Hardware Accelerated Image Stitching Camera

ECE 445: Design Document

Cole Herrmann (colewh2)
Gautum Pakala (gpakala2)
Jake Xiong (yuangx2)

Fall 2022

1. Introduction

1.1 Problem

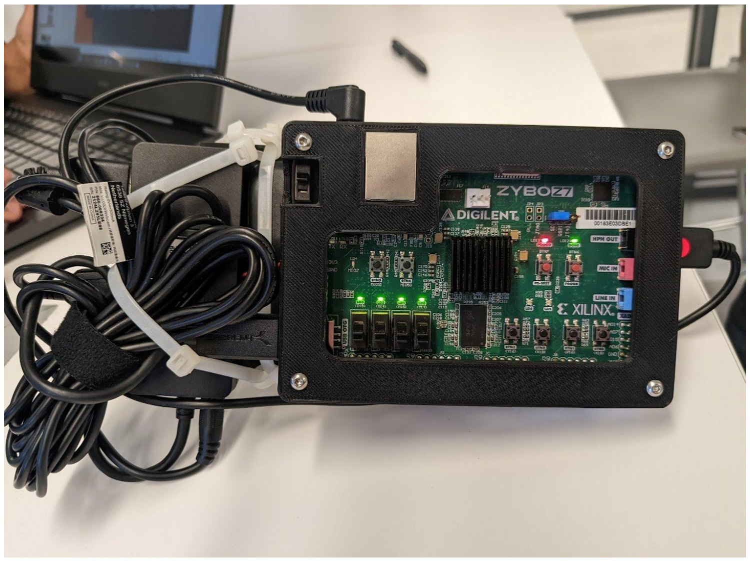
Time and energy are resources that aren't plentiful in UAVs. Traditionally when a UAV is used for aerial mapping, it will take a picture every time it flies a predetermined distance interval. Since UAVs must be kept lightweight, it's uncommon to find any with enough onboard processing hardware and energy reserves to stitch hundreds of frames into a map. That's why most mapping UAVs perform the map generation offsite on more powerful hardware than the onboard camera and flight controller. In time sensitive emergencies (open combat, search and rescue, etc), it may not be possible to land the UAV to render an aerial map, and it would be much more convenient if the drone could render the map itself.

1.2 Solution

We designed a camera that has onboard hardware acceleration capability to stitch images together. When stitching images together into a panorama or map, several repetitive operations are required to "prep" the images for stitching. Operations to grayscale, blur, and convolute images can be performed on a traditional CPU, but the processing time and power consumption can be improved when such repetitive operations are pipelined through an FPGA. With Cole's ECE 397 funding from last semester, he acquired a Diligent Embedded Vision bundle (https://digilent.com/shop/embedded-vision-bundle/), from which we used the Zybo Z7020 and PCAM 5C as the basis for the camera. After completing this project, Cole plans to integrate the camera into one of his drones, including adding serial communication between the flight controller and the Zybo board (another pro of building on PetaLinux), which would give access to a plethora of sensors such as GPS, airspeed, etc that could bring a live rendering aerial mapping drone into reality!

1.3 Physical Design

Zybo Z7020 Development Board with Zynq-7000 SOC



1.4 High-Level Requirements

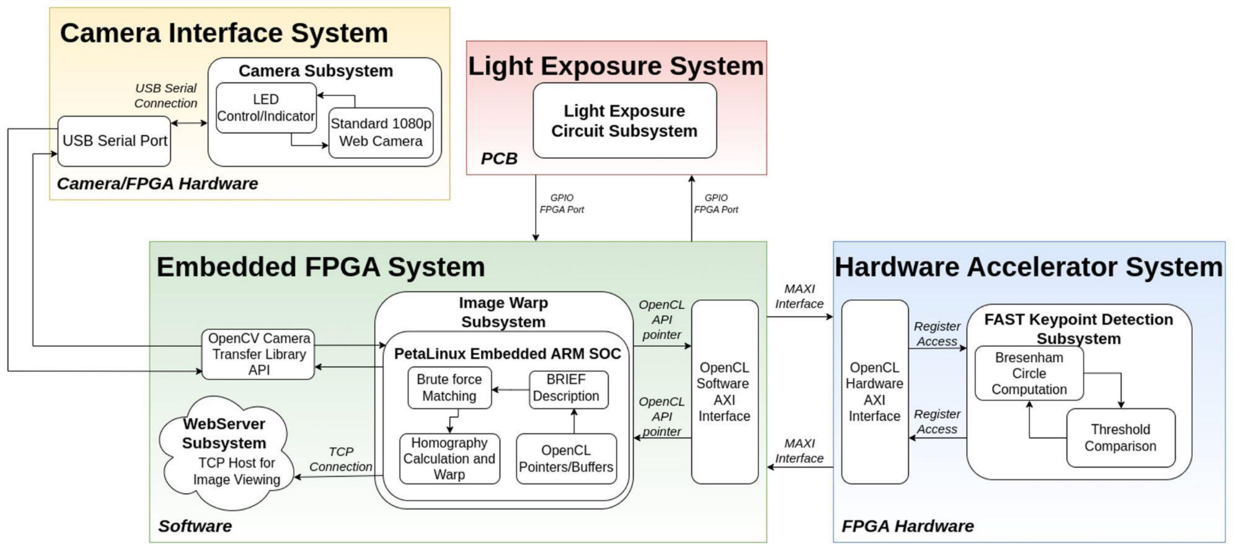
1.4.1 A picture is taken by pressing a button on the external camera control PCB. The Camera will store all pictures as YUV files on the Ext4 filesystem (with 32 GB of space), accessible by the PetaLinux OS.

