

# Preliminary Report

Smart Systems Lab: 360-degree camera

de Guzman, Carlos  
Ng, Alexander

Fruge, Alan  
Park, Daniel

Kirby, William

## I. Abstract

**With the advancement of computer vision, the applications of cameras have dramatically increased. In this project, the goal is to create a 360° camera using an Nvidia Jetson Nano stitching several camera frames. To achieve this goal, a custom case and stitching software were created for this project. The algorithm for the stitching software will be improved iteratively and is the greatest challenge in creating a reliable camera system. Also included is a basic and general-purpose GUI that communicates with the Jetson from a host computer.**

**This report begins with how the general-purpose framework of this project could be reused for a wide variety of applications. Next, it will provide the technical background regarding computer vision as well as our test results with different camera and stitcher combinations. These informed the decision to create a custom stitching algorithm. The implementation of the custom stitcher is documented in the design section. This section will also include the communication protocols that interface between each of the subsystems and how the overall system should interact from both a user and developer perspective. Lastly, it will cover features that have been completed, are in progress and are planned along with estimated dates. Along with this, a brief discussion of future modifications and applications will be discussed.**

## II. Introduction

This project aims to create an inexpensive and reliable 360° camera for both consumers and businesses. As the completed product will be able to capture an entire scene around it, it has a wide variety of applications. Examples include panoramic video conferencing, virtual experience capturing for VR, etc.

While there are currently existing 360° cameras available to consumers, it is important to note that the software and design artifacts are made open source on Github. This will allow an end user to build upon the general-purpose architecture to fulfill the requirements of a specific application. Also, documented there is the specific hardware that was used. Because the software was built on the UNIX platform, an end user could also choose to change the hardware it runs on to better fit its use case. An example of changes could include the Nvidia Jetson Nano to a Raspberry Pi, the cameras to USB 3.0, and redesigning a case to include more cameras.

An example that was proposed in the SmartSystems Lab was the use of computer vision to improve businesses'

efficiency with tasks like inventory and security statistics. In this case, the user could experiment with different camera parameters including the number of connected cameras or the stitcher being used. Another possibility is the use of an embedded system with higher computational power that could be used to run a lightweight neural network for item detection. Regardless of the specific changes, the goal of the project is to have a well-documented and general-purpose platform that is easy to build upon for different applications.

## III. Background

### A. Camera Calibration

As computer vision is the field within artificial intelligence that studies how computers interpret and understand the visual world, this paper will begin with cameras. Camera calibration is a foundational concept that determines how a camera represents an external scene as an image based on the extrinsic and intrinsic parameters of the camera [14]. Extrinsic parameters correspond to rotation and translation vectors from 3D points in the real world to the camera coordinate system. Conversely, intrinsic parameters convert the internal coordinate representation to the image plane. These include the focal length and the optical center of the camera which are unique to a camera based on its physical geometry, sensor, etc.

To compute these matrices, several images of a well-defined pattern like a chessboard with known real-world lengths should be taken. From these, OpenCV can interpolate the values of these matrices and save them. With these values known, it is possible to correct distortions which are important for stereo applications such as this project. In this project, it is also explored how intrinsic matrices are used for cylindrical warping explained below.

$$\text{camera matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 0: General Camera matrix (focal length and optical center) [14]

### B. Homography

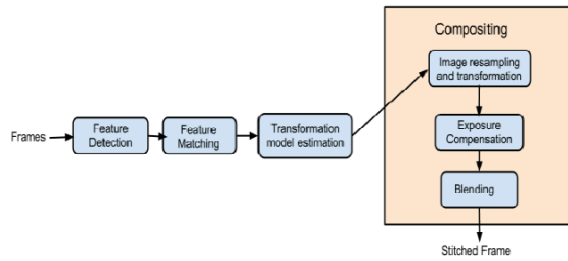


Figure 1: Image Stitching Pipeline

In the generalized image stitching pipeline, shown above, the stages can be grouped into transformation estimation (first three) and compositing (last three). One method of the former is using homography matrices that relate [15]. It is composed of a set of rotations, translations, and plane normal matrices. By extracting and matching features between 2 images, it is possible to estimate this homography matrix. Using a homography matrix, it is possible to warp the perspective of one image and overlay it over the other. This is shown in the 2 figures below.

