Multi-Camera Stereoscopic Vision: Technology Review

Erin Sullens, John Miller, Sam Schultz

CS461
Fall 2016
Group 54
Sponsor: Kevin McGrath
Group Name: ImMaculaTe Vision

Abstract

This Technology Review outlines the types of technologies that are available for specific requirements in our project, Multi-Camera Stereoscopic Vision. Each requirement has three possible technologies that can be utilized, and a description of each is provided. After describing research on all options, the technology that will work the best is selected and that will be the option that we use in our project.

Table of Contents

I. Introduction..... 2
II. Technologies 2
1. User Interface of Mobile App..... 2
2. User Interface of Desktop App 3
3. Capture Videos 3
4. Stabilization of Videos 4
5. Cropping videos for similar FOV 4
6. Parsing BLOB 5
7. Correlation of Video Frames 5
8. Viewing 3D video in VR Headset 6
9. Conversion of 2 Videos into 1 3D Video 7
III Conclusion..... 7
IV Bibliography..... 7

I Introduction

In our project, Multi-Camera Stereoscopic Vision, there are nine requirements that each need a specific technology in order to be completed to our client's specifications. In this Technology Review, we cover the different options that are available for each requirement and then decide which one will be the best suited to complete that requirement. The three of us will each take responsibility for three of the nine requirements. John will be responsible for the following: the User Interface for the Mobile App, the User Interface for the Desktop App, and the process of capturing the videos. Erin will be responsible for the following: Stabilization of the videos, Trimming the videos so they have similar FOV, and Parsing the BLOB files from the video cameras. Sam will be responsible for the following: the Correlation of video frames between the two videos, viewing the 3D video in a VR headset, and the Conversion of the two videos into one 3D video.

II Technologies

1. User Interface of Mobile App

a. Options: React Native, Xamarin, Android Studio

b. Goals for use: Create a mobile application capable of running on either an android phone/tablet or a ios phone/tablet. It does not need to be cross platform, but if it does not interfere with the completion of the project it could be. The purpose of the application is to provide a medium for the user to upload, view, and interact with video data captured by the cameras. It will also allow the user to instruct our algorithm to convert compatible videos into a single stereoscopic video. If anything goes wrong during any stage of the process, it will also serve the purpose of giving the user feedback telling them what happened.

c. Criteria being evaluated: Quality of product, resources available, ease of use for the developer, and application appearance.

d. Comparison of technologies:

1. React Native is an open source, cross platform, Javascript based framework developed and maintained by Facebook. It allows the developer to write code in any combination of javascript, objective-c, java, and swift [21].

2. Xamarin is a Microsoft owned company/software that allows the developer to create cross platform applications using C. It requires the developer to create the application in either Xamarin Studio or Visual Studio [22].

3. Ionic is developed and maintained by Driftyco. Ionic is another "web based" mobile application platform that allows the developer to create applications in a familiar environment.

It encourages the developer to use Angular2 along with Ionic to create the application [23].

e. Discussion: All three technologies have fairly strong followings, all have at least a moderate amount of documentation, and are all actively maintained. Xamarin being proprietary (with a paid option) doesn't allow the developer to use their own environment [] which will make the development process a little more intensive. Both React Native and Ionic allow the developer to use any IDE and both allow the developer to use CSS to style the application [21].

f. Best Option: React Native. It is used by the facebook team for their mobile applications (Facebook app, instagram, etc.). It has strong documentation maintained by the Facebook team and a strong community (over 40,000 stars on github [21]). One of our team members is already familiar with React which would decrease the time necessary to learn the technology. With it's ability to create simple components in javascript and any more complex portions in lower level languages, it seems like a great option for the needs of this project.

2. User Interface of Desktop App

a. Options: Electron with React, QT, Xamarin

b. Goals for use: Create a desktop application for Mac OS X, windows, or both. It's other purposes should be the same as the mobile application with the exception of it taking full advantage of the desktop's/laptop's more powerful hardware and larger storage capabilities.

c. Criteria being evaluated: Quality of product, resources available, ease of use for developer, and application appearance.

d. Comparison of technologies:

1. Electron with React: Both are open source technologies, electron being developed and maintained by github [24] and React by Facebook [25]. Both have large followings with electron at 38,000 stars on github [24] and React with almost 54,000 [26]. Electron is a technology that allows the developer to create an application using web technologies and package it into a desktop application. The desktop applications are fully cross platform, are based off of chromium, and are capable of calling outside scripts that make use of lower level code [24]. React is a web framework made for developing user interfaces in web applications [25].