1.4.2 After at least two images have been saved to the filesystem, the user can press a button on the external camera control PCB, which stitches together the images taken into a panorama. By accelerating the stitching process with hardware, the image processing only takes a few milliseconds, depending on how many images are being stitched together.

1.4.3 After all images have been processed and a resultant image composed of the original images stitched together is completed, a client computer can access a local web server on the development board to view an HTML image gallery.

2. Design

2.1 Block Diagram



The FAST algorithm utilizes computer vision techniques to check pixel intensity and identify key points in images. Key points are pixels in an image that are significant such as the tip of a building or the license plate on a car. The FAST Keypoint detection subsystem accelerates this process through hardware comparisons on the FPGA.

After the key points are identified in the images, they are sent back to the Image Warp subsystem on the FPGA. This system performs calculations on the images in order to warp and blend them together into a proper panorama.

The final stitched together images are hosted on the WebServer subsystem through a tcp connection between the SOC and the local subnet.

While the images are being taken, the light sensing circuit on the PCB sends a signal to the FPGA. The FPGA will communicate with the PCB telling it whether to flash the LEDs for light exposure in the images.

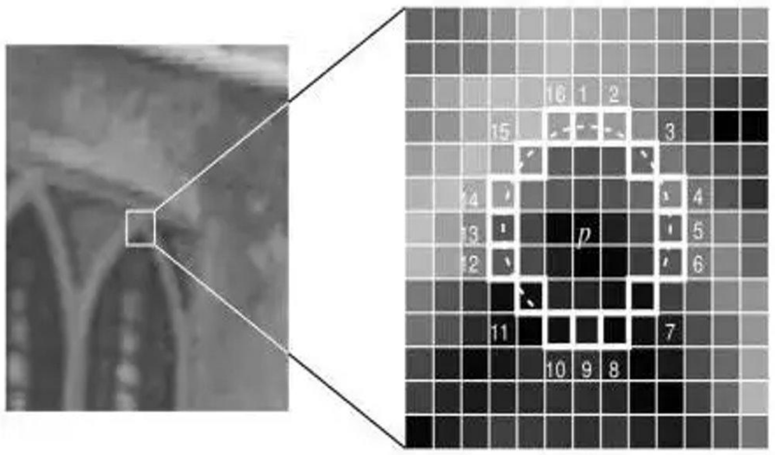
2.2 Subsystems

2.2.1 FAST Key Point Detection

Keypoint Detection is the process of identifying key points in an image that are recognizable from different angles, lighting, and scale. Many computer vision algorithms accomplish this goal such as SIFT, SURF, and FAST to name a few. We are choosing to implement the FAST algorithm for keypoint detection, not just because it is faster than most other algorithms, but also because it is the least resource intensive for the FPGA to execute. These algorithms already take into account scale and rotational invariance for the images.

The reasoning behind FAST is simply because the algorithm only uses the intensity of the pixel (meaning the grayscale value) to compute whether the point is a keypoint. The FAST algorithm takes pixel values around the current pixel being tested in a circle called Bresenham's circle. The algorithm then checks whether there are 12 or more pixels in this circle that have a greater intensity than the tested pixel plus a threshold. The threshold determines how significant you want the detected key points to be.

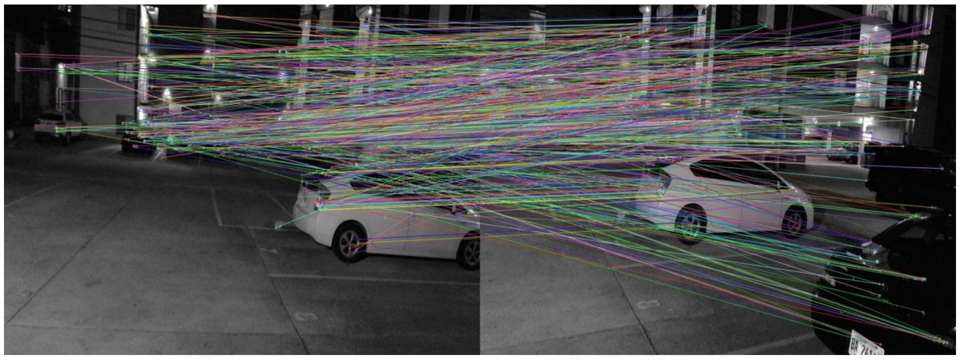
Bresenham's Circle



2.2.2 Key Point Description, and Matching

Keypoint Description gives each identified keypoint a unique descriptor that can be used to identify each keypoint on the image. There are many methods of doing this, but the simplest is to compile a matrix of the gradient vectors around each keypoint that can be obtained through convolving the image with specific filters.

Keypoint matching occurs when the keypoints are detected and described in each image. If the difference between the descriptors is below a certain error threshold, the key points in each image are said to be a match, as shown by the matches in the image below. Typically, a minimum of 4 keypoint matches is needed for Homography Transformation. The method we used for keypoint matching is the Brute Force Algorithm, which does what it says and brute force compares all iterations of the descriptors until it finds the highest confidence match.



