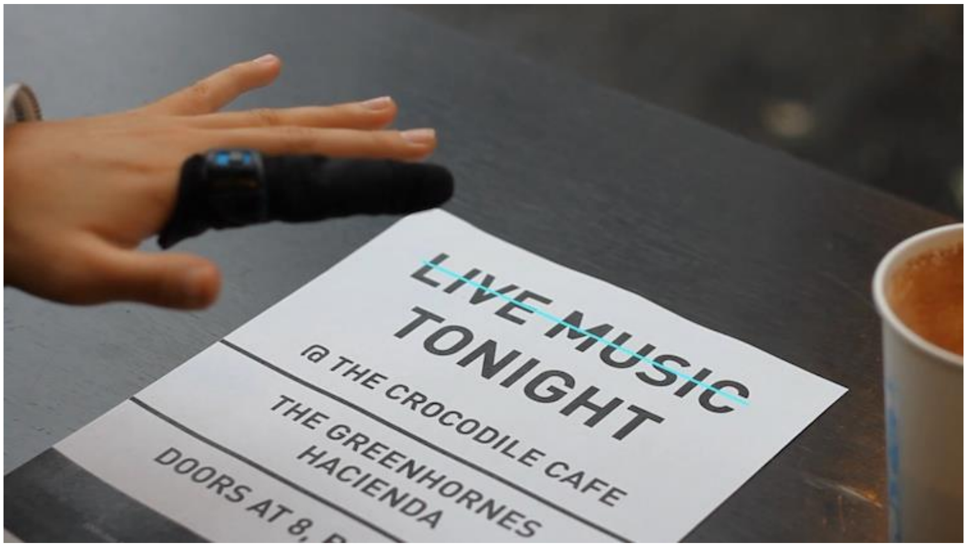


Figure 5: Electro tactile braille display

Braille is a written language that has raised dots that represent characters, therefore the limitations in braille are not resolution. The limitations in braille include the boundaries of the description to only be available for text conversion and ability to present information within a timely manner. Aside from the Electro tactile braille display, all other assistive representation technologies are currently limited in resolution [3], [9].

Besides the electro-tactile display, another approach to a haptic display is the vibro-tactile display. A vibro-tactile display utilizes vibrating elements rather than electrode elements for the ‘tactile pixels’. The vibro-tactile display has the advantage of not requiring placement on the tongue, which allows a much larger potential surface area for placement of the device. The larger surface area would also potentially allow for a high resolution.

All of the haptic display solutions found in the literature have two primary complications: The low resolution and the one pin per element problem. Resolution is the number of pixel contained in an image. A visual digital image commonly has a resolution such as 640x480 or 720x1080 pixels. However, the current highest- resolution tactile display has a resolution of less than 32x32. Since a visual image becomes more difficult to interpret the lower the resolution is, it is expected that an image with a resolution as low as 32x32 would not be able to be interpreted by a viewer. Therefore, the low resolution of the haptic displays is a significant impediment to their success.

The one pin per element problem stems from the need to individually control each tactile pixel independently of the other. In order to accomplish this individual control, a single pin of a microcontroller is needed for each tactile element. Thus, in order to achieve a resolution of even 32x32, 1024 individual pins would be required, as well as 1024 individual pulse width modulation signals and their corresponding clocks. Both of these complications are addressed with the proposed solution of the resonant microbeam vibrotactile haptic display.

Both audio and haptic display solutions depend upon the brain's ability of 'sensory substitution'. Sensory substitution is a subcategory of neuroplasticity that allows the brain of an individual to interpret information received through one sense as if it were presented through another sense. Sensory substitution is necessary for patients to make new connections in the brain to comprehend surroundings. The ability for the brain to make new connections successfully by repeating stimuli is the reason that the haptic display solutions have shown good initial success with the visually disabled [5].

1.1 Resonant Microbeam Vibrotactile Haptic Display Concept

An alternative solution to the vibrotactile haptic display is currently under development at Arizona State University. This alternative solution has the potential to greatly increase the resolution of a haptic display. This research project aims to investigate the necessary control system for this type of display. Thus, a brief background of the proposed vibrotactile haptic display is given here.

The development of the resonant microbeam vibrotactile haptic display concept is based on the patient's ability to utilize sensory substitution [5], [14], [15]. The resonant microbeam vibrotactile array is a mechatronic system that is based on two subsystems: a beam array and a beam array controller. The beam array consists of stainless steel beams, each fixed to a base at one end and free at the other end as shown in Figure 6. Each beam in the array is designed to have a unique length and/or cross-sectional area, so that each bean has a unique natural frequency. The base of the beam array is attached to a surface transducer that is actuated to vibrate. When the frequency of vibration of the surface transducer matches the natural frequency of one of the beams, that beam 'resonates'- vibrates with a large amplitude. When the frequency of vibration of the surface transducer does not match the natural frequency of a beam, the beam does not vibrate or vibrates with a low amplitude. Because the natural frequency of each beam is unique, the birational amplitude of each beam can be controlled independently, each beam can represent a different pixel of an image. The construction of the beam array is through an electrical discharge machine. This manufacturing process uses current discharges between two electrodes separated by a dielectric liquid to remove material from the electrodes.

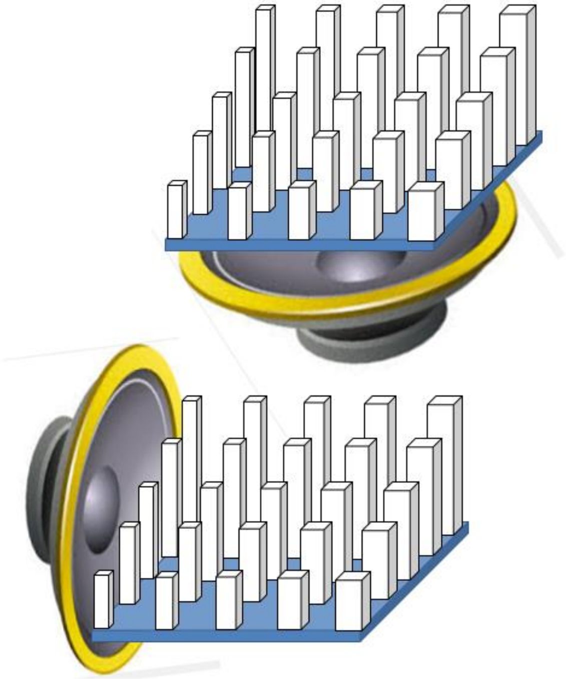
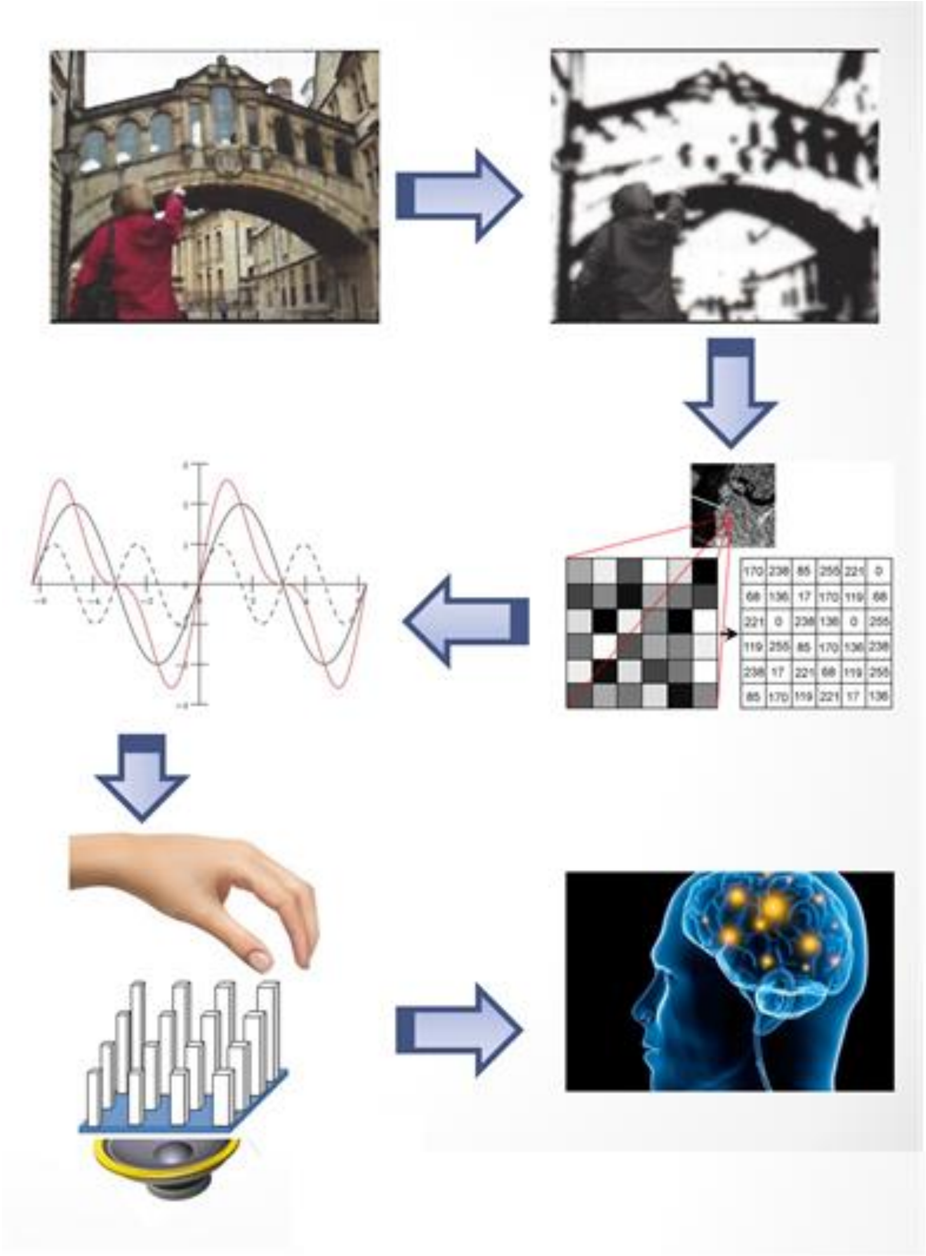


Figure 6: Array of Cantilever Beams

The second subsystem is the control the vibration beam array. The control system converts streamed images to a single soundwave that will resonate the corresponding beams of the image. The sound wave that is produced by the control device, phone or tablet, is the sum of every individual element sinusoidal wave. The Fourier transform is the decomposition of the function of time. This transformation can be used against the sinusoidal sound wave to determine individual frequencies. This decomposition is broken up into each individual beam that represents each pixel in the image. This is expressed in the fourth step of the flowchart of the vibrotactile display shown in Figure 7.



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Figure 7: Flow chart of vibrotactile display

This study addresses the design of the second subsystem. In order to be a viable approach to replace vision, the overall system needs to have a frame rate that is fast enough that the user can perceive each picture as motion. However, due to the number of calculations that must be computed at the time of streaming, achieving a reasonable frame rate is difficult. To perceive each picture as motion, the frame rate will ideally be 30 frames per second (FPS). This value is slightly above of the current cinematic frame rate of 24 FPS. In addition to having a reasonable frame rate, the solution needs to be cost effective for mass production. The frame rate is dependent on the computation development for effective and efficient controls. The cost-effective development is dependent on how the device is fabricated at scale.

In addition to fast implementation, the solution must also be easily accessible by a user. The user will have easy accessibility by utilizing Kivy, which allows python code to run on Linux, Windows, OS X, Android, and iOS. With a control system that is supported among all types of platforms the user will have the ability to use their current technology; such as a phone, or tablet. This feature also addresses the device's cost-effective prototyping.

The proposed solution addresses the two areas of complications in current haptic displays. The first problem is the limitation in resolution. The number of pixels required to represent an image is determined by the number of beams in the beam array. In the example of the electro-tactile display, the display is limited to 32 x 32 because the display must be placed on the tongue; therefore, the area available for the electrodes is limited. With the resonant microbeam vibrotactile haptic display, the resolution is not limited by available surface area. The patient will be able to utilize any surface area that can feel motion. This means they have the option to have this device anywhere on their skin. In addition, each vibrating element in the proposed array is expected to be 0.1mm. This allows a large quantity of elements to fill a limited space. If the micro-cantilever beams are made 0.1mm in diameter with 0.1mm space between beams, 640x480 beams could fit in a space approximately the size of the palm of a hand.

One cause for limited resolution is the one pin per element problem. In a haptic display, each pixel needs to be controlled individually. One downfall of the current vibratory systems is the use of individual motors as each tactile element. Each tactile element is independent and requires its own signal to drive the motors such as the ones shown in Figure 8. With the resonant microbeam vibrotactile haptic display, the one pin per element is not limiting. To excite each of these pins individually, the excitation is produced by a dynamic soundwave that excites each of the beams individually at their natural frequencies.

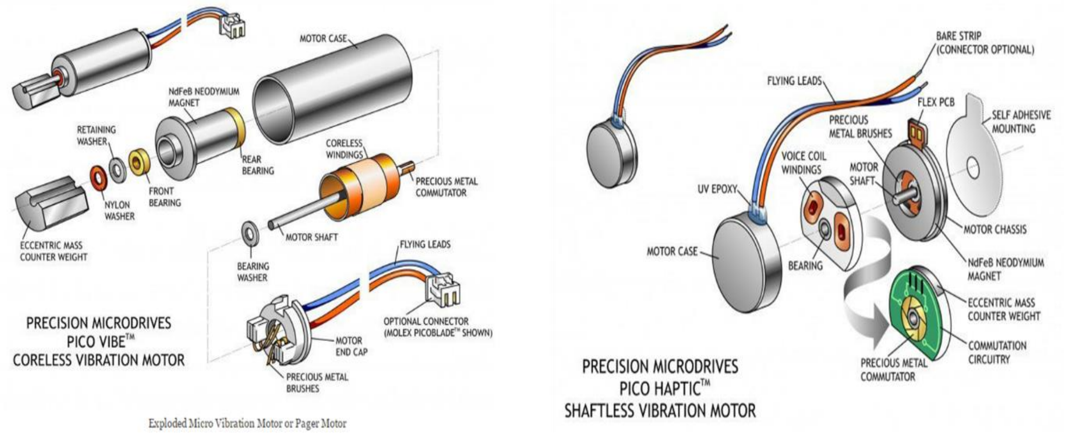


Figure 8: Motors Currently being used for Vibration Application

2. Control Problem Analysis

The control problem analysis chapter includes the system overview and benchmarking sections. The system overview is a system relates how the resonant haptic display conceptually works with benchmarking data. The explanation includes the theoretical process including how the video feed input is manipulated into a soundwave output. The benchmarking section explains the importance of the benchmark and how the benchmarking experiment was implemented as well as the benchmarking results.

2.1 System Overview

The control system of the vibrotactile display consists of software to stream in the video feed in which is then translated to soundwaves that excite the corresponding beams. This process proceeds as follows: A single image is extracted from a video stream. This single image is then reduced in resolution to match the number of beams in the beam array. Then, the image is converted from color to grayscale. The image is required to be converted to grayscale because the device is not able to represent a spectrum of colors. This limitation is due to the design of the device which allows for either full excitation representing a white color or no excitation representing a black color, as well as all excitations of grayscale in-between. Each pixel has a grayscale value from 0 – 255 that represents the pixel's brightness. The 255 pixel value limit is based on the 8 bit-depth that is a standard for image processing. The pixel's grayscale value is used as the amplitude of the individual sinusoidal wave. All individual sinusoidal waves are summed, and the single resultant wave is produced through the speaker attached to the base of the beam array.

For example, suppose the image is of a dark room where all pixels in the image are black. In this case, the grayscale value of all pixels is 0 and there is no excitation to the beams. However, if the image is completely light and all pixels in the image are white, the grayscale value of all pixels is 255 and all of the beams will be excited with maximum amplitude. The maximum amplitude of sound for any individual wave is based on the required excitation for a human to feel the excitation of the beam. This is dependent on the size of the beams, the spacing of the beams, as well as the user's sensitivity.

The video streaming will ideally be on a device that is common to the user such as a phone or tablet. Due to the type of device to be used, the software selection to develop the control system is limited to object oriented languages that can be used on multiple operating systems. Python is one of the most popular languages used in data science. In addition to being reliable and efficient with libraries that offer cross platform support, python is accessible. This software can be run on mobile devices such as a phone or tablet. Because the execution speed is variable to the device, the experimentation was done on standard current equipment. Python also has the ability to work on the web based execution method known as, Jupiter.

The most computationally-intensive portion of the proposed system is the array management of each image. The grayscale values will be stored into the random-access memory of user's computational device, such as their tablet or phone. These stored values in memory are an array type that is referenced to determine the amplitude for the corresponding beam. In addition to image processing libraries and array management, the device will also need to produce the sinusoidal wave to excite the beams. The total harmonic distortion of the produced sound wave is addressed with the speaker selection.

2.2 Benchmarking

To determine if the python code needs to be optimized or if the standard libraries are reasonable enough, benchmarking was done on basic code that implements all of the primary steps of the control system: capture an image, convert to grayscale, decrease resolution, calculate waves, sum the waves, and produce the sound wave. The hypothetical beam array to be excited by the control code is a 64 x 64 beam array. The dimension of 64 x 64 was chosen because this resolution is double the current resolution in haptic displays. The benchmarking evaluates the effects of resolution by starting the time study at an 8 x 8 resolution and increases the resolution until 64 x 64 is reached. Each operation uses the resolution information to determine the number of loops required for the image dimensions. For example, if the resolution is 8 x 8 there are 64 pixels in the image, and the quantity of pixel loops is 64 in this case. However, if the resolution is 64 x 64, the there are 4096 pixels in the image and the loop needs to run 4096 times. The results of the time study shows that the pixel loop is directly related to the time increase. This is due to the number of times the loop is required to run. These results determine what part of the code will be optimized. The correlation between resolution and time for each operation is considered and analyzed to verify the loop optimization.

