Life Cycle Reliability and Safety Engineering
Vol.4 Issue 2 (2015) 40-51

Non-contact and Real-time Dynamic Displacement Monitoring using Smartphone Technologies

Jae-Hong Min[1], Nikolas J. Gelo[2], and Hongki Jo[1*]
[1]*Department of Civil Engineering and Engineering Mechanics,*
[2]*Department of Computer Science, The University of Arizona, Tucson, AZ 85721, USA.*
Email: jaehongmin@email.arizona.edu

Abstract

Many of the available approaches for Structural Health Monitoring (SHM) can benefit from the availability of dynamic displacement measurements. However, current SHM technologies rarely support dynamic displacement monitoring, primarily due to the difficulty in measuring absolute displacements. The newly developed smartphone application in this study allows measuring absolute dynamic displacements in real time using state-of-the-art smartphone technologies, such as high-performance graphics processing unit (GPU), in addition to already powerful CPU and memories, embedded high-speed/resolution camera, and open-source computer vision libraries. A carefully designed color-patterned target and user-adjustable crop filter enable accurate and fast image processing, allowing up to 120Hz frame rate for complete displacement calculation. The performances of the developed smartphone application are experimentally validated, showing comparable results with those of conventional laser displacement sensor.

Key words: structural health monitoring, smartphone technology, computer vision, dynamic displacement monitoring, real time image processing

1. Introduction

Structural displacement or deformation information is particularly important when permanent deformation occurs and is often more accurate than acceleration measurements in lower-frequency ranges. However, current SHM technologies rarely support displacement monitoring, primarily due to the difficulty in measuring absolute displacement; which mostly require fixed reference points. Though the reference-free nature of GPS-based methods conveniently measures absolute displacements, relatively low sampling rates are only available and the associate cost for survey-level dual-frequency GPSs that support sub-centimeter accuracy is still too high for routine use [1~3]. Single-frequency, low-cost GPSs, generally used for navigation purposes, have shown the feasibility for dynamic displacement monitoring [4], however, the technologies of single-frequency GPSs have not been sufficiently advanced yet for practical use in SHM applications.

Computer vision-based methods have received broad interests in measuring static and dynamic displacements of structures due to various merits of the methods. The capability of non-contact, distant measurement of absolute displacements significantly reduces the difficulties in providing stationary reference points; ensuring such reference points is a critical challenge for contact-type displacement sensors [5~8]. Moreover, these vision-based measuring systems are available at relatively low cost, combining a video camera, an optical zoom lens, lighting lamps and a precision target attached to the location of interest on the structures [9].

In most civil engineering applications, computer vision-based measuring systems have been considered particularly interesting for low-frequency vibration measurements. One of the reasons is that amplitudes of the high-frequency displacements are generally smaller than those of low-frequency displacements, causing difficulty in identifying the vibration with the limited resolution of conventional cameras. Another important reason is that the maximum frame rates (fps: frames per second) of the most of conventional video cameras are limited to 30 ~ 60fps at the best resolutions [10, 11]. Though such low frame rates would be sufficient for measuring low-frequency and high-amplitude vibrations of long-period structures, such as high-rise buildings and long-span cable-supported bridges, ensuring higher frame rate is still essential for appropriate monitoring the dynamic behavior of many of small-to-mid scale common civil structures.

One of the issues in these types of dynamic measurements is that anti-aliasing filters are not available for vision-based measuring systems. The only way to minimize such aliasing problem is to increase the frame rate. A high-speed camera allowing up to 2000fps has been investigated for dynamic displacement monitoring [12]. However the practical use of such expensive cameras for civil engineering applications is still in question, because the level of cost and the difficulty of achieving real-time processing can be the main restraint for the success of the technique itself.

Recent advances in smartphone technologies provide various onboard sensing capabilities, including, but not limited to, accelerometer, temperature sensor, GPS, altimeter, gyroscope, and etc. Particularly, embedded cameras show great advances in providing higher-resolution and higher-speed video features, often better than many conventional camcorders. Moreover, their powerful processors and memories allow for onboard processing capabilities, eliminating the need for additional computers to perform extensive image processing. However, such advanced vision and embedded processing capabilities of smartphones have not been effectively utilized for dynamic displacement monitoring applications yet.

This study investigates the feasibilities of such smartphones for dynamic displacement monitoring of civil structures. A new smartphone application is developed for real-time measurement and processing of dynamic displacements using the rear camera of the iPhone 6 Plus and OpenGL (Open Graphic Library) in the iOS environment, which enables easy and low-cost monitoring of absolute dynamic displacements. For real-time applications, a user-selectable crop filter is implemented to optimize the image size and minimize associated processing time, considering the size and distance of target, which allow up to 120fps. To clearly discriminate from the background, carefully designed targets with unique color patterns are used. To make this smartphone application be more widely embraced for practical uses, several useful features, such as autonomous detection of target centroid, email transmission of measured data, user-adjustable color hue range, high-precision time stamps, and automated onboard calibration of measured data are implemented. Following a description of the software developed herein, the performances of the smartphone application are experimentally validated with shake table tests under both indoor and outdoor conditions.

2. State-of-the-art Smartphone Technologies

Recent advances in smartphones technologies provide many attractive features in addition to its original function for mobile telecommunication (see Table 1). Various types of sensors embedded in the smartphones allow the devices to be used for various purposes; for example, motion sensing for game software, proximity sensing for screen power saving, GPS for navigation, fingerprint sensor for device security, and so on. Most of all, what makes the smartphone actually be smart is its onboard computing capability. The speed and size of their microprocessors