2.2.3 Homography Transformation and Warping

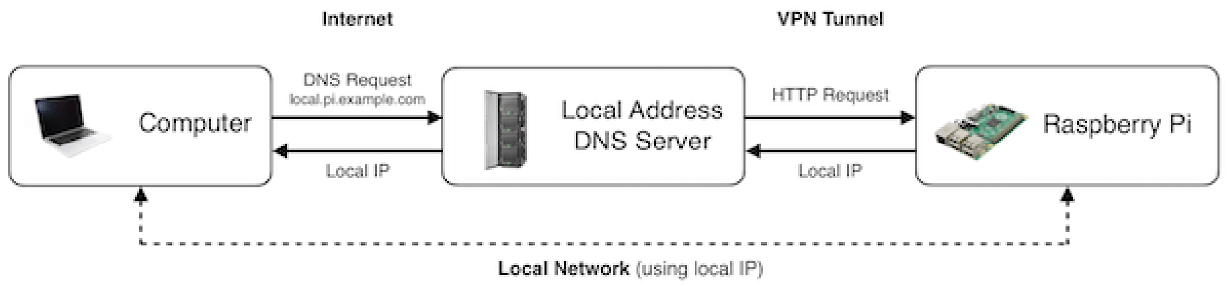
When image stitching, the angle of the images needs to be rectified to create a clean output panorama. Homography Transformation is a common problem that transforms the coordinate system of an image into the plane of the reference image through a 3x3 homography matrix. The homography matrix can be calculated using the keypoint matrix and solving a constrained least squares problem in order to find the eigenvector with the lowest eigenvalue. The method of calculating this matrix is shown below with x being the source image pixels and Y being the destination image pixels. This transformation is then applied to the destination image. One issue with the homography transformation is that the result can be skewed with outliers in the keypoint matching process, where there are keypoint matches detected, but they are not really matches. A common solution to any outlier problem like this is the RANSAC algorithm. This is used to make the computation of the homography matrix more robust. After the images are warped (transformed) and overlapped through matrix multiplication.

$$\begin{vmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3Y_3 & -y_3Y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 \end{vmatrix} \bullet \begin{vmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{vmatrix} = \begin{vmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{vmatrix}$$

2.2.4 Web Server Image Gallery

We configured our development board to run Petalinux 2020.2, with the network card's default IP addressing configuration set to static before building the filesystem. It's important that certain features like static IP addressing be built into the filesystem before each boot because the development board uses a volatile file system upon each boot. A volatile filesystem is handy in preventing file corruption through bad programs or sudden power loss, and is a common feature on embedded devices. The development board was also configured with a Busybox HTTP daemon being built into the kernel and filesystem. Upon boot, the development board obtains a static IP (10.10.10.3 in our configuration), and then starts the Busybox daemon once a valid network connection is established. We configured Busybox to point all HTTP requests to the '/srv/www/gallery/' directory, where we hosted all the output images from the camera, and a single file HTML page with built in javascript functions and CSS styling. Any computer on the same local network was able to access the camera's image gallery by visiting 'http://10.10.10.3/gallery/gallery.html' in a modern web browser.

An HTTP web server is the ideal method for viewing the output from the camera because it offers flexibility on the quantity and size of the images being displayed. For example, rather than displaying a single image at a fixed resolution through the HDMI out port, viewing many images in a single webpage is more versatile for camera applications. The diagram below shows the high level process of how any client machine can send an HTTP request to the FPGA. Please note that the Zybo development board takes the Raspberry Pi's place in our implementation.



Two clients are on the same local network if they share a local subnet; which is likely if they are connected to the same router. Having this local connection, they can send HTTP requests one of two ways: either through the local network using IP addresses for the naming scheme, or through a local address DNS server using a registered domain name for addressing. To determine the optimal method, one must consider whether Static IP or DHCP is being used for IP address assignments. Static IP infers that the development board's IP address will not change, while DHCP (Dynamic Host Control Protocol) generally randomizes IP assignments, giving a new IP each time the device connects. Registering a domain on the DNS server for the development board is only necessary if DHCP is turned on, because a constant URL is needed for accessing the image gallery. If static IP addressing is used, the URL will remain constant and the client computer connecting to the web server can always use the IP of the development board to access the webpage.

2.2.5 Camera

To capture the images to be stitched together, we had to interface a camera into the system that can capture an image, and save it in RAM so the stitching application can manipulate the image. After some complications interfacing a MIPI camera into the linux system, we decided to interface a Logitech C920 1080p USB webcam. The USB webcam offers auto focus and auto exposure which are important features when stitching images into panoramas. To integrate the camera into the system, we had to compile the Petalinux OS with V4l2 drivers which can instantiate a USB webcam as a video device in the '/dev/video0' directory. We also had to instantiate a USB PHY host controller in the linux device tree that allowed USB peripherals to be controlled. When the user triggers an image capture, the stitching program launches a python gstreamer application that captures an image from '/dev/video0' and saves it to the RAM.

To control the camera, we interfaced the Petalinux OS to talk to the onboard push buttons and LEDs on the Zybo Z7020 development board. In our block diagram, we instantiated two AXI-GPIO interfaces, with one linking the onboard LEDs to an AXI memory address, and the other linking the pushbuttons to an AXI memory address. We then added these AXI-GPIO interfaces to the linux device tree so the Petalinux OS knows what memory addresses the IO is located at. The image stitching application uses the push buttons as inputs to trigger images to be stitched, and uses the LEDs to show the status of the image capture and stitching process.

