Sabancı Üniversitesi

Term Project - EE417

Stereo Panorama
Image Stitching - Depth Estimation

Barış Almaç
Mert Ekici

Supervisor: Dr. Mustafa Ünel
Faculty of Engineering and Natural Sciences

January 20, 2023

Contents

1 Introduction

Panoramic imaging is an increasingly important topic in computer vision as it allows for capturing wide-angle views of a scene. However, current techniques for panoramic image stitching often suffer from distortion and loss of depth information. In this project, we aimed to implement, test and compare various tools of panoramic image stitching, rectification, disparity, despth estimation and stereo panoramic view. Then, evaluate the performance of single camera panoramic stitched images' depth estimation with stereo panoramic images disparity maps and other helpful tools to provide more insight on the concepts of stereo panoramic image stitching and depth estimation.

The main objective of this project is to improve the quality of panoramic images and make them more useful for a variety of applications. The proposed methods will incorporate image stitching and depth adjustment techniques to create a seamless panoramic image with accurate depth information. Other objectives of this project were to understand the problems that occur during the process of image stitching, stereo panorama, depth estimation, and 360-degree view visualizations and to provide solutions for future research and applications. To accomplish these objectives, we used existing libraries such as OpenCV, Intel's MiDaS, and an online stereopanorama viewer tool, as well as in-lab codes written in MATLAB and an implemented algorithm to calculate disparities of a left and right image. The project aligns with the curriculum of computer vision by applying computer vision techniques to solve a real-world problem. This research is motivated by the need for an accurate and visually pleasing representation of the scene, as stereo panoramic images can provide even more information about the scene by capturing depth information. It is important to note that while the proposed methodology and evaluation provided insight into the concepts of stereo panoramic image stitching and depth estimation, unexpected challenges may have arisen during the course of the project. The project researchers had to carefully consider and address these issues in order to achieve the goal of providing a comprehensive evaluation of stereo panoramic image stitching and depth estimation.

The literature review of previous research in the field of stereo panoramic image stitching and depth estimation showed that there are several challenges in the process of image stitching. One of the most important challenges is to find a way to accurately align and stitch images together. This process is crucial in ensuring that the resulting panoramic image is seamless and holds accurate depth information. In "Stereo panoramic image stitching with a single camera" by Cho et al. the authors proposed a single-camera stereo panoramic image stitching method that extracts feature points using SIFT (Scale-Invariant Feature Transform) and matches them using a ratio test. Then, they use the estimated homography matrix to warp and blend the images. In "Real-time panoramic depth maps from omni-directional stereo images for 6 dof videos in virtual reality" by Lai et al. the authors proposed a method to generate real-time panoramic depth maps from omni-directional stereo images for 6 degree of freedom videos in virtual reality. The method involves the use of a deep learning model trained on a dataset of omni-directional

stereo images, which is able to estimate depth maps in real-time. In "Megastereo: Constructing high-resolution stereo panoramas" by Richardt et al. the authors proposed a method for constructing high-resolution stereo panoramas using a single camera and a mirror ball. The method involves capturing multiple images of a scene using a mirror ball, and then using these images to construct a high-resolution stereo panorama. In "Post-stitching depth adjustment for stereoscopic panorama" by Wang et al. the authors proposed a method for adjusting the depth of stereoscopic panoramas after they have been stitched. The method involves adjusting the depth of each image in the panorama individually, in order to ensure that the final panorama has accurate depth information. In "Casual stereoscopic panorama stitching" by Zhang and Liu the authors proposed a method for stitching stereoscopic panoramas that is robust to camera motion and partial overlap. The method uses a combination of feature matching and RANSAC (Random Sample Consensus) to estimate the homography matrices between images.

Overall, the project aimed to provide a comprehensive evaluation of stereo panoramic image stitching and depth estimation by researching and implementing existing libraries and in-lab codes, and testing on a variety of stereo panorama images. The findings of the project highlighted the problems that occur during the process and the potential for future research and applications in the field of computer vision.

2 Methods

The proposed evaluative study of stereo panoramic image stitching and depth estimation will involve a combination of literature review, implementation of existing libraries, designed algorithms and in-lab codes, and testing on a variety of stereo panorama images. The following steps will be taken to achieve the objective of the project:

1. Literature review: This step will involve a thorough literature review of previous research in the field of stereo panoramic image stitching and depth estimation. This review will provide the theoretical background and the state of the art techniques for our project.
2. Image preparation: In this part, several images are obtained around the campus with a single non-stereo smartphone camera and additional image datasets provided from the Megastereo research are also utilized in this project to be tested as inputs from the implemented and developed algorithms.
3. Implementation and testing of existing libraries and in-lab codes with additionally designed algorithms: This step will involve the implementation of existing libraries and in-lab codes on a dataset of stereo panorama images. The images will be selected based on their suitability for testing the proposed evaluation. The implemented libraries and codes will be tested on these images to evaluate the performance of single camera panoramic stitched images' depth estimation with stereo panoramic images disparity maps.
4. Panoramic Image Stitching: Slided photos are stitched panoramically. These photos were taken with a single camera, with the goal of obtaining a stereo panoramic

image. Stitching is conducted by feature extraction and feature matching algorithms of OpenCV. Lastly, output image is trimmed (in the black regions around the corners) into a proper rectangular shape.

5. Depth Estimation: Disparity maps and red-cyan panoramic visuals are formed with both built-in Matlab and OpenCV functions and also compared with the enhanced in-lab disparity algorithm. Furthermore, same input images (mostly panoramic or stereopanoramic) are used with a pyTorch ML model called MiDaS by Intel ISL for depth estimation. Once the stereo panoramic images have been processed, disparity maps and red-cyan panoramic visuals will be created to visualize the depth information. This step will involve the use of existing libraries or software, (such as Intel's MiDaS) and an online stereopanorama viewer tool. These tools will be used to generate disparity maps from the stereo panoramic images, which will be used to evaluate the accuracy of the depth estimation. Additionally, the red-cyan panoramic visuals will be created to provide a visual representation of the depth information.
6. Comparison of results: The results of the proposed evaluation will be compared in the field of stereo panoramic image stitching and depth estimation. This comparison will provide insight into the performance of the proposed evaluation and its potential for future applications. Additionally, the results will be analyzed to identify any limitations or challenges that may have been encountered during the evaluation process.
7. Prepare final report: In the final stage of the project, a final report will be prepared to summarize the findings and conclusions of the study. The report will include a description of the proposed evaluation, the results of the tests, and a discussion of the implications of the study for future research and applications.

In summary, our methodology involved the use of existing libraries and in-lab codes for image stitching, feature matching, depth estimation and disparity map calculation. We used mathematical and algebraic notation to explain the process in detail and also referenced formal scientific sources to provide a deeper understanding of the concepts. We also tested our results on a variety of stereo panorama images to evaluate the performance of the algorithm and to identify potential areas for improvement.

2.1 Panorama Stitching

The first step in the project was to take photos to be stitched panoramically. These photos were taken with a single camera, with the goal of obtaining a stereo panoramic image.

In order to stitch the images together, we used the OpenCV library in python to perform feature matching. In our methodology, we used the Scale-Invariant Feature Transform (SIFT) algorithm for feature extraction from the images. The SIFT algorithm is based on the difference of Gaussian (DoG) function, which is calculated by subtracting the result of two Gaussian filtered images with different standard deviations. The DoG function is used to detect local extrema, which are then used as keypoints in the image. The SIFT descriptor is then calculated for each keypoint by considering the gradient

information in a local region around the keypoint. The SIFT algorithm is a robust feature detection method that is invariant to image scale, rotation and affine distortion.

For feature matching, we used the BFMatcher and knnMatch functions provided by the OpenCV library. The BFMatcher function uses the brute-force method to match the SIFT descriptors between the images, while the knnMatch function uses k-nearest neighbor matching to find the best matches between the descriptors.

Once the feature points were matched, we used the cv2.Stitcher function to stitch the images together. The Stitcher class provides an interface for stitching a set of images. The class uses the matched feature points to estimate the images' relative positions and then warps and blends the images to create the panoramic image.

Finally, we developed a code for trimming an image after panorama stitching to remove black regions around the corners of the stitched image. It does this by using image processing techniques such as thresholding, contour detection, and erosion.

The first code segment is using feature matching to match the same features in two images. It starts by reading two images, resizing them to a specific height, and converting them to grayscale. Then it uses the SIFT algorithm to detect keypoints and compute descriptors for each image. The keypoints and descriptors are used to match similar features in the two images.

The matching algorithm used is the KNN-matching, it takes the two descriptor sets and finds the two closest descriptors for each descriptor in the first image. Then it filters the matches using a ratio test, which only keeps the matches where the distance of the closest descriptor is less than 35

The final step is to draw the matches between the two images and show the result.

The second code segment is for stitching the images together. It starts by importing the necessary libraries and defining a function called 'trim'. The function takes an image as input, which is supposed to be the stitched image, and it applies a border to the image, converts it to grayscale and applies a threshold. Then it finds the contours of the image, finds the largest contour which is the contour of the stitched image, creates a mask for this contour and removes the black pixels outside of the mask. Finally, it shows the result.

1. The input image is first copied and a border of 10 pixels is added around it.
2. The image is converted to grayscale, and then a threshold is applied to it such that all pixels greater than zero are set to 255 (foreground) while all others remain 0 (background).
3. All external contours in the threshold image are found, and the largest contour is identified, which will be the contour/outline of the stitched image.
4. A rectangular bounding box is created around the stitched image region.
5. The minimum rectangular region is created by eroding the mask and subtracting the thresholded image from the minimum rectangular mask.
6. The final stitched image is extracted using the bounding box coordinates.

The purpose of the algorithm is to remove black regions around the stitched image that are created by the panorama stitching process. It does this by identifying the region of interest in the image and cropping the image to that region.

The first code segment (Appendix 1) is using feature matching to match the same features in two images. It starts by reading two images, resizing them to a specific height, and converting them to grayscale. Then it uses the SIFT algorithm to detect keypoints and compute descriptors for each image. The keypoints and descriptors are used to match similar features in the two images. The matching algorithm used is the KNN-matching, it takes the two descriptor sets and finds the two closest descriptors for each descriptor in the first image. Then it filters the matches using a ratio test, which only keeps the matches where the distance of the closest descriptor is less than 35% of the distance of the second closest descriptor. This step is important to remove false matches. The final step is to draw the matches between the two images and show the result.

The second code segment (Appendix 2) is for stitching the images together. It starts by importing the necessary libraries and defining a function called 'trim'. The function takes an image as input, which is supposed to be the stitched image, and it applies a border to the image, converts it to grayscale and applies a threshold. Then it finds the contours of the image, finds the largest contour which is the contour of the stitched image, creates a mask for this contour and removes the black pixels outside of the mask. Finally, it shows the result.

2.2 Rectification and Disparity

Image rectification is a process in which two images of a scene taken from different viewpoints are aligned so that corresponding points in the images have the same position. This is done by applying a geometric transformation to one or both of the images, such as a rotation, translation, or perspective transformation. One common application of image rectification is in stereo vision, where two images of a scene are taken from slightly different viewpoints and used to create a 3D representation of the scene. In order to create an accurate 3D representation, the stereo images must be rectified so that corresponding points in the images are aligned. In our project, we used the uncalibrated rectification method.