Table 1 shows each operation performed within the program and the time it took to perform each operation. Table 1 reports the amount of time required for each part of the benchmarking code under different resolutions. For example, row 1 for Table 1 shows that the ‘Opening PyAudio’ function require 0.534 seconds to complete regardless of the resolution. From Table 1, we can see that only a few functions are affected by the change in resolution. For example, the ‘Create Waves; function only requires 0.080 seconds to complete with a resolution of 8x8, but requires 4.663 seconds with a 64x64 resolution. Roughly 11% of the program is not affected by the increase of resolution. This is because the resolution only affects the number of waves that are being created. One of the key benefits of the micro-cantilever beam resonant frequency vibratory haptic display is the lack of limitations regarding pixel resolution. The proposed resonant frequency approach will allow all elements to be excited with a single sound wave. Thus, it is important to consider the effects of increasing resolution on computation time. The time study shows that only the pixel loop is dependent upon resolution. This is because the number of times the loop is required to run is dependent upon the number of pixels in the image.

Table 1: Benchmarking of Basic Python Control

	8x8 image	16x16 image	24x24 image	32x32 image	40x40 image	48x48 image	56x56 image	64x64 image
Opening Pyaudio	0.534 sec.	0.534 sec.	0.534 sec.	0.534 sec.	0.534 sec.	0.534 sec.	0.534 sec.	0.534 sec.
Access Webcam	0.172 sec.	0.172 sec.	0.172 sec.	0.172 sec.	0.172 sec.	0.172 sec.	0.172 sec.	0.172 sec.
Take an Image	0.026 sec.	0.002 sec.	0.002 sec.	0.001 sec.	0.001 sec.	0.001 sec.	0.001 sec.	0.001 sec.
Grayscale	0.001 sec.	0.002 sec.	0.001 sec.	0.001 sec.	0.001 sec.	0.001 sec.	0.000 sec.	0.001 sec.
Adjust Resolution	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.
Create Waves	0.080 sec.	0.340 sec.	0.701 sec.	1.212 sec.	1.886 sec.	2.687 sec.	3.611 sec.	4.665 sec.
Sum Waves	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.	0.000 sec.
Pixel Loop	0.080 sec.	0.340 sec.	0.701 sec.	1.212 sec.	1.886 sec.	2.687 sec.	3.611 sec.	4.665 sec.
Write Wave	0.160 sec.	0.188 sec.	0.189 sec.	0.189 sec.	0.158 sec.	0.188 sec.	0.188 sec.	0.190 sec.

3. Proposed Solution

In this chapter, the solution options overview and solution details are discussed. The solution options overview discusses the conceptual details of the three experiments conducted. This section also evaluates the hypothesis of the experiments. The solution details section reviews the flowchart used and libraries required to conduct the experiments.

3.1 Solution Options Overview

There are three solutions that we propose to improve the computational time of the control system: (1) Threading, (2) Change Filtering, and (3) Wave Library. Each of these solutions are related to improving the logic sequence that the control system uses to compute each resonant frequency sinusoidal excitation.

3.1.1 Threading

The first solution is to break the image into multiple parts that will be computed at the same time, also known as threading. Threading is a method that is used for parallel programing. This allows the execution of image processing to occur multiple times within the same time frame. By allowing the imaging to be broken up into multiple processes, the time to compute each of these processes will be reduced. The evaluation of this improvement considered multiple threads to determine if increasing the threads will increase the frame rate to 30 frames per second.

3.1.2 Change Filtering

The second and third solutions both utilize a decreased grayscale range. By default, the captured image has an 8-bit grayscale depth, giving 256 different grayscale levels. In the vibratory haptic display, each grayscale level corresponds to the amplitude of the sound wave, which determines the amplitude of vibration of the corresponding beams. By utilizing a decreased grayscale range, the number of different possible sound amplitudes and, thus, vibration amplitudes of the beams will be reduced. This means that the user will not be presented with 255 different levels of vibration amplitude but instead will be presented with fewer, such as 12, levels of amplitude. The 12 levels of amplitude is designed by software limitation for the Wave Library design and is utilized throughout all grayscale experimentation. By decreasing the grayscale levels to 12, the number of possible waves is reduced by more than 95%. Although this does not decrease the calculation time directly, this improvement is required for both the Change Filtering and the Wave Library improvements.

The Change Filtering improvement is to determine if each pixel has changed within the new grayscale range before calculating the wave. If the pixel has significant change, then the new wave will be calculated; however, if the pixel is determined to not have a significant change then the same wave can be used as in the previous calculation. This can improve the time by ~100% if the entire image does not change. The determination of the what pixel tolerance is acceptable to consider the pixel to be unchanged is determined in the experimentation. This solution will be most beneficial in scenarios where the environment does not have significant change such as a conversation or standing still. This solution does need to utilize additional memory to determine if there is change from the original image. In the case that the first image is found to not have significant change to the second image, the first image is stored. The third image is compared against the first image to determine if there is significant change. If the third image is significantly different then the third image is computed and stored in place of the first image to be the new comparison to future images.

3.1.3 Wave Library

The third solution is to change the addition of the Wave Library to the control system. The Wave Library is the creation of the waves in the initialization process. The waves created are all of the pixel options both in position and grayscale. For example, in position 1x1 the Wave Library includes twelve waves for each grayscale. Due to the number of waves per position that is calculated, this solution adds time to the initialization process by calculating all the predetermined waves. The benefit to increasing the initialization time is that the initialization only occurs one time and becomes less significant the longer the program is running. In addition to adding to the initialization time, all the waves are stored in memory which decreases the processing time slightly. After the waves are created they are stored in memory, they are called for each beam's pixel values of the images.

These solutions are implemented individually to determine the increase of time per solution. As each of the solutions are executed the baseline will also be executed with the same input variables for consistency. The individual experiments determine their benefits and can be combined for the most optimized the control system based on these findings.

3.2 Solution Details

The control in Python is based on an initialization with two loop back systems shown in Figure 9. After the startup of the program, there are two main initializations that are required in addition to the libraries. The first main initialization is a PyAudio, a Python binding for Port Audio. This library is used to produce the frequency to excite the corresponding beams. The second main initialization is an imaging module that uses multiple library to stream the video for the patient's view.

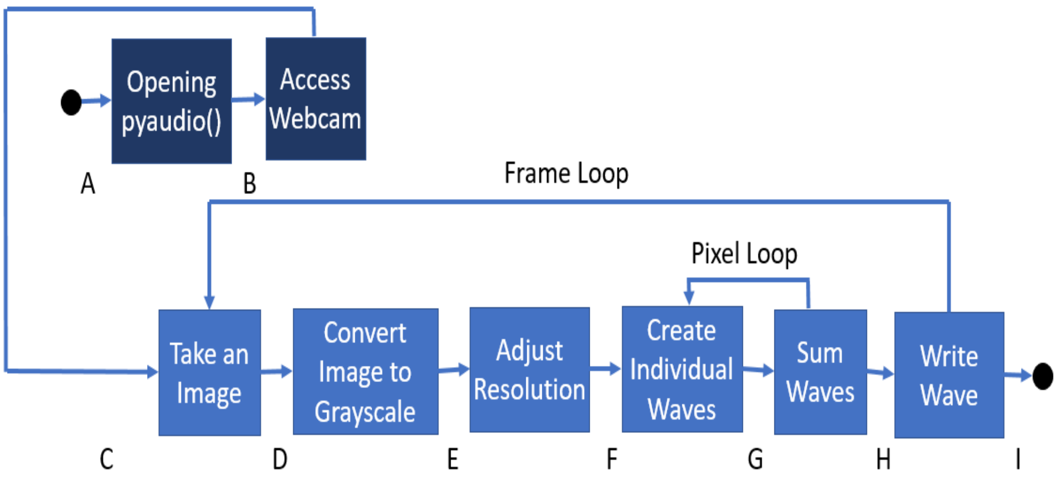


Figure 9: Flow chart of Python Control System

This initialization utilizes several dynamic link libraries. Another purpose for using python as the control system language are the dynamic link libraries readily available. The dynamic link libraries that are used utilize a process that does not require compilation into the main program and therefore does not use the random-access memory to load programs. Table 2 shows the dynamic link libraries that were used within the control system.

Table 2: Dynamic Link Libraries

Library	Syntax
Image	from PIL import Image
Resizeimage	import resizeimage
Scipy	Scientific computing library
Mathplotlib.pyplot	2D plotting
CV2	Open CV used for array operations and preserved data types
Pyaudio	Audio input/output library
Numpy	Highly stable and fast array processing library
Time	Representing time under the control of CPU
Math	Mathematical
GC	Garbage Collector

The first library used is the image resizeimage which is imported from the Image PIL. This library saves the streamed video as a single image. In addition to converting video to image, the library also adjusts the resolution of image to the quantity of beams available given as an input variable. This library is imported from Image module. The image module provides a class with many functions to load images from files and create new images. This module is used to capture different scenarios for a controlled experimental procedure.

The program requires the ability to compute and represent the results. These results and representations are based on multiple dynamic link libraries shown in all code within the Appendix III. The first library used is the Scipy library which is used in the calculations of the results. Another library used is the matplotlib library that includes the pyplot function. The pyplot function allows changes to a figure such as plotting area, plot labels, and creating a plot. This is used to compare and represent the results of each experiment. The Numpy library is the extension of the matplotlib library and is an array package that is also used in OpenCV. Another dynamic library used is the OpenCV library that is for array operations, preserved data types, as well as image processing. Within OpenCV, Array processing is used specifically with the third solution. Time is a function that is used to determine the time the CPU spend on executing each operation of the program. This is used for experimental purposes and is not a required function for the prototyping of the device's control system.

After the program has all the required initializations including importing all of the necessary libraries shown in Table 2, the program enters the first main loop. The first main loop ensures that the patient has a consistent stream of their environment, as the haptic display is produced. Within the first main loop there are six executables that are broken into three main sections as shown in Figure 9. The first main section is the image manipulation which changes the image pixel values. The second main section is the construction of waves which relates the pixel values to a single wave. The last main section is the sound production which sums all of the waves and produces a sound. The first main loop is also referred to as the frame loop because the loop iterates every time the user interprets a new frame.

The first section of the frame loop is the image manipulation, which consists of three executables. The first executable is to take an image of the steamed video in real time. The process for taking an image is extracting a single image that the video is streaming at the exact moment the code is executed. After the image is taken, it is stored in the device's random-access memory. After the image is stored, the image is then converted to grayscale. The last executable in the image manipulation is adjusting the resolution of the image to represent the quantity of beams available. The image is converted to grayscale before adjusting the resolution due to image compression model that python organizes when executing the ‘resizeimage’ module. The resolution is set to 64 x 64 for the proposed solution, however this is not a maximum resolution. The maximum resolution is dependent on the size of the beam array of the device. This means that the user could have a standard dvd resolution of 720 x 480 if the beams are small enough to fit 345,600 within a surface area that has nerves such as the user’s back.

After the image is manipulated, the information from the image is used for the construction of the waves. Construction of the waves is a nested Pixel Loop within the Frame Loop. Each wave is related to a single pixel of the manipulated image. This wave holds an amplitude value from 0-255, that is based on the grayscale value of the pixel that is used for the corresponding pixel’s beam on the haptic display. Each of the pixels will have a beam with an individual natural frequency that the amplitude is applied to. As each of the waves are calculated they are summed together to form a single wave, referred to as the Sum Wave. The Sum Wave is used to produce the sound that excites the beams on the haptic display. After the sound is produced, the Frame Loop goes back to take another image.

The improvements of the control system are to save time of the pixel loop so the frame rate is fast enough to ensure the user can sense motion and get information at a reasonable time. The current cinematic industry considers 24 FPS as an acceptable standard for reasonable time. The control system is evaluated for each improvement independently to determine what improvements impact the frame rate the most. The first improvement is threading the control by separating the image into multiple parts. Separating the image into multiple parts allows parallel execution of the frame loop shown in Figure 10.

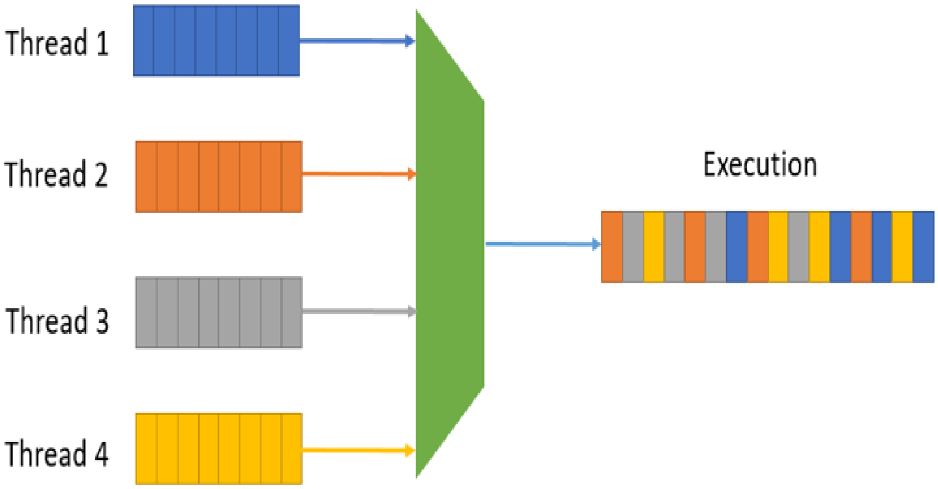


Figure 10: Threading Representation of Image Dividing for Parallel Execution

Figure 10 shows that the threading operations do not simply go from parallel operation to serial execution. This is due to how each of the threads are processed. For example, if thread 3 takes longer to execute, thread 1,2, and 4 will continue to execute within the same time. This process allows for some threads to finish executing before others. Therefore, the time used is the time for the last thread of the last pixel is considered total time for the frame execution time.

Threading is one opportunity to improve the frame rate. There are additional opportunities to increase the frame rate by limiting the image’s information. The last two improvements limit the image’s information by decreasing the gray scale range to 12 levels shown in Figure 11. The limited grayscale range is executed between the image manipulation section and the construction of the pixel waves. Changing the image’s bit depth from 8 to 4 decreases the number of grayscale levels. By decreasing the grayscale levels the calculations required for each frame is also decreased.

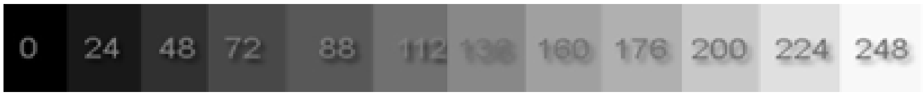


Figure 11: 12 Levels of Grayscale

Decreasing the bit depth is accomplished by changing the pixel value to the closest grouped value within Table 2. With a limited grayscale value, the number of waves to be calculated are decreased in the majority of the image, and in some cases images will not be represented effectively.

Table 2: 12 Levels of Grayscale

Level	Pixel Value Range
1	0 – 21
2	22 – 43
3	44 – 65
4	66 – 87
5	88 – 109
6	110 – 131
7	132 – 153
8	154 – 175
9	176 – 197
10	198 – 219
11	220 – 241
12	242 - 255

It is assumed that decreasing the grayscale levels does not affect the patient’s ablility to identify the environment with less grayscale. Figure 12 shows two common scenarios that the patient may experience when walking by either buildings or nature. Visually the images are different however, the objects in the images are still identifiable.

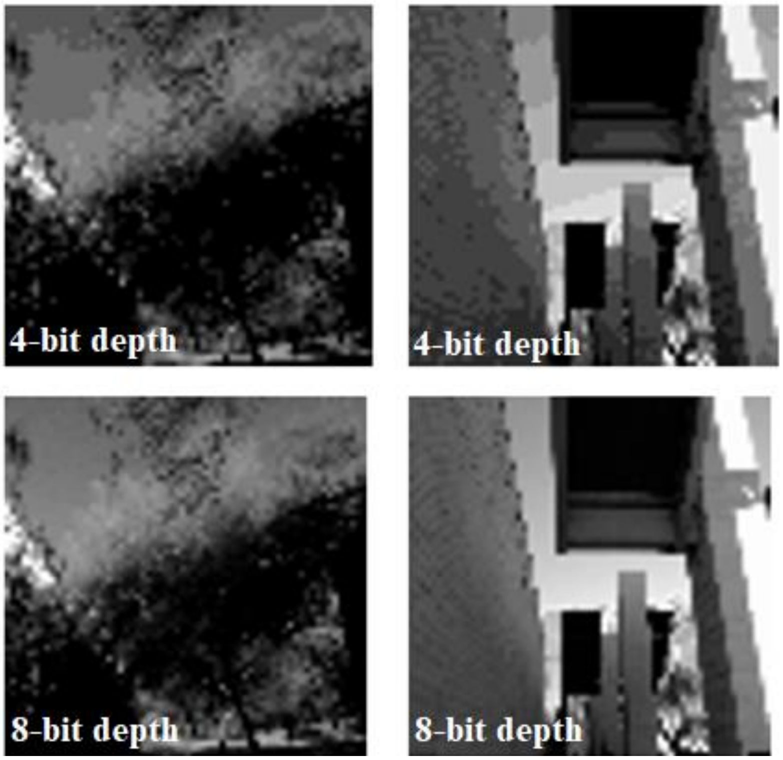


Figure 12: Comparison of 8-bit depth (bottom) and 4-bit depth (top) of the same image

The Change Filtering proposed improvement is to determine if the image has significant change from the previous image. This improvement is completed after the image has limited gray scale values, in the form of a conditional statement. This conditional statement comes before the pixel loop shown in Figure 13. The significant change improvement has the short coming of more memory to store previous image values; however, the benefit is that the image has the potential to not require recalculation.

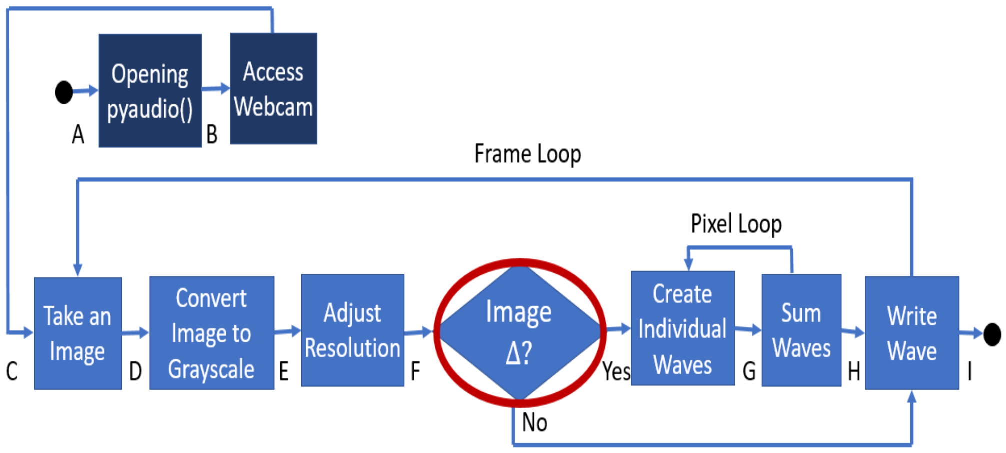


Figure 13: Flow chart of Python Control System with Conditional Statement for Significant Image Change

The Wave Library solution, is a change of logic in the flow of the system. This solution creates all of the possibilities of each wave for all individual beams within the initialization stage. Although the creation of the waves requires a longer initialization time, the calculation time for this initialization will only be required one time. After the waves for each beam are calculated, they are stored in memory. After the waves are stored in memory they can be called on, as shown in Figure 14.