2.2.6 PCB Light Exposure Circuit

The light exposure subsystem is a simple LED and resistor 5V circuit connected to the FPGA through the GPIO power pin. This way the FPGA can monitor the circuit and determine when the LEDs should be turned on for better energy efficiency. The circuit is currently configured to be turned on at start up.

2.3 Tolerance Analysis

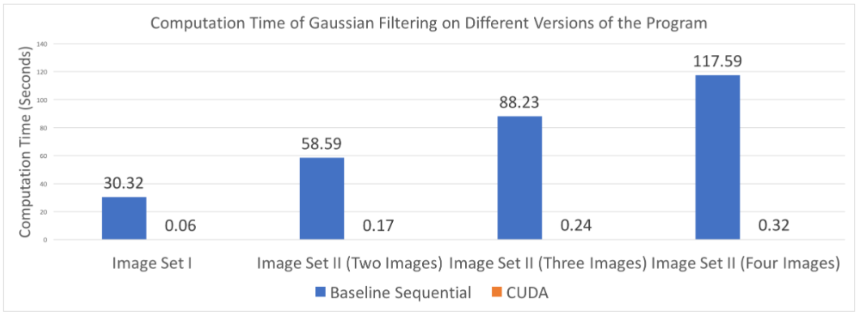
Two algorithms are usually applied for key point matching, parallel and sequential image stitching with their tradeoffs.

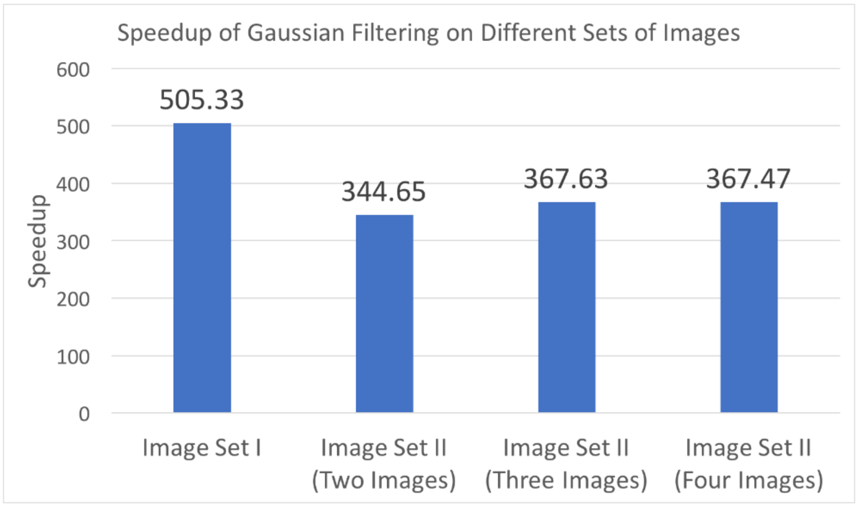
Key point matching utilizes the difference of gaussian approach in which we blur the image and subtract the images to find the difference with different levels of gaussian blur. The key points are the pixels that are locally distinct, and we utilize the gaussian pyramid to find multiple key points with the approach. The next step is computing the descriptor in which we compute the gradient of the area and collect the gradient for histogram and find similar local key points.

Computing the gaussian pyramid is usually time consuming and different approaches can be used in the steps. The process of parallelizing gaussian filtering benefits from the parallelism, which reduces the process within seconds[3].

Another challenge of gaussian filtering is the memory constraint. We find out that although the actual computation on a CUDA device is only 0.43s, transferring the images to a CUDA device takes 0.73s, which is almost twice as much as the actual computation time.[4]

The sequential image stitching approach stitch images with optimal seam finding and transition smoothing processes. The sequential panorama stitching procedure enables us to process large source images and create high resolution panoramic images on limited-resource devices such as mobile phones.[5] The steps of the procedure are optimal seam finding and transition smoothing.





During panorama stitching, the approach only needs to keep the panoramic image and the current source image other than all source images in memory, which is good for implementation on mobile devices.[5]