Image rectification for uncalibrated pictures is a process of aligning two or more images that were taken from different viewpoints without the knowledge of the camera parameters. This can be a challenging task because the camera parameters, such as the intrinsic and extrinsic parameters, are not known. Our approach for rectifying uncalibrated images is to use feature-based methods that rely on detecting and matching features across the images. This involves identifying corresponding points in the different images and then estimating the geometric transformations that align these points. Once the transformations have been estimated, they can be applied to the images to rectify them.

Disparity mapping is the process of determining the distance of each point in a scene from the camera based on the difference in position of that point in the two rectified

stereo images. The motivation for aligning stereo images for the disparity is that it allows for more accurate depth perception. If the images are not aligned, the disparity will be incorrect and the resulting depth map will be inaccurate. By rectifying the stereo images, the disparities are more accurate and the depth map is more reliable.

In addition to using the single camera panoramic images, we also used stereo panorama images from a dataset provided by "Megastereo: Constructing high-resolution stereo panoramas" paper to derive their disparity maps for depth estimation. We used our in-lab MATLAB code and the algorithm we implemented to calculate the disparities of the left and right images. The disparity map is a representation of the depth information of the scene, where each pixel in the map corresponds to the difference in horizontal position of a corresponding feature in the left and right images. The disparity map is calculated using the following mathematical equation:

$$d(x, y) = xL - xR$$

Where d(x,y) is the disparity of a pixel at position (x,y), xL is the x-coordinate of the same feature in the left image, and xR is the x-coordinate of the same feature in the right image.

Code for Disparity Mapping is implemented within the following steps:

1. The code first loads two images (ImLeft and ImRight) and converts them to grayscale. It then sets some variables: k and omega are constants used in the calculation of the similarity between sub-images, and offset is the amount by which the images will be padded.
2. The code then pads the images by offset amount on all sides using the padarray function. This adds a border of zeros around the images.
3. Next, the code initializes a zero matrix called dispar of the same size as the padded images. This matrix will be used to store the calculated disparity values for each pixel in the image.
4. The code then enters a nested loop where it iterates over all pixels in the left image, starting from the offset-th column and row and ending at the xdim-offset-1 and ydim-offset-1 columns and rows, respectively.
5. For each pixel in the left image, the code extracts a small sub-image centered at that pixel (subL) and another sub-image of the same size in the right image at the same vertical position but starting omega pixels to the left (subR).
6. It then calculates the similarity between these two sub-images using sum of squared differences (SSD) and stores the result in a matrix called dist along with the displacement d (i.e., the difference in the horizontal positions of the sub-images in the left and right images).

The similarity between the sub-images for each displacement d in R as follows:

$$C(d) = \sum_{k=-W}^{k=W} \sum_{l=-W}^{l=W} \Psi(f(i+k, j+l), g(i+k, j+l))$$

Where ψ is the similarity measure such as SSD which can be calculated as follows:

$$\Psi(f, g) = -(f - g)^2$$

7. Once all sub-images have been processed, the code finds the maximum similarity value in the dist matrix and uses it to determine the corresponding displacement value d. This value is then stored in the dispar matrix at the position corresponding to the center of the left sub-image.
8. After the loop is finished, the code displays the original stereo pair as a red-cyan anaglyph and the calculated disparity map with a colorbar, Then results are compared with built-in disparity map result and the better output (accurate estimation) is selected.

To further visualize the depth information, we also created red-cyan anaglyphs for each stereo panorama image. Anaglyphs are stereo images that are created by superimposing two images taken from slightly different angles, with one image being in red and the other in cyan. By viewing the anaglyphs through red-cyan glasses, the viewer can perceive the depth information of the scene.

2.3 Depth Estimation

The next step in the process was to use the trimmed stitched panoramic images to estimate the depth of the scene. We passed the images to Intel's MiDaS, which is a monocular depth estimation model that uses a deep convolutional neural network (CNN) to predict depth maps in real-time. We used the pre-trained model provided by Intel to estimate the depth of the scene and saved the results. MiDaS computes relative inverse depth from a single image. The repository provides multiple models that cover different use cases ranging from a small, high-speed model to a very large model that provide the highest accuracy. The models have been trained on 10 distinct datasets using multi-objective optimization to ensure high quality on a wide range of inputs. Its functionality is utilized in this project with an official Python notebook (Google Colab) and the pre-trained model is used for obtaining accurate AI-augmented depth estimations.

2.4 Stereo-Panorama Viewer

Finally, we used the stereo panorama images released by Megastereo paper to create 3D VR visualizations by using an online stereopanorama viewer tool. These visualizations allowed us to further examine the depth information of the scene and to compare the results of the single camera panoramic images' depth estimation with the stereo panoramic images' disparity maps. The web ui simply takes an input left-right stereo panorama image (left-top and right-bottom) and visualizes it in a 360-degree VR view.