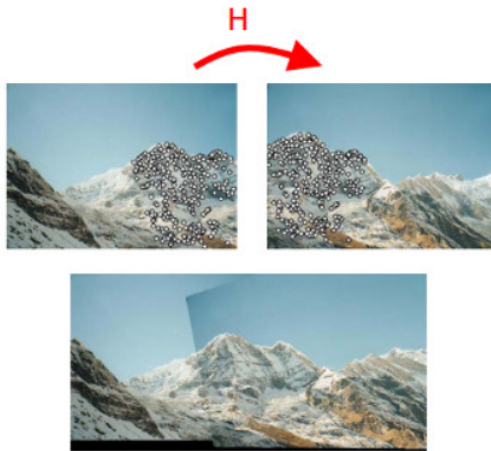


Figure 2: Homography with matched features [15]

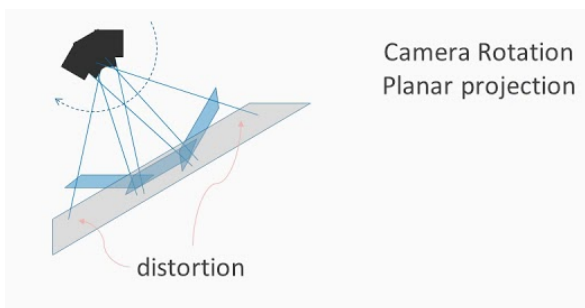


Figure 3: Planar projection/warping

### C. Compositing and Cylindrical Warping

Compositing applies the estimated transforms on the original images to create the final stitched image. The first

step involves warping the images. Then exposure compensation and blending can be applied to improve the quality of the stitch. These two will not be discussed as complex techniques like exposure estimation and finding the best seam masks are beyond the scope of this project.

Cylindrical warping can be used for image compositing to project all images onto a common cylindrical plane. It requires the camera's intrinsic matrix and rotation matrices to be known. The image can then be "unrolled" to be equirectangular. While this may cause some distortion, it will allow the stitched image to span greater than 180 degrees. This is ideal for cases where there is one rotating camera or multiple cameras with a common center.

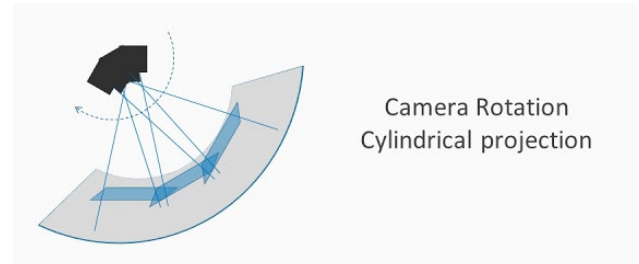


Figure 4: Cylindrical projection/warping

### D. Testing

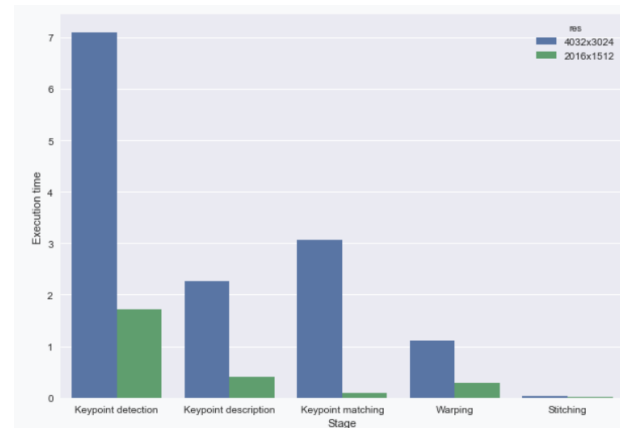


Figure 5: Runtimes of Image Stitching Stages [10]

Above is a chart from a study from Harvard University regarding Real-Time image stitching. It is important to notice that transformation estimation (keypoint/feature detection, description, matching) takes a significant majority of the execution time when compared with compositing (warping and stitching). This informed the decision to create a custom stitcher for this project. A testing methodology for the stitcher was created to ensure that optimizations improved the performance over the baseline (OpenCV) stitcher.