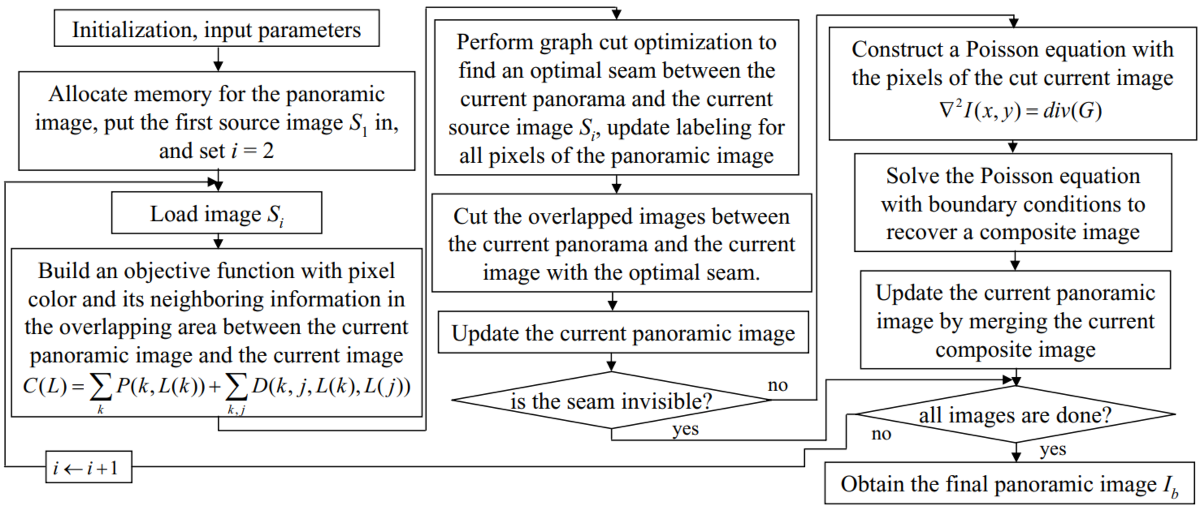
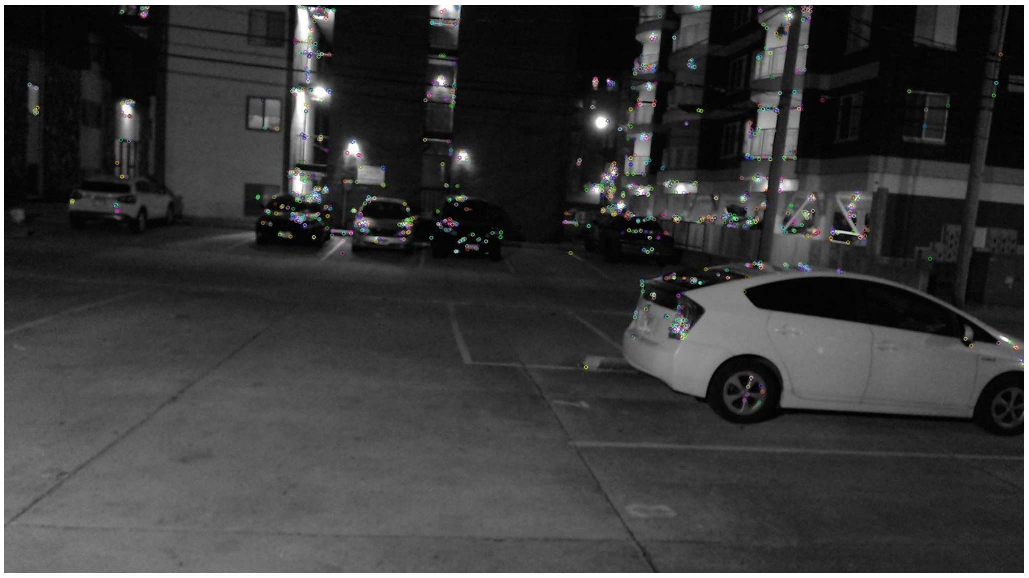

Initialization, input parameters
Allocate memory for the panoramic image, put the first source image S_1 in, and set $i = 2$
Load image S_i
Build an objective function with pixel color and its neighboring information in the overlapping area between the current panoramic image and the current image
$C(L) = \sum_k P(k, L(k)) + \sum_{k,j} D(k, j, L(k), L(j))$
$i \leftarrow i+1$
Perform graph cut optimization to find an optimal seam between the current panorama and the current source image S_i, update labeling for all pixels of the panoramic image
Cut the overlapped images between the current panorama and the current image with the optimal seam.
Update the current panoramic image
is the seam invisible?
no
yes
Construct a Poisson equation with the pixels of the cut current image
$\nabla^2 I(x, y) = div(G)$
Solve the Poisson equation with boundary conditions to recover a composite image
Update the current panoramic image by merging the current composite image
all images are done?
no
yes
Obtain the final panoramic image I_b

Fig. 1. Work flow of the sequential image stitching approach for creating mobile panoramic images.

3. Verification

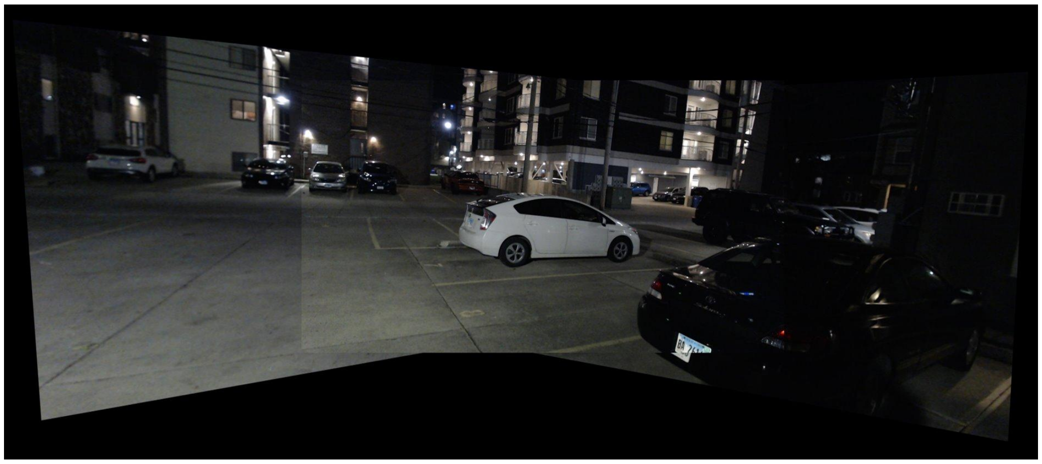
3.1 FAST Keypoint Detection

The Fast algorithm is suitable for our real-time video processing application because of its high-speed performance. With 20 threshold for the Bresenham Circle, we are able to achieve 20ms average latency of the algorithm, accounting for I/O, for an average of 131 keypoints detected per image. The high speed is reached with our optimization of the algorithm, and choice of threshold. As an additional comparison, running a software FAST detection algorithm took 85 ms on average, so our acceleration provided a 4.25x speedup over the software.