3 Results

3.1 Panorama Stitching

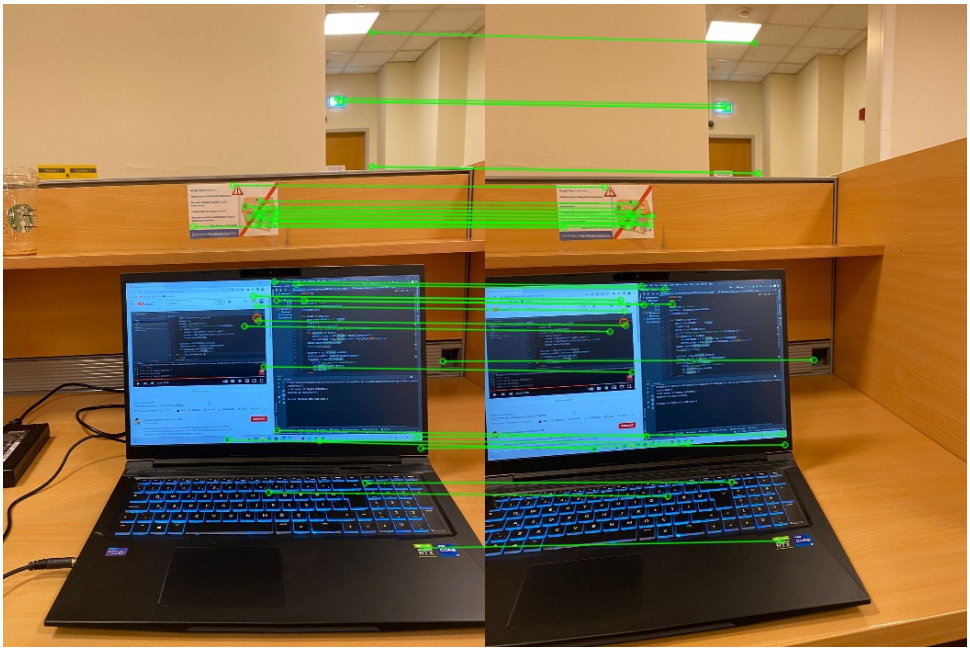

Figure 1: Feature Matching - I

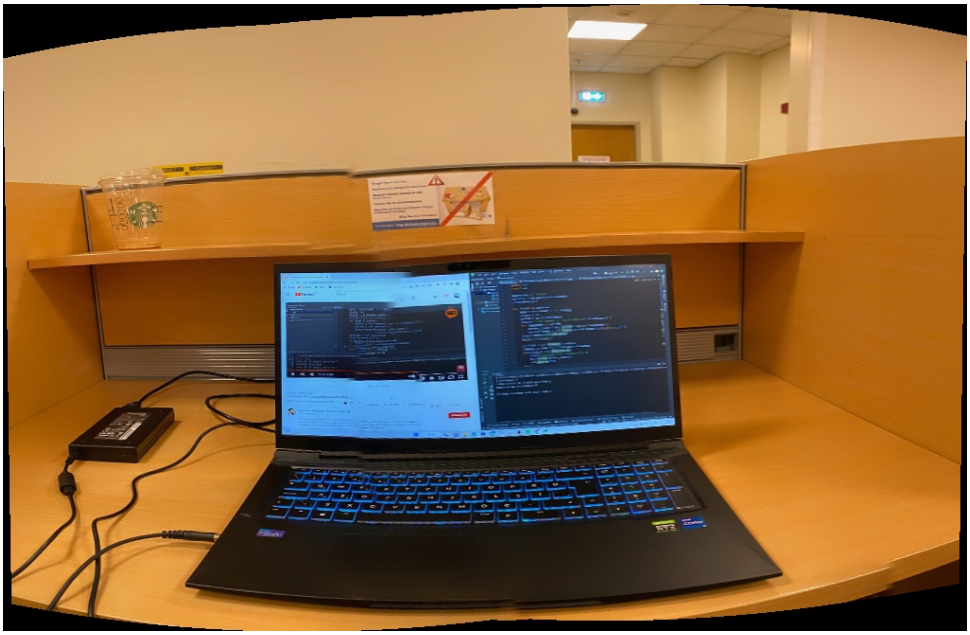

Figure 2: Stitched Image - I

Figure 3: Trim Step 1 - I

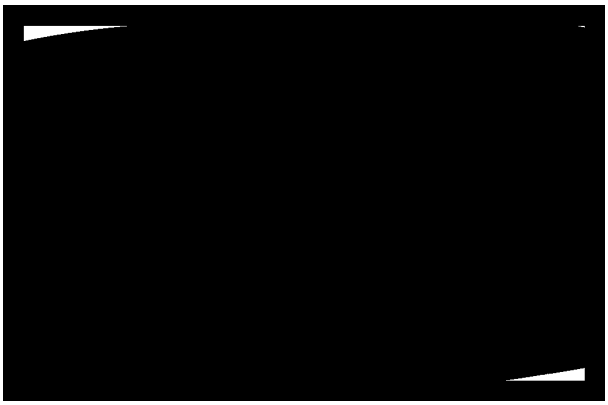

Figure 4: Trim Step 2 - I

Figure 5: Trim Step 3 - I

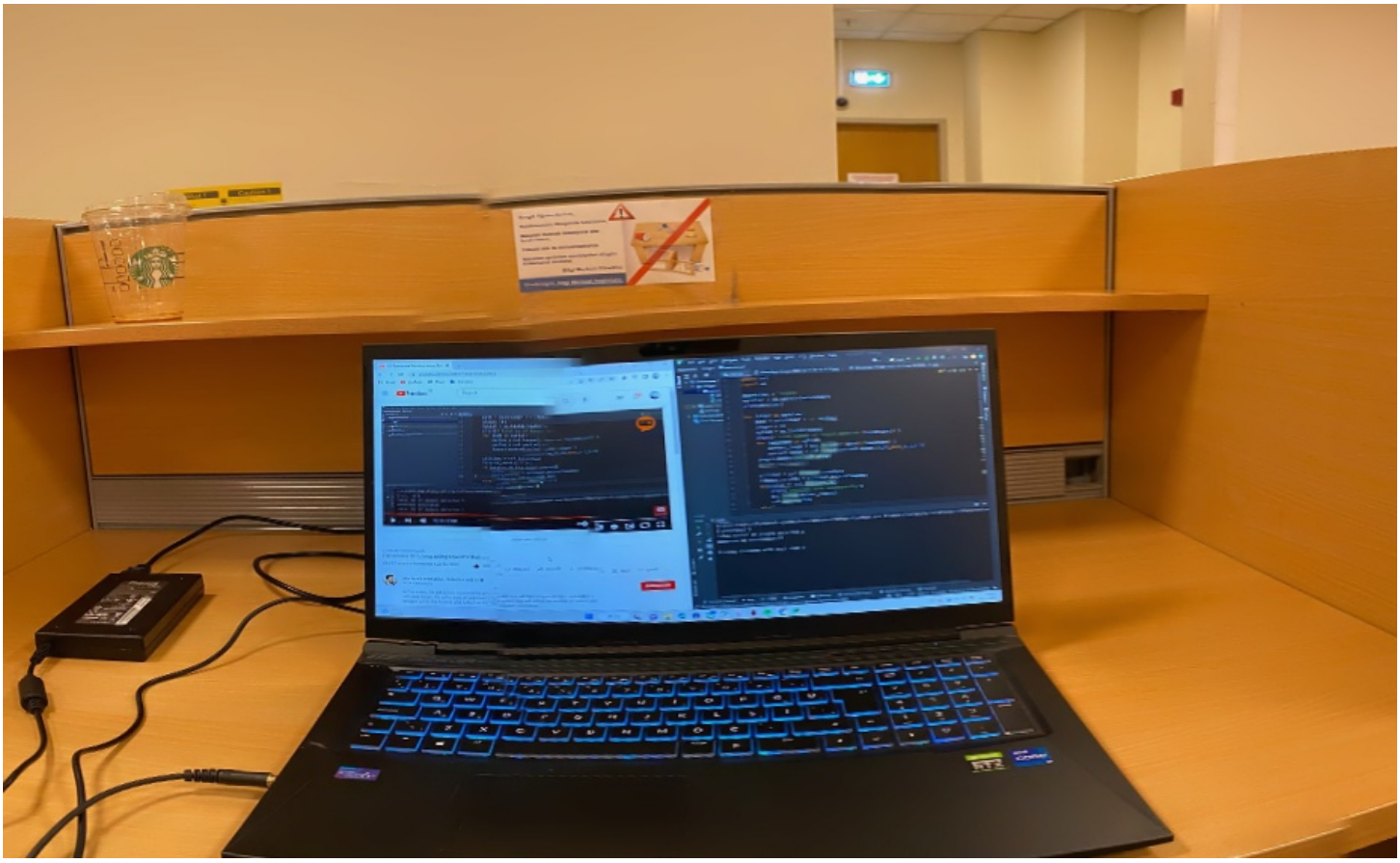

Figure 6: Panorama - I

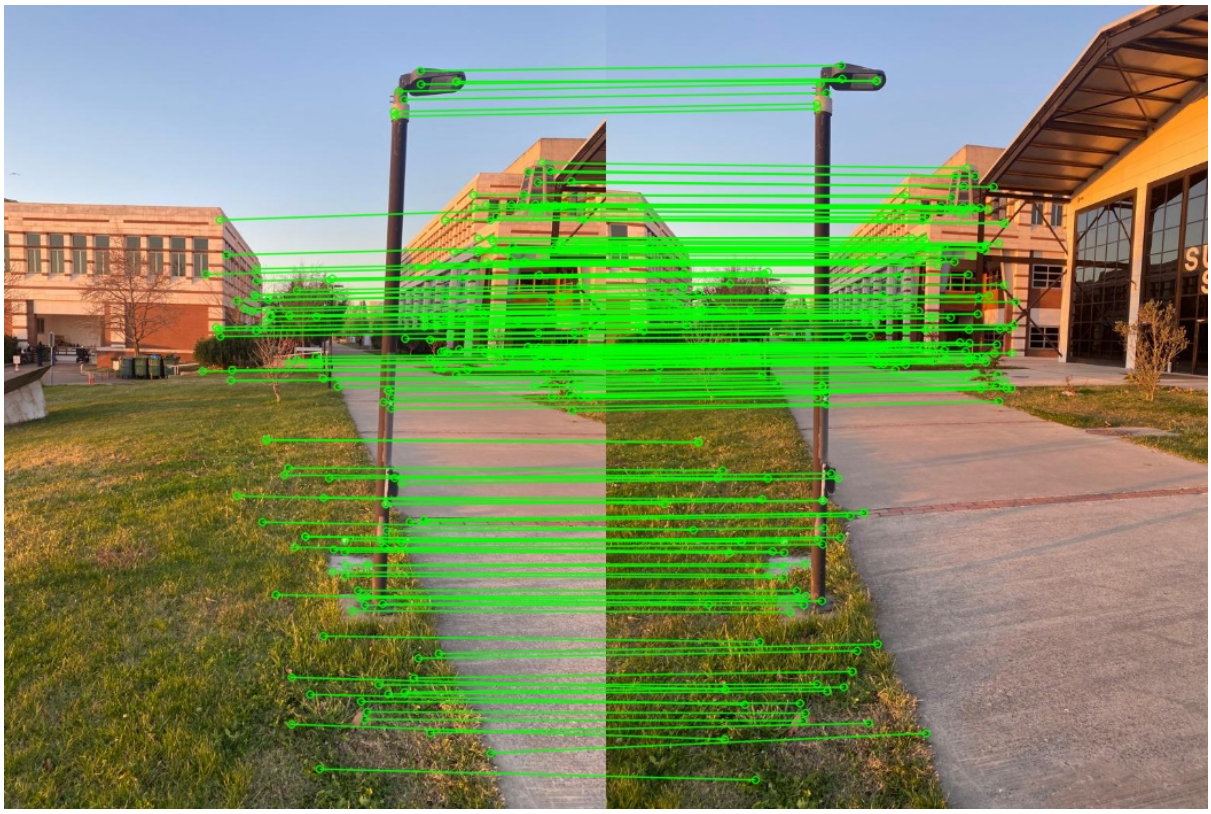

Figure 7: Feature Matching - II

Figure 8: Stitched Image - II

Figure 9: Panorama - II

Figure 10: Stitched Image - III

Figure 11: Panorama - III

Figure 12: Stitched Image - IV

Figure 13: Panorama - IV

3.2 Rectification and Disparity

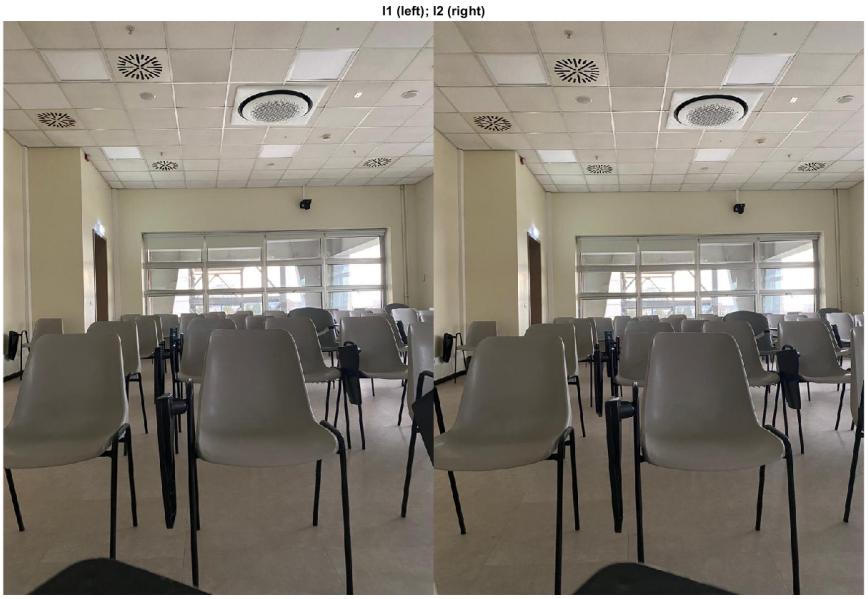

Figure 14: LR Image - I

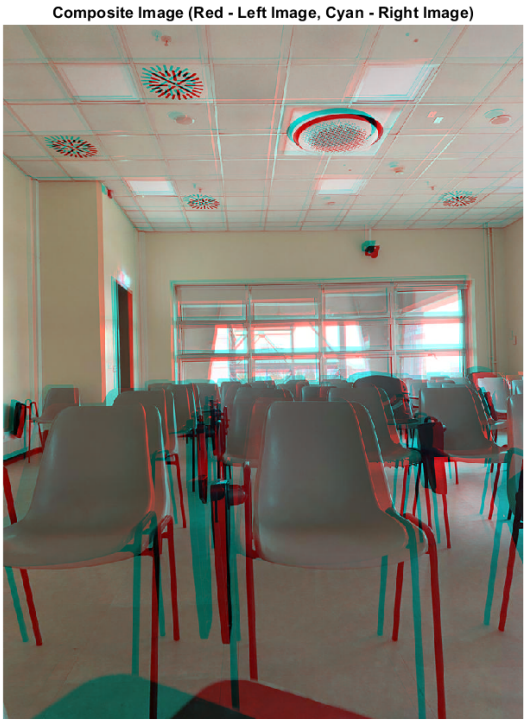

Figure 15: Anaglyph - I

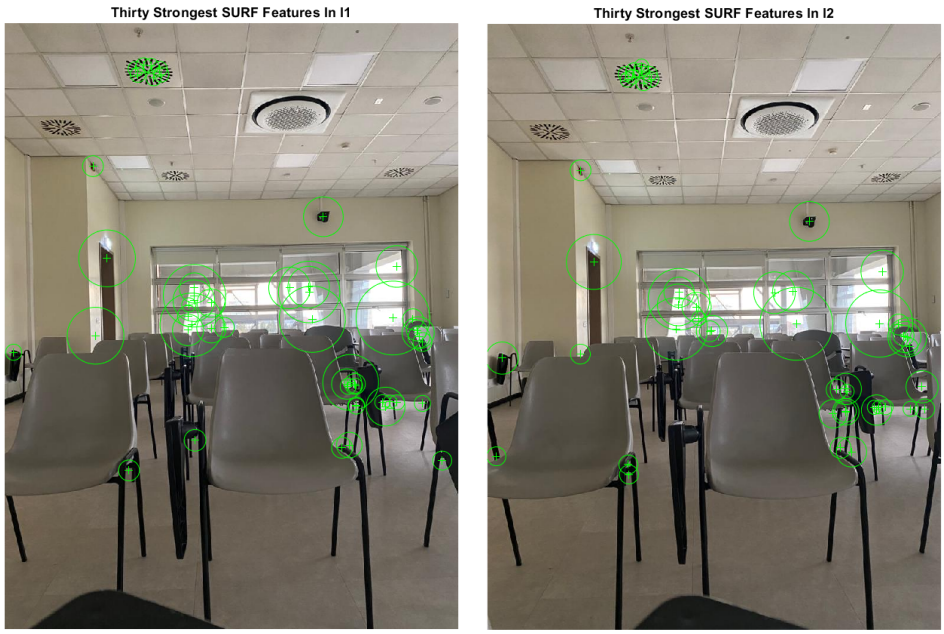

Figure 16: SURF Features - I

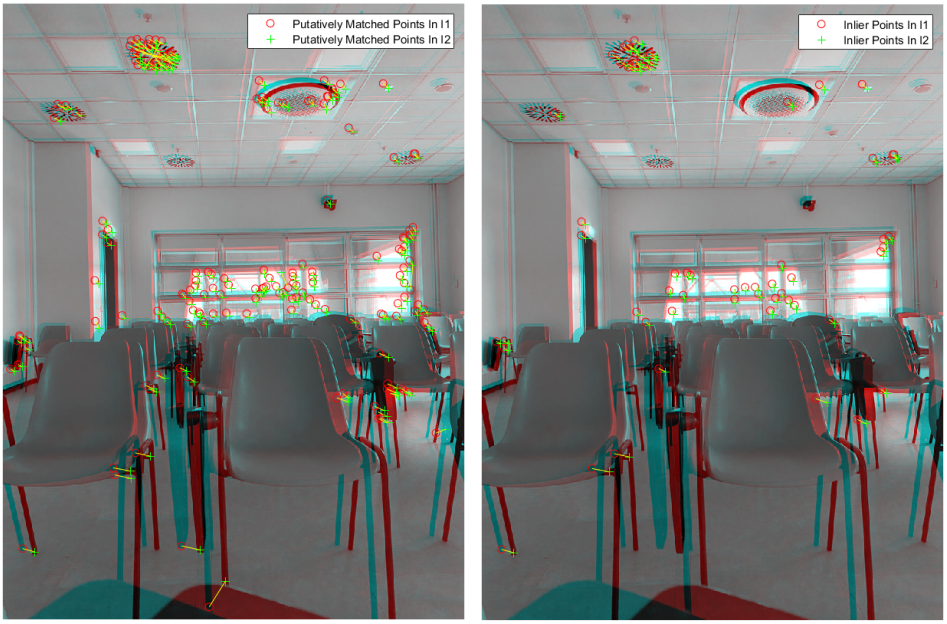

Figure 17: Matched Points - I

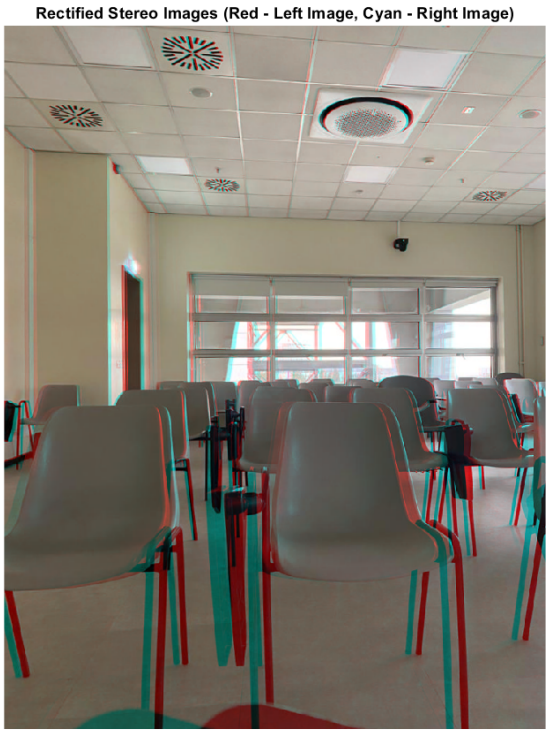

Figure 18: Rectified - I

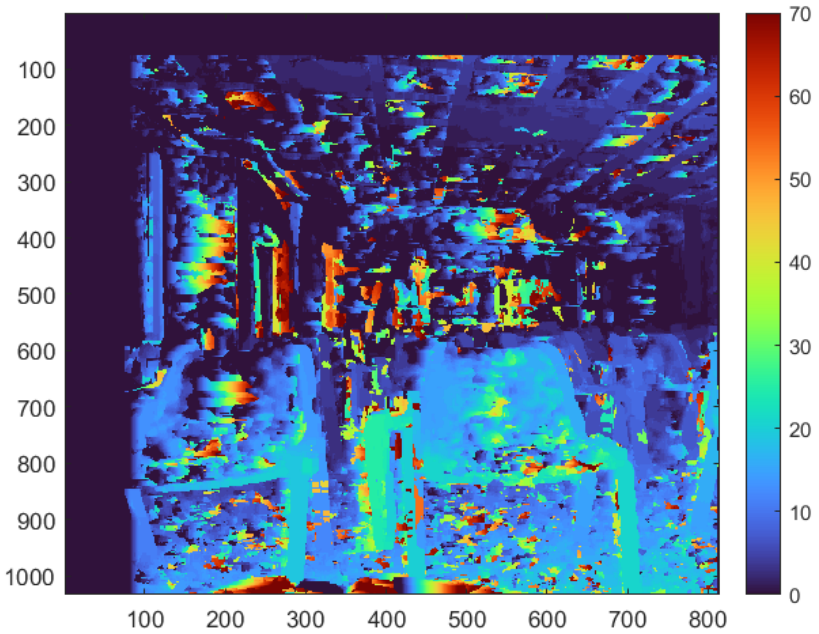

Figure 19: Disparity Map (Developed Algorithm) - I

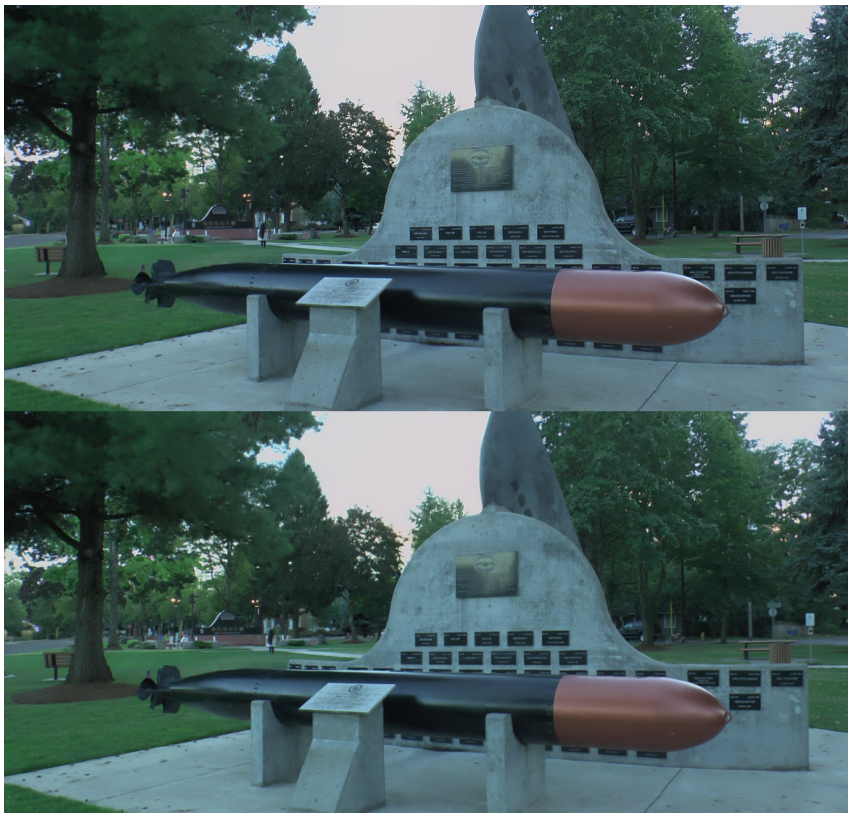

Figure 20: LR Image - II

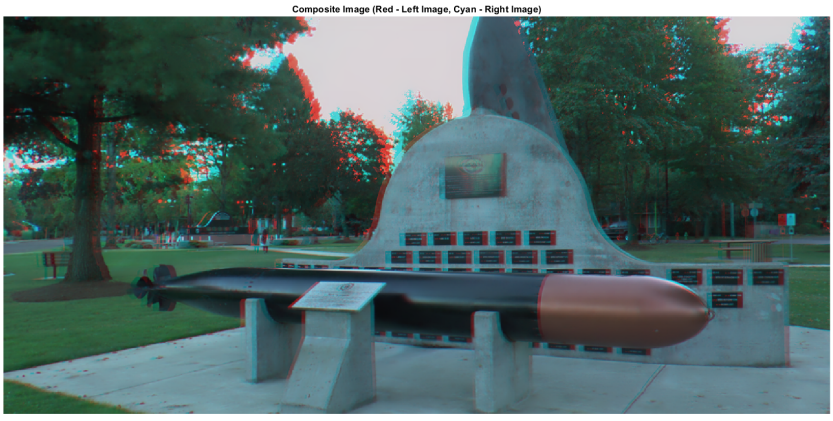

Figure 21: Anaglyph - I

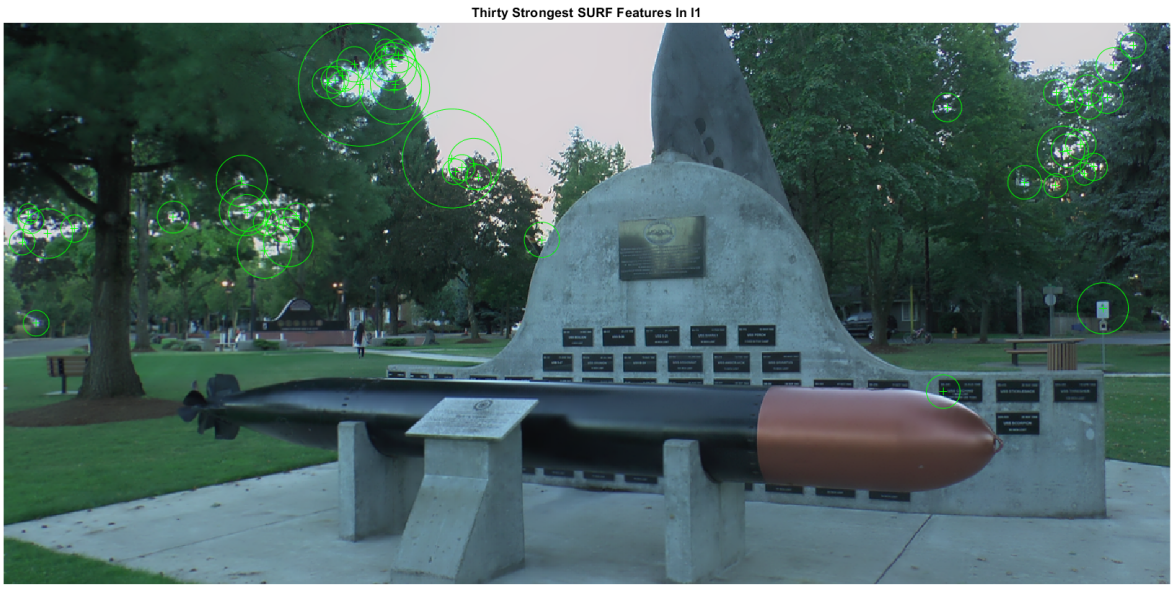

Figure 22: SURF Features Left - II

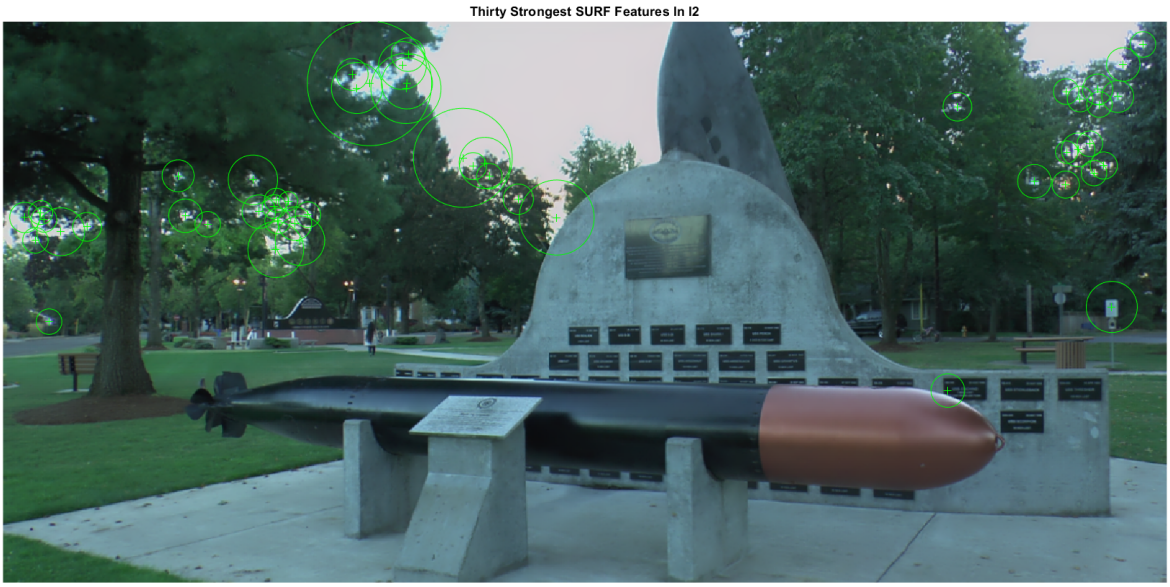

Figure 23: SURF Features Right - II

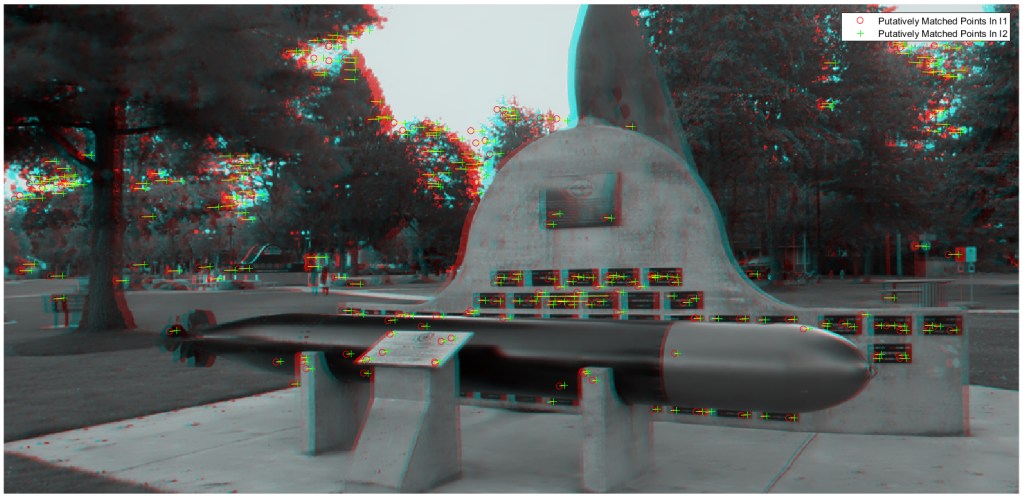

Figure 24: Matched Points Left - II

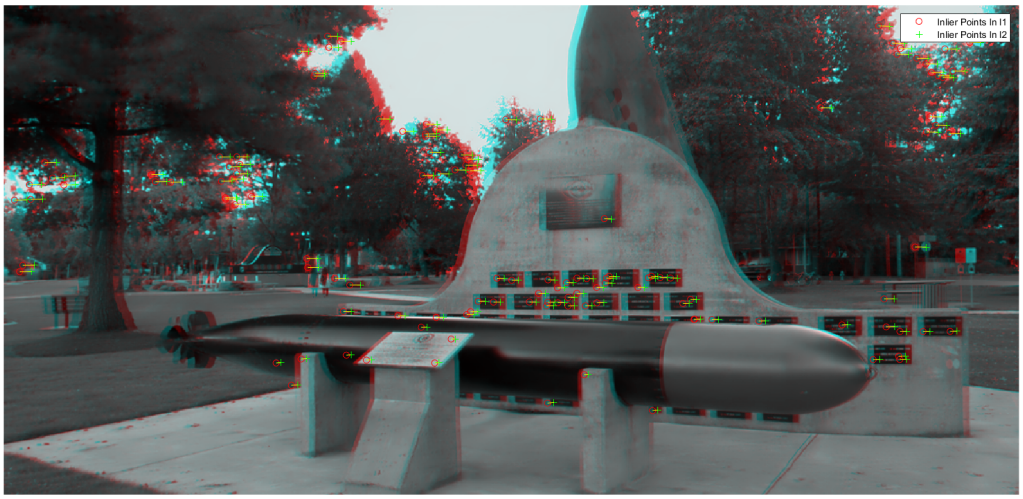

Figure 25: Matched Points Right - II

Figure 26: Rectified - II

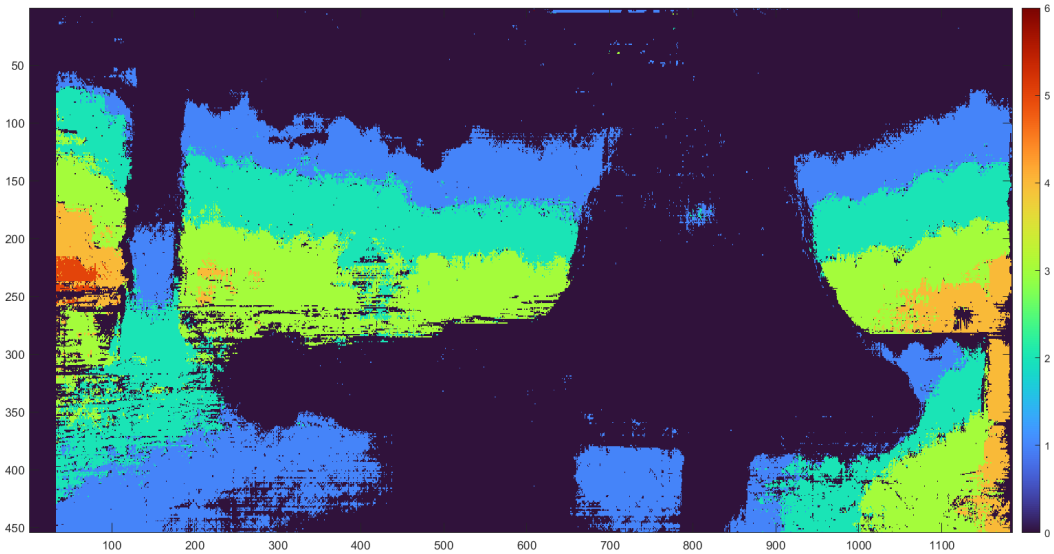

Figure 27: Disparity Map (Built-In) - II

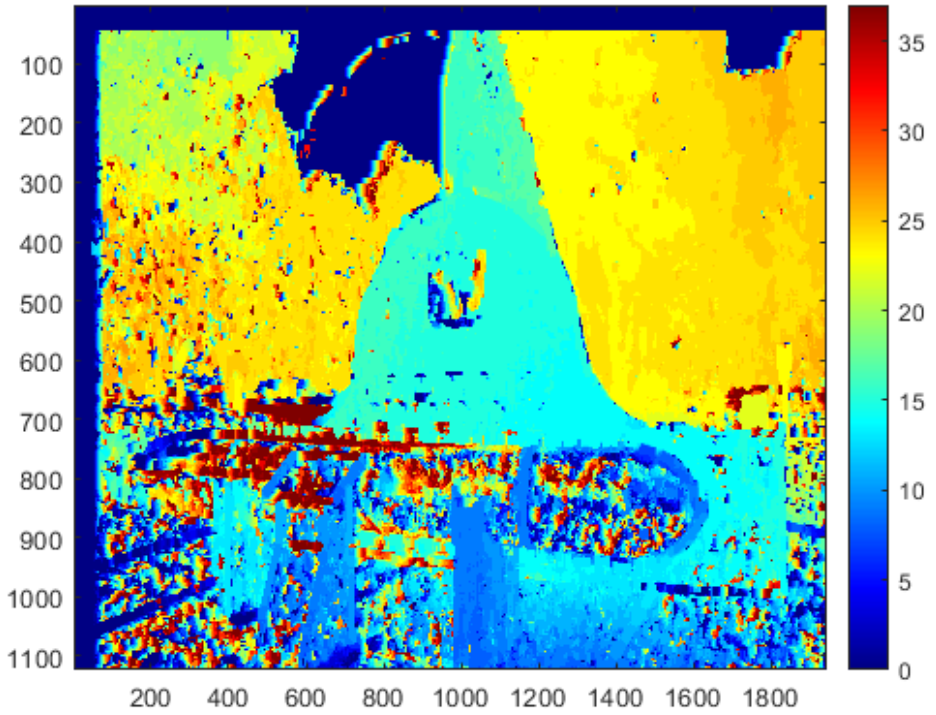

Figure 28: Disparity Map (Developed Algorithm) - II

3.3 Depth Estimation