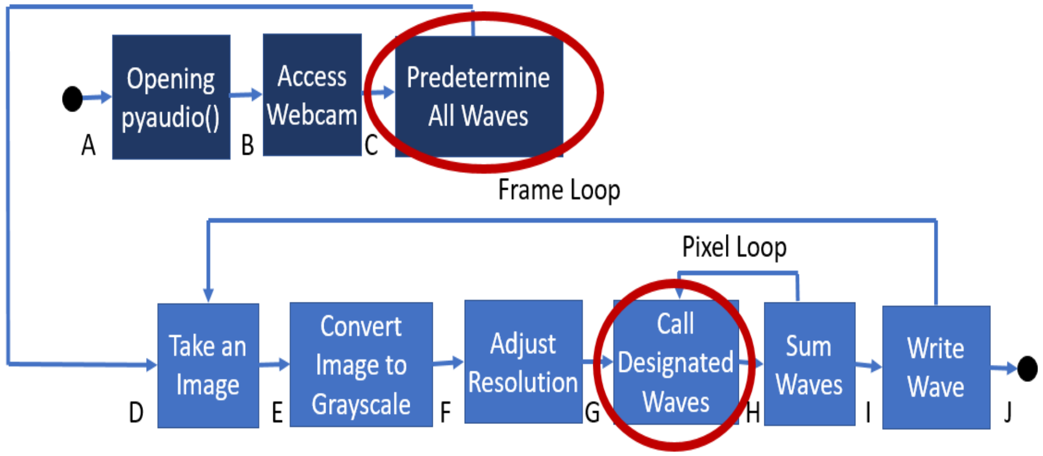


Figure 14: Flow chart of Python Control System with Predetermined Waves

These solutions are independent of each other and are studied under multiple conditions to determine individual efficiency. This means that each of the experiments were conducted under the same baseline and not dependent on each other. The time savings per frame rate can we improved further by combining all three of the improvements together.

4. Experimentation and Results

This chapter describes how the experimentation of each solution is compared with a standard baseline that includes the results of the control system. The baseline code that is run for each individualized experiment can be found in Appendix I. This code sets the measured parameters of interest for the starting point of each experiment. Without this baseline the effects cannot be quantified or interpreted as a measurement. Based on these results the system solutions can be organized into the most optimized system.

4.1 Experimental Setup

One experiment is performed for each proposed solution to evaluate its effectiveness. The Threading experiment uses the same image values across 10 images for an average time to complete a baseline, a single thread, two threads, and four threads. This experiment uses randomly selected values to represent the 10 images. For this experiment, it is hypothesized that the computation time will be halved when the number of threads is doubled. Therefore, with four threads, the time will be 25% of the time as a single thread. Due to the increase of time in the initialization for creating each thread, the comparison is made against the single thread instead of the baseline. However, the comparison with the baseline is hypothesized to show significant decrease in time.

The Change Filtering experiment utilizes videos of five common scenarios selected to include different amounts of motion, or change, in the video. These five scenarios are: (1) a conversation with someone, (2) walking down the hall, (3) walking next to a landscaped path, (4) driving in a car, and (5) walking on a city path as shown in the Appendix. The conversations scenario includes images from a laptop’s webcam to show what a person would see in a conversation. The setting of this scenario is within a conference room with a nonactive background. The second scenario is walking down a hall. The hall is also basic, however the motion of walking allows the background to be mildly active. This is similar to walking down a landscaped path. The main difference between walking down a landscaped path and a hallway is that the hallway has a simple background and the landscaped path has a very detailed background. In-between these two scenarios is walking down a city path. The city path includes mild landscaping as well as buildings that are large simple structures. The last scenario is the driving scenario. This includes nonactive areas such as a dashboard or visor, as well as extremely active parts of the image which is what the user sees through the windshield which can include either a landscaped path or a city path. These scenarios were chosen to represent five likely scenarios with different motion rates and different motion representation such as partial frame motion.

Each of these scenarios have different levels of motion that occur in each scene. It is hypothesized that the scenes with the least amount of motion will benefit the most from the Change Filtering solution. The benefit is potentially up to 100% of the computation time if there is no change. This solution is limited because it is only beneficial in low motion scenarios. The scenarios of the user being high motion are more likely and will therefore limit the time savings. The Wave Library experiment also utilizes videos of the five common scenarios.

4.2 Experimental Results

In a review of the literature, no standards were found for evaluating real time video processing methods for efficiency. Thus, a method is proposed here to carry out such an evaluation. The proposed evaluation is based on levels of complex motion and amount of motion within the frame. In order to evaluate the success of the proposed methods across a range of scenarios with varying levels of complexity and motion, five common scenarios were selected to be video recorded and evaluated. The level of motion within each video was subjectively evaluated.

The scenarios have 5 different levels of motion that are used as the metric to determine time savings of both the Change Filtering as well as the Wave Library solutions. These levels of motions are the following: hardly any motion, motion in partial frame, fast changing motion, moderate changing motion, and slow changing motion. Each of the experimental scenarios are rated in level of motion. The first video was taken during a conversation. In this experiment the code executes 10 frames per second to determine how many pixels were considered changed after increasing the tolerance of each pixel to consider it changed.

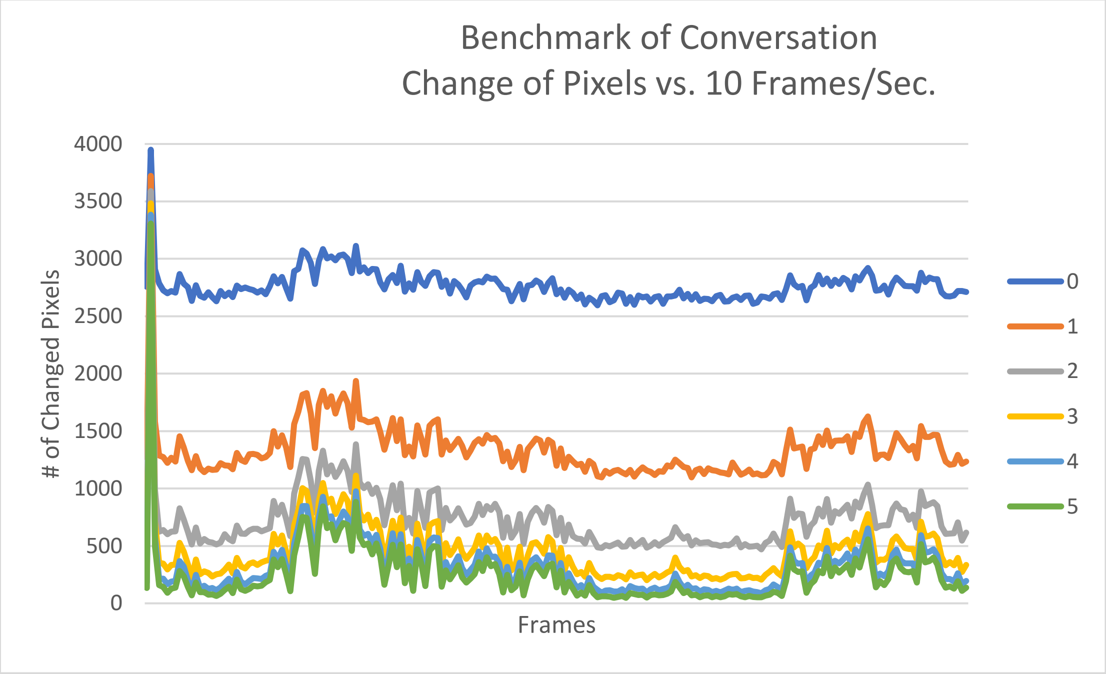


Figure 15: Tolerance of 0-5 pixel changes for Conversation Benchmark

Figure 15 shows the number of pixels that changed their grayscale value relative to the previous frame for each frame of the 'Conversation' video, for each of 6 levels of 'change threshold'. For example, the blue curve shows the number of changed pixels in each frame when any change greater than 0 is detected as a change. The green curve shows the number of changed pixels in each frame when only a change greater than 5 is detected as a change. The differences in the curves in Figure 15 show that the baseline (change threshold of 0) requires recalculation for roughly 2700 pixels which is slightly above 60% of the image. When the tolerance of the pixel changes from 0 to 1 there is significant reduction in the number of changed pixels. Since only changed pixels require recalculation, this would give a significant reduction in the amount of time required for calculation. The image has the most significant improvement at the transition from 0 to 1 pixel tolerance however the difference between 1 to 2 pixel tolerance is also significant and continues to increase as the tolerance increases.

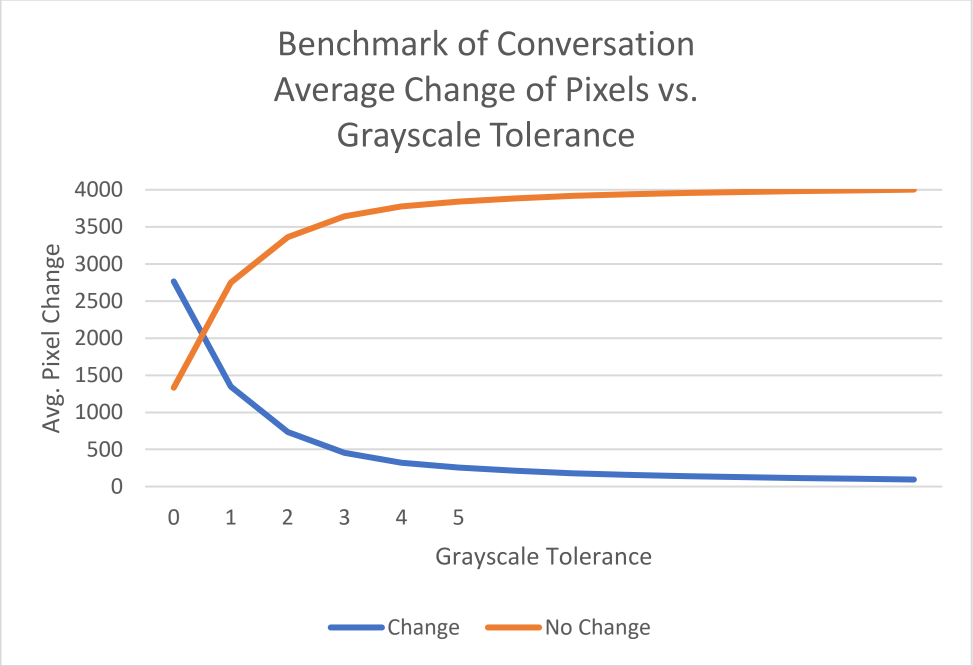


Figure 16: Average Change of Pixels vs. Pixel Differential in Conversation Scenario

Figure 16 shows the change of pixels against the grayscale levels in the conversation scenario. As the grayscale tolerance increases to the point of plateauing the solution becomes ineffective because the image will not show as changing and motion will not be interpreted by the user.

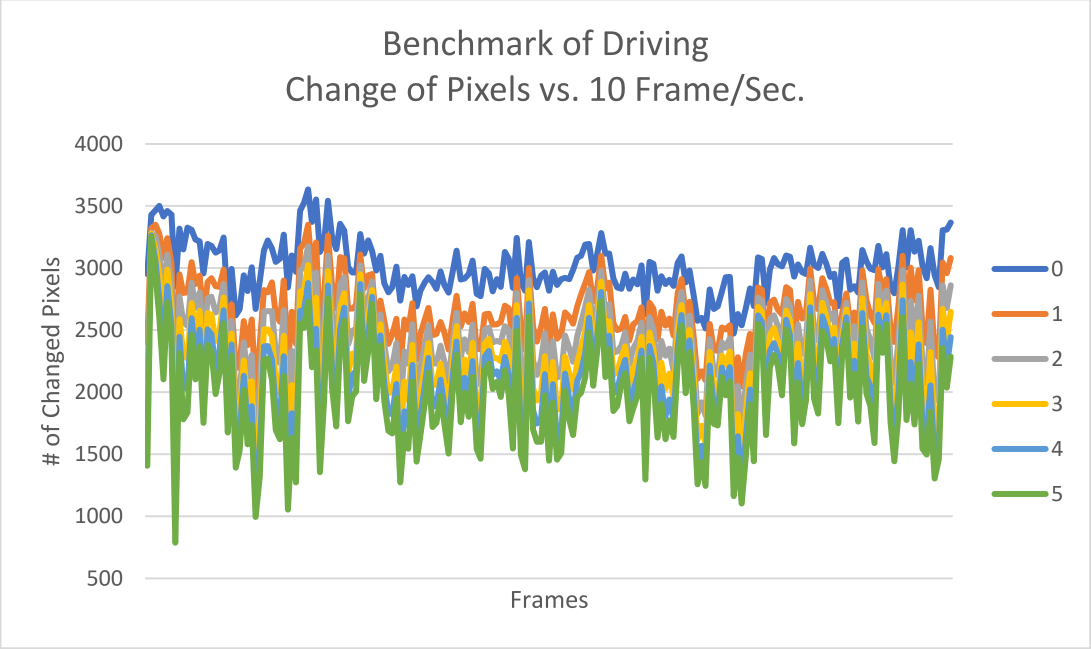


Figure 17: Tolerance of 0-5 pixel changes for Driving Benchmark

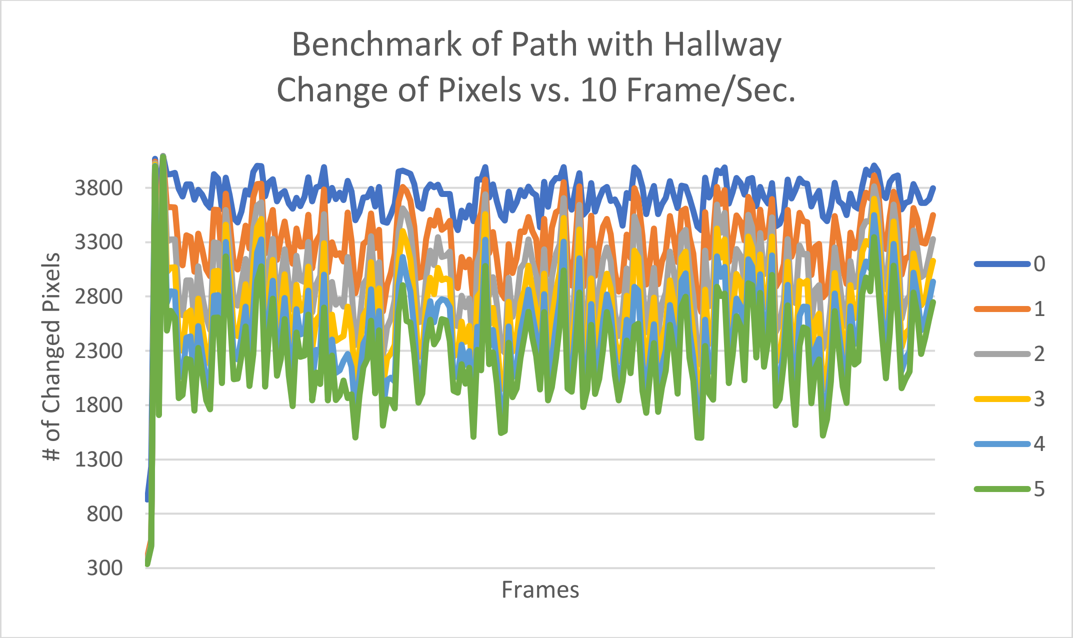


Figure 18: Tolerance of 0-5 pixel changes for Hallway Benchmark

Figure 17 is the plot of the number of pixels changed in the Driving scenario with six different grayscale levels. Figure 18 is the same plot with the Hallway scenario. The Driving scenario shows that as the pixel tolerance increases, the number of changed pixels within the image is slightly decreased. This difference between the conversation scenario and the driving scenario is that the motion changes consistently. The driving scenario has motion in partial frame in streaming and walking down the hall with slow changing motion. Due to the motion being slow changing there is not a large change in the pixels changed as shown in the Conversation scenario, Figure 15. Figure 17 and Figure 18 have similar patterns such that the difference between each grayscale level is roughly the same. However, the averages per grayscale level is shifted, Figure 18 the Hallway scenario has a higher average of pixels changing overall.