Table 1. Hardware specifications of example smartphones

	iPhone 5 [13]	iPhone 5S [14]	iPhone 6/Plus [15]	Galaxy S5 [16]	LG G3 [17]
Release Date	9/12/2012	9/20/2013	9/19/2014	4/11/2014	5/28/2014
CPU	Apple A6:32-bit 1.3 GHz dual core	Apple A7:64-bit 1.3 GHz dual-core	Apple A8:64-bit 1.4 GHz dual-core	1.9 GHz quad-core Cortex-A15 1.3 GHz quad-core Cortex-A7	2.5 GHz quad-core Krait 400
Memory	1 GB LPDDR2-1066 RAM	1 GB LPDDR3 RAM	1 GB LPDDR3 RAM	2 GB LPDDR3 RAM	3 GB (for 32 GB model)
Other Sensors	- Gyroscope - Accelerometer - Digital compass - Proximity sensor - Ambient light sensor	- Gyroscope - Accelerometer - Digital compass - Proximity sensor - Ambient light sensor	- Gyroscope - Accelerometer - Digital compass - Proximity sensor - Ambient light sensor - Barometer	- Gyroscope - Accelerometer - Digital compass - Proximity sensor - Ambient light sensor - Barometer - Infrared (IR) LED sensor	

Table 2. Camera performances of example smartphones

	iPhone 5 [13]	iPhone 5S [14]	iPhone 6/Plus [15]	Galaxy S5 [16]	LG G3 [17]
Rear Camera	- 8 MP iSight camera with 1.5μ pixels - Autofocus - f/2.4 Aperture - True tone flash - Hybrid IR filter	- 8 MP iSight camera with 1.5μ pixels - Autofocus - f/2.4 Aperture - True tone flash - Hybrid IR filter	- 8 MP iSight camera with 1.5μ pixels - Autofocus with focus pixels - f/2.2 Aperture - True tone flash - Hybrid IR filter	- 16 MP - Autofucos - f/2.2 Aperture - LED flash	- 13 MP - Hybrid Infrared Autofocus - f/2.4 Faperture - Dual tone LED flash
HD Rear Camera Capture	- 720p@30/60fps - 1080p@30fps	- 720p@30/60/120 fps - 1080p@30fps	- 720p@30/60/120/240fps - 1080p@30/60fps	- 1080p@30/60fps - 4K@30fps	- 720p@60fps - 1080p@30fps - 4K@30fps
GPU	- PowerVR SGX543MP3	- PowerVR G6430 (four cluster @ 450 MHz)	- PowerVR Series 6 GX6450 (4 clusters)	- ARM Mali T628MP6	- Adreno 330

and memories are sufficiently comparable with decent laptop computers.

The particular focus of this study, related with smartphones' performance, is in the cameras (see Table 2). For example, the latest version of Apple's iPhone (i.e. iPhone 6/Plus) supports up to 240 fps at 720p resolution and 120 fps at 1080p (HD) resolution. Samsung's Galaxy S5 and LG's G3 support up to 4K (UHD) resolution, but with slower frame rate (30 fps). Moreover, the integrated graphics processing unit (GPU) can rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display.

For this study, Apple's iPhone is selected as its higher frame rate (up to 240 fps) is a more reasonable option for reducing the aliasing issues in dynamic displacement measurements. Moreover, Apple iOS mobile operating system is more appropriate for the development of this application because within a year of its release about 91% of Apple iOS users were running iOS 7, Apple's latest release. Compare this with Android where five of the latest releases each hold about a 10% share of installations [18]. This operating system fragmentation often leads to developers having to support a variety of new and deprecated APIs to accomplish the same tasks. Although Google has taken steps to minimize the effects of operating system fragmentation [19], avoiding compatibility issues was avoidable.

The issues of fragmentation for Android devices are prevalent not only in software but also in the devices' hardware. For example, when comparing the pictures taken with 4 different Android devices under the same conditions, the differences in color and brightness are immediately clear [20]. The core functionality of the application depends on computer vision analysis and introducing variability into this process would make the initial development of the application more difficult.

Developing the app with consistent performance was established as a goal early on in the development process. By eliminating variables in the devices on which the application ran, the first version of the application could be more easily be developed. Developing the app so that it would perform consistently on devices was necessary for the first version. Developing for Android is a goal for future iterations.

3. Object Tracking using Computer Vision

To track the objects using computer vision, i) the objects of interest should be detected in video sequence first, then ii) classify the detected objects, and iii) track the movements of identified targets. Though this is a straightforward process, practical implementation of this object tracking is not that simple, because of possible variability of color, shape, and texture of the objects for each video frame [21]. Many different methods for object tracking using computer vision have been developed for past few decades, which can generally be divided into three categories; i) region-based methods, ii) contour/edge-based methods, and iii) point-based methods.

Region-based methods basically utilize the color features of the objects, as all objects can be represented by their specific color distribution. This method may not be efficient when several objects move at the same time in a video frame, because of the possible occlusion among the moving objects. However, this approach is well known to be computationally much faster than other algorithms and effective for tracking fast moving objects.Despite the shape information possibly being distorted by fast movement, the objects can be traced due to their color [22, 23].

Shape of an object can be characterized by its contour/edge/outline. Edge-based tracking can be represented in active contour model [24] and is known to be efficient in tracking moving objects, because a number of points distributed along the edge of the object can provide higher probability of accurate tracking. However, the performance of this edge-based approach is very sensitive to the initial shape of the object and not so good for 3D movement tracking.

Specific points can also be the characteristic feature of the object. This feature points should be sufficiently different from its neighbor. Usually, such feature points are extracted and detailed by descriptors that can recognize the object [25, 26]. Then the extracted feature points are matched from sequential frames to track the object. The success of this approach is dependent on how well the feature points of the object are extracted and matched, because many other similar points can be regarded as the feature points in the next frame, causing uncertainty.

If desktop or laptop computers are used for processing the images, real-time operation of the vision-based measuring systems may not be an issue. For example, a simple test shows total processing time per each frame to measure the dynamic movement of a target is just about 1~3 millisecond (ms) using a region-based object tracking method with a desktop computer that has Intel's i7-4770 CPU (quad core 64bit 3.40Hz), Nvidia GeForce GTX 650 Ti graphic card (GDDR5 128-bit, OpenGL 4.3, Memory Bandwidth 86.4 GB/sec), and 16GB RAM. Though the hardware capabilities of recent smartphones are comparable to some laptop computers, the performance is less than conventional PCs. Considering the minimization of the processing time as the key factor in realizing higher frame rate for dynamic displacement monitoring using smartphones, an appropriate object tracking method should be selected.