2. QT: A cross platform C++ development tool used to create UIs and applications [27]. QT is developed and maintained by the QT Project and is dual-licensed under commercial and open source licenses with the open source license protecting developers creating other non-profit open source projects [28].

3. Xamarin: Microsoft owned company/software that allows the developer to create native cross platform applications using C [22].

e. Discussion: All three technologies allow the developer to create cross-platform (OS X/windows) applications using a single code base. QT and Xamarin require the use of specific IDEs [30], but seem to offer performance advantages over Electron in complex applications. QT has been used for 20 years [27], Xamarin has over 6 years [22], and Electron has only been around for 3 years [24]. All three technologies have reputable and highly used software build using them. No member of the team has experience with C, but all members have at least experience with C++, and one team member is very familiar with web technology and react. QT requires special QT stylesheets for editing styles [31], Xamarin requires the use of Xamarin.Forms [32] and Electron/React makes use of standard CSS [25].

f. Best Option: Client has shown lack of initial interest in Electron, so QT would is the best option.

3. Capture Videos

a. Options: Garmin Virb XE, GoPro with Altiforce GPS recorder, Garmin Virb Action Cam

b. Goals for use: Use two of the listed options to capture video and metadata about the video (gps, timestamps). The video files and their associated metadata will be used to create a single stereoscopic video.

c. Criteria being evaluated: Availability, cost, resolution, frames per second captured.

d. Comparison of technologies:

1. Garmin Virb XE: Our client is already able to provide two of these cameras making them readily available. They produce video at 60fps and 1080p quality [33] and has built in video

stabilization. This device is already owned by the client and costs 300-400 dollars for each camera [33].

2. GoPro Hero5 Black: This model of GoPro can capture video in 4k at 30fps, 2.7k at 60fps, and 1080p at 120fps [34] and has built in video stabilization. This range of resolutions and frame capture speeds allows a lot of flexibility for users who may experience motion sickness when using a Virtual Reality device displaying video at a lower frame rate [35]. This device costs 400 dollars for each camera [34].

3.GoPro Hero5 Session with Altiforce Sensor: Captures video at 1440 at 30fps or 1080p at 60fps [35]. This device costs 200 dollars for each camera [34] and would have to be combined with an external gps that could mount to the camera like the Altiforce Sensor which would add another $80-100$ to the total cost [36].

e. Discussion: The GoPro Hero5 Black with its ability to capture video at 120fps at 1080p resolution is the superior camera. The Garmin Virb XE is the most readily available device because it is already owned by the client. It can capture frames at what should be an adequate rate, at least for development purposes. The GoPro Hero5 Session with Altiforce Sensor is the cheapest option and can capture video at the same rate/resolution as the Garmin Virb XE, but involves using two separate devices.

f. Best Option: Because of its availability, the Garmin Virb XE is the best choice for the project during development.

4. Stabilization of Videos

a. Options: OpenCV, Microsoft Cognitive Services, Vid.stab

b. Goals for use: The goal of stabilizing the videos is to take a video that is shaky, and stabilize it. The footage taken is from video cameras that are mounted on the front of a truck, so they are going to be shaky on some level. If we don't stabilize the videos, then viewing the 3D video in a VR headset could cause motion sickness, and it would also make the video look cleaner and be more pleasurable to view.

c. Criteria being evaluated: The technology we use for stabilizing the videos needs to be an API that we can utilize that takes in a video, and outputs a stabilized video file.

d. Comparison of technologies:

1. OpenCV has a large library of video stabilization API's that is well suited for the stabilization that we need to accomplish. There are numerous examples of videos that have been stabilized with OpenCV in this source [1], one of which is a road trip, which is exactly the kind of video we need to stabilize.

2. Microsoft Cognitive Services has an API for video stabilization that takes in MP4, MOV, and WMV video file types. The video has to be no more than 100 MB long, which could be a problem since some of the video files we will be handling are much longer [2]. This site [3] has some examples of videos that were stabilized using this API.