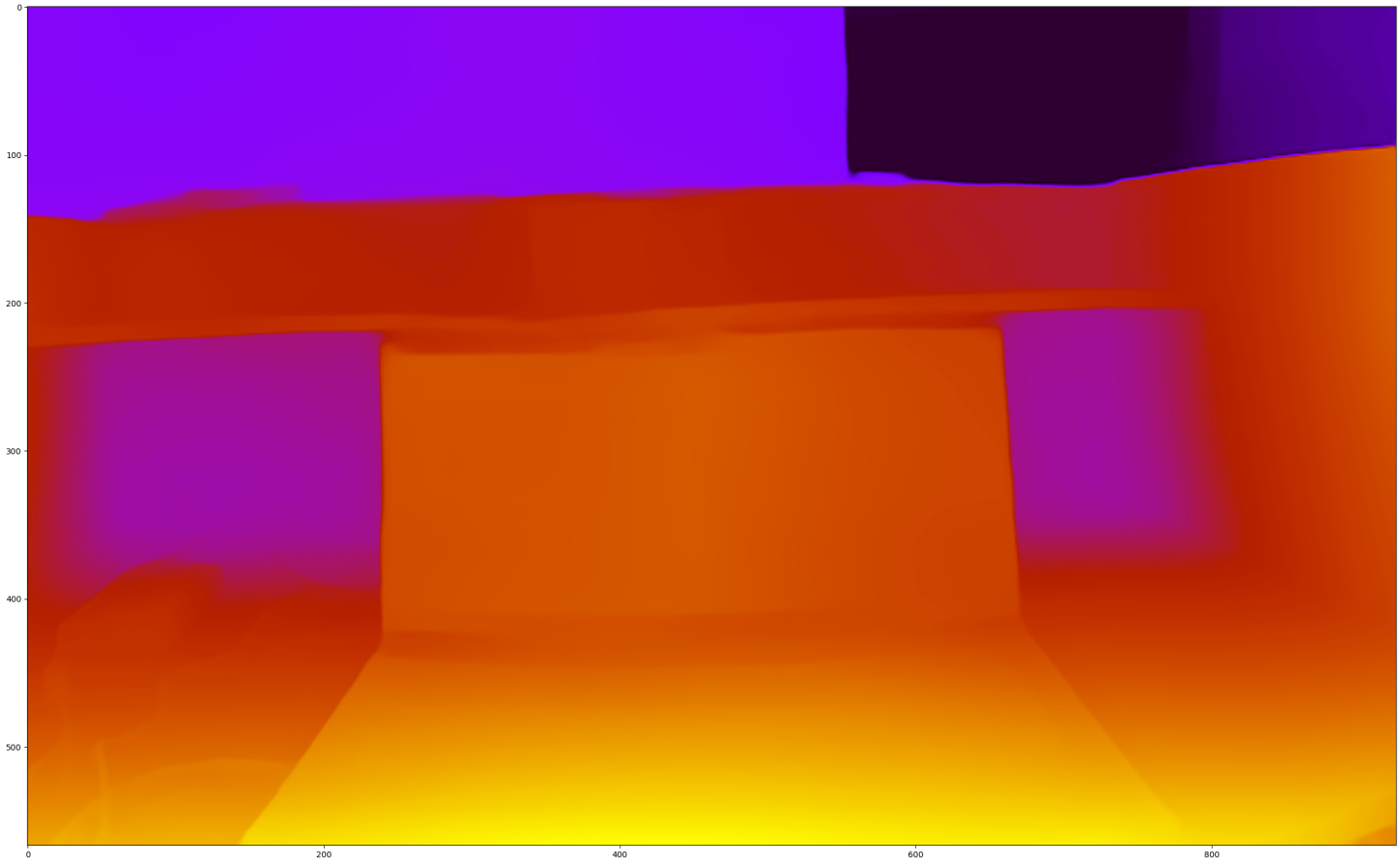

Figure 29: Depth Estimation - I

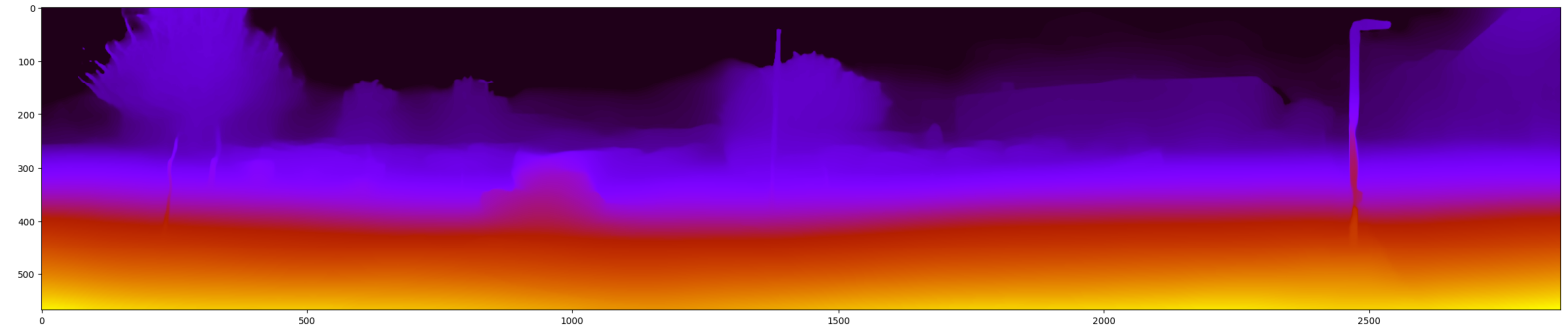

Figure 30: Depth Estimation - II

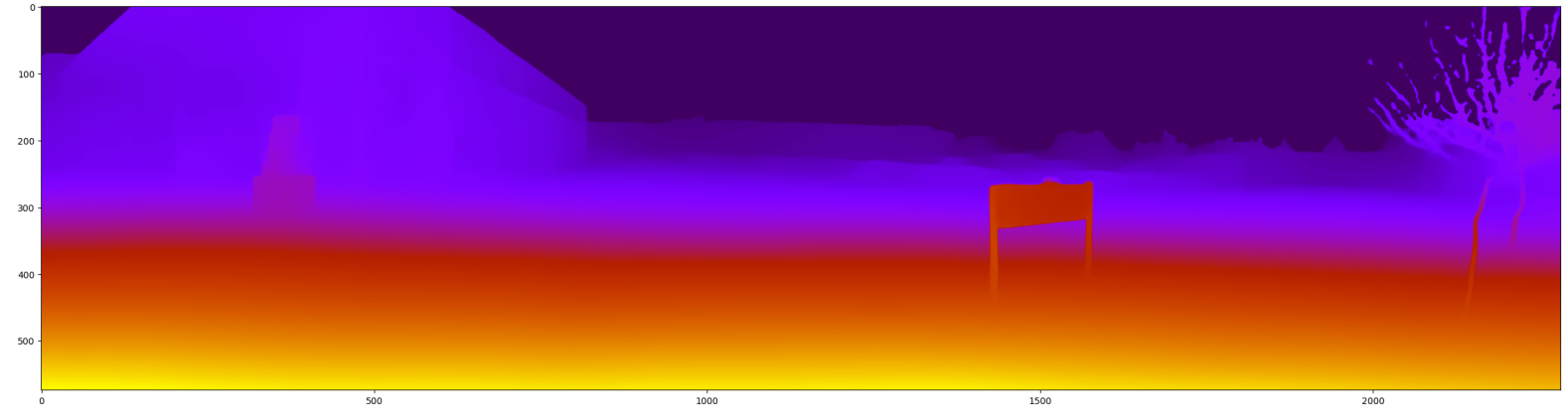

Figure 31: Depth Estimation - III

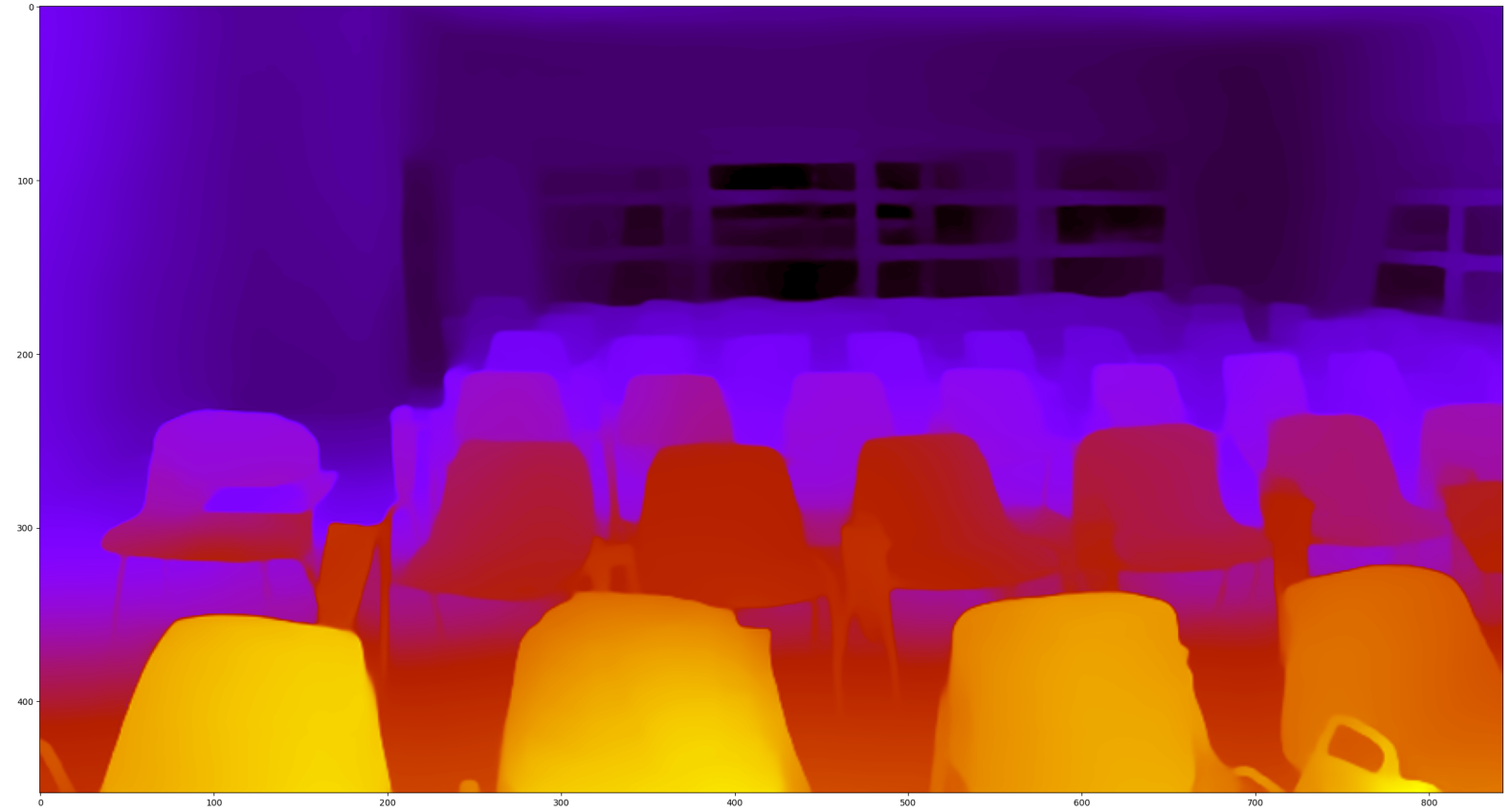

Figure 32: Depth Estimation - IV

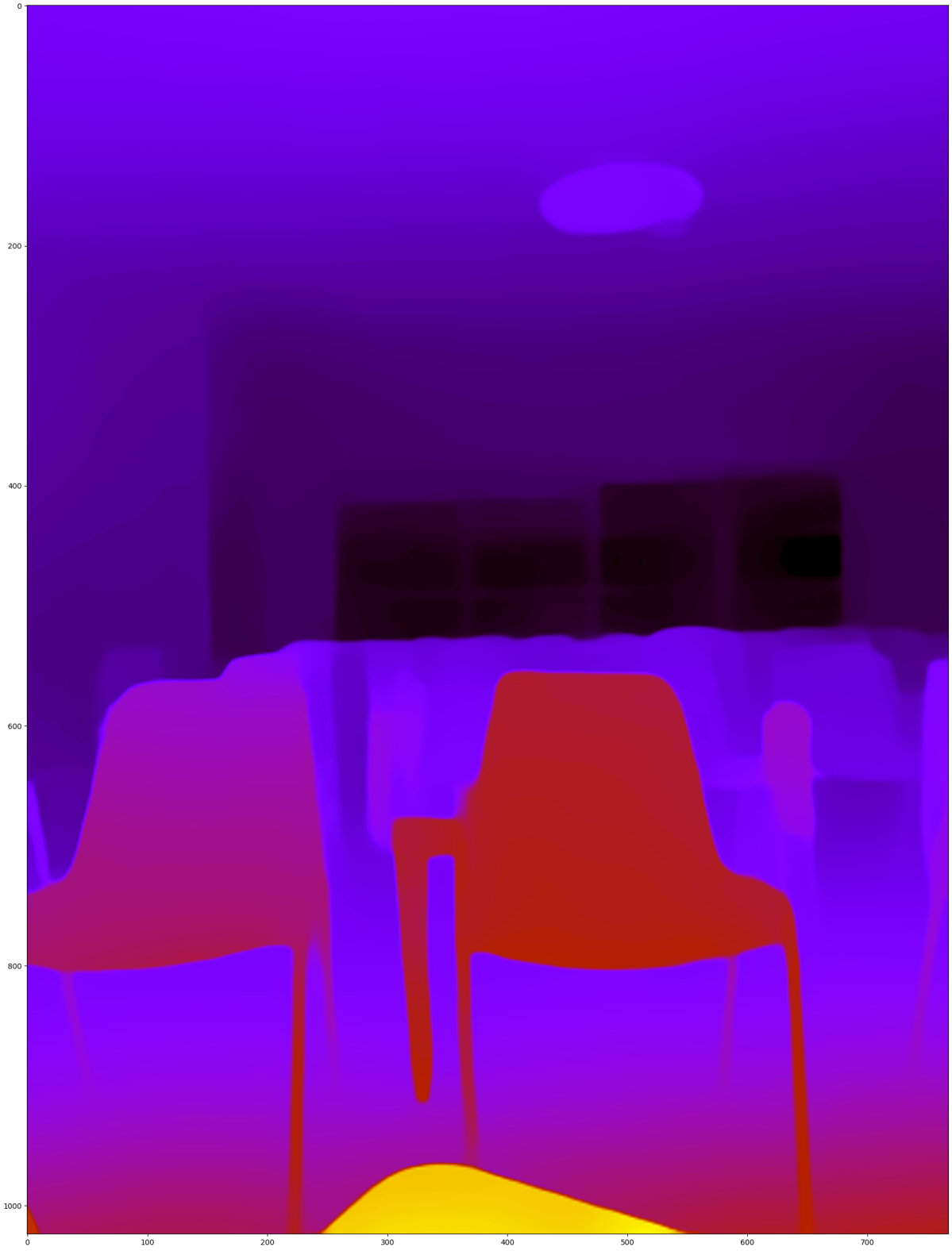

Figure 33: Depth Estimation - V

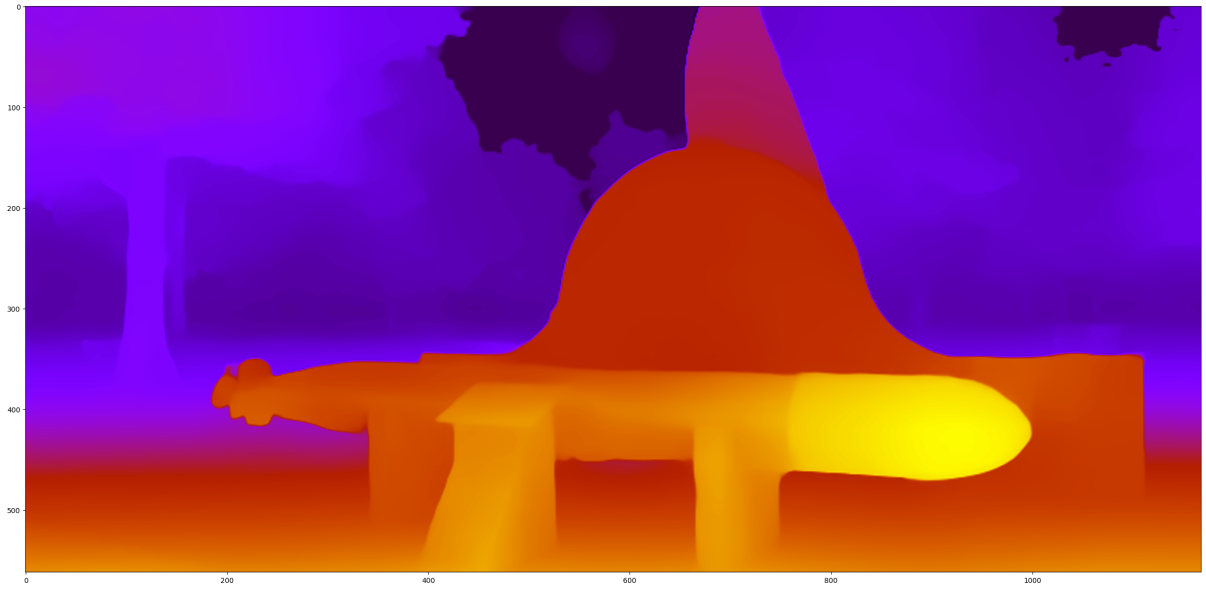

Figure 34: Depth Estimation - VI

3.4 Stereo-Panorama Viewer

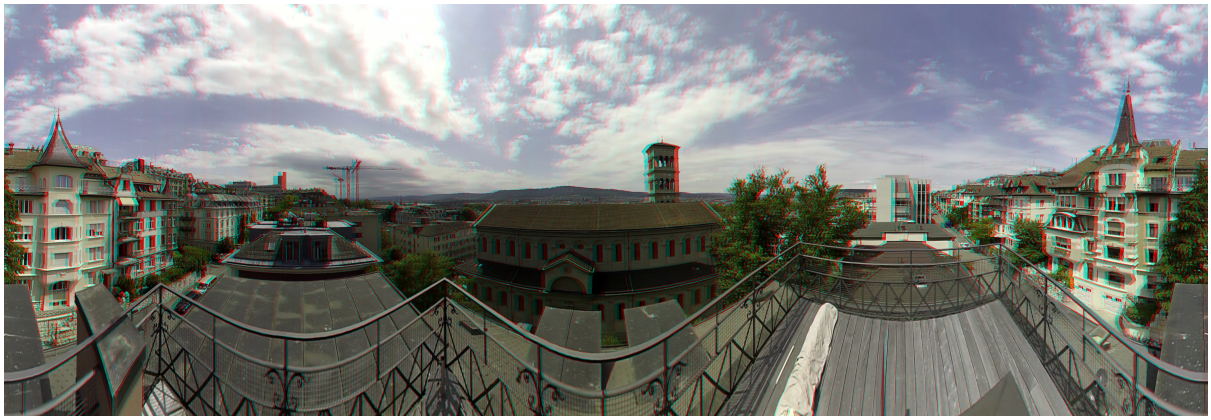

Figure 35: Panorama Anaglyph - I

Figure 36: Stereo Panorama - I

Figure 37: StereoPanoramic View A - I

Figure 38: StereoPanoramic View B - I

Figure 39: StereoPanoramic View C - I

Figure 40: Panorama Anaglyph - II

Figure 41: Stereo Panorama - II

Figure 42: StereoPanoramic View A - II

Figure 43: StereoPanoramic View B - II

Figure 44: StereoPanoramic View C - II

4 Discussion

In the discussion section of our report, we will analyze and interpret the results of our stereo panoramic image stitching and depth estimation algorithms. We will examine the performance of the algorithms in terms of accuracy, distortion, and the overall quality of the panoramic images. Additionally, we will compare the results of the single-camera panoramic images' depth estimation with the stereo panoramic images' disparity maps and examine the effectiveness of the red-cyan anaglyphs and the 3D VR visualizations in visualizing the depth information of the scene. Our goal is to provide a comprehensive evaluation of the algorithm and to identify potential areas for future research and improvement.