4. OpenCV vs. OpenGL (GPUImage)

The OpenCV library [27] is a popular open-source library for computer vision applications with thousands of optimized algorithms. The iOS-compatible release is written in C++, which can be compiled alongside Objective-C (the language used to write iOS apps), and contains many classes for easily integrating the device's camera with the OpenCV library. Despite OpenCV being a powerful computer vision library, its main drawback for usage in developing a real-time computer vision application is its lack of usage of the iPhone's GPU. Instead, all image processing is performed on the iPhone's CPU. Simple image processing can be accomplished on the CPU in real-time such as adjusting the RGB values or inverting an image. But more complex algorithms like corner detection are more expensive and cannot be performed in real-time in the CPU side of iPhone.

In order to utilize the GPU to perform complex computer vision analysis, the GPUImage library [28] was used in this study. The GPUImage applies GPU-accelerated filters and other effects to images. Filters used with GPUImage are written using the OpenGL Shading Language, using OpenGL ES 2.0 (iPhone 5 and iPhone 5S) or 3.0 (iPhone 5S and later), and are compiled at runtime. The combination of using these filters and processing them on the GPU allows complex image analysis algorithms to run at much higher speeds. Table 3 shows a comparison of rendering speeds on the CPU and GPU.

Table 3. Rendering speed comparisonGPU vs. CPU (larger fps is better)

Calculation	GPU (fps)	CPU (fps)
Thresholding x 1	60.00	4.21
Thresholding x 2	33.63	2.36
Thresholding x 3	1.45	0.05

Source: http://nshipster.com/gpuimage/

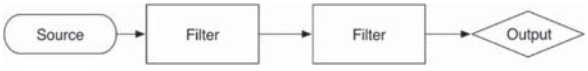

Figure 1. Example image rendering process using GPUImage Source: http://nshipster.com/gpuimage/

By specifically targeting the hardware of the iPhone in this study, GPUImage allows for faster image processing compared to that of the CPU. To accomplish image rendering on the GPU, GPUImage is built upon an OpenGL rendering pipeline that takes a source image, passes it through a series of filters (or OpenGL shaders), and produces an output (see Figure 1). The simple integration of custom OpenGL shaders

that GPUImage creates allows for easy access to the often complicated OpenGL rendering pipeline.

5. Samrtphone App Development for Real-time Dynamic Displacement Monitoring

5.1 GPUImage library

The iOS platform and the GPUImage library were established as the starting point for the development of the application. Integrating the GPUImage library was accomplished using the Cocoapods dependency manager. After the base application was configured, all future modifications to the application were managed using a private GitHub repository.

Various object tracking methods have been explored under the iOS environment to effectively detect the target of interest and track its centroid movements of each frame front the camera. This started with examining the filters included in the GPUImage library.

Each filter in GPUImage uses OpenGL shaders to process the incoming image and produces a resulting image that can be extracted from the GPU and presented to the user. Color adjustment filters, such like RGB Levels, Hue, and Luminance Threshold filters, were studied to learn how OpenGL can manipulate the colors of an incoming frame from the camera. The color adjustment filters were able to run very quickly because of their heavy reliance on the GPU for manipulating the image. Additionally, image processing filters such as the *Harris Corner Detection*, *Sobel Edge Detection*, and *Hough Transform Line Detection* filters showed how computer vision algorithms could be implemented with OpenGL. The main difference between the color adjustment and image processing filters was the reliance of the image processing filters on the CPU to get access to the raw pixel data from the GPU and the rasterization across each pixel in the image per frame processed. This bottleneck was necessary so that data like the number of corners and the locations of these corners in the image could be determined. By simply running the image processing filters with just their OpenGL shaders and not doing any processing on the CPU, the filters were able to run very quickly. However, introducing the reading of raw pixel data from the GPU onto the CPU led to significant drops in the filters' performance.

5.2 Crop filter

None of the filters included with the GPUImage library could accurately and efficiently identify targets in a full-sized, 720×1280p, image in real-time. Because of the large amount of pixels that had to be processed in each frame, a crop filter was introduced into the rendering pipeline that could crop an image from 720×1280p to 720×100p. The result was a significant drop in processing time per frame for the image processing filters.

5.3 Target design

The color adjustment filters could successfully identify a target (e.g. red-colored targets) in an image by filtering the colors in the incoming image and produce a binary image. The binary image indicated all areas that passed the filter by highlighting the valid areas as white and all ignored areas were black. Despite the ability to identify a target with a certain color, much of the resulting output from color adjustment filters contained false positives of features in the image that were not apart of the intended target but whose RGB values could pass the filter. Similarly, many of the image processing filters like the Harris Corner Detection filter could identify a square target by its corners but its resulting output would often include highlighted areas that met the threshold for detecting a corner but were not apart of the target.

The best target to identify had to be both unique in its appearance and easy and fast to identify by an OpenGL Shader. While researching for good targets to use for identification, the yellow and black color pattern used in crash test simulations by automobile manufacturers became the best choice (see Figure 2). The pattern could be both easily identified by its color pattern and by the corners and edges of the target.

Identifying the target was best found by examining the hue values of each pixel in the image. While the hue values of a color slightly change with minor variations, the RGB values of colors can change dramatically even with minute changes in their appearance. By

Figure 2. Yellow-black color pattern target on a dummy for car crash test

Source: http://www.carkoon.com/blog/nhtsa-introduces-new-crash-test-dummy-child-safety-seat-evaluation

filtering an image by using a hue range rather than an RGB range, changing the colors that were to be filtered became much easier to change on the fly. The hue scale ranges from 0 to 360 (see Figure 3) and by passing upper and lower hue values to an OpenGL shader, the image could filter out all hues outside of a range. In the OpenGL shader, the value of a pixel's color had to be converted from RGB to hue using a simple algorithm. This was necessary because of the way OpenGL handles colors. For each pixel being processed, OpenGL stores its colors value in RGB. As a result, in order to filter out hue values outside of a given range, the shader must calculate the pixel's hue value.