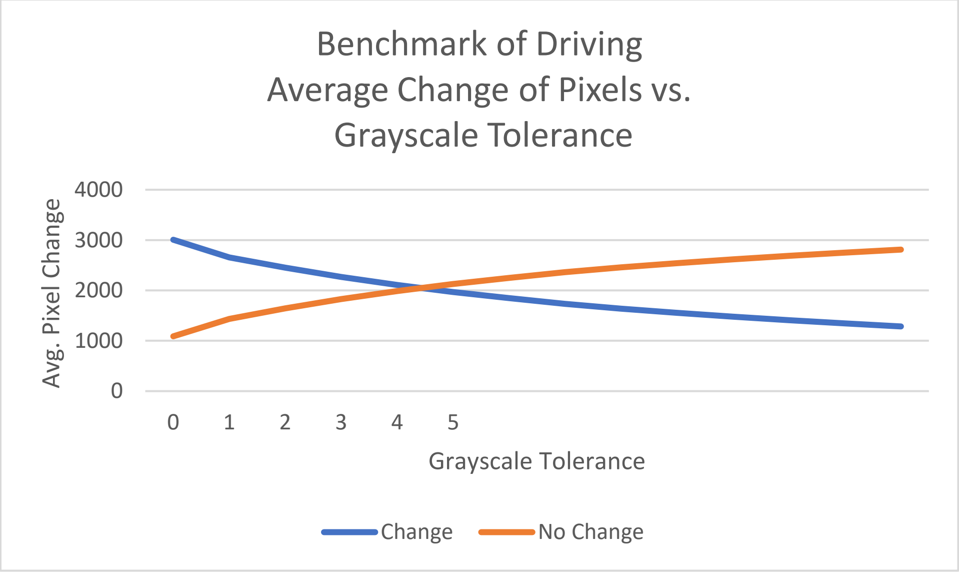


Figure 19: Average Change of Pixels vs. Pixel Differential in Driving Scenario

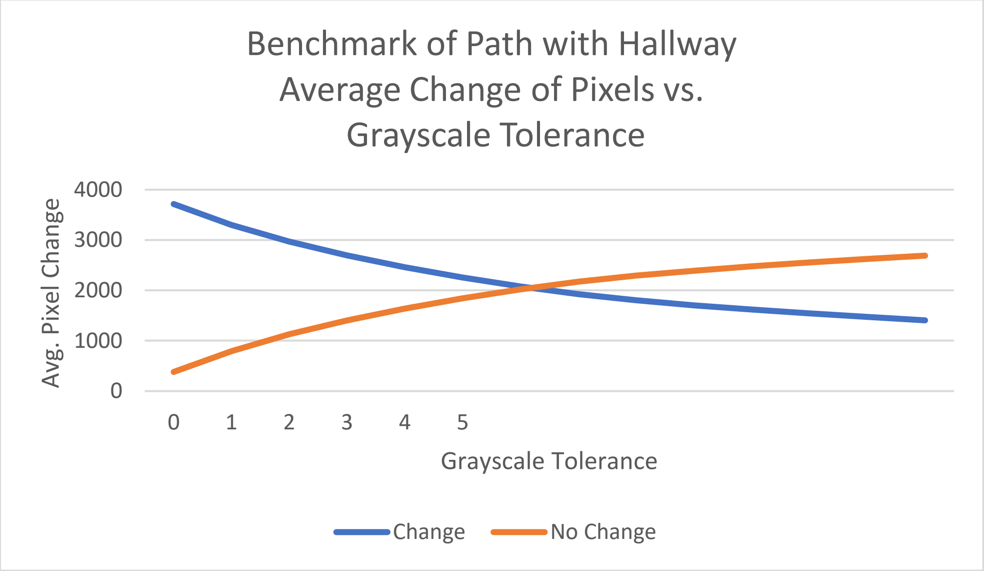


Figure 20: Average Change of Pixels vs. Pixel Differential in Walking Down a Hall Scenario

Figure 19 shows the average change of pixels against the grayscale tolerance for the Driving scenario. Figure 20 shows the same plot as Figure 19 with the Hallway scenario. Figure 20 shows that there is consistent climb of the average pixel change and the grayscale tolerance increases. Due to the number of changed pixels decreasing at a slow rate, the grayscale cannot be set for multiple scenarios. For example, if both conversation and driving are set at the same value of grayscale there will be significant sacrifice in either case. If the grayscale tolerance is set to 3 for the conversation optimization, the driving scenario will have limited benefits.

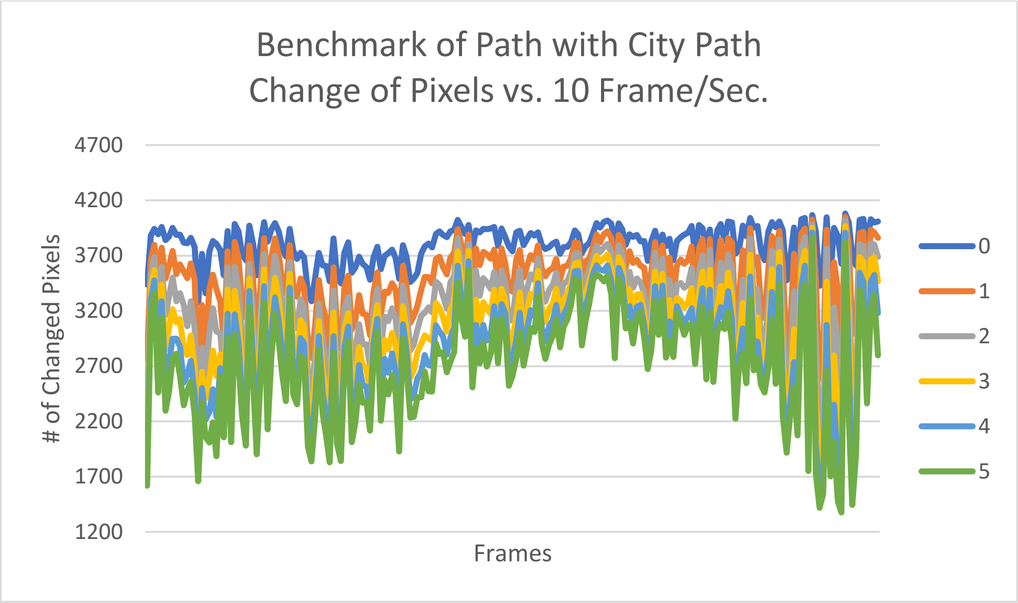


Figure 21: Tolerance of 0-5 pixel changes for City Path Benchmark

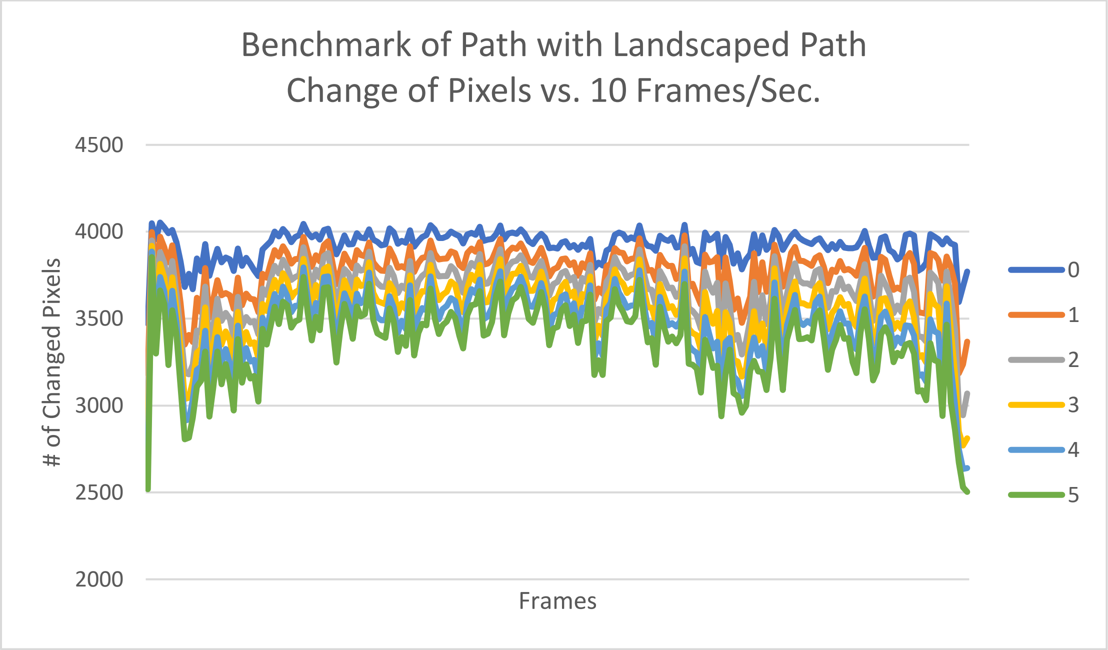


Figure 22: Tolerance of 0-5 pixel changes for Landscaped Path Benchmark

Figure 21 shows the number of pixels changed for six grayscale levels for the City Path scenario. Figure 22 shows the same plot for the Landscaped Path scenario. Walking down a path that is naturally landscaped or on a path next to city buildings, the change in image is significant. Although the images streamed include images that are changing, in the driving scenario and walking down the hall scenario the change is only in partially the frame; whereas the landscaped and building paths, the users experiences change within the entire frame. Due to this difference in frame change, the number of pixels changed for each scenario has no correlation to each other and therefore cannot depend on a single grayscale tolerance.

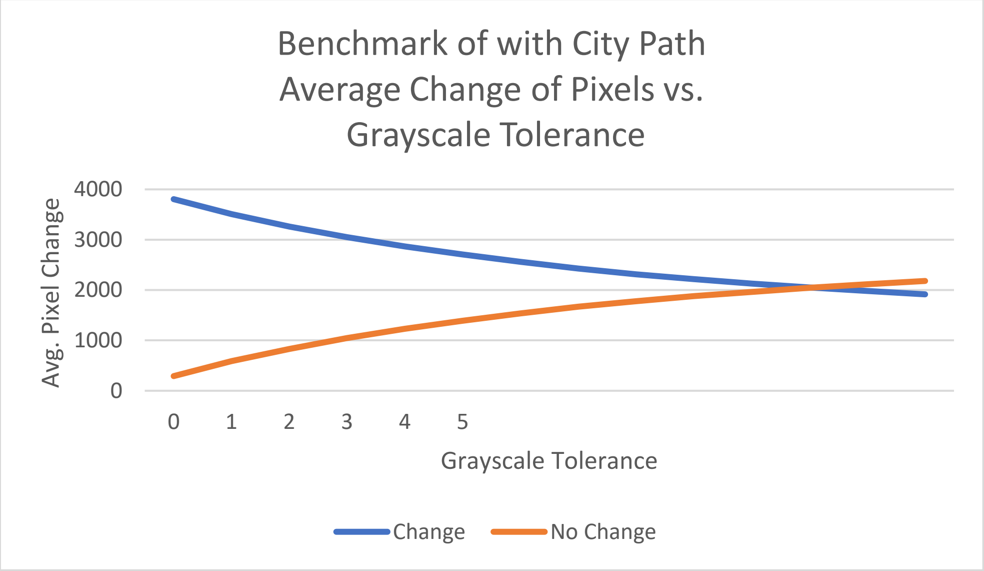


Figure 23: Average Change of Pixels vs. Pixel Differential in Walking Down a City Path Scenario

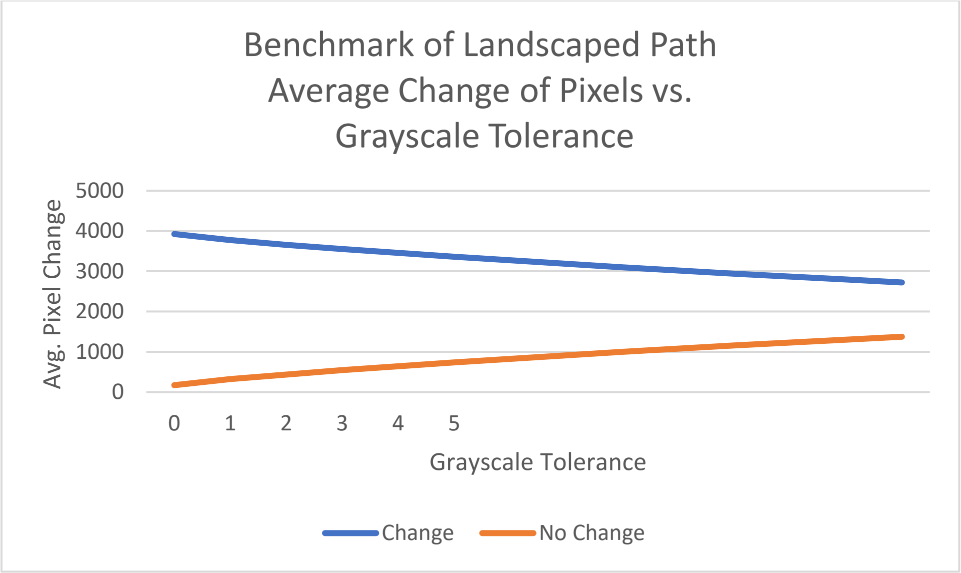


Figure 24: Average Change of Pixels vs. Pixel Differential in Walking Down a Landscape Path Scenario

Figure 23 show the average change of pixels against the grayscale tolerances. Figure 24 shows the same plot except instead of a City Path scenario, Figure 24 is of the Landscaped Path scenario. The City Path scenario is moderately changing in motion while streaming. The Landscape scenario is fast changing motion in streaming. The Conversation scenario shows that the tolerance of grayscale can be as low as 4 pixels before climbing up to majority of the frame being considered changed. Figure 25 shows that at less than 500 frame changes all scenarios stay at roughly none of frame changed up.

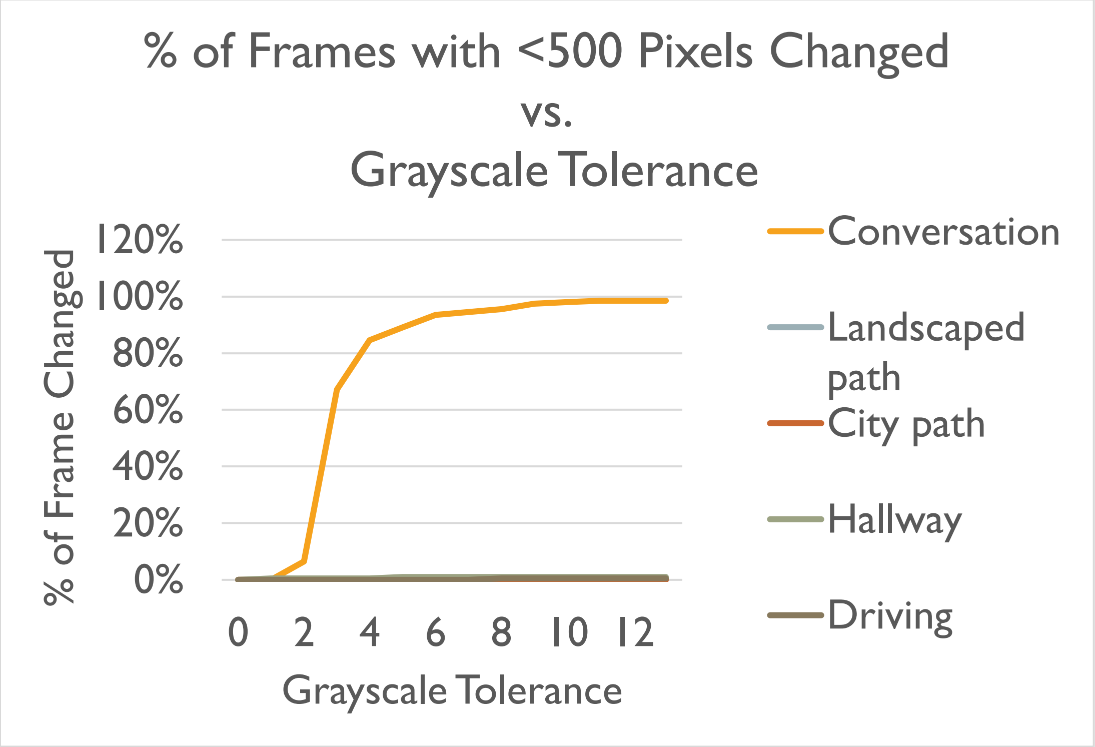


Figure 25: Percent of Frame Changed with Tolerance Increase for Pixel Changes < 500

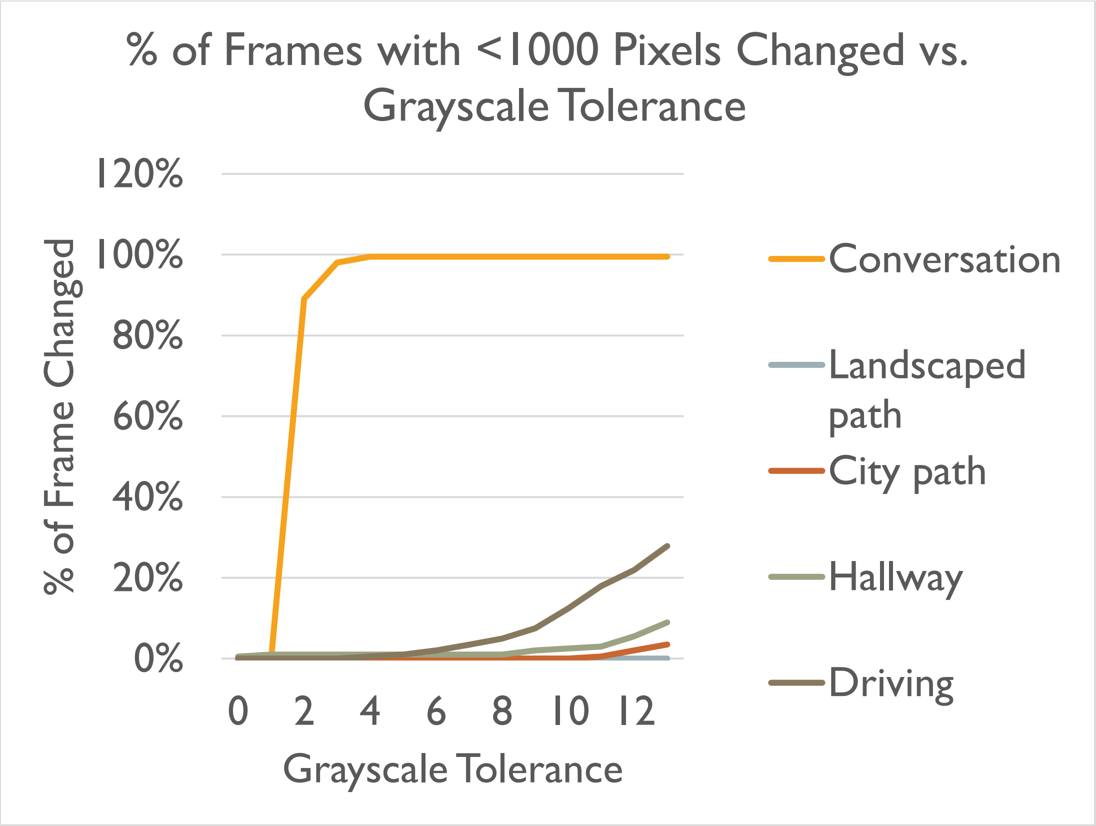


Figure 26: Percent of Frame Changed with Tolerance Increase for Pixel Changes < 1000

Figure 25 shows the percentage of the frame that is changed with a tolerance of 500 pixels for the grayscale tolerance levels for each scenario. Figure 26 shows the same plot as Figure 25 with a tolerance of 1000 pixels. As the pixel change increases from 500 to 1000 shown in Figure 26, the driving scenario does start to represent a frame change at roughly 20%. This means that with partial frame change the pixel change can be represented with less than 1000 pixels with a grayscale tolerance of 11. By adding another 500 pixel tolerance, the pixel change within a frame includes all five scenarios except for the landscaped path. The compromise for increasing the pixels changed in frame by 1000 is that the gauge for low motion conversation scenario includes 100% of the frame changed with a grayscale tolerance of 1.