3. Vid.stab is an API on Github for video stabilization. It is only supported on a Linux based system, so it would only work on the desktop application and if we went with a Linux only application [4]. Here [5] is an example of a stabilized video that used Vid.stab.

e. Discussion: All three options supply an API for video stabilization, but the technology that would best fit our needs is OpenCV because it supports large video files, is well documented, and supports the operating systems we are planning on using for the applications, both desktop and mobile.

f. Best Option: OpenCV

5. Cropping videos for similar FOV

a. Options: OpenCV, MoviePy, FFmpeg

b. Goals for use: The purpose of trimming the videos so that they have a similar field of view, is so that when the user looks to the edges of the video inside of the VR headset, the edge is a clean line, and the videos overlap in the correct way. The videos are going to need trimming after we stabilize them because it will create a black bar at the top and bottom of the videos. The FOV at the left and right edges of the videos are also going to be slightly different because the video cameras are placed about five feet apart on the front of the truck. So the technology will also have to trim the sides of both videos as well.

c. Criteria being evaluated: The technology we use for this requirement needs to be able to trim the field of view of the videos.

d. Comparison of technologies:

1. OpenCV provides basic editing tools such as cropping, resizing, and rotating images. We would have to edit each frame individually, cropping off a certain amount depending on how much black space is created from the stabilization process [6]. It is cross platform and will work on the desktop and mobile apps.

2. MoviePy is an open source API for editing videos. It is in Python, and works on Windows, Mac and Linux [7]. It can process common video formats. The code is on Github [8].

3. FFmpeg is cross platform and provides an API for manipulating videos and it handles all video types [9]. The most well documented way of cropping videos with FFmpeg is in the form of a command line command, so if this is the technology we use, there has to be a way to use the command line in order to crop the videos.

e. Discussion: OpenCV is probably the best option because it is cross-platform and will work on the desktop and mobile app. FFmpeg would be a good option because of the range of file types that it can handle, but the use of the command line inputs could be problematic for the mobile apps.

f. Best Option: OpenCV

6. Parsing BLOB

a. Options: IBM BLOB Parser, Garmin-connect-export, Python library to parse ANT/Garmin.fit files.

b. Goals for use: The BLOB we obtain from the cameras contains the GPS timestamp information that we need in order to correlate the frames. The technology we choose for this task needs to be able to take the BLOB files as input, parse them, and output a file that we can extract data from in order to correlate the frames of the two videos. Without doing this, we won't be able to convert the two videos into one 3D video.

c. Criteria being evaluated: The technology to parse the BLOB file has to decode the BLOBs into a format that GPS data can be extracted from. It is hard to know whether any of the technologies we have found will work because the BLOB files from the camera are not known at this time.

d. Comparison of technologies:

1. The IBM BLOB parser takes in a BLOB message in the form of a bit stream or message tree and then outputs a new tree or bit stream [10].

2. The Garmin-connect-export parser takes data from a Garmin-connect, which possibly has the same type of GPS data as the camera we are using, since they are both from Garmin. If they are the same file type, then we might be able to use it to parse the BLOB from the camera using it [11].

3. The Python library to parse ANT/Garmin.fit files could work because the BLOB files from the cameras might be .fit files, and therefore could be parsed by this software. Not much is known at this point if this could work [12].

e. Discussion: The Garmin-connect-export parser seems like it has the most potential to work since the Garmin Connect has GPS data, like the camera does. The IBM BLOB parser is the least likely to work, since it has nothing to do with Garmin. The Garmin.fit parser is 4 years old, so it could not work just for that reason. So the technology we will use is the Garmin-connect-export.

f. Best Option: Garmin-connect-export.

7. Correlation of Video Frames

a. Options: GPS data, PIL/SciPy libraries, OpenCV

b. Goals for use: The purpose of this is to validate that the two videos were taken at the same time and location.

c. Criteria being evaluated: The criteria being evaluated is how processing intensive each solution is as well as the difficulty of the approach.

d. Comparison of technologies:

1. In order to validate that the video files are able to be combined into a stereoscopic 3D video file the frames of each video must be associated with each other. There are several

solutions to this problem. The first method to solve this problem is using the gps files associated with each video file. The gps files will contain information about the time and location as well as accelerometer data. Using this data we will be able to validate that both video files were captured at the same time and location. Once the files have been validated they will move to the next processing stage. This solution is our first choice and is the easiest to implement once the BLOB parser is complete.