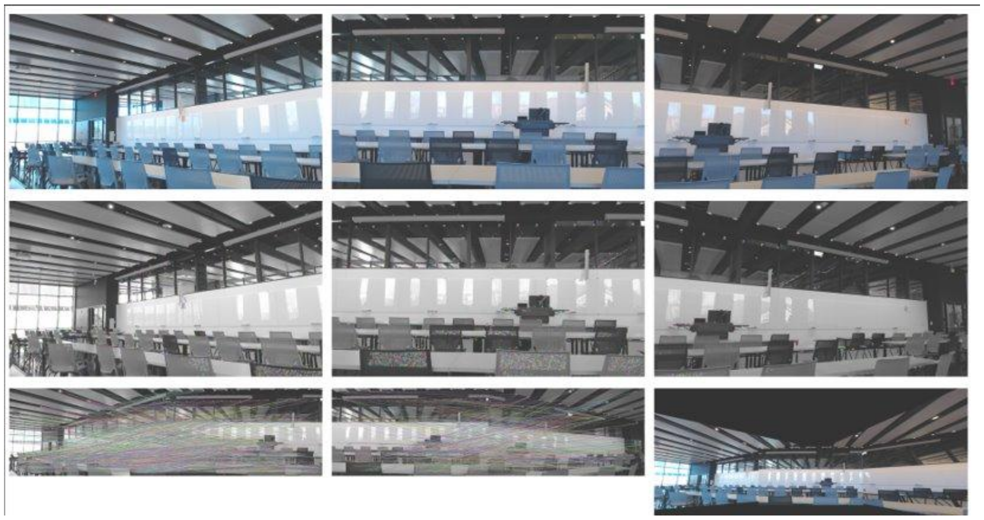
3.2 Homography Transformation and Blending

At least 4 points are necessary for the calculation, but our FAST is robust enough to provide over 100 key points. With our system, the latency of blending and overlay varied from 3-10 seconds depending on the number of keypoints. Artifacts in the images are mainly only noticeable through light exposure which is a tradeoff we made with cost. With a more expensive camera with light-balancing we would reduce the artifacts from light exposure.



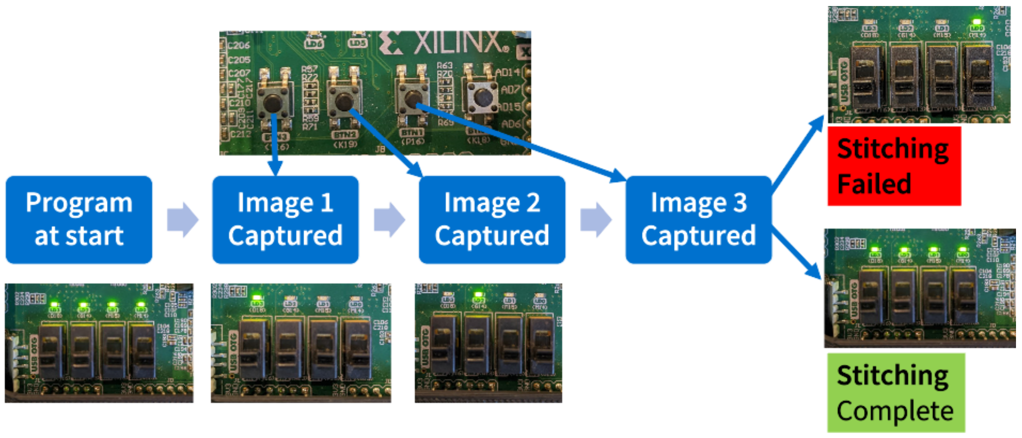
3.3 Web Server Hosted Image Gallery

After the camera application finishes stitching the completed panorama, it sends the three original pictures (left.jpg, middle.jpg, and right.jpg), grayscale keypoint descriptors, grayscale matched keypoints, and the resultant panorama to the image viewer directory. The images can then be viewed by visiting 'http://10.10.10.3/gallery/gallery.html', with 10.10.10.3 being the IP of the development board. The image layout in 'gallery.html' is shown below.