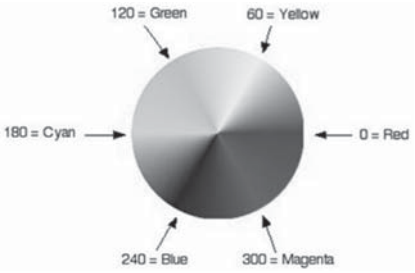

Figure 3. Hue chart of color (Source: http://www.fmwconcepts.com/imagemagick/huemap/)

To identify the center of the color patterned target, a combination of using two hue ranges and the *GPUImage3x3ConvolutionFilter* were required. Two hue ranges were required so that both of the colors in the target could be identified. So a new custom filter was developed by modifying the *GPUImage3x3ConvolutionFilter*, so that the hue values of neighboring pixels in trial-directions could be accessed as shown in the Figure 4. The neighboring pixels of a given pixel were used for identifying the center of the target. For example, as shown in the Figure 5, near the center of the left target, the blue and yellow colors border each other. This pattern of alternating colors (e.g. hue value difference between blue and yellow: 180) can be used to effectively identify the center of the target.

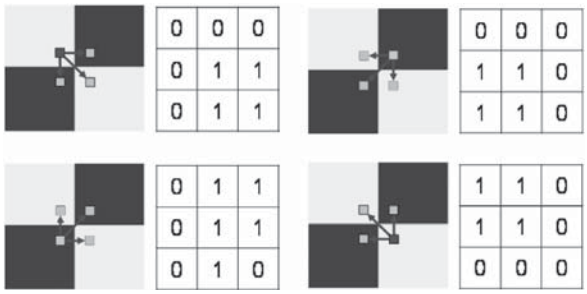

Figure 4.Modified GPUImage3x3 Convolution Filter

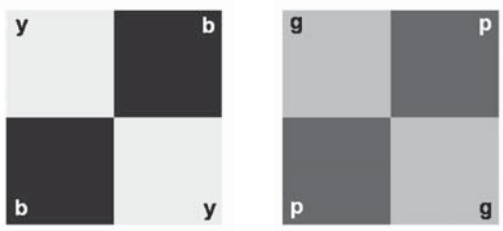

Figure 5.Example pattern of alternating colors: yellow &blue (left), green &purple (right)

5.4 Target centroid calculation

Then, pixels near the center of the target that could be identified using this color pattern were highlighted in the OpenGL shader. Pixels that were not identified by the shader as being pixels near the center of the target were not highlighted. After the shader had processed a frame, the CPU would load the processed image into memory and find all of the pixels that were highlighted. The 2D coordinates of each highlighted pixel were averaged to find the center of the target. The coordinate of the center of the target would be recorded by the application for the use of calculating the displacement of the target at any given frame.

5.5 Real-time displacement calculation and onboard calibration

To calculate real displacement values, two of these targets were required. The second target used the same pattern but with different colors as shown in the Figure 5 (right). The colors used in the second target had similar characteristics to those in the first target in that the level of contrast between the two colors (i.e. green & purple) was very high. By using high contrast colors, the two colors in each target could easily be identified and the risk of the two colors blending together was reduced. By knowing the distance between the two targets in the frame and keeping the distance fixed, the real displace ment of the two targets can be calculated. With GPUImage, the custom OpenGL shader for finding the center of the two targets, and the ability to calculate real displacement of the target in the camera frame, the core functionality of the application was complete.

5.6 Real-time display of processed displacement

Displaying the calculated information on the screen to the user required Apple's Core Animation library and the open source Core Plot library. Core Animation was used to display three dots on the screen that would show where the centers of the targets were. The first two dots were located at the

center of each colored target and a third dot was placed in between the two targets to indicate the center of the entire pattern (see the top of Figure 6).

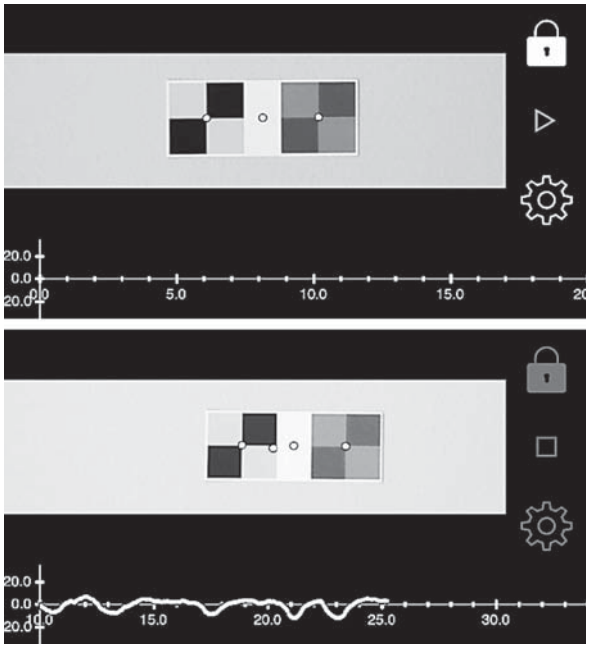

Figure 6.Screen shot of the developed iOS app: before running the app (top), real-time display of measured displacement (bottom)

Once the application is started, a forth dot is appeared where the third dot was initially located, which is fixed. Then actual displacement is calculated with respect to the fourth dot (i.e. distance between the third and fourth dots), and is displayed at the bottom of the application screen in real time as shown in the bottom of Figure 6.

5.7 User configurable setting parameters

With each of these components added to the application, many parameters were introduced to control the functionality of each component. A settings feature was introduced into the application that managed how each component should perform. The settings were separated into three sections: Camera, Filter, and Graph. The options under each section are listed in the Table 4.

5.8 Setting parameter optimization

While the application could identify the target in the camera frame and calculate its displacement at each frame, its performance was slightly above the target of being below 10 milliseconds on the iPhone 6 Plus. The slowest portions of the application were identified through profiling the application using the

Table 4. Setting parameters

Camera Settings	
Frame Rate	Manually adjust the camera's frame rate 30, 60, 120 and 240 fps.
Crop Size (width & height)	Change the size of the crop filter Width (px):100/200/400/720/1280 Height (px):100/200/400/720
Auto-Focus Range	Change the autofocus range of the camera between near, far and none.
Show Camera View	Whether or not the feed from the camera should be shown on the screen.
Show Benchmark	Whether or not the current and average processing time from capturing to filtering, to displaying the image should be calculated and shown on the screen.