2. The second way to solve this problem is to use PIL and SciPy python libraries to create a measurement of similarity between the two frames. In order to validate that the two frames were taken of the same object at the same time, a each frame will be compared on a per pixel level. The colors will be stored in an array in order to be processed by a cross correlation function within the SciPy library. The files will be validated and moved onto the next stage of processing if the pixel color arrays are within an acceptable difference margin of each other. This method is more computational heavy due to having to process each frame of the video file by pixel [13].

3. The third way to solve this problem is using OpenCV to process each of the files. OpenCV offers object recognition capabilities which will be used to to identify the objects in each frame. If the objects are considered the same by the OpenCV algorithm then the two files can be assumed they were taken at the same location at the same time. Problems that may occur with this implementation is that we have no way to verify time of the video, however it is unlikely that two different videos taken at different times will contain the same object at the same camera angle. The validation will be based off of this assumption [14].

e. Discussion: Using the gps data from the files will be less processing intensive while also validating all components having to do with time and location of the videos.

f. Best Option: GPS Data

8. Viewing 3D video in VR Headset

a. Options: HTC Dev Api, OpenCV, Direct3D Api

b. Goals for use: The goal is to create a suitable file format for the stereoscopic 3D video to be played on a viewing device.

c. Criteria being evaluated: Difficulty of file type conversions.

d. Comparison of technologies:

1. The first solution to this problem is to use the HTC Dev API. The purpose of using this API is to generate the correct file format for the display device that will be used to play the converted video. The API is already built for certain display devices such as the Vive and SteamVR. The API has the ability to create an mpo file which can be treated as a jpg file while being processed. This solution is simple to implement and the file types created are very versatile and will be able to be played on any display device capable of displaying jpg files. This solution is the most likely choice because of how easy it looks to manipulate the end file format as well as the cross-platform capabilities of the file format [16].

2. The second option for creating a suitable file format for the selected viewing device is to use OpenCV. OpenCV is not as flexible with it's options for file output, however it is capable of writing the converted video to a specified video file type using the VideoWriter class. The class offers some flexibility when choosing the output format. However the biggest issue I see with this approach is the lack of a specific designated stereo 3D file format. This could mean more file processing work on our end in order to make the display file to be used for a specific display [19].

3. This api would be used after the converted video is created. The api has the ability to output video files into a suitable viewing file format such as the SBS file format which is widely used on virtual reality viewing devices. The file format contains images that are projected on a per eye basis from the viewing device and are treated like separate video streams in order to perform the 3D effect. This solution looks to be very applicable to our problem and will produce a file format suitable for many different viewing devices. However it may over complicate the project to introduce an additional api just for the purpose of correcting the file format when other api's can do that plus other processing tasks we need [20].

e. Discussion: Out of all these options the HTC dev api looks like the easiest option to manipulate the file format into the option that would work best.

f. Best Option: HTC Dev Api

9. Conversion of 2 Videos into 1 3D Video

a. Options: OpenCV, HTC Dev Api, MeshLab

b. Goals for use: Each of these software packages offers the ability to convert the videos into the stereoscopic 3D format.

c. Criteria being evaluated: Ease of implementation.

d. Comparison of technologies:

1. The first solution to merge the validated files is to use OpenCV. OpenCV contains classes implementing stereo correspondence as well well as epipolar geometry. The stereo correspondence class uses epipolar geometry to align the two video frames into a stereoscopic frame. This will happen on a frame by frame basis which will have the end result of creating a stereoscopic video based off of the two video files. OpenCV offers lots of tools for working with video files as well as creating stereoscopic images and videos, this is the solution that we will most likely choose to implement [15].

2. The second solution to this problem is to use the HTC Dev api. This api uses a side by side technique in order to create the stereo 3D effect. The frames will already be validated for stereo 3D creation at this point in processing. The api will create a new stereoscopic 3D file based off each video file being played for each eye. This solution looks to be easily implementable due to the public api and that the api is designed for the target platform we are trying create the video for [16].