'gallery.html' image layout

3.4 Camera

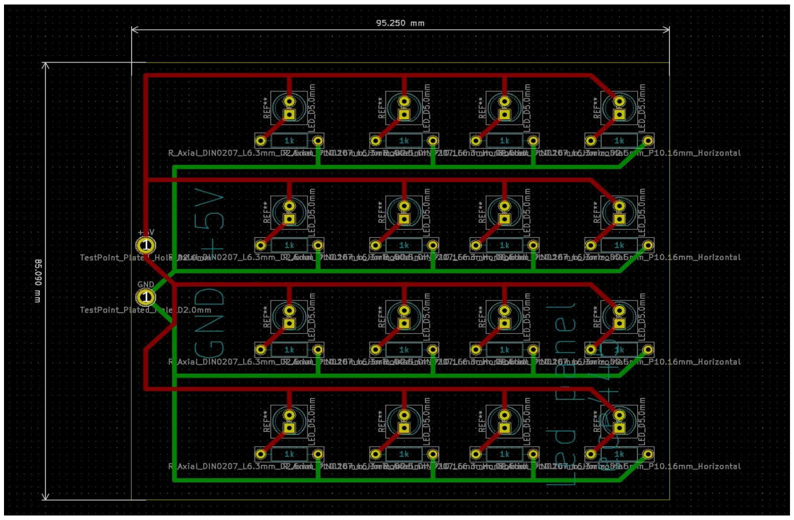
The camera system consists of a USB webcam, GPIO push buttons, GPIO LEDS, and drivers to control the image capturing process. The USB webcam is instantiated as a device upon boot as '/dev/video0' in the Petalinux filesystem. The camera application uses a Python program that accesses a gstreamer to capture a frame from the webcam. However, the camera application waits for GPIO input before executing the python script. Once the program receives GPIO input, the python script is run and the program outputs a signal to a GPIO LED to signify that a picture has been captured. The diagram below shows this process.



The rigidity of the camera application can also be tested by purposely taking three faulty pictures (one method would be to cover the lens with a piece of paper), and let the camera application attempt to stitch them together. Since the images of a piece of paper will not result in any key points from the fast algorithm, the stitching will fail and the program moves to the 'Stitching Failed' state in the block diagram. However in normal operation, the stitching will succeed and move to the 'Stitching Complete' state.

3.5 Light Exposure Circuit

PCB verification was tested through its GPIO connection to the FPGA. The LEDs on the PCB only turned on when the FPGA was turned on due to our setting of the GPIO pin only to turn on with the FPGA. Before connecting to the FPGA, the PCB was tested through battery power with 5 volts and a switch. The circuit operated functionally under these conditions, with none of the LEDs turning on when the switch was off. Due to these tests, we were able to confirm functionality of the PCB to connect to the FPGA.


95.250 mm
85.090 mm
+5V
GND
TestPoint_Plated_Hole_D2.0mm
R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
LED_D5.0mm
REF**
1k
Led panel

PCB layout

4. Cost and Schedule

The cost of our project is shown in the table below. We do not require much hardware as our project is mainly based on FPGA. We also take the labor cost into consideration. We expect the team members to work at least 1 hour per day with a salary 45 dollars per hour. Therefore the labor cost is 45 dollars/hour × 3 members × 1 hour/day × 90 days = 12150 dollars.

Description	Manufacture	Cost
Embedded Vision Bundle	Digilent	400$
LED signal circuit	PCB supplier	50$
HDMI cables	KabelDirekt	10$

The total cost = \$12150 + \$460 = \$12610

5. Ethics

Our project, accelerated panorama image stitching camera, upholds the code of ethics I as it has societal implications and great potential applications in dangerous situations.[1]

The image stitching camera can be used for traffic control and cartography. As it provides fast and high quality image output. The image stitching camera also has commercial prospects because it reduces the burden of communication systems as only one panorama is sent per frame contrary to sending several images through the wireless system.

Besides the societal implications, drones equipped with our camera can be deployed in various urgent and perilous natural hazards such as forest fire and earthquake when the wireless stations are shut down or disabled. The information can be quickly processed and the danger is responded to faster compared to traditional methods of communication.

6. Safety

We were aware of the safety and ethical problems that would come with the project. As there are few high buildings in the Champaign urbana area, it's safer to control the drone in an open area with few crowds. We were aware of the intrusion of private properties and right of portrait when experimenting with the camera as it potentially violated the IEEE code 2 "hold paramount the safety, health, and welfare of the public".[2]

The PCB only requires a low, safe voltage. But we should payed attention to testing the board, which might have caused a short circuit and damage to the board. We would follow the fire procedure when the burning gets out of control.

Personal safety is also important when we are working in the lab. We will not engage in experiments with hazardous materials or high voltage electronics, therefore, we will abide by the laboratory safety guidance provided by the University of Illinois.[2] For example, not working alone in the laboratory, not touching wires with two hands, not eating, drinking, or applying cosmetics.

7. Conclusions

While we are thrilled that our camera did function as intended, we also found a few areas where we can improve our design. One flaw of our design is our utilization of the FPGA. Granted, we did hardware accelerate the FAST algorithm which took up a decent portion of the FPGA resources. However, we could have also accelerated the matching algorithm and the image warping algorithm which would have significantly decreased stitching time. We did attempt to do this, but ran into issues with buffer data type sizes. But theoretically, accelerating the matching and warping algorithms is possible. We also could have chosen a much faster matching algorithm. Our matching algorithm, the brute force matcher, is one of the main bottlenecks in our design. Switching to an optimized algorithm such as the Fast Library for Approximate Nearest Neighbors-Based Matcher (FLANN) would remove the bottleneck, especially since it's optimized for the FAST keypoint detection algorithm.

We also discussed expanding the design to support multi-spectral sensor inputs. This would allow for imaging techniques using radar and NDVI that would provide more useful data other than only 2D imaging. We have a base design that can be used for drone aerial mapping once we add a larger frame buffer to support larger maps. But adding more sensors would justify the use of the FPGA in an aerial mapping drone, since the FPGA is able to process large amounts of data quickly.