Filter Settings	
Show Filtered View	Whether the resulting output from the OpenGL shader should be shown in place of the raw camera view.
Pixel Search Distance	The distance between neighboring pixels used in the OpenGL shader.
Set Filter Colors	Select the hue ranges of the two colors used in each target.

Graph Settings	
Show Graph View	Whether the graph should be updated with displacement calcultions from each frame.
Target Distance (cm)	The distance (in centimeters) of the two colored targets in the camera frame.
Show X & Y Displacement	Whether the X, Y, or both displacement lines should be calculated and shown on the graph.

Instruments software included with Apple's Xcode. By identifying potential areas of slow performance and memory leaks the application's processing time from capture to display was drastically increased.

- The biggest improvement in processing time was achieved simply by not displaying the camera feed on the screen. This resulted in about a 2 to 3 millisecond drop in processing time.

- Another improvement in processing time was achieved by running all camera related operations on a separate thread and running all UI (user interface) operations on the main thread.

However, much of the slowest portions of the application could not be resolved so easily. Portions of the application could not be removed as they were integral to the application. Reducing the processing time in components like reading the raw pixel data to the CPU from the GPU, calculating the center of each target on the CPU by averaging the coordinates of all found pixels, and performing displacement calculations were difficult due to their lack of complexity and necessity to the functionality of the application.

After resolving performance issues in the application, the application's processing time was reduced to below 7 milliseconds on the iPhone 6 Plus. Due to the constraints of the hardware on the iPhone 5, such results could not be achieved. The Table 5 shows a breakdown of the processing time (average from several tests) for the application.

Table 5.Processing time breakdown (with 720×100p @ 120 fps)

Step	iPhone 5	iPhone 6 Plus
GPUImage (no display)	1.10 ms	1.10 ms
GPUImage + Crop Filter (no display)	1.34 ms	1.42 ms
GPUImage + Crop Filter + Custom Filter (no display)	35.2 ms	6.30 ms
Total w/ Displaying	37.1 ms	11.25 ms
Total w/o Displaying	35.2 ms	6.30 ms

5.9 Email transmission of measured data

Additional features were developed to allow the user to view the graph full screen after a recording session had ended and to export the collected data to a CSV file that can be emailed to a personal account for further analysis. These changes made the application suitable for testing and practical use.

Figure 7 shows the simplified block diagram of the developed iOS application in this study.

6. Experimental Validations

In order to evaluate the performances of the developed iOS app, including sampling time accuracy and the ability to track the dynamic movements of targets, a series of laboratory-scale tests have been carried out using a shaking table with single-frequency and multi-frequency sinusoidal motions.

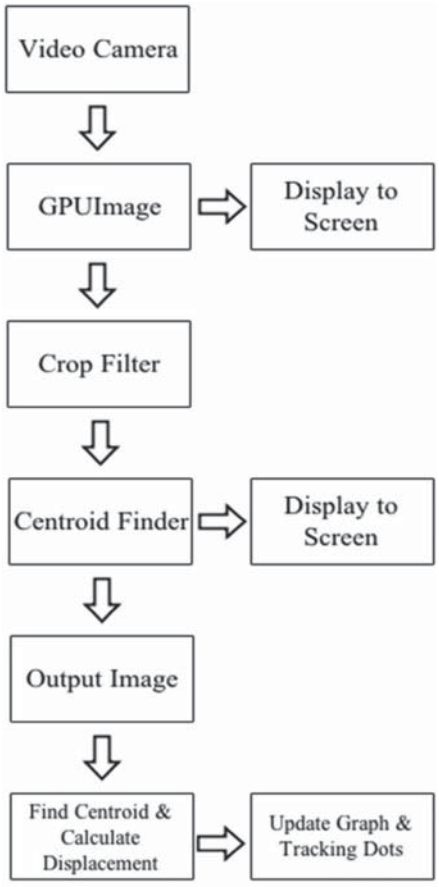

Figure 7. Software block diagram of developed iOS app

6.1 Experiment setup

An APS Dynamics' APS 400 Electro-Seis shaker was used for the evaluation tests (see Figure 8). The input excitation of the shaker was amplified by the APS 145 amplifier. To compare the performance of the developed iOS App with that of a conventional displacement sensor, a laser displacement sensor (KEYENCE, IL-100, 4-μm resolution) was used as a reference. The analog voltage outputs from the laser sensor were measured by the National Instruments' NI-9234 ADC module (24-bit

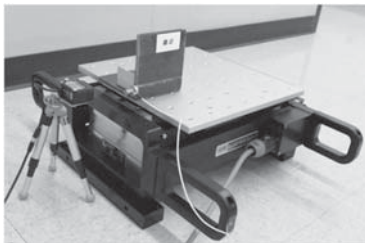

Figure 8. Shaking table test setup

Delta-sigma ADC) with CompactDAQ chassis (cDAQ-9178 model). At the same time, NI-9269 voltage output module, which was housed in the same CompactDAQ, was used to generate the excitation signals for the shaker.

To overcome the limited resolution of the iPhone camera for long-distance and small-target measurements, optical zoom lenses were used in conjunction with the iPhone (see Figure 9). With commercially available low-cost zoom lenses,

Figure 9.12× (left) and 50× (right) optical zoom lens designed for smartphones

precisely designed smartphone cover cases that allow easy connection of the lens to the phone come with the package. 12× and 50× zoom lens were considered initially, but the 50× lens only was used for this tests.

6.2 Sampling Time Accuracy

Consistency of the sampling rate or sampling time is important to ensure the quality of dynamic vibration measurements. Even when conventional analog-to-digital converters are used, the time intervals between adjacent samples are not always consistent [29]. Particularly, because this kind of computer vision-based measuring systems handle extensive image processing, making sure the consistency of the sampling time is important.