3. The third solution to this problem is to use an open source library called MeshLab. This library offers a full toolkit for creating 3D images. The videos would be processed at a frame by frame basis in order to combine the two frames into a stereoscopic 3D frame. This solution to the problem seems to be more complicated than the previous ones. The software package offers a lot of capabilities that we wouldn't need to utilize and would most likely add to confusion while trying to implement the 3D file creation [17].

e. Discussion: OpenCV has the most robust toolset for the actual merging of the files and the API also offers solutions to other problems in the project. Meshlab looks like it will over complicate things, and the HTC API offers everything that the OpenCV API does.

f. Best Option: OpenCV

III Conclusion

For each of our nine requirements, we described three different technologies that could be used to complete that requirement. The best option for the UI of the mobile app is React Native. The best option for the UI of the desktop is QT. The best option for the capture of video is the Garmin Virb XE. The best option for stabilizing videos, cropping the videos, and converting two videos into one 3D video, is OpenCV. The best option for parsing BLOB files is Garmin-connect-export. The best option for correlating video frames is GPS data. The best option for viewing 3D videos in a VR headset is HTC Dev API. With these technologies we will be able to complete the requirements needed to finish our project.

IV Bibliography

[1] NGHIAH012, 'Simple Video Stabilization Using OpenCV', 2014. [Online]. Availeble: http://nghiaho.com/?p=2093. [Accessed: 2- Nov- 2016].

[2] 'Video API', 2016. [Online]. Available: https://dev.projectoxford.ai/docs/services/565d6516778daf15800928d5/operations/56 [Accessed: 12- Nov- 2016].

[3] 'Video API', 2016. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/video-api. [Accessed: 12- Nov- 2016].

[4] Martius, Georg, 'georgmartius/vid.stab', 2015, May 29. [Online]. Available: https://github.com/georgmartius/vid.stab. [Accessed: 12- Nov- 2016].

[5] Martius, Georg, 'vid.stab - Transcode video stabilization plugin'. [Online]. Available: http://public.hronopik.de/vid.stab/. [Accessed: 12- Nov- 2016].

[6] Rosebrock, Adrian, 'Basic Image Manipulations in Python and OpenCV: Resizing (scaling), Rotating, and Cropping'. [Online]. Available: http://www.pyimagesearch.com/2014/01/20/basic-image-manipulations-in-python-

and-opencv-resizing-scaling-rotating-and-cropping/. [Accessed: 12- Nov- 2016].

[7] ‘MoviePy’, 2014. [Online]. Available: http://zulko.github.io/moviepy/index.html. [Accessed: 12- Nov- 2016].

[8] Zulko, ‘Zulko/moviepy’, 2016, March 16. [Online]. Available: https://github.com/Zulko/moviepy. [Accessed: 12- Nov- 2016].

[9] ‘About FFmpeg’. [Online]. Available: https://www.ffmpeg.org/about.html. [Accessed: 12- Nov- 2016].

[10] ‘BLOB parser and domain’, 2016, August 12. [Online]. Available: http://www.ibm.com/support/knowledgecenter/SSMKHH $12 - Nov - 2016]$.

[11] kjkjava, ‘kjkjava/garmin-connect-export’, 2015, December 22. [Online]. Available: https://github.com/kjkjava/garmin-connect-export/blob/master/gcexport.py. [Accessed: 12- Nov- 2016].

[12] dtcooper, ‘dtcooper/python-fitparse’, 2012, December 8. [Online]. Available: https://github.com/dtcooper/python-fitparse. [Accessed: 12- Nov- 2016].

[13] "SciPy", SciPy.org, 2016. [Online]. Available: https://docs.scipy.org/doc/scipy-0.18.1/reference/misc.html. [Accessed: 14- Nov- 2016].

[14] "Features2D + Homography to find a known object — OpenCV 2.4.13.1 documentation", Docs.opencv.org, 2016. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/features2d/feature$_homography/feature_homography.html.[Ac$ $14 - Nov - 2016]$.

[15] "OpenCV: cv::stereo::StereoBinarySGBM Class Reference", Docs.opencv.org, 2016. [Online]. Available: http://docs.opencv.org/trunk/d1/d9f/classcv$_{11}stereo_{11}StereoBinarySGBM.html.[Accessed : 14 - Nov - 2016]$.