In the Panorama Stitching part, the stitching process of the first output is shown in Figure 1 – Figure 6. First, common features are extracted and matched between both images. Based on the resulting points, images are stitched into one. In Figure 2, a stitched image appeared with black regions around the corners. This problem is caused by the stitching process and necessary alignments during the connections. To remove the undesired black parts, our developed algorithm for trimming is used. Black regions are masked and iteratively subtracted until they are completely disappear from the resulting image (the process can be followed respectively with Figures 3, 4, and 5). Since it was our first trial, the first resulting panorama stitched image has small distortions on it. For the second and third panorama trials, we took multiple (12-18) images around the campus to obtain wide panoramic views. Again in Figure 7, the feature matching results are shown but this time the result is more robust and many features are extracted and matched correctly. Therefore the panorama stitched image in Figure 9 is satisfactory. Similarly, other panorama stitch results (raw and trimmed) are given in Figures 10 and 11. The last output of the first step consists of shifted images from a classroom. To prevent potential problems that can affect the results of trimming (it masks and eliminates the black or 0 density points), detected 0 density pixels are reassigned as 1 (one pixel is detected in the bag on the left side of the image).

For the Rectification and Disparity firstly, two uncalibrated but same-sized images were taken for the algorithm, then two images were overlapped to see the differences. As can be seen, from Figure 15, the red image corresponds to the left image, and the cyan colored image corresponds the right image. Then for each image, we have taken strongest features that