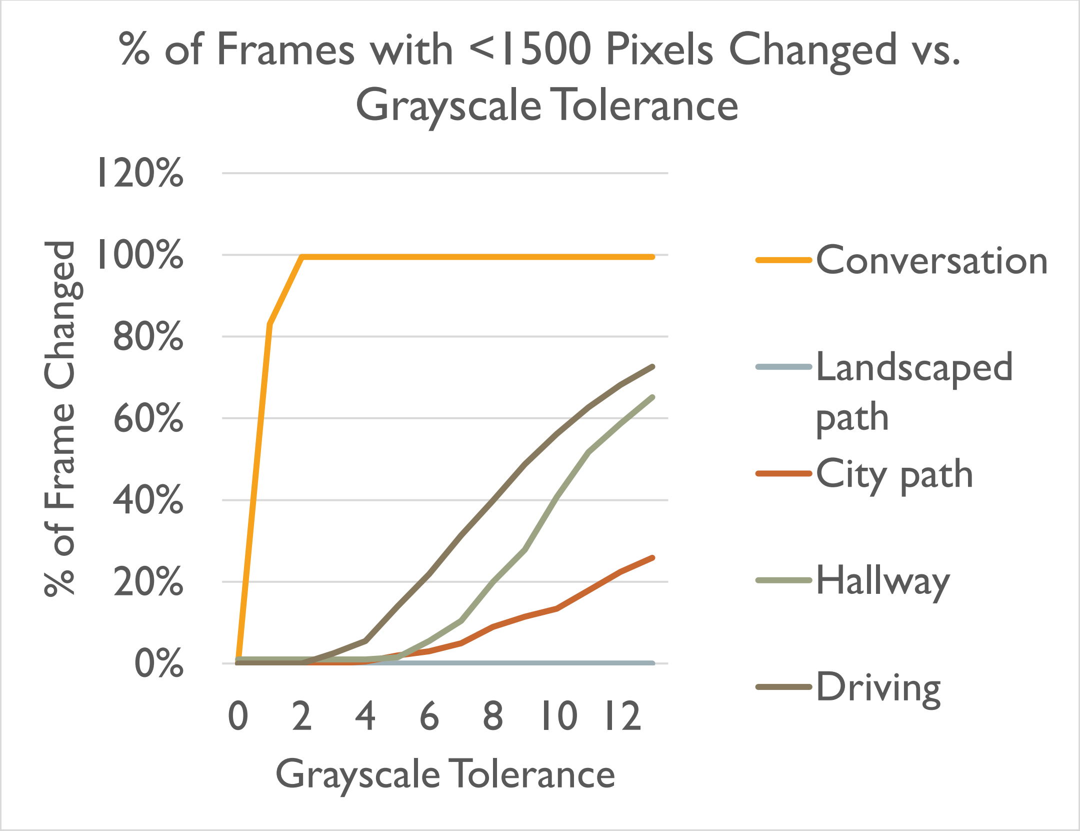


Figure 27: Percent of Frame Changed with Tolerance Increase for Pixel Changes < 1500

Figure 27 shows the percentage of the frame that is changed with a tolerance of 1500 pixels for the grayscale tolerance levels for each scenario. In Figure 27 the frame rate change with less than 2000 pixels, the driving and hallway scenarios where the frame goes up to 80% changed at less than 10 grayscale tolerances is ideal however, the city path is roughly half of driving/hallway frame rate change at 10 grayscale tolerance with 40% of the frame changed.

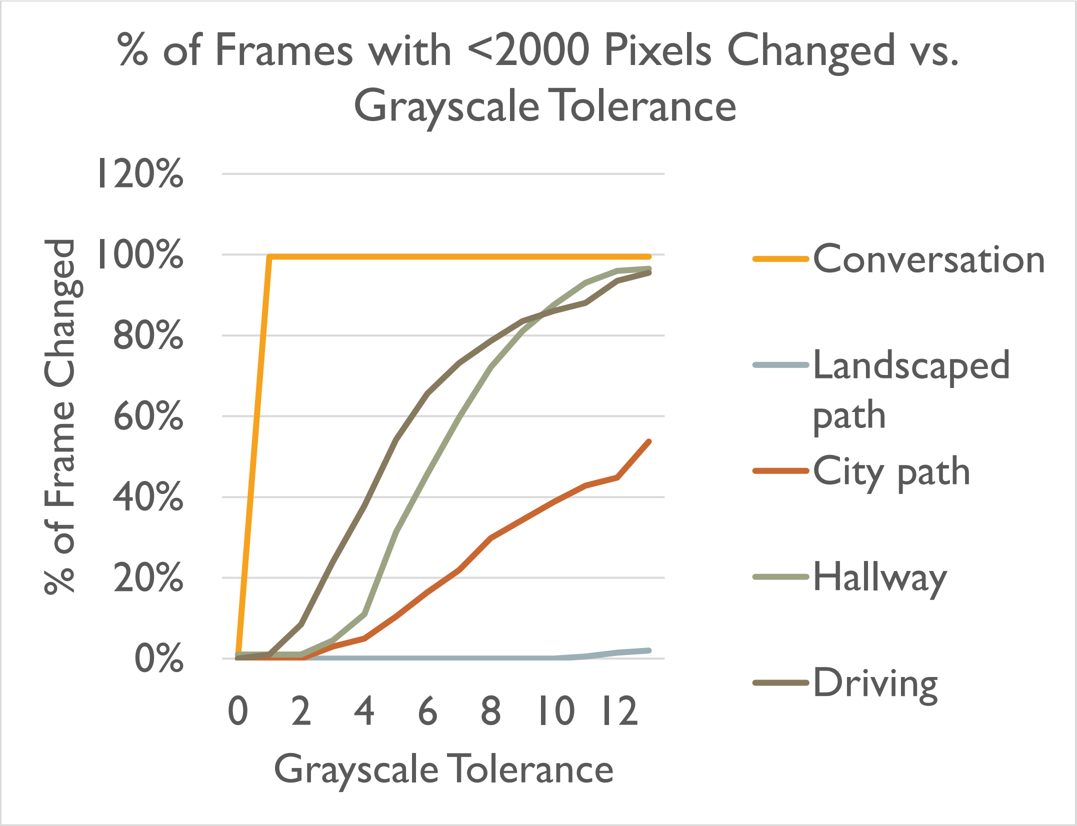


Figure 28: Percent of Frame Changed with Tolerance Increase for Pixel Changes < 2000

Figure 28 shows the percentage of the frame that is changed with a tolerance of 2000 pixels for the grayscale tolerance levels for each scenario. With the total resolution at 4096 pixels the experiment with pixel change of less than 2500, more than half of the available image that changes are considered within the tolerance difference. Therefore, the landscaped path where the images were changing often, the percentage of frame changed starts to increase to roughly 20% at a grayscale tolerance of 12 shown in Figure 28. This pixel change yields the same results for images that were not changing often such as the conversation scene.

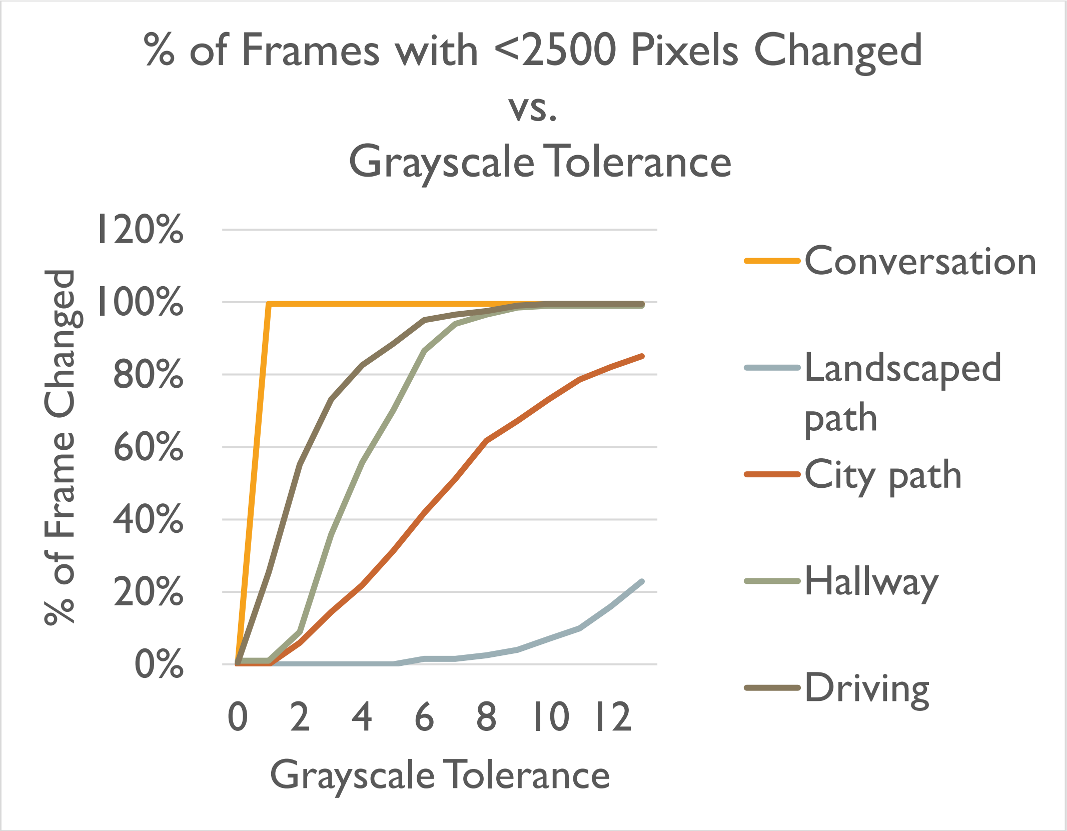


Figure 29: Percent of Frame Changed with Tolerance Increase for Pixel Changes < 2500

Figure 29 shows the percentage of the frame that is changed with a tolerance of 2500 pixels for the grayscale tolerance levels for each scenario. The last comparison of frame changes with a tolerance increase for pixels changed less than 3000. When considering this high of pixel change the conversation as well as the driving scenarios start with changes at ~50% and ~90% of the frames respectively. The results are analyzed when determining the grayscale range as well as planning for future development. This future development is to consider additional algorithms for efficient computing based on feedback.

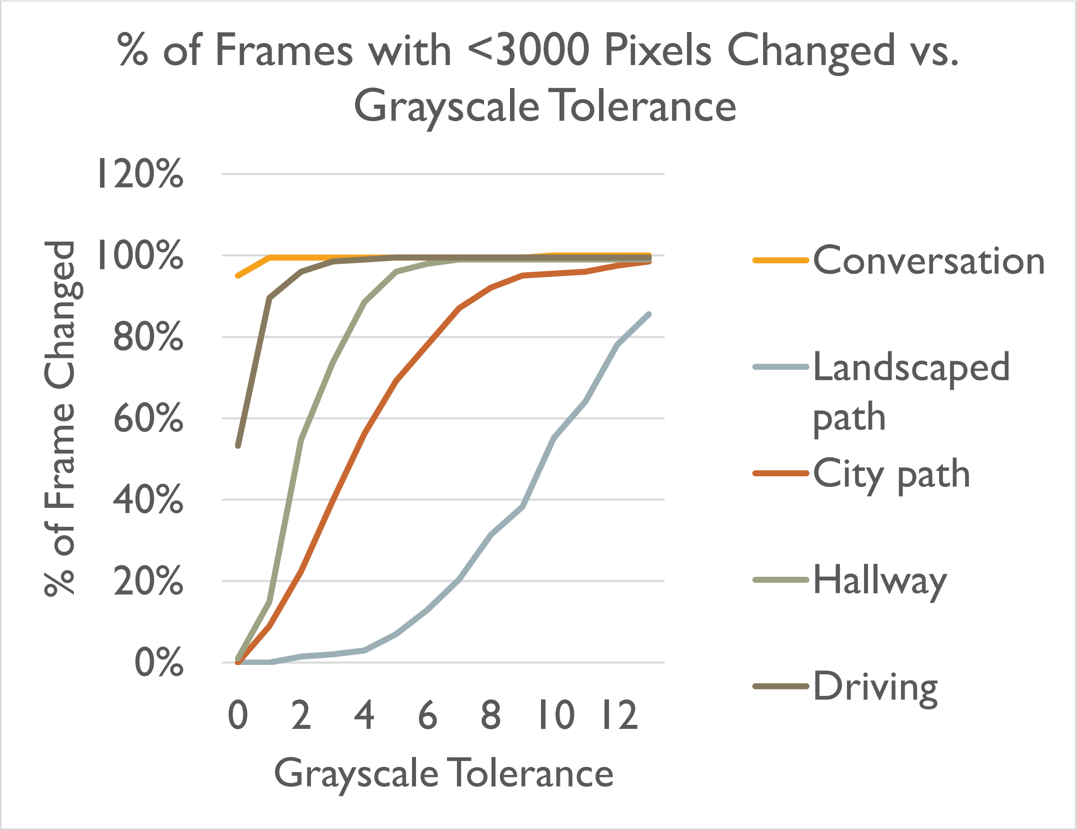


Figure 30: Percent of Frame Changed with Tolerance Increase for Pixel Changes < 3000

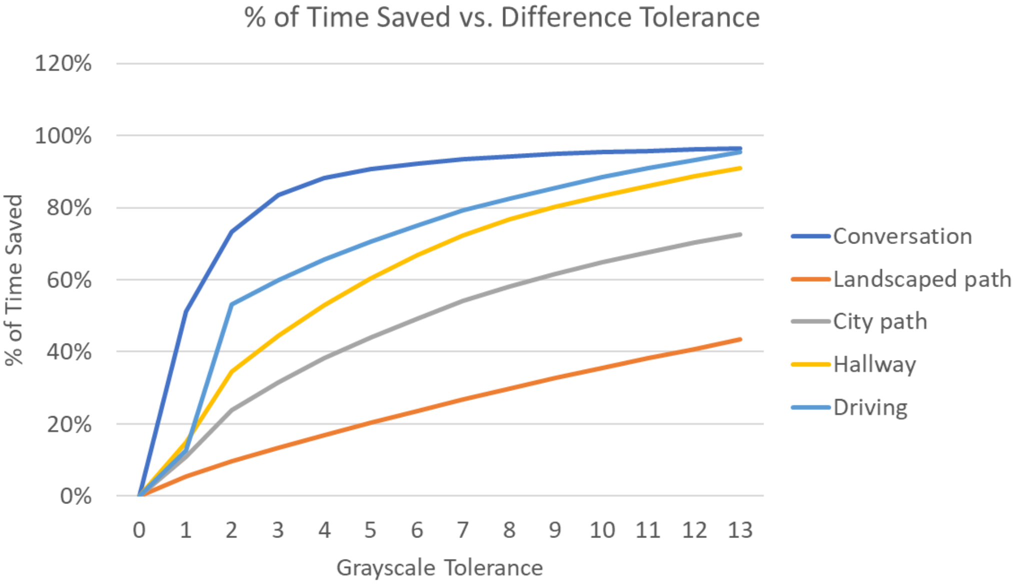


Figure 31: Percent of Time Saved vs the Difference Tolerances

Figure 30 shows the percentage of the frame that is changed with a tolerance of 3000 pixels for the grayscale tolerance levels for each scenario. At this tolerance the Conversation scenario starts off at almost 100% of the frame changed and the Driving scenario is roughly 50% already changed with no grayscale level increase. Figure 31 shows the percentage of the time saved with multiple grayscale tolerance levels for each scenario. The Wave Library improvement increases the initialization time by creating all predetermined waves. The predetermined waves include the 12 levels of grayscale ranged amplitudes for every beam. The benefit of this process is that the calculations will only need to be run one time. This will require an increased initialization time, however as the length of time that the program runs increases, the less significant the initialization time becomes. After the waves are created in an array during the initialization, the program analyzes which wave to call based on the pixel grayscale value or the corresponding beam.

Based on all experiments the results show that every solution does improve the frame rate. The first solution of threading with four threads improve time by roughly 52%. However, this solution alone does not bring the frame rate to the target 30 frames/second, and will have to be considered when combining multiple solutions. The second solution of storing original image and comparing images after to determine the change of image improves time by roughly 99.9% if the image does not change. The big downfall to this solution is that in this solution has the potential to not only add time with additional storage but could not improve the time at all if the images are always significantly changing. The last solution of predetermined waves and change of logic to call stored waves improves time by roughly 96.5%. This solution alone improves the frame rate to reach the target 30 frames/second.

5. Conclusions and Discussion

In all the experiments there was improvement in the frame rate. The threading experiment resulted in three different improvements. Table 3 shows the results of this threading experiment. The first thread takes slightly longer than the baseline. This is expected because the baseline does not include the threading initialization. Although this is not required every iteration of the code, it is required one time. As the number of frames increases the initialization becomes negligible. The hypothesis of this experiment was that there would be a proportional decrease of frame rate with an increase of threading. When the number of threads increase to two, the frame rate was hypothesized to be roughly 50% because the image to be processed in each thread is half. As the threads increase to four, the image is broken into four sections with the expectation of the frame rate to decrease to roughly 25%. The results show that the two-thread experiment resulted in roughly 60% decrease of time, however the four-thread experiment resulted in roughly 50% decrease of time.

Table 3: Results of the Threading Time Study

	Baseline (4096)	1 Thread (4096/thread)	2 Threads (2048/thread)	4 Threads (1024/thread)
Average Time to Complete (10 images)	4.787	4.905	2.971	2.491
STDDev	0.056	0.049	0.064	0.181

The Change Filtering solution of determining if there is significant change has high variability. The time study shown in Table 4 shows the potential of time to be saved because the time to call the previous wave is significantly less than the time to make a new wave. The high variability of the time savings is due to the dependent nature on the user's environment.

Table 4: Results of the Change Filtering Experiment

Time to Call Previous Wave/ Pixel (sec.)	Time to Make Wave/ Pixel (sec.)	Time to Append Wave (sec.)	Time for Pixel Change
3.674e-6	0.00194	0.002	0.00394

The Wave Library solution of improving the frame rate by rearranging the sequence of the control system to include predetermined waves that are called. Based on arbitrary images the improvement is substantial enough to meet the target frame rate.