[16] "Stereoscopic 3D Api Overview", HTC Dev Api, 2016. [Online]. Available: https://www.htcdev.com/devcenter/opensense-sdk/legacy-apis/stereoscopic-3d/. [Accessed: 14- Nov- 2016].

[17] "VCC-3D", 3d-coform.eu, 2016. [Online]. Available: http://www.3d-coform.eu/index.php/vcc3d. [Accessed: 14- Nov- 2016].

[19] "Reading and Writing Images and Video — OpenCV 2.4.13.1 documentation", Docs.opencv.org, 2016. [Online]. Available: http://docs.opencv.org/2.4/modules/highgui/doc/reading$_and_writing_images_and_video.html.[Accessed :$ $14 - Nov - 2016]$.

[20] "Windows 8 Direct3D stereoscopic 3D sample in C++ for Visual Studio 2013", Code.msdn.microsoft.com, 2016. [Online]. Available: https://code.msdn.microsoft.com/windowsapps/Direct3D-111-Simple-Stereo-9b2b61aa. [Accessed: 14- Nov- 2016].

[21] Facebook, "Facebook/react-native," in GitHub, GitHub, 2015. [Online]. Available: https://github.com/facebook/react-native. Accessed: Nov. 14, 2016.

[22] Xamarin, "Developer Center - Xamarin,". [Online]. Available: https://developer.xamarin.com/. Accessed: Nov. 14, 2016.

[23] Drifty, "Build amazing native Apps and progressive web Apps with ionic framework and angular," in ionicframework. [Online]. Available: http://ionicframework.com/. Accessed: Nov. 14, 2016.

[24] GitHub, "Electron/electron," in GitHub, GitHub, 2013. [Online]. Available: https://github.com/electron/electron. Accessed: Nov. 14, 2016.

[25] Facebook, "A JavaScript library for building user interfaces - react," in GitHub. [Online]. Available: https://facebook.github Accessed: Nov. 14, 2016.

[26] Facebook, "Facebook/react," GitHub, 2013. [Online]. Available: https://github.com/facebook/react. Accessed: Nov. 14, 2016.

[27] The QT Company, "The future is written with Qt: Cross-platform software development for embedded desktop," in Qt, Qt, 2015. [Online]. Available: https://www.qt.io/. Accessed: Nov. 14, 2016.

[28] The QT Company, "Legal | licensing," in Qt, Qt, 2014. [Online]. Available: https://www.qt.io/licensing/. Accessed: Nov. 14, 2016.

[29] The QT Company, "Product | the IDE," in Qt, Qt, 2015. [Online]. Available: https://www.qt.io/ide/. Accessed: Nov. 14, 2016.

[30] Xamarin, "Advanced IDE for iOS and Android," in Xamarin. [Online]. Available: https://www.xamarin.com/studio. Accessed: Nov. 14, 2016.

[31] The Qt Company, "Qt style sheets," in Qt. [Online]. Available: http://doc.qt.io/qt-4.8/stylesheet.html. Accessed: Nov. 14, 2016.

[32] Xamarin, "Styles - Xamarin," in Xamarin. [Online]. Available: https://developer.xamarin.com/guides/xamarin-forms/user-interface/styles/. Accessed: Nov. 14, 2016.

[33] Garmin. ltd, "VIRB XE," in Garmin, Garmin. [Online]. Available: https://buy.garmin.com/en-US/US/prod165499.html. Accessed: Nov. 14, 2016.

[34] GoPro, "GoPro - Cameras," in GoPro. [Online]. Available: https://shop.gopro.com/cameras. Accessed: Nov. 14, 2016.

[35] L. Goodall, "Motion sickness in virtual reality and how to prevent it," in TruVision VR, TruVision VR. [Online]. Available: http://truvisionvr.com/blog/motion-sickness-in-virtual-reality-and-how-to-prevent-it/. Accessed: Nov. 14, 2016.

[36] Altiforce, "Alti-Force sensor pack - GoPro recorder video Synced data," in Altriforce, Alti-Force for GoPro video. [Online]. Available: http://www.altiforce.net/shop/info-sp22/. Accessed: Nov. 14, 2016.