8. References

[1] A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS. https://docs.xilinx.com/v/u/en-US/xapp1170-zynq-hls

[2] Zybo Reference Manual.
https://digilent.com/refrence/programmable-logic/zybo/reference-manual

[3] Yingen Xiong and Kari Pulli, *Sequential image stitching for mobile panoramas,* Information, Communications and Signal Processing conference 2010.
https://www.researchgate.net/publication/224107399_Sequential_image_stitching_for_mobile_panoramas

[4] Xin Xu and Zhuoqun Chen, *Parallelizing Image Stitching,*
https://github.com/JamesOnEarth/Parallel-Image-Stitching

[5] IEEE code of ethics I. IEEE Policies, Section 7 - Professional Activities (Part A - IEEE Policies). https://www.ieee.org/about/corporate/governance/p7-8.html**.**

[6] J. Crawford, "Easy access to my pi on a local network," *Jordan Crawford*, 31-Oct-2016. [Online]. Available: https://jordancrawford.kiwi/local-address-dns/. [Accessed: 07-Dec-2022].

9. Appendix A: Requirements and Verification Table

Subsystem 1: FAST Keypoint Detection

Requirements:	Verification:
The FAST algorithm should be hardware accelerated to output more than 100 identifiable keypoints per image.	The equipment for verification would be the Xilinx FPGA The camera inputs image arrays to the FAST keypoint accelerator. The output of the algorithm is sent to the web server for visibility and accuracy checks. The keypoint count is printed to the terminal and recorded.
The FAST algorithm should be able to return all images' keypoint arrays in under 30 ms for a clock speed of 200 MHz.	The equipment for verification would be the Xilinx FPGA. The code is edited to include the built in Linux chrono library for timing. The FPGA runs the program and prints the time taken by the FAST functions to the terminal.

Subsystem 2: Image Warp

Requirements:	Verification:
From the returned keypoint arrays from the hardware, the SOC is able to compute homography, warp, and stitch the panorama in under 10 seconds.	The equipment for verification would be the Xilinx FPGA. The code is edited to include the built in Linux chrono library for timing. The FPGA runs the program and prints the time taken by the homography and warping functions to the terminal.
From the identifiable key points, the SOC calculates the homography matrix, and performs the transformation. The software should be able to warp and blend the panoramas to where end-users could only notice the difference if they stepped closed and searched for artifacts in the image, disregarding black space for background.	The webserver hosted by the SOC would be used for this test. The image is inputted into the Linux environment from the accelerator. The image is sent to the web host and viewed on the monitor. The results of the image can be qualitatively observed and recorded by the users.

	The image on the web server can be used to verify the projective transformation. Some signs of error we would pay attention to: 1. Differences in Light Exposure leading to color mismatches. 2. Differences in rotation of images leading to correctional warping rather than distorting the image.

Subsystem 3: Web Server

Requirements:	Verification:
Users can access a web page that displays all the images taken by the camera. Users can view the original images captured, the greyscale images with defined key points, images tracing the matched keypoints, and the final output panorama. Users should also be able to click on an image to enlarge it and inspect the details.	To connect to the web server, users must set their computer's NIC to any IP in the 10.10.10.X subnet. Then after connecting an ethernet cord between the computer's NIC and the FPGA's NIC, they can access the image gallery web page by going to the following URL: 10.10.10.3/gallery/gallery.html. 10.10.10.3 is the static IP of the FPGA.
All images that are successfully stitched are sent to the webserver from the stitching program, any images that failed the stitching process should not be sent and the stitching process should start from the beginning.	The user can take three pictures, moving the camera left to right after each picture. Then assuming the stitch was successful, the results are copied to the web server directory. If the user covers the camera lens with their finger and takes three pictures, the program won't be able to stitch the images and restarts. Since the stitching failed, the images are not copied to the web server directory.

Subsystem 4: Camera

Requirements:	Verification:
Users can use the push buttons to trigger the camera to capture a picture, then once all three images are taken, the images are	Once the program loads, all 4 status LEDs are lit signifying that the application is ready to capture pictures. The user then

stitched together and the program waits for the user to take the next round of pictures.	can press pushbutton 1 to start the capture of image 1. Once the capture is over, only status LED 1 is lit, signifying the image is saved in RAM. The same process is repeated for capturing images two and three, except the user will use pushbuttons and LEDs 2 and 3 to finish capturing the pictures.
If the user captures images that can't be stitched together, the application notifies the user by turning on LED 4, signifying that the application has reset and 3 new images must be taken.	The user can capture 3 images while keeping their finger over the lens, and once the stitching application returns 0 key points to be paired, LED 4 will turn on telling the user to capture 3 new images.

Subsystem 5: Light Exposure Circuit

Requirements:	Verification:
The LED is controlled by a switch and is supplied with 5 Volts from the battery. The PCB provides consistent light for photo taking.	We would test the functionality of the PCB with the lab equipment. 1. Supply 5V DC to the PCB and ensure LED lights up without short circuit or disconnection on the board.
Switching of the circuit is controlled by a hardware switch or the GPIO of the FPGA.	The GPIO is easier to work with internal signals on the FPGA and requires a driver and relay circuit therefore, we finally decided to use a hardware switch for simplicity and easier control over the circuit. 1. Switch the circuit and test the durability.