Table 5: Results of the Wave Library Experiment

Original Initialization (sec.)	Logic Improvement Initialization (sec.)	Original Frame Loop for 50 Frames (sec.)	Logic Improvement Frame Loop for 50 Frames (sec.)	Original Total Time for 50 Frames (sec.)	Logic Improvement for 50 Frames (sec.)
0.706	3.891	266.744	5.471	267.45	9.363

This improvement was validated through experimentation results shown in Table 5. In all five experimental scenarios the frame rate exceeds the target frame rate shown in Table 6.

Table 6: Wave Library Improvements for Experimental Conditions

	Conversation	Landscaped Path	City Path	Hallway	Driving
Time Saved (sec.)	88.747	94.863	93.635	99.902	97.432
% Saved	94.84%	94.88%	95.07%	95.29%	94.96%
Frame Rate (FPS)	41.658	39.266	41.383	40.697	38.833

The average percentage of time savings is ~95% seconds across the five experimental conditions. This time savings results in the average frame rate across the five experimental conditions to be 40 frames/second. The Wave Library solution is a significant improvement from the original 2 frames/second and meets the goal of 30 frames/second.

6. Future Work

There are potential future works in three main categories of this control system. The first is improving the current solutions. The python control system is primarily based on the concept of converting image to sound. The development of this process can be further expanded in a number of different applications. In addition to the application of converting image to sound, each of the solutions have the opportunity to expand. The first solution, running the code in parallel, also known as threading has the opportunity to run the program on a GPU instead of a CPU. The GPU has thousands of cores with the ability to process parallel workloads more efficiency than the CPU which only contains multiple cores. The second solution of determining if the pixels have experience significant value change can be expanded to determine if the pixels of an image have experienced significant change. This development would require focus on determining the effect on the frame rate. The last solution has the opportunity to store the predetermined waves on a network so more pixel grayscale values are available to the user on the cloud instead of in the device’s memory.

The second future improvement is expanding on the Change Filtering proposed solution. Currently, the amount of motion in the videos used to evaluate the proposed solutions was evaluated subjectively. This means that these were determined low to high activity based on human evaluation. As an item for future work, the use of the Change Filtering method could be evaluated as a way to objectively quantify the amount of change within a video to determine the processing difficulty.

The last future improvement is total system testing. This control system is designed for a vibrotactile haptic display currently in development. Full system integration can include testing for harmonic distortion, overlapping natural frequencies, as well as user implementation. User implementation testing could include determining the most sensitive part with the largest surface area of the human body. The implementation can also include how much pressure the skin needs to contact each beam.

Although there is opportunity to develop each solution, visually impaired scenario metric, as well as the total system integration testing, this control system concept also has opportunity for further development as well as application. Conversion from image to sound can be further expanded with artificial intelligence as well as image compression standards. The application can also be further researched to determine if this solution would improve current system processes.

References

[1] X. Zhang *et al.*, “Human echolocation: waveform analysis of tongue clicks,” *Electron. Lett.*, vol. 53, no. 9, pp. 580–582, 2017.

[2] W. Liu and M. S. Humayun, “Artificial retinal prosthesis to restore vision for the blind,” in *2000 Digest of the LEOS Summer Topical Meetings. Electronic-Enhanced Optics. Optical Sensing in Semiconductor Manufacturing. Electro-Optics in Space. Broadband Optical Networks (Cat. No.00TH8497)*, 2000, pp. I61–I62.

[3] Y. H.-L. Luo and L. da Cruz, “A review and update on the current status of retinal prostheses (bionic eye),” *Br. Med. Bull.*, vol. 109, no. 1, pp. 31–44, Mar. 2014.

[4] J. M. Ong and L. da Cruz, “The bionic eye: a review,” *Clin. Experiment. Ophthalmol.*, vol. 40, no. 1, pp. 6–17, Jan. 2012.

[5] P. Bach-y-Rita, *Brain mechanisms in sensory substitution.* New York,: Academic Press, 1972.

[6] E. Sampaio, S. Maris, and P. Bach-y-Rita, “Brain plasticity: ‘visual’ acuity of blind persons via the tongue,” *Brain Res.*, vol. 908, no. 2, pp. 204–207, Jul. 2001.

[7] A. C. Nau, C. Pintar, A. Arnoldussen, and C. Fisher, “Acquisition of Visual Perception in Blind Adults Using the BrainPort Artificial Vision Device,” *Am. J. Occup. Ther.*, vol. 69, no. 1, p. 6901290010p1-6901290010p8, 2015.

[8] T. Gonnot and J. Saniie, “Integrated machine vision and communication system for blind navigation and guidance,” in *2016 IEEE International Conference on Electro Information Technology (EIT)*, 2016, pp. 0187–0191.

[9] T. H. Nguyen, T. H. Nguyen, T. L. Le, T. T. H. Tran, N. Vuillerme, and T. P. Vuong, “A wearable assistive device for the blind using tongue-placed electrotactile display: Design and verification,” in *2013 International Conference on Control, Automation and Information Sciences (ICCAIS)*, 2013, pp. 42–47.

[10] Z. Liu, Y. Luo, J. Cordero, N. Zhao, and Y. Shen, “Finger-eye: A wearable text reading assistive system for the blind and visually impaired,” in *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2016, pp. 123–128.

[11] “Combining haptic and braille technologies: design issues and pilot study.” [Online]. Available: http://www.sigchi.org/chi96/proceedings/papers/Ramstein/CR_BRL.HTM. [Accessed: 23-Feb-2017].

[12] T. Park, J. Jung, and J. Cho, “A method for automatically translating print books into electronic Braille books,” *Sci. China Inf. Sci.*, vol. 59, no. 7, p. 072101, Jul. 2016.

[13] M. Romero, B. Frey, C. Southern, and G. D. Abowd, “BrailleTouch: designing a mobile eyes-free soft keyboard,” in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, 2011, pp. 707–709.

[14] P. Bach-y-Rita and S. W. Kercel, “Sensory substitution and the human–machine interface,” *Trends Cogn. Sci.*, vol. 7, no. 12, pp. 541–546, Dec. 2003.

[15] “Blind Sight: The Next Generation of Sensory Substitution Technology,” *The Crux*, 28-Apr-2014.