Figure 10 shows the example record of the sampling time for 60fps and 120fps cases (720×100p crop filter used for both). The case with 60fps (dotted line) shows very consistent sampling time of 16.67 milliseconds over entire measurements. However, when 120fps (solid line) was used, little inconsistencies are observed in the beginning of the measurements for couples of samples, of which phenomenon is attributed by the dropped samples (see the bottom of Figure 10). To achieve 120fps, all the image processing required to get the displacement information should be done within

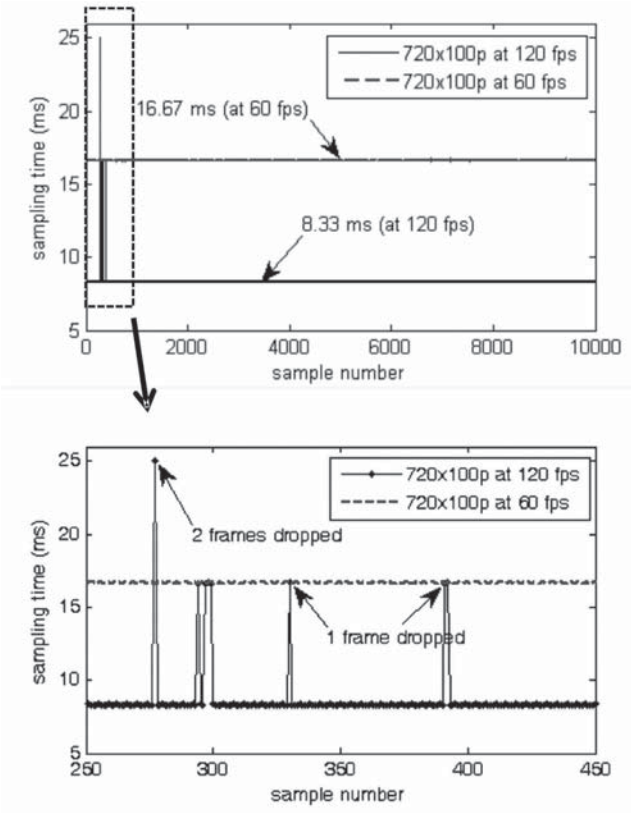

Figure 10.Sampling time accuracy (bottom: zoomed around sample # 350)

8.33ms for each frame. If the processing takes more than 8.33ms, then the software automatically drops the corresponding sample out, to not cause any delay or interference to following samples. Because the case of 60fps ensures sufficient time for processing, such dropped samples were not observed in this test.

6.3 Shake Table Tests

For the initial shake table tests indoors, the iPhone with the zoom lens was placed 3.0m away from the target attached on the shake table. The target size was 1.0×2.5cm, which was composed of two rectangular alternating color patterns having 1.5cm center distance between them. 720×100p crop filter was used to track the target in a horizontal direction in an optimized way. The distance between the two color patterns (i.e. 1.5cm) was occupied by about 300~400 pixels, corresponding resolution for this particular set up could be estimated about 0.0375~0.05mm; actual size of each pixel was autonomously calibrated in the software and used for displacement calculation.

Figure 11 shows the shake table test results for the 1Hz, 10Hz, 20Hz sinusoidal excitations, and multi-

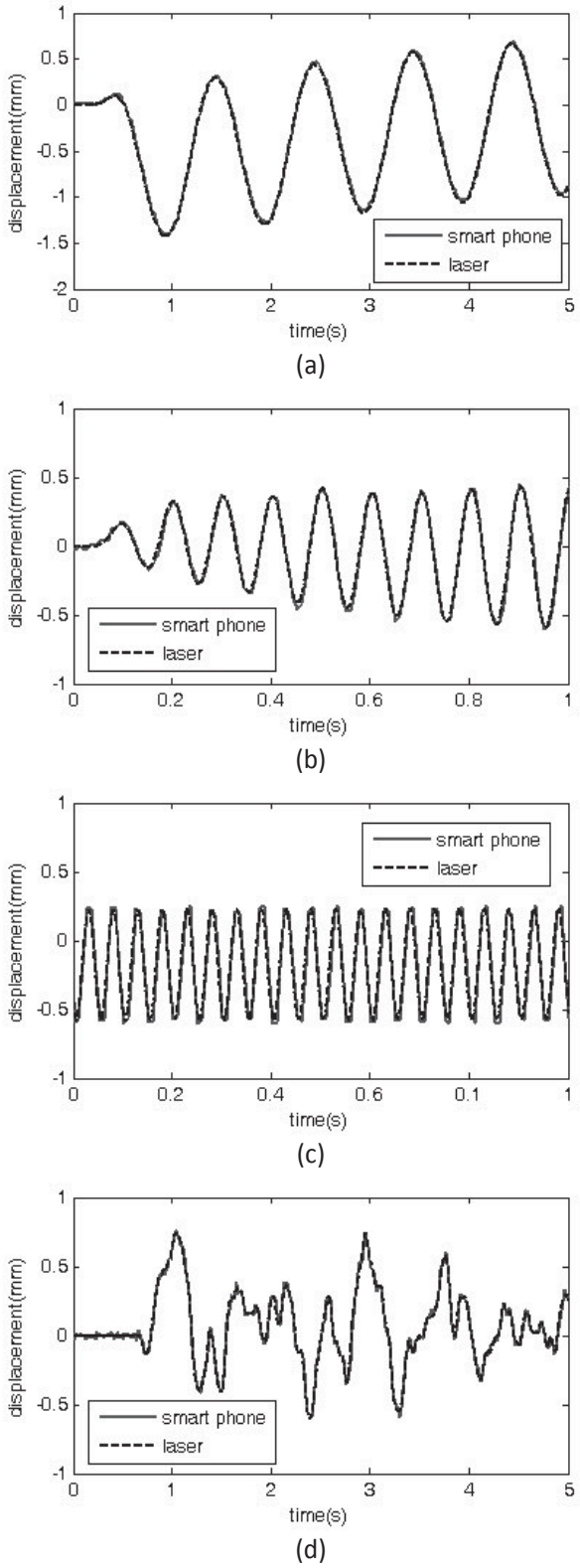

Figure 11.Indoor shake-table test results (3m from target): (a) 1Hz sine at 120fps, (b) 10Hz sine at 120fps, (c) 20Hz sine at 120fps, and (d) multi-tone sine signal at 120fps.

tone excitation composed of 1~20Hz (0.2Hz step) sinusoidal signals. Vibration levels were kept below 2 mm amplitude (peak to peak), and 120 fps was used in this test. As shown in the Figure 11, the dynamic displacements measured by the iPhone 6 Plus with the developed iOS app (solid line in the Figure) agree very well with those by the laser displacement sensor (dotted line in the Figure).