exceed our given threshold value. From figure 16, the strongest features are visible for each right and left image. After feature extraction, our algorithm finds the corresponding point of the feature at the other image. Then the algorithm align each feature point by creating a transformation of the image. From Figure 17, the matched points and corresponding transformation of the image could be seen. Lastly, our rectified image is created by the transformation that we have. After having rectified the images, we have started to create the disparity-based dept estimation algorithm. In our image, we have chosen omega as 70 and window size value as 5 to get the best results of our dept estimation. We have both compiled our algorithm by built-in functions and by writing our own code. As the built-in output shown in Figure 27, while the far features could be compiled, the closer features were seen as completely dark. However, in our own codes we

have overcome the problem of not seeing closer features, and both far and close objects could be determined in our output as Figure 28 shows.

In the Depth Estimation part, the outputs of the first (Panorama Stitching) and the second (Rectification and Disparity) parts are used as test inputs for the pre-trained MiDaS model to obtain their high-accuracy depth estimations (Figure 29-34). As one of the main objectives of our project suggests, we developed a combined program (with our image stitching algorithm and integration of the MiDaS model) and obtained high quality panorama images with depth estimation. Last two outputs of this project step are generated depth estimations of the second part's outputs. When we compare the disparity maps from the second part with the same images' depth estimations produced by MiDaS, it is understood that our fine-tuned disparity algorithm gives better estimations on depth which converges the baseline depth estimations of MiDaS.

The last part of our project, Stereo-Panorama View, is an additional study to explore the benefits of stereo panoramic images more. The large dataset provided by MegaStereo research is used. Some of the selected stereo panorama images from the dataset are passed as an input image for a Virtual Reality tool called Stereo Panorama View (https://spano.pyrik.dev/). Three left-right stereo panorama images are converted to a single image (left on top and right on bottom) and given to the tool. The results were 360-degree virtual reality visualization (similar to Google Street View) and some of the screenshots can be seen between Figures 35 to 44. Additionally, red-cyan anaglyphs and input images are also provided in these figures to give an insight to understand the process of generation. There are many usecase where stereopanoramic outputs can be used, and we aimed to approach this methodologies conceptually in the last part of our project.

The results of the project provided insight into the challenges that arise during stereo panoramic image stitching and depth estimation. We were able to observe the distortion and fish-eye effect caused by panoramic stitching, and the limitations of using a single camera for stereo panoramic image capturing. We also found that the disparity maps derived from stereo panorama images provided more accurate depth information compared to the single-camera panoramic images' depth estimation. Additionally, we found that the red-cyan anaglyphs and the 3D VR visualizations were useful tools for visualizing the depth information of the scene.