APPENDIX I

PHOTOS OF VIDEOS FOR EXPERIEMTNATION

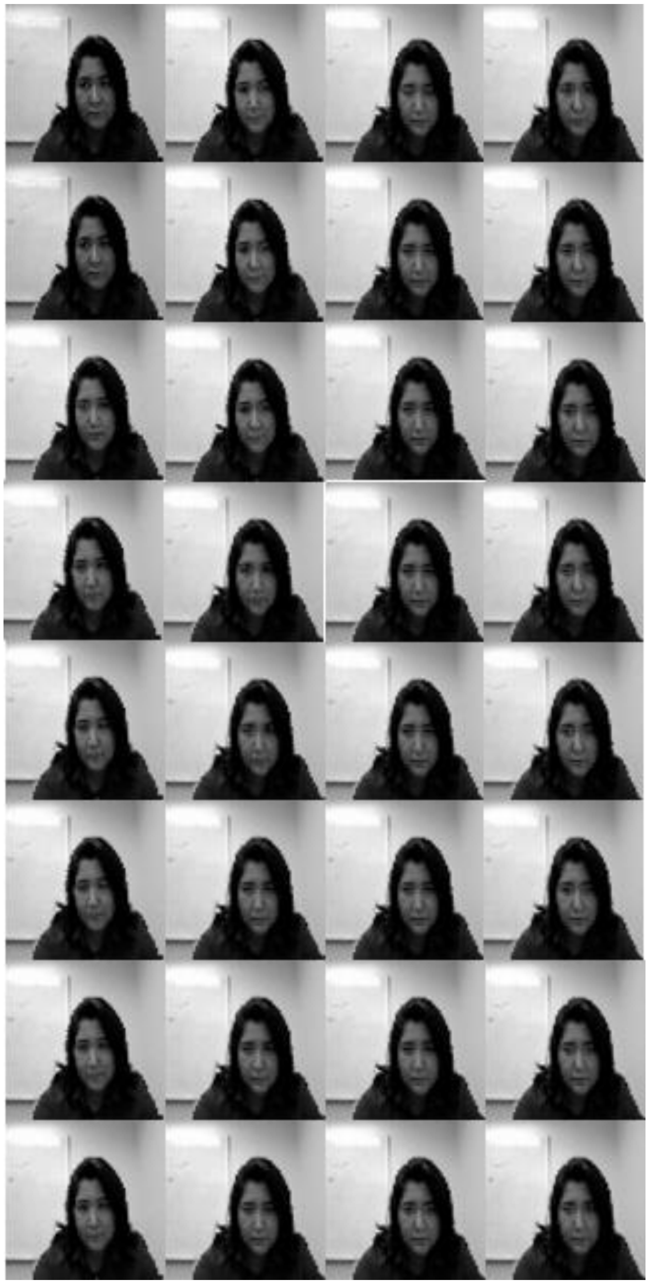


Figure 32: Images of *Conversation* Video

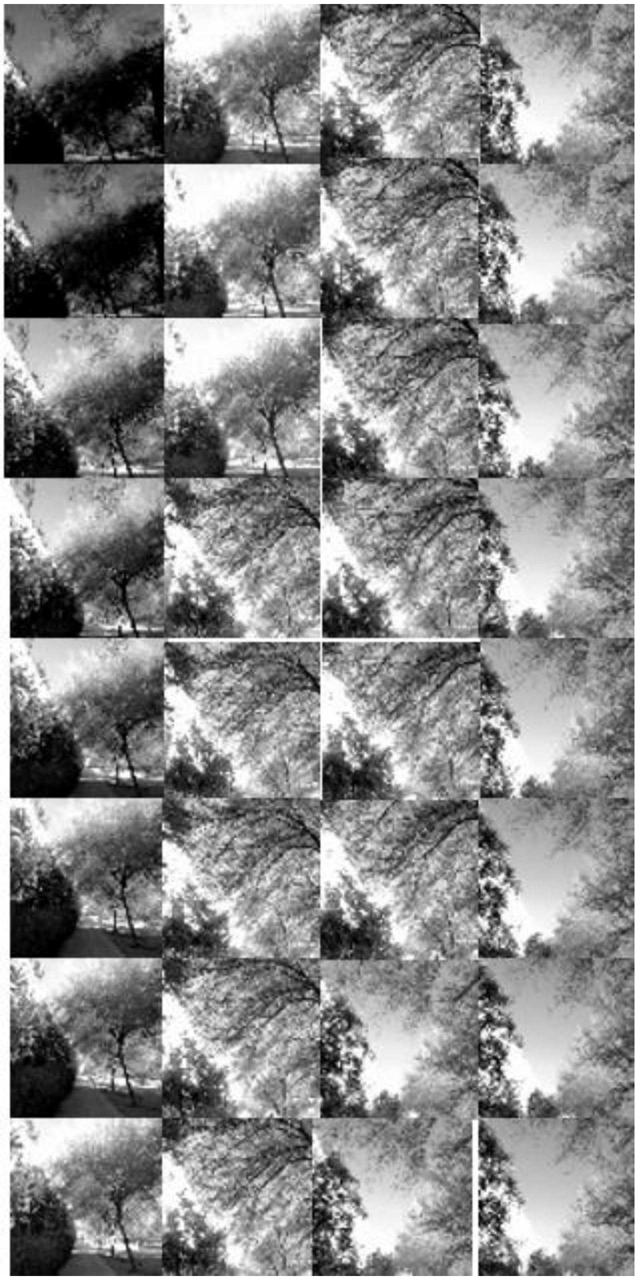


Figure 33: Images of Walking down *Landscaped Path* Video

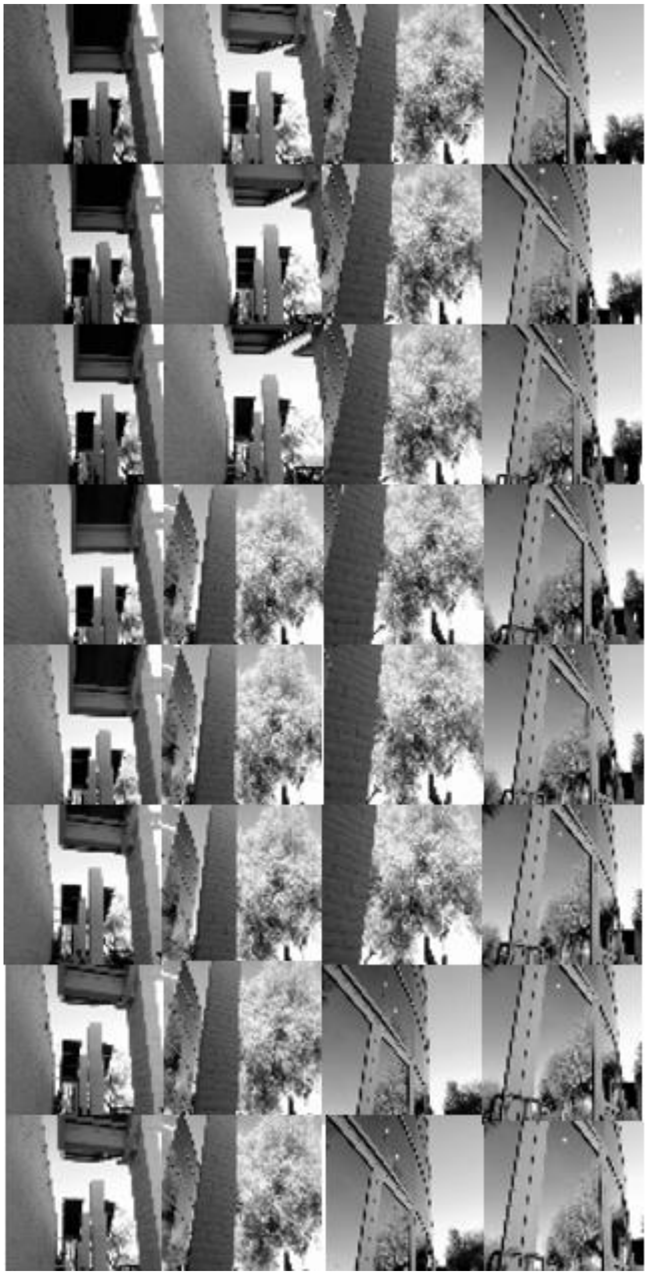


Figure 34: Images of Walking down a *City Path* Video

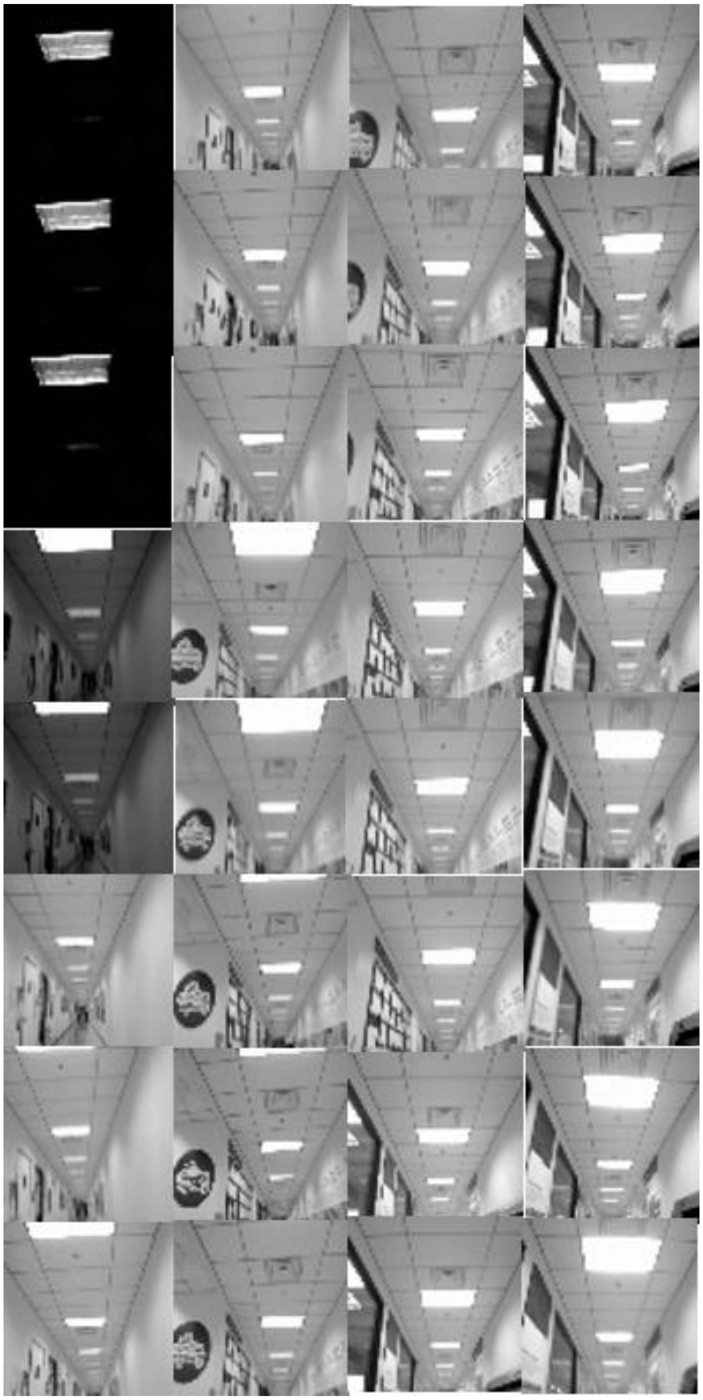


Figure 35: Images of Walking down a *Hallway* Video

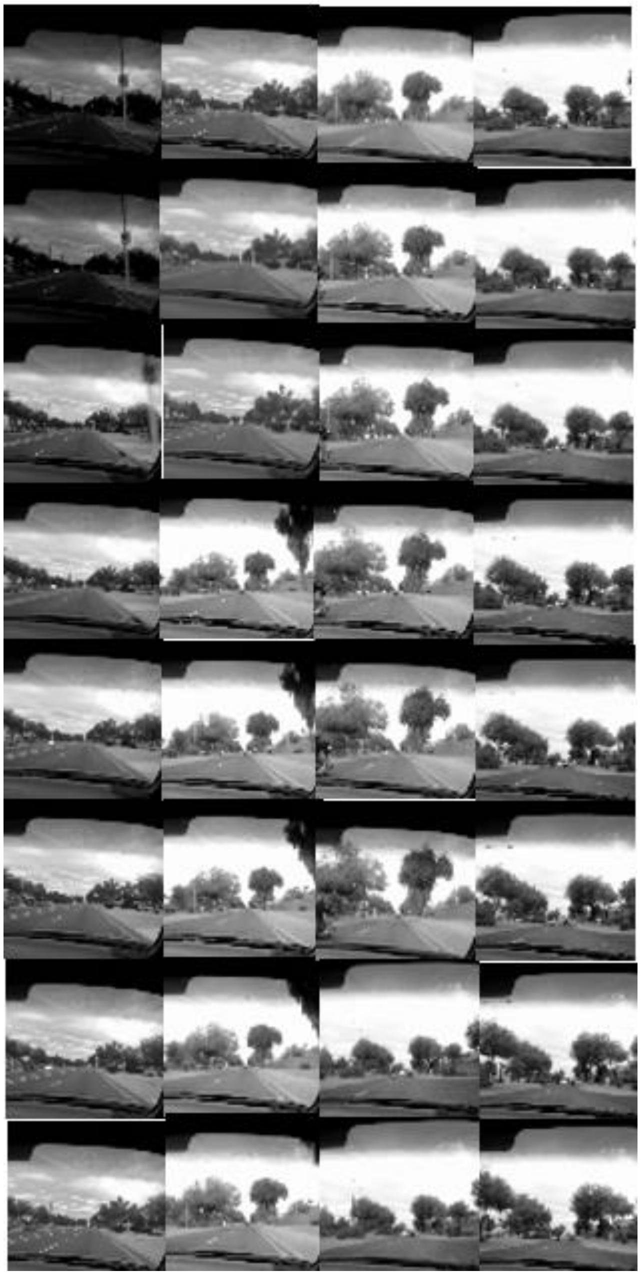


Figure 36: Images of *Driving* Video

APPENDIX II

ADDITIONAL GRAPHS

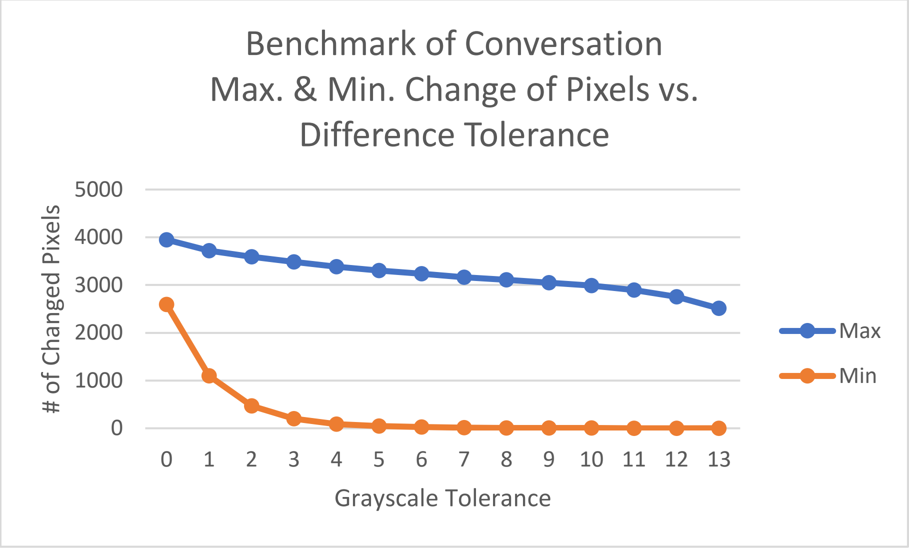


Figure 37: Minimum and Maximum Pixel Change vs Grayscale Tolerance for the *Conversation* Video

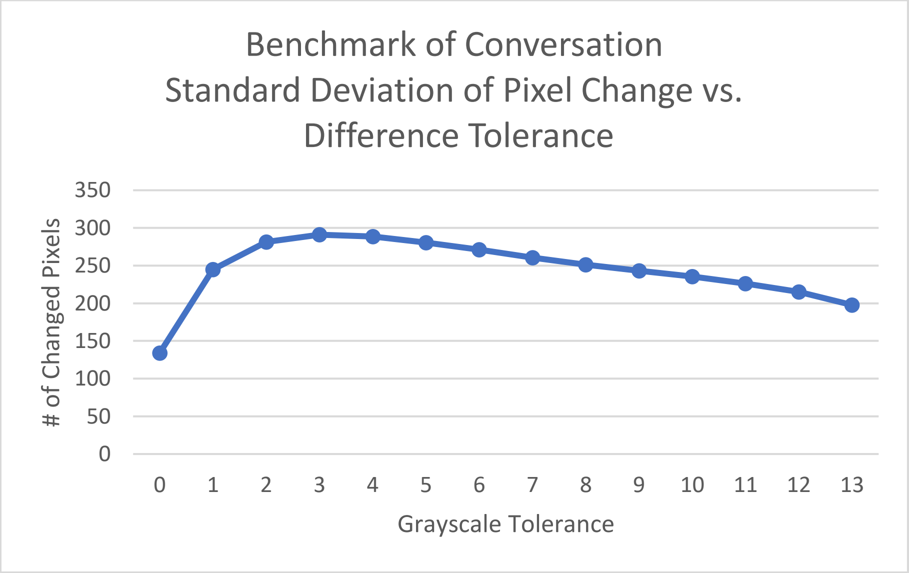


Figure 38: Standard Deviation of the Pixel Change vs Grayscale Tolerance for the *Conversation* Video

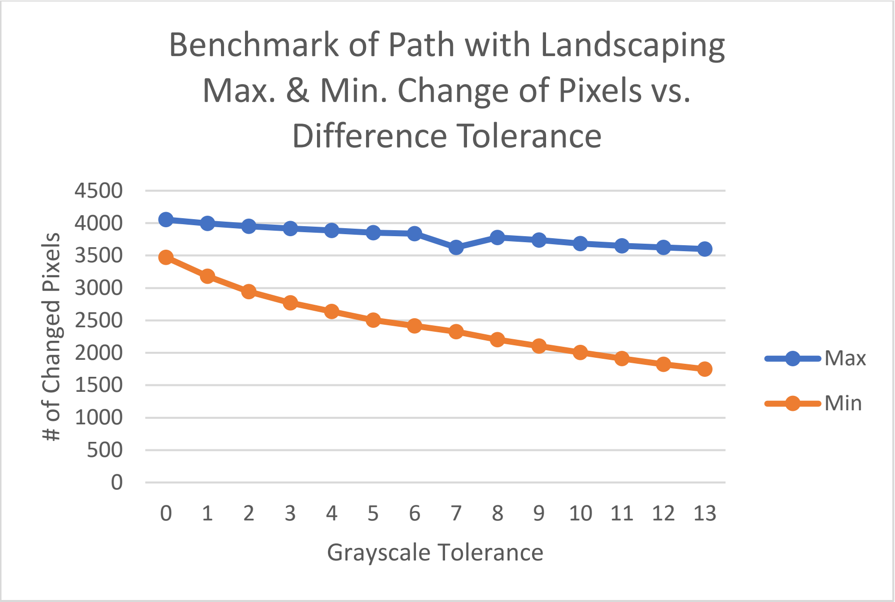


Figure 39: Minimum and Maximum Pixel Change vs Grayscale Tolerance for the *Landscaping* Video

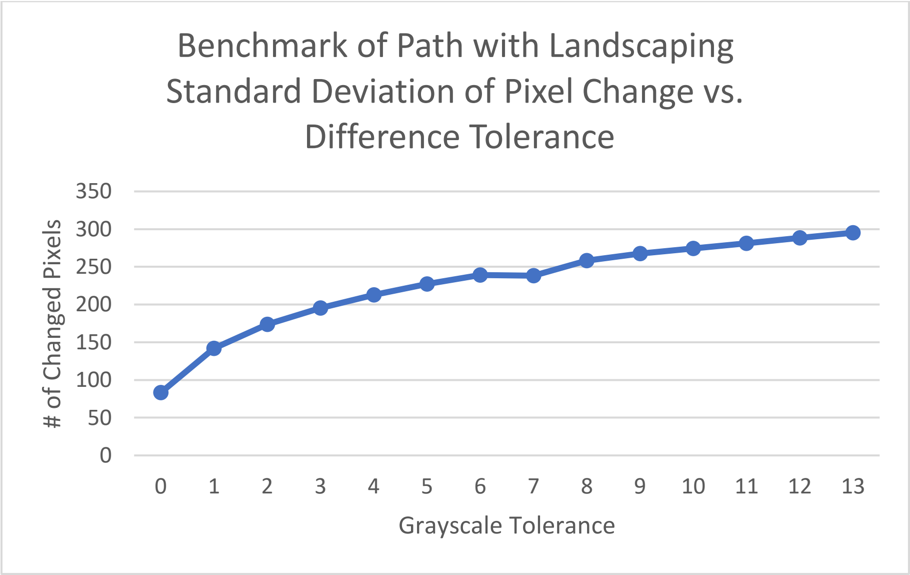


Figure 40: Standard Deviation of the Pixel Change vs Grayscale Tolerance for the *Landscaping* Video

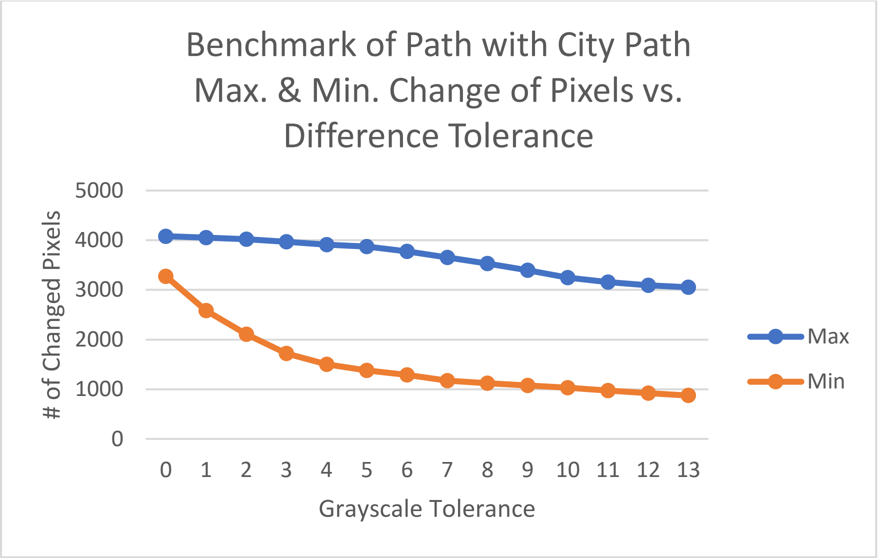


Figure 41: Minimum and Maximum Pixel Change vs Grayscale Tolerance for the *City Path* Video

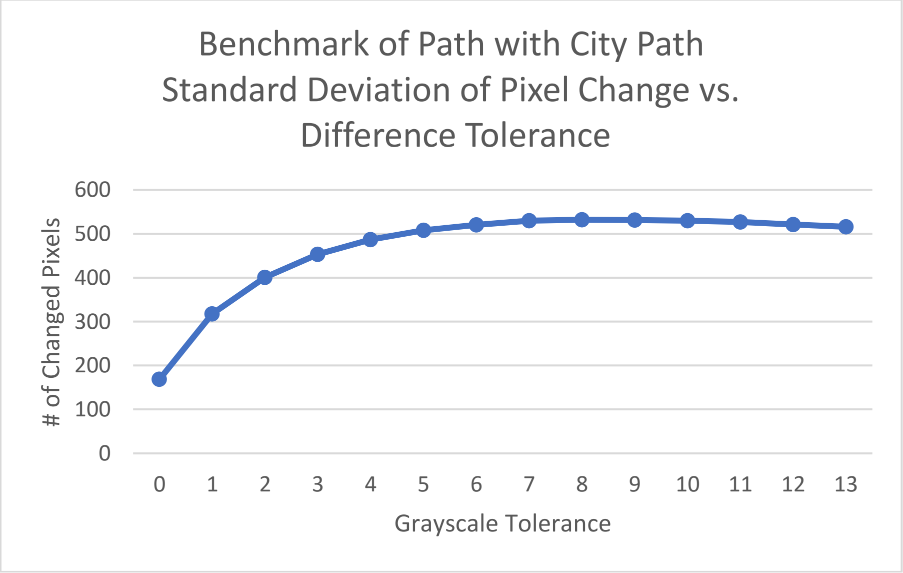


Figure 42: Standard Deviation of the Pixel Change vs Grayscale Tolerance for the *City Path* Video

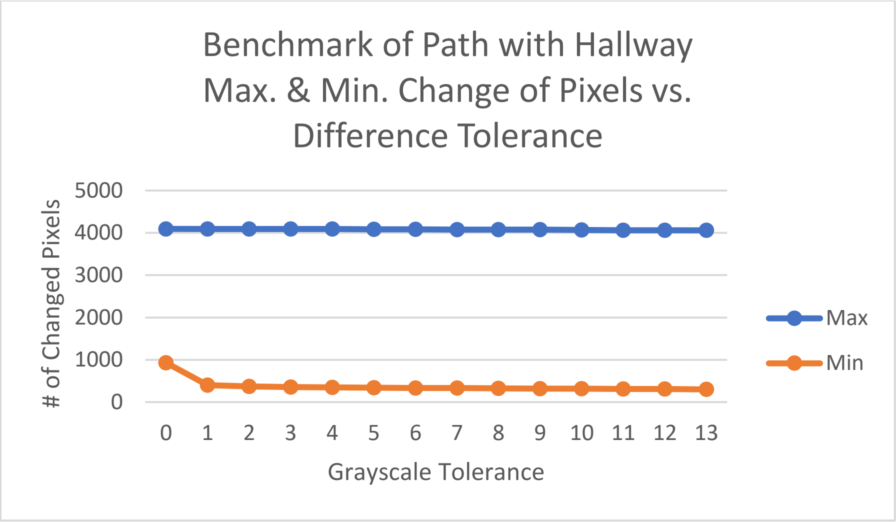


Figure 43: Minimum and Maximum Pixel Change vs Grayscale Tolerance for the *Hallway* Video

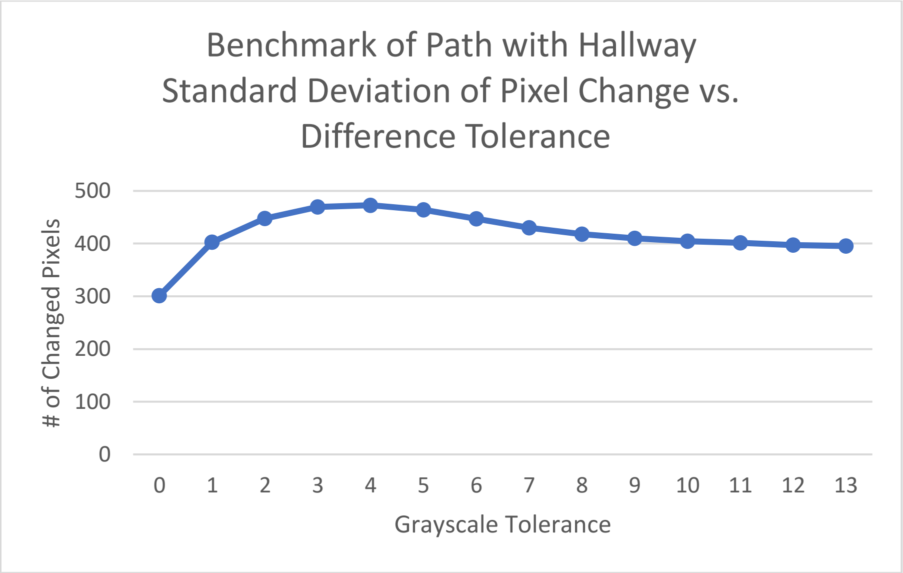


Figure 44: Standard Deviation of the Pixel Change vs Grayscale Tolerance for the *Hallway* Video

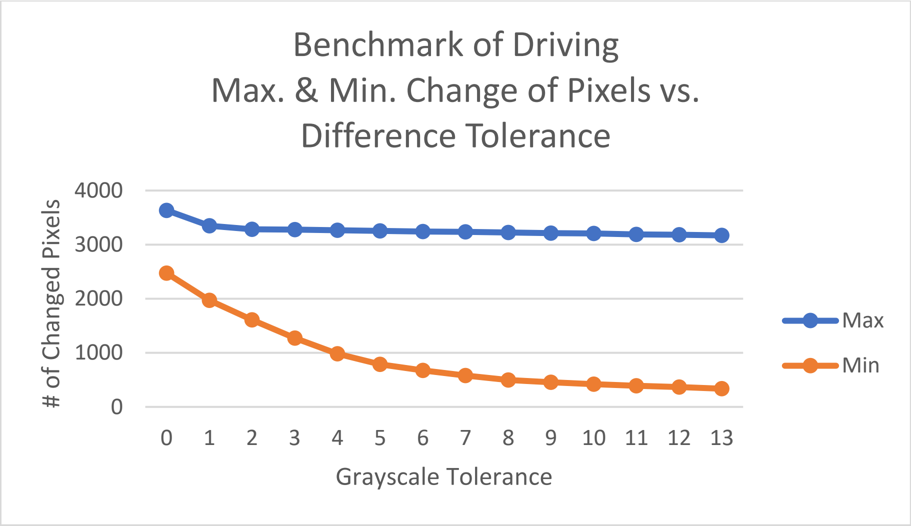


Figure 45: Minimum and Maximum Pixel Change vs Grayscale Tolerance for the *Driving* Video