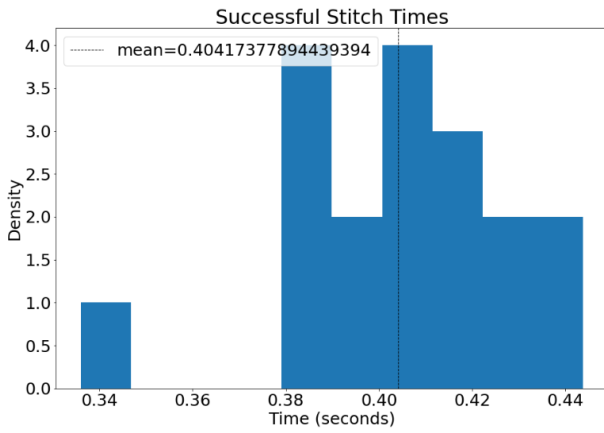


Figure 6: Stitch test results with OpenCV Stitcher

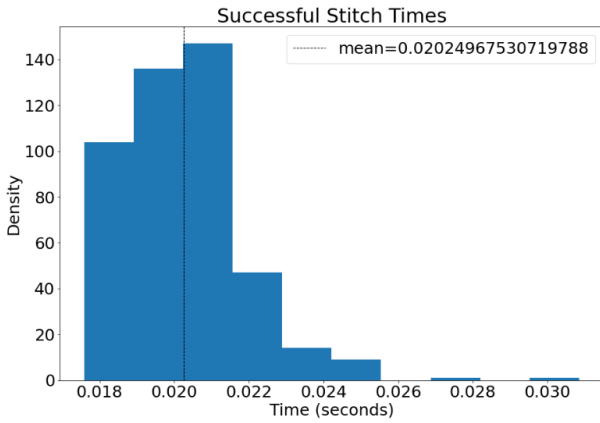


Figure 7: Stitch test results with Custom Stitcher

The key results from testing were as follows. As desired, the custom stitcher had a nearly 20x faster execution time when compared with the OpenCV stitcher. It also had an effectively 100% stitch rate compared with only 17% for OpenCV. It is also noteworthy that the execution time of the custom stitcher did not have outliers like those of its counterpart. This contributed to a more consistent experience.

#### IV. Design

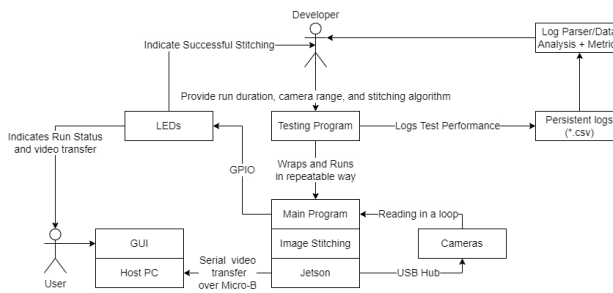


Figure 8: Behavior Diagram

This project was developed to run on the Nvidia Jetson Nano. This was chosen due to the CUDA acceleration which

can help with many vectorized operations in computer vision and opens the possibility for expansions on this project like deployed image detection, etc. The Jetson connects to 6 cameras through an externally powered USB hub. These cameras are held in place with 60 degrees rotation along with the Jetson in a 3D printed case. The case was also designed to hold an LED indicator.

As discussed above, it was decided that a custom stitcher should be used. This works by taking images, estimating the transform, and then saving. It then uses this saved transformation estimation to compost all subsequent image frames, so the output appears as a unified smooth video. Overall, this option was chosen as it was decided that with the computation constraints, fast execution and consistency were more important to the end user experience than perfect image quality.

The Jetson was configured to create a Flask web server on startup. This sets up the cameras and image stitcher and has routes to read stitched video. The Jetson can be connected to a host computer through serial Micro USB. This allows the host computer to communicate over IP. There is also an LED indicator for when the Flask server is running.

From the host computer, the user can open a web-based GUI. This program interacts with Jetson's flask web server through HTML requests. The interface includes buttons to refresh the video feed and recalibrate the cameras by estimating the transformation of images with the current environment (the cameras calibrate on start-up). Currently, it is being updated to use Pannellum which uses WebGL to allow the user to pan around the video. In addition, the user will have access to LEDs that indicate the stitching software's status.

On the developer side, the project includes a testing tool for different configurations in a repeatable way. This application takes a stitcher, execution time, and the number of cameras as parameters and runs the test while logging the results. From these results, a report can be generated. This includes graphs about stitch timing like figures 6 and 7 above, distribution of stitch errors along with a sample image to visually show the quality of the stitching.



Figure 9: Sample image with cylindrical warping (release candidate)

#### V. Status

## V.1 Completed

### Hardware

1. Prototype case created
  - a. Holds 6 cameras at a fixed angle
2. Cameras connected to the Jetson
  - a. Able to read from 6 concurrently
3. USB power requirements of cameras fulfilled

### Software

1. Custom Stitching Algorithm
  - a. Homography matrix for sets of 3-camera inputs
2. Alternate GUI
  - a. Sends SSH commands from the host to Jetson
  - b. Utilizes socket to receive video data
3. Flask Server
  - a. Runs on startup
  - b. Initializes Camera System
  - c. Recalibrates to environment
  - d. Routes created to send video through HTTP responses
4. Testing Program
  - a. Able to run for a given duration
  - b. Able to test the different camera and stitcher configurations
  - c. Creates detailed log files
  - d. Takes an image before running the test
  - e. Detailed testing results analyzer
    - i. Creates statistics and charts from logs and images