Overall, this project has contributed to the understanding of stereo panoramic image stitching and depth estimation and has highlighted the potential for future research and applications in the field of computer vision. Further work could focus on improving the accuracy of the algorithm by using multiple cameras and developing methods to effectively minimize the distortion and fish-eye effect caused by panoramic stitching.

5 Appendix

1 - Feature Extraction and Matching

```
import cv2
import numpy as np


img_ = cv2.imread('2.jpeg')


ratio = img_.shape[1]/img_.shape[0] # keep original width

height = 700
width = int(height*ratio)
dim = (width, height)
img_ = cv2.resize(img_, dim, fx=0.3, fy=0.3)
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)

img = cv2.imread('3.jpeg')
img = cv2.resize(img, dim, fx=0.3, fy=0.3)
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
# find the key points and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
#cv2.imshow('original_image_left_keypoints',
cv2.drawKeypoints(img_,kp1,None))

#FLANN_INDEX_KDTREE = 0
#index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
#search_params = dict(checks = 50)
#match = cv2.FlannBasedMatcher(index_params, search_params)
match = cv2.BFMatcher()
matches = match.knnMatch(des1,des2,k=2)

good = []
for m,n in matches:
    if m.distance < 0.35*n.distance:
        good.append(m)

    draw_params = dict(matchColor = (0,255,0),
    # draw matches in green color
                       singlePointColor = None,
```

```
                    flags = 2)

    img3 = cv2.drawMatches(img_,kp1,img,kp2,good,None,**draw_params)
    cv2.imshow("original_image_drawMatches.jpg", img3)
cv2.waitKey(0)
```

2 - Panorama Stitching

```
import cv2
import os

import imutils
import numpy as np

def trim(stitched):
    stitched = cv2.copyMakeBorder(stitched, 10, 10, 10, 10,
                                  cv2.BORDER_CONSTANT, (0, 0, 0))

    # convert the stitched image to grayscale and threshold it
    # such that all pixels greater than zero are set to 255
    # (foreground) while all others remain 0 (background)
    gray = cv2.cvtColor(stitched, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]

    # find all external contours in the threshold image then find
    # the *largest* contour which will be the contour/outline of
    # the stitched image
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
                            cv2.CHAIN_APPROX_SIMPLE)

    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)

    # allocate memory for the mask which will contain the
    # rectangular bounding box of the stitched image region
    mask = np.zeros(thresh.shape, dtype="uint8")

    cv2.imshow(folder, mask)
    cv2.waitKey(0)

    (x, y, w, h) = cv2.boundingRect(c)
    cv2.rectangle(mask, (x, y), (x + w, y + h), 255, -1)

    # create two copies of the mask: one to serve as our actual
    # minimum rectangular region and another to serve as a counter
```

```
    # for how many pixels need to be removed to form the minimum
    # rectangular region
    minRect = mask.copy()
    sub = mask.copy()

    cv2.imshow(folder, minRect)
    cv2.waitKey(0)

    # keep looping until there are no non-zero pixels left in the
    # subtracted image
    while cv2.countNonZero(sub) > 0:
        # erode the minimum rectangular mask and then subtract
        # the thresholded image from the minimum rectangular mask
        # so we can count if there are any non-zero pixels left
        minRect = cv2.erode(minRect, None)
        sub = cv2.subtract(minRect, thresh)

        cv2.imshow(folder, minRect)
        cv2.waitKey(0)
        cv2.imshow(folder, sub)
        cv2.waitKey(0)

    # find contours in the minimum rectangular mask and then
    # extract the bounding box (x, y)-coordinates
    cnts = cv2.findContours(minRect.copy(), cv2.RETR_EXTERNAL,
                            cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)
    (x, y, w, h) = cv2.boundingRect(c)

    # use the bounding box coordinates to extract the our final
    # stitched image
    stitched = stitched[y:y + h, x:x + w]
    return stitched



mainfolder = 'images'
myfolder = os.listdir(mainfolder)
print(myfolder)

for folder in myfolder:
    path = mainfolder + '/' +folder
    images = []
    mylist = os.listdir(path)
```

```
    print(f'total number of images detected {len(mylist)}')
    stitcher = cv2.Stitcher.create()
    for imgN in mylist:
        current_image = cv2.imread(f'{path}/{imgN}')
        current_image = np.where(current_image == 0, 1, current_image)
        current_image = cv2.resize(current_image,(700,700),None)
        images.append(current_image)
        print(len(images))
        if(len(images)>1):
            (status,result) = stitcher.stitch(images)
            if(status == cv2.STITCHER_OK):
                print('panorama done successfully')
                cv2.imshow(folder,result)
                cv2.waitKey(10)
            else:
                print ('panorama is unsuccessfull')
    rect = result
    rect = trim(rect)
    cv2.imwrite(f'{folder}.jpg', result)
    cv2.imwrite(f'{folder}rect.jpg', rect)


img = cv2.imread(f'{folder}.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect corners
corners = cv2.goodFeaturesToTrack(gray, 100, 0.01, 10)
corners = np.int0(corners)

# Find the perpective transform matrix
src_points = np.float32([corners[i] for i in range(corners.shape[0])])
dst_points = np.float32([[i, j] for i in range(corners.shape[0]) for j in r
matrix = cv2.getPerspectiveTransform(src_points, dst_points)

# Warp the image
result = cv2.warpPerspective(img, matrix, (img.shape[1], img.shape[0]))
cv2.imshow("Result", result)
```

3 - Rectification and Disparity

```
clear all ; close all ; clc;

I1 = imread("okul_sol.jpeg");
I2 = imread("okul_sag.jpeg");
```

```
% Convert to grayscale.
I1gray = rgb2gray(I1);
I2gray = rgb2gray(I2);
figure;
imshowpair(I1, I2,"montage");
title("I1 (left); I2 (right)");
figure;
imshow(stereoAnaglyph(I1,I2));
title("Composite Image (Red - Left Image, Cyan - Right Image)");
blobs1 = detectSURFFeatures(I1gray,MetricThreshold=2000);
blobs2 = detectSURFFeatures(I2gray,MetricThreshold=2000);
figure;
imshow(I1);
hold on;
plot(selectStrongest(blobs1,50));
title("Thirty Strongest SURF Features In I1");
figure;
imshow(I2);
hold on;
plot(selectStrongest(blobs2,50));
title("Thirty Strongest SURF Features In I2");
[features1,validBlobs1] = extractFeatures(I1gray,blobs1);
[features2,validBlobs2] = extractFeatures(I2gray,blobs2);
indexPairs = matchFeatures(features1,features2,
Metric="SAD", MatchThreshold=50);
matchedPoints1 = validBlobs1(indexPairs(:,1),:);
matchedPoints2 = validBlobs2(indexPairs(:,2),:);
figure;
showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2);
legend("Putatively Matched Points In I1",
"Putatively Matched Points In I2");
[fMatrix, epipolarInliers, status] =
estimateFundamentalMatrix(matchedPoints1,
matchedPoints2,Method="RANSAC",NumTrials=10000,
DistanceThreshold=0.01,Confidence=99.99);

if status ~= 0 || isEpipoleInImage(fMatrix,size(I1)) ...
  || isEpipoleInImage(fMatrix',size(I2))
  error(["Not enough matching points were found or "...
         "the epipoles are inside the images. Inspect "...
         "and improve the quality of detected features ",...
         "and images."]);
end

inlierPoints1 = matchedPoints1(epipolarInliers, :);
```

```
inlierPoints2 = matchedPoints2(epipolarInliers, :);

figure;
showMatchedFeatures(I1, I2, inlierPoints1, inlierPoints2)
legend("Inlier Points In I1","Inlier Points In I2");
[tform1, tform2] = estimateStereoRectification(fMatrix, ...
  inlierPoints1.Location,inlierPoints2.Location,size(I2));
[I1Rect, I2Rect] = rectifyStereoImages(I1,I2,tform1,tform2);
figure;
imshow(stereoAnaglyph(I1Rect,I2Rect));
title("Rectified Stereo Images (Red - Left Image,
Cyan - Right Image)");
figure
subplot(1,2,1)
imshow((I1Rect))
title("I1");
subplot(1,2,2)
imshow((I2Rect))
title("I2");

img_right = I1Rect;
img_right = rgb2gray(img_right);
img_right = double(img_right);
[row, column,ch] = size (img_right);
    %img_right = imresize(img_right,[4000,4000])
img_left = I2Rect;
img_left = rgb2gray(img_left);
img_left = double(img_left);
    %img_left = imresize(img_left,[4000,4000])

[row, column,ch] = size (img_left);

k= 5;
    omega = 70;
    offset = omega + k;

    img_L = padarray(img_left,[offset offset],'both');
    img_R = padarray(img_right,[offset offset],'both');
    [ydim,xdim]=size(img_L);

    for xL = offset+1:1:xdim-offset-1
        for yL = offset+1:1:ydim-offset-1
        dist = [];
        subL =img_L(yL-k:yL+k,xL-k:xL+k);
            for xR = xL:-1:xL-omega
```

```
                    subR = img_R(yL-k:yL+k,xR-k:xR+k);
                    C = sum(sum(-1.*(subL-subR).^2));

                    dist=[dist;xL-xR C];
                end
            ind = find(dist(:,2) == max(dist(:,2)));
            d = dist(ind(1),1);
            disparity(yL,xL)=d;
            %disparity=disparity(yL,xL);
            end
        end

imshow(stereoAnaglyph(uint8(img_L),uint8(img_R)));
% Show disparity map with colorbar
figure; imagesc(uint8(disparity)); colormap turbo; colorbar

disparityRange = [0 32];
dmap = disparitySGM(img_left,img_right,'DisparityRange',
disparityRange,'UniquenessThreshold',40);

% Show disparity map
imshow(dmap);
colormap jet
colorbar
imshow(stereoAnaglyph(uint8(img_left),uint8(img_right)));
% Show disparity map with colorbar
figure; imagesc(uint8(dmap)); colormap turbo; colorbar
```

4 - MiDaS Colab Notebook

https://colab.research.google.com/github/pytorch/pytorch.github.io/blob/master/assets/hub/intelisl_midas_v2.ipynb

5 - Stereo Panorama Image Viewer - Web UI VR

https://spano.pyrik.dev/

6 References

[1] J. Cho, J. H. Cha, Y. M. Tai, Y.-S. Moon, and S. Lee, "Stereo panoramic image stitching with a single camera," in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, 2013, pp. 256–257. DOI: 10.1109/ICCE.2013.6486884.

[2] P. K. Lai, S. Xie, J. Lang, and R. Laganière, "Real-time panoramic depth maps from omni-directional stereo images for 6 dof videos in virtual reality," in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, pp. 405–412. DOI: 10.1109/VR.2019.8798016.

[3] C. Richardt, Y. Pritch, H. Zimmer, and A. Sorkine-Hornung, "Megastereo: Constructing high-resolution stereo panoramas," Jun. 2013, pp. 1256–1263. DOI: 10.1109/CVPR.2013.166.

[4] H. Wang, D. J. Sandin, and D. Schonfeld, "Post-stitching depth adjustment for stereoscopic panorama," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 2282–2286. DOI: 10.1109/ICASSP.2019.8682724.

[5] F. Zhang and F. Liu, "Casual stereoscopic panorama stitching," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2002–2010. DOI: 10.1109/CVPR.2015.7298811.