Then, the shake table set ups were moved out for outdoor testing with longer target distance. The shake table set ups were placed in the outdoor hallway of the civil engineering building at the University of Arizona, of which hallway can ensure up to 50m clear line-of-sight (see Figure 12). Target distance from the iPhone camera was 33m and the same zoom lens was used, but with little bigger target (4×10cm target size and 6cm center distance between two color patterns).

Figure 12.Shake-table test setup in the outdoor hallway

Figure 13 shows some example results from the outdoor shaking table tests. The performances of the iPhone with the developed app were not so impressive, compared with indoor tests. Particularly when 120fps was used, substantial high-frequency noises were observed in the measurements by iPhone (solid line in the Figure) as shown in the Figure 13 a) and c), while the results from 60fps were acceptable, successfully resolving millimeter-level displacements. Possible reasons for these high-frequency noises in outdoor tests may be attributed to, but not limited to, possibilities that i) the captured image at 120fps might be exposed to less amount of light as the higher frame rate allows the shorter exposure time, which could change the color properties in the image, ii) the phone might be subjected to unexpected high-frequency vibrations due to wind and/or building vibrations, resulting in such noisy measurements; though it is a very little vibration, its effects on the captured images would be substantial, as the target is located further and further away.

No matter what the reasons for causing such high-frequency noises, possible vibrations of the phone itself should be compensated for the practical use of this approach for dynamic displacement measurements in

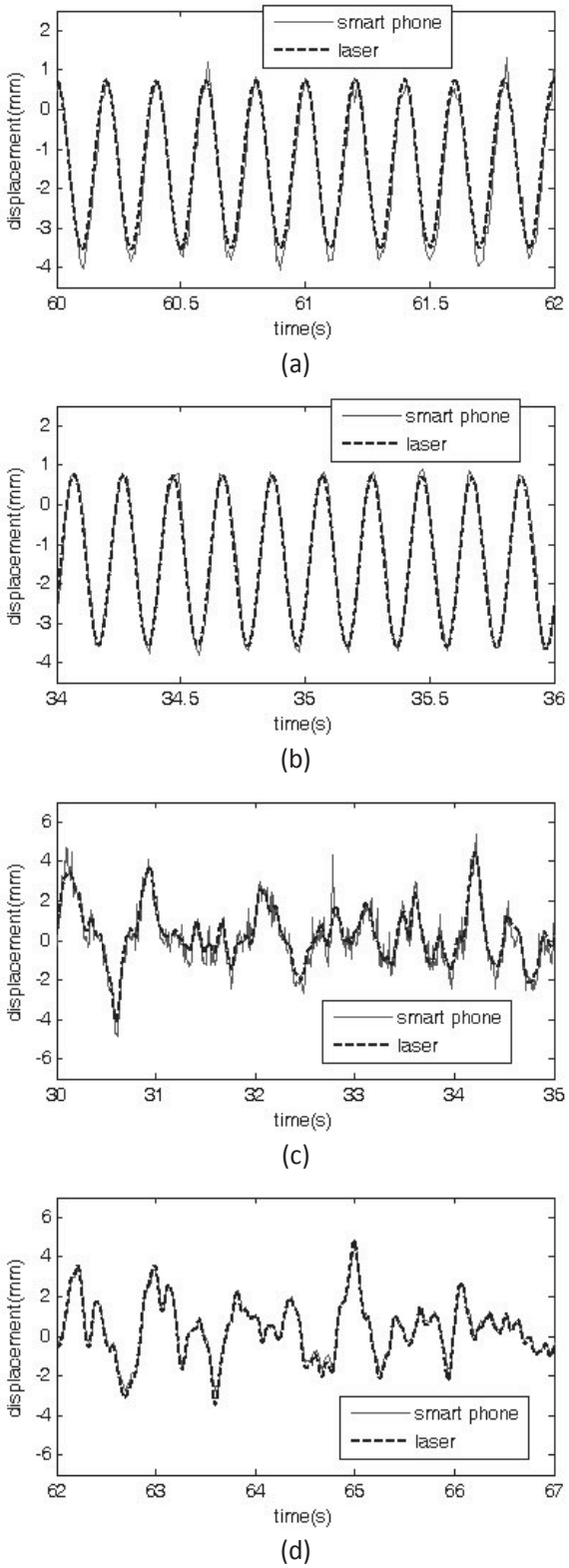

Figure 13.Outdoor shake-table test results (33m from target): (a) 5Hz sine at 120fps, (b) 5Hz sine at 60fps, (c) multi-tone sine at 120fps, and (d) multi-tone sine signalat 60fps.

the field. Other sensors (e.g. accelerometer, gyroscope) embedded in the smartphone (see Table 1) may be utilized for the phone vibration compensation. To ensure sufficient amount of light for outdoor tests, a self-light emitting target (e.g. LED) may be used for future tests. In addition, a low-pass filtering can be implemented in the iOS app to reduce such high-frequency noises.

7. Conclusions

The feasibility of smartphone technologies for real-time dynamic displacement monitoring has been investigated in this study. A new smartphone application was developed under iOS environment for the iPhone. Various methods for moving object tracking have been explored, then, a region/ color-based tracking method was adapted in this study because of its computational efficiency in image processing and robustness in tracking fast moving objects. In order to fully utilize the GPU capabilities of smartphones, the GPUImage library was used in developing the iOS app. A crop filter was implemented for users to compromise between the image size and frame rate without sacrificing accuracy. Onboard calibration of the image pixel size to a given-dimension target was implemented in the developed iOS app. And other various features for controlling camera, filter, and graph settings and email transmission of measured data were also incorporated in this iOS app development. All the functions required for measuring the dynamic movements of the target could successfully be operated in real time, allowing up to 120fps with iPhone 6 Plus. And the performances of the iPhone hardware and the iOS app developed herein were experimentally validated. Although some high-frequency noises were observed from outdoor shake-table tests, the performances of the developed app were comparable to those of a conventional laser displacement sensor, allowing down to sub-millimeter resolutions at 33m distance from the target. The possibilities and limitations of the smartphone (iPhone) and its camera for real-time dynamic displacement monitoring applications have been explored in this study, pointing in the direction of the following research.