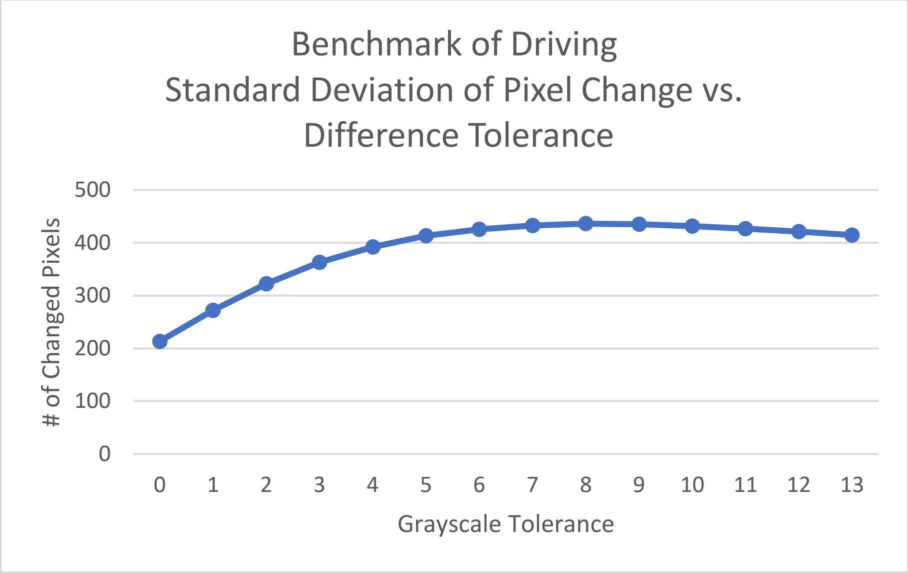


Figure 46: Standard Deviation of the Pixel Change vs Grayscale Tolerance for the *Driving* Video

APPENDIX III

CODE

Original Time

Code to show the time of each step per resolution from 8x8 to 64x64

```
    import cv2
    from PIL import Image
    from resizeimage import resizeimage
    import pyaudio
    import numpy as np
    import time
    import matplotlib.pyplot as plt
    import math
    import scipy
    import pylab
    from numpy.random import randn

    img_counter = 0
    p = pyaudio.PyAudio()
    fs = 44100                  # sampling rate, Hz, must be integer
    duration = .10              # in seconds, may be float                # sine
frequency, Hz, may be float
    f = 20                      # sine frequency, Hz, may be float
    waves=[]                    # array of wavesa= .0002
    a=0#.0002                   #amp/volume range [0.0, 1.0] for 4096
    t = np.linspace(0, duration, fs)   # used in plot
    benchinc=0
    count=0
    freq=[]
    amp=[]
    sumsamples=0
    check =0
    waves=[]
    countTest=0


    #**********A*********************
    startA=time.time()
    # open pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paFloat32,
                    channels=1,
                    rate=fs,
                    output=True)
    endB=time.time()
    #**********B*********************
```

```
#access webcam
startB=time.time()
cap=cv2.VideoCapture(0)
endC=time.time()

while countTest <= 7:
#****Benchmarking*****************
#*********************************
    if countTest == 0:
        sumMax = 8*8
    if countTest == 1:
        sumMax = 16*16
    if countTest == 2:
        sumMax = 24*24
    if countTest == 3:
        sumMax = 32*32
    if countTest == 4:
        sumMax = 40*40
    if countTest == 5:
        sumMax = 48*48
    if countTest == 6:
        sumMax = 56*56
    if countTest == 7:
        sumMax = 64*64

    # access the correct values for the soundwave (freq & amp)
    for count in range (benchinc, sumMax):
        freq.append(f)
        amp.append(a)
        f=f+4
        a=a+0.0002
#*********************************
#*********************************

#**********C**********************
    #capture an image
    startC=time.time()
    ret, frame = cap.read()
    endD=time.time()
#**********D**********************
    #convert image to grayscale
    startD=time.time()
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    endE=time.time()
```

```
#**********E**********************
  #resize image
  startE=time.time()
  new_img = gray.resize((64,64))
  endF=time.time()

#**********F**********************
  #calculating soundwave
  startF=time.time()
  for inc in range (benchinc, sumMax):
    #create sinewaves for each pin
    f=freq[inc]
    a=amp[inc]
    w = 2. * np.pi * f
    samples = a*np.sin(w * t)
    endG=time.time()
#**********G**********************
    startG=time.time()
    #sum each pin wave into a single wave
    sumsamples = samples + sumsamples
    endH=time.time()
#**********H**********************
  startH=time.time()
  stream.write(sumsamples)
  endI=time.time()

  countTest=countTest+1
#**********I**********************

#****Benchmarking*****************
#*********************************
  AtoB=endB-startA
  BtoC=endC-startB
  CtoD=endD-startC
  DtoE=endE-startD
  EtoF=endF-startE
  FtoG=endG-startF
  GtoH=endH-startG
  FtoH=endH-startF
  HtoI=endI-startH
  print "Resolution: ", sumMax
  print "A > B", AtoB
  print "B > C", BtoC
  print "C > D", CtoD
```

```
   print "D > E", DtoE
   print "E > F", EtoF
   print "F > G", FtoG
   print "G > H", GtoH
   print "F > H", FtoH
   print "H > I", HtoI
#**********************************
#**********************************
stream.close()
p.terminate()
cap.release()
```

Threading Solution

Code to show the time of each threading experiment

```
import cv2
from PIL import Image
from resizeimage import resizeimage
import pyaudio
import numpy as np
import time
import matplotlib.pyplot as plt
import math
import scipy
import pylab
import threading
from threading import Thread



img_counter = 0
p = pyaudio.PyAudio()
fs = 44100                  # sampling rate, Hz, must be integer
duration = .10               # in seconds, may be float                    # sine
frequency, Hz, may be float
f = 20                    # sine frequency, Hz, may be float
waves=[]                    # array of wavesa= .0002
a=0#.0002                   #amp/volume range [0.0, 1.0] for 4096
t = np.linspace(0, duration, fs)   # used in plot
benchinc=0
count=0
freq=[]
amp=[]
```

```
sumsamples=0
check =0
waves=[]
countTest=0
sumMax=4096

p = pyaudio.PyAudio()

stream = p.open(format=pyaudio.paFloat32,
                channels=1,
                rate=fs,
                output=True)
cap=cv2.VideoCapture(0)

def singlescreen():
    import cv2
    from PIL import Image
    from resizeimage import resizeimage
    import pyaudio
    import numpy as np
    import time
    import matplotlib.pyplot as plt
    import math
    import scipy
    import pylab
    cap=cv2.VideoCapture(0)

    img_counter = 0
    fs = 4410                  # sampling rate, Hz, must be integer
    duration = .1              # in seconds, may be float                   # sine
frequency, Hz, may be float
    f = 20.0                   # sine frequency, Hz, may be float
    waves=[]                   # array of wavesa= .0002
    a=.0002                   #amp/volume range [0.0, 1.0] for 4096
    t = np.linspace(0, duration, fs)   # used in plot
    benchinc=0
    count=0
    freq=[]
    amp=[]
    sumsamples=0
    check =0
    waves=[]
    countTest=1
```

```
        while countTest <= 10:
            sumMax = 4096
            freq=[f]*sumMax
            amp=[a]*sumMax
            t1=[t]*sumMax
            ret, frame = cap.read()
            gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
            new_img = gray.resize((64,64))
            for inc in range (benchinc, sumMax):
                f=freq[inc]
                a=amp[inc]
                samples = a*np.sin(2. * np.pi * f * t)
                sumsamples = samples + sumsamples
                #print "Thread 1 sumsample ", sumsamples
            stream.write(sumsamples)
            countTest=countTest+1
            print "Thread 1 of Resolution: ", sumMax
        stream.close()
        p.terminate()
        cap.release()


    def singlescreen2():
        import cv2
        from PIL import Image
        from resizeimage import resizeimage
        import pyaudio
        import numpy as np
        import time
        import matplotlib.pyplot as plt
        import math
        import scipy
        import pylab
        cap=cv2.VideoCapture(0)

        img_counter = 0
        fs = 4410                    # sampling rate, Hz, must be integer
        duration = .1                # in seconds, may be float                    # sine
frequency, Hz, may be float
        f = 20.0                     # sine frequency, Hz, may be float
        waves=[]                      # array of wavesa= .0002
        a=.0002                     #amp/volume range [0.0, 1.0] for 4096
        t = np.linspace(0, duration, fs)   # used in plot
        benchinc=0
        count=0
        freq=[]
        amp=[]
        sumsamples=0
        check =0
        waves=[]
        countTest=1

        while countTest <= 10:
            sumMax = 64*64
            freq=[f]*sumMax
            amp=[a]*sumMax
            t1=[t]*sumMax
            ret, frame = cap.read()
            gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
            new_img = gray.resize((64,64))
            for inc in range (benchinc, sumMax):
                f=freq[inc]
                a=amp[inc]
                samples = a*np.sin(2. * np.pi * f * t)
                sumsamples = samples + sumsamples
            stream.write(sumsamples)
            countTest=countTest+1
            print "Thread 2 of Resolution: ", sumMax
        stream.close()
        p.terminate()
        cap.release()

    def main():
        global k, lock

        lock= threading.Lock()

        k=0
        ScreenTesting=threading.Thread( target=singlescreen, name =
"Screen_Testing" )
        ScreenTesting.start()
        print ('Start of Thread 1')
        ScreenTesting2=threading.Thread( target=singlescreen2, name =
"Screen_Testing2" )
        ScreenTesting2.start()
        print ('Start of Thread 2')

    if (__name__=="__main__"):
```

```
    main()
```

Grayscale

Code to show the images manipulated at 12 grayscale levels

```
import cv2
from PIL import Image
from resizeimage import resizeimage
import pyaudio
import numpy as np
import time
import matplotlib.pyplot as plt
import math
import scipy
import pylab
import gc
from scipy.misc import imsave


Amp=[0]*4096
Amp2=[0]*4096
imgValue = Image.open("Building.png")
gray = imgValue.load()

xMax = (64)
yMax = (64)
x=0
y=0
count=0
for x in range (x,xMax):
    for y in range (y,yMax):
        a= gray[x,y]
        Amp[count]=a
        if a <= 21:
            a=0
        elif a <= 43:
            a=21
        elif  a <= 65:
            a=43
        elif a <= 87:
            a=65
        elif a <= 109:
            a=87
        elif a <= 131:
            a=109
        elif  a <= 153:
            a=131
        elif  a <= 175:
            a=153
        elif a <= 197:
            a=175
        elif a <= 219:
            a=197
        elif a<= 241:
            a=219
        else:#if a >241& a <= 255:
            a=241
        Amp2[count]=a
        count=count+1
        if y==63:
            y=0
img = Image.new('L',(64,64),color=None)
img.putdata(Amp2)
img.save('Building1.png')
```

Wave Library

Code to show the implement the Wave library

```
import cv2
from PIL import Image
from resizeimage import resizeimage
import pyaudio
import numpy as np
import time
import matplotlib.pyplot as plt
import math
import scipy
import pylab
from numpy.random import randn

img_counter = 0
p = pyaudio.PyAudio()
fs = 44100                  # sampling rate, Hz, must be integer
```

```
        duration = .10                # in seconds, may be float                    # sine
frequency, Hz, may be float
        f = 20                      # sine frequency, Hz, may be float
        waves=[]                      # array of wavesa= .0002
        a=0#.0002                     #amp/volume range [0.0, 1.0] for 4096
        t = np.linspace(0, duration, fs)   # used in plot
        benchinc=0
        count=0
        freq=[]
        amp=[]
        sumsamples=0
        check =0
        waves=[]
        countTest=0


        #**********A**********************
        startA=time.time()
        # open pyaudio.PyAudio()
        stream = p.open(format=pyaudio.paFloat32,
                        channels=1,
                        rate=fs,
                        output=True)
        #endB=time.time()
        #**********B**********************
        #access webcam
        #startB=time.time()
        cap=cv2.VideoCapture(0)
        endC=time.time()

        startTest=time.time()
        while countTest <= 50:
        #****Benchmarking****************
        #*******************************

           sumMax = 64*64

           # access the correct values for the soundwave (freq & amp)
           for count in range (benchinc, sumMax):
              freq.append(f)
              amp.append(a)
              f=f+4
              a=a+0.0002
        #*******************************
```

```
#*********************************

#**********C**********************
    #capture an image
    startC=time.time()
    ret, frame = cap.read()
    #endD=time.time()
#**********D**********************
    #convert image to grayscale
    #startD=time.time()
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    #endE=time.time()
#**********E*********************
    #resize image
    #startE=time.time()
    new_img = gray.resize((64,64))
    #endF=time.time()

#**********F*********************
    #calculating soundwave
    #startF=time.time()
    for inc in range (benchinc, sumMax):
        #create sinewaves for each pin
        f=freq[inc]
        a=amp[inc]
        w = 2. * np.pi * f
        samples = a*np.sin(w * t)
       # endG=time.time()
#**********G**********************
       # startG=time.time()
        #sum each pin wave into a single wave
        sumsamples = samples + sumsamples
       # endH=time.time()
#**********H**********************
   # startH=time.time()
    stream.write(sumsamples)
    #endI=time.time()

    countTest=countTest+1
#**********I*********************

#****Benchmarking*****************
#*********************************
##    AtoB=endB-startA
##  BtoC=endC-startB
##  CtoD=endD-startC
##  DtoE=endE-startD
##  EtoF=endF-startE
##  FtoG=endG-startF
##  GtoH=endH-startG
##  FtoH=endH-startF
##  HtoI=endI-startH

##  AtoC=endC-startA
##  CtoI=endI-startC
##  print AtoC
##  print CtoI

endTest=time.time()

test=endTest-startTest
print test

#*********************************
#*********************************
stream.close()
p.terminate()
cap.release()
```

Frame Rate Calculation

Code to show the calculate time savings and frame rate

```
import cv2
from PIL import Image
from resizeimage import resizeimage
import pyaudio
import numpy as np
import math
import scipy
import time

print 'driving'


fs=44100
duration = 0.1
```

```
t = np.linspace(0, duration, fs*duration)
f=20.0
BansalWaves0=[]
BansalWaves1=[]
BansalWaves2=[]
BansalWaves3=[]
BansalWaves4=[]
BansalWaves5=[]
BansalWaves6=[]
BansalWaves7=[]
BansalWaves8=[]
BansalWaves9=[]
BansalWaves10=[]
BansalWaves11=[]

check=1
sumwave=0
wave=0
i=0
j=0
k=0



startCreate=time.time()
p = pyaudio.PyAudio()

stream = p.open(format=pyaudio.paFloat32,
                channels=1,
                rate=fs,
                output=True)

#_______________________CREATE WAVES___________________
for i in range (4096):
   a=0
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves0.append(b)
#print len(BansalWaves0)

for j in range (4096):
   a=21
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves1.append(b)
#print len(BansalWaves1)
```

```
for k in range (4096):
   a=43
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves2.append(b)
#print len(BansalWaves2)

for l in range (4096):
   a=65
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves3.append(b)
#print len(BansalWaves3)

for m in range (4096):
   a=87
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves4.append(b)
#print len(BansalWaves4)

for n in range (4096):
   a=109
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves5.append(b)
#print len(BansalWaves5)

for o in range (4096):
   a=131
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves6.append(b)
#print len(BansalWaves6)

for p in range (4096):
   a=153
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves7.append(b)
#print len(BansalWaves7)

for q in range (4096):
   a=175
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves8.append(b)
#print len(BansalWaves8)

for r in range (4096):
   a=197
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves9.append(b)
#print len(BansalWaves9)

for s in range (4096):
   a=219
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves10.append(b)
#print len(BansalWaves10)

for n in range (4096):
   a=241
   b=a*np.sin(2. * np.pi * f * t)
   BansalWaves11.append(b)
#print len(BansalWaves11)

endCreate=time.time()


#_______________________CHECK IMAGE___________________
startBansal=time.time()
check=0
while check <= 201:
   startvideo=time.time()
   imgValue = Image.open("TinyGS_{}.png".format(check))
   gray = imgValue.load()
   endvideo=time.time()

   xMax = (64)
   yMax = (64)
   x=0
   y=0
   count=0

   for x in range (x,xMax):
       for y in range (y,yMax):
           a= gray[x,y]
           if a <= 21:
               wave=BansalWaves0[count]
           elif a <= 43:
               wave=BansalWaves1[count]
           elif  a <= 65:
               wave=BansalWaves2[count]
           elif a <= 87:
               wave=BansalWaves3[count]
           elif a <= 109:
               wave=BansalWaves4[count]
           elif a <= 131:
               wave=BansalWaves5[count]
           elif  a <= 153:
               wave=BansalWaves6[count]
           elif  a <= 175:
               wave=BansalWaves7[count]
           elif a <= 197:
               wave=BansalWaves8[count]
           elif a <= 219:
               wave=BansalWaves9[count]
           elif a<= 241:
               wave=BansalWaves10[count]
           else:
               wave=BansalWaves11[count]

           count=count+1
           sumwave=wave+sumwave
           if y==63:
               y=0
   stream.write(sumwave)
   check=check+1



endBansal=time.time()
BansalTime=endBansal-startBansal
CreateTime=endCreate-startCreate
VideoTime=endvideo-startvideo
print sumwave
print ('Bansal time', BansalTime)
print ('Create time', CreateTime)
print ('Video Open time', VideoTime)



#______________________ORIGINAL IMAGE__________________
startOriginal=time.time()
check=0
while check <= 201:
   startvideo=time.time()
```

```
imgValue = Image.open("TinyGS_{}.png".format(check))
gray = imgValue.load()
endvideo=time.time()

xMax = (64)
yMax = (64)
x=0
y=0
count=0

for x in range (x,xMax):
    for y in range (y,yMax):
        a= gray[x,y]
        if a <= 21:
            a=0
            wave=a*np.sin(2. * np.pi * f * t)
        elif a <= 43:
            a=21
            wave=a*np.sin(2. * np.pi * f * t)
        elif  a <= 65:
            a=43
            wave=a*np.sin(2. * np.pi * f * t)
        elif a <= 87:
            a=65
            wave=a*np.sin(2. * np.pi * f * t)
        elif a <= 109:
            a=87
            wave=a*np.sin(2. * np.pi * f * t)
        elif a <= 131:
            a=109
            wave=a*np.sin(2. * np.pi * f * t)
        elif  a <= 153:
            a=131
            wave=a*np.sin(2. * np.pi * f * t)
        elif  a <= 175:
            a=153
            wave=a*np.sin(2. * np.pi * f * t)
        elif a <= 197:
            a=175
            wave=a*np.sin(2. * np.pi * f * t)
        elif a <= 219:
            a=197
            wave=a*np.sin(2. * np.pi * f * t)
        elif a<= 241:
                219
            else:
                a=241
                wave=a*np.sin(2. * np.pi * f * t)

            count=count+1
            sumwave=wave+sumwave
            if y==63:
                y=0
    stream.write(sumwave)
    check=check+1

endOriginal=time.time()
OriginalTime=endOriginal-startOriginal
print ('Original time', OriginalTime)

stream.close()
#p.terminate()
#cap.release()
```