## V.2 In Progress

### Hardware

1. Redesign of 3D printed case
  - a. Include space for LED indicators
  - b. Incorporate more space for Nvidia Jetson and USB Hub

### Software

1. Cylindrical Warping
2. Panellum (WebGL) video viewer
3. Code refactoring
4. GUI improvements

## V.3 Planned

### Hardware

1. Finalized 3D-printed shell accommodating all desired hardware including cameras, hub, LED indicators, and Jetson.

### Software

1. CUDA Acceleration
2. Improved camera calibration and matrix adjustments for cylindrical wrapping.

## VI. Conclusion

This report covered the computer vision concepts required in this project. Combining this with the results from Alpha and Beta testing, it is shown why the decision was made to create a custom stitcher for this project.

In the preliminary iteration of the 360° camera, all basic functionality including the ability to view & stitch from all cameras has been implemented. With the pending addition of cylindrical wrapping, the image coherency can be improved and provide a full 360° image instead of a dual 180°-stitched display output. This will deliver on the overall goal of this project, with future goals focused on image quality through improved stitching & composting. This makes this project suitable to be built upon with future projects as discussed in the Introduction.

## VII. References

- [1] Microchip, "USB5807" *Microchip Technology Inc.* [Online]. Available: <https://www.microchip.com/en-us/product/USB5807> [Accessed: 28-Feb-2022].
- [2] Microchip, "USB2517" *Microchip Technology Inc.* [Online]. Available: <https://www.microchip.com/en-us/product/USB2517> [Accessed: 02-Mar-2022].
- [3] Microchip, "EVB-USB5807" *Microchip Technology Inc.* [Online]. Available: <https://www.microchip.com/en-us/development-tool/EVB-USB5807> [Accessed: 02-Mar-2022].
- [4] Microchip, "EVB-USB2517" *Microchip Technology Inc.* [Online]. Available: <https://www.microchip.com/en-us/development-tool/EVB-USB2517> [Accessed: 02-Mar-2022].
- [5] Findchips, "USB2517" *Findchips.com* [Online]. Available: <https://www.findchips.com/search/USB2517> [Accessed: 03-Mar-2022].
- [6] Findchips, "USB5807" *Findchips.com* [Online]. Available: <https://www.findchips.com/search/USB5807> [Accessed: 03-Mar-2022].
- [7] Electronicsnotes, "USB Standards: USB 1, USB 2, USB 3, USB 4 - capabilities & comparisons" *Electronicsnotes.com*. [Online]. Available: <https://www.electronics-notes.com/articles/connectivity/usb-universal-serial-bus/standards.php> [Accessed: 01-Mar-2022].
- [8] Microchip Support, "Cascading Hubs" *Microchip Technology Inc.* [Online]. Available: <https://microchipsupport.force.com/s/article/USB-HUBs---Cascading-hubs> [Accessed: 03-Mar-2022].
- [9] J. R. Picture and Jay Rambhia Senior Android Developer Also, "Stereo calibration," *Jay Rambhia's Blog*, 30-Mar-2013. [Online]. Available: <https://jayrambhia.com/blog/stereo-calibration>. [Accessed: 04-Apr-2022].
- [10] Harvard University, "Real-time image stitching," *CS205 Real Time Image Stitching*, 2018. [Online]. Available: <https://cs205-stitching.github.io/>. [Accessed: 07-Apr-2022].
- [11] Nghonda, Erman, "Pose Estimation and Triangulation" *University of Florida, Event: Research*. [Online]. Available: File request. [Accessed: 07-Apr-2022].
- [12] Gray, Dan and Lee, Keven, "360 Camera" *Hackaday.IO*. [Online]. Available: <https://hackaday.io/project/11604-360-camera>. [Accessed: 07-Apr-2022].
- [13] Rosebrock, Adrian, "Image Stitching with OpenCV and Python" *pyimagesearch*. [Online]. Available: <https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/> [Accessed: 07-Apr-2022].

[14] OpenCV, "Camera calibration," OpenCV. [Online]. Available: [https://docs.opencv.org/3.4/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html). [Accessed: 07-Apr-2022].

[15] "Basic concepts of the homography," *OpenCV*. [Online]. Available: [https://docs.opencv.org/4.x/d9/dab/tutorial\\_homography.html](https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html). [Accessed: 27-Oct-2022].