References

1. Fujino, Y., Murata, M., Okano, S., and Takeguchi, M. (2000), "Monitoring system of the Akashi Kaikyo Bridge and displacement measurement using GPS," *Proc. of SPIE, nondestructive evaluation of highways, utilities, and pipelines IV*, 229–236.
2. Casciati, F. and Fuggini, C. (2009), "Engineering vibration

monitoring by GPS: long duration records," *Earthquake Engineering and Engineering Vibration*, Vol. 8(3), 459-467.

3. Yi, T.H., Li, H.N., and Gu, M. (2012), "Recent research and applications of GPS based monitoring technology for high-rise structures," *Structural Control and Health Monitoring,* Published online, DOI: 10.1002/stc.1501.
4. Jo, H., Sim, S.H., Tatkowski, A., Spencer Jr., B.F., and Nelson, M.E. (2012), "Feasibility of Displacement Monitoring using Low-cost GPS Receivers", *Structural Control and Health Monitoring*, 20(9), 1240-1254.
5. Malesa M, Szczepanek D, Kujawinska M, Swiercz A, Kolakowski P. (2010), "Monitoring of civil engineering structures using digital image correlation technique", *ICEM 14–14th International conference on experimental mechanics*. Poitiers, France.
6. Caetano, E., Silva, S., Bateira, J., (2011), "A vision system for vibration monitoring of civil engineering structures", *Experimental Techniques,* 74–82.
7. Bell E, Peddle J, Goudreau A. (2012), "Bridge condition assessment using digital image correlation and structural modeling", *IABMAS'12 – Bridge maintenance, safety, management, resilience and sustainability*, Dubrovnik, Croatia, 330–7.
8. Ji, Y.F. and Zhang, O.W. (2012), "A novel image-based approach for structural displacement measurement", *Proc. 6th Int. Conf. Bridge Maintenance, Safety Manag.*, 407–414.
9. Ribeiro, D., Calcada, R., Ferreira, J., and Martins, T. (2014), "Non-contact measurement of the dynamic displacement of railway bridges using an advanced video-based system", *Engineering Structures*, 75, 164-180.
10. Ho, H.N., Lee, J.H., Park, Y.S., & Lee, J.J. (2012), "A Synchronized Multipoint Vision-Based System for Displacement Measurement of Civil Infrastructures", *The Scientific World Journal*.
11. Fukuda, Y., Feng, M., Narita, Y., Kaneko, S. I., & Tanaka, T. (2010), "Vision-based displacement sensor for monitoring dynamic response using robust object search algorithm", *.Sensors, IEEE,* 1928-1931.
12. D'Emilia, G., Razzè, L., & Zappa, E. (2013), "Uncertainty analysis of high frequency image-based vibration measurements", *Measurement*, *46*(8), 2630-2637.
13. "IPhone 5." *Wikipedia*. Wikimedia Foundation, 11 Apr. 2014. Web. 04 Nov. 2014 (http://en.wikipedia.org/wiki/IPhone_5).
14. "IPhone 5S." *Wikipedia*. Wikimedia Foundation, 11 May 2014. Web. 04 Nov. 2014 . (http://en.wikipedia.org/wiki/IPhone_5S).
15. "IPhone 6." *Wikipedia*. Wikimedia Foundation, 11 July 2014. Web. 04 Nov. 2014 (http://en.wikipedia.org/wiki/IPhone_6).
16. "Samsung Galaxy S5." *Wikipedia*. Wikimedia Foundation, 11 June 2014. Web. 04 Nov. 2014 (http://en.wikipedia.org/wiki/Samsung_Galaxy_S5).
17. "LG G3." *Wikipedia*. Wikimedia Foundation, 11 Jan. 2014. Web. 04 Nov. 2014 (http://en.wikipedia.org/wiki/LG_G3).
18. http://appleinsider.com/articles/14/08/22/while-91-of-apple-users-run-ios-7-five-different-versions-of-android-hold-10-share
19. http://www.theguardian.com/technology/2014/aug/22/android-fragmented-developers-opensignal
20. "LG G3 Review." *TechSpot*. N.p., n.d. Web. 07 Nov. 2014 (http://www.techspot.com/review/847-lg-g3/page5.html).
21. Deori, B., and Thounaojam, D.M. (2014), "A SURVEY ON MOVING OBJECT TRACKING IN VIDEO", *International Journal on Information Theory (IJIT)*, 3(3).
22. Li, L., Ranganath, S., Weimin, H., and Sengupta, K. (2005), "Framework for Real-Time Behavior Interpretation From Traffic Video", *IEEE Tran. On Intelligen Transportation Systems*, 6(1), 43-53.
23. Kumar, P., Weimin, H., Gu, I.U., and Tian, Q. (2004), "Statistical Modeling of Complex Backgrounds for Foreground Object Detection", *IEEE Trans. On Image Processing*, 13(11), 43-53.
24. Serby, D., Meier, E.K., and Gool, L.V., (2004), "Probabilistic Object Tracking Using Multiple Features", *IEEE Proc. of International Conf on Pattern Recognition Intelligent Transportation Systems*, 6, 43-53.
25. Zivkovi, Z. (2004), "Improving the selection of feature points for tracking", *Pattern Analysis and Applications*, 7(2).
26. Lou, J., Tan, T., Hu, W., Yang, H., and Maybank, S.J. (2005), "3D Model-Based Vehicle Tracking", *IEEE Trans. on Image Processing*, 14, 1561-1569.
27. "OpenCV." Web. 07 Nov. 2014 (http://opencv.org/).
28. "BradLarson/GPUImage." *GitHub*., Web. 04 Nov. 2014 (https://github.com/BradLarson/GPUImage).
29. Nagayama, T., and Spencer, B. F., Jr. (2007). "Structural health monitoring using smart sensors." *Newmark Structural Engineering Laboratory (NSEL) Rep. Series No. 1*,Univ. of Illinois at Urbana-Champaign, Champaign, IL