Introduction

The primary issue with using 2-D cameras to visualize a 3-D space, is that depth is not a feature that can be accurately determined. If an image can be considered as the (x, y) plane with each coordinate point representing a single pixel, the depth (z axis) of each pixel cannot be determined due to the 2D nature of the image. This issue is tackled in stereovision, which uses more than one camera to calculate depth based on different algorithms. A binocular stereovision system uses only two cameras separated by a **baseline distance**, similar to Figure 1, to calculate the depth of the image [1]. When comparing the images obtained from both cameras, there are locations that can be viewed by both cameras, known as **features**. Features will have different positions in the left and right camera and the pair of image points is called a **conjugate pair**. The distance between a conjugate pair (measured in pixels) is known as **disparity**, and is the primary factor used in calculating depth [2].

Figure 1: binocular stereovision system [14]

My chosen image was “sword1-perfect”, and the original image taken from the right camera can be seen in Figure 2 [3].

**(A) Assumptions**

Figure 2: Sword1-perfect [3]

To simplify the process of obtaining a 3D model and/or calculating the disparity of my chosen picture, several assumptions were made:

**A.1** The cameras are on the same plane (as seen in Figure 1) so no displacement on the y axis was assumed to be made. This allows me to simplify my algorithm, reducing its runtime as I only need to focus on the pixels in the same row.

**A.2** The disparity value has a maximum of , where ndisp is an upper bound on the number of disparity levels given in calib.txt and factor is the value used to resize my image. This means that a pixel on the left image will be matched with a pixel on the right image within a certain interval, rather than the entirety of the row, which reduces the runtime.

**A.3** There is a negligible change in quality when resizing the image, so an image with a smaller resolution will still accurately represent the original image. Decreasing the image resolution greatly reduces runtime because there is a smaller number of pixels that have to be analyzed

**A.4** Turning the image into grayscale preserves the accuracy of the disparity map compared to retaining the RGB values, which would greatly reduce runtime as calculations on three separate variables are not required. This was assumed since the formula that OpenCV (the python library that I used to analyse the images) uses to convert an image in RGB to grayscale is [13]. As can be seen, the grayscale value is linearly proportional to the RGB values, indicating that a conversion to grayscale would accurately account for all individual RGB values.

**A.5** The principle axes of the cameras are assumed to be parallel. This simplifies my algorithm in two major ways. Firstly, I do not have to account for angle rectification. Secondly, the disparity value will be greater than zero under this assumption. A feature mapped in the right camera cannot be on the left side of the same feature mapped in the left camera, unless the principal axes are not parallel.

**(AE) Algorithm & Equations**

The following algorithms and associated equations were used in this assignment:

**AE.1** Correlation Coefficient

The correlation coefficient is a measure of similarity between two windows centered around a feature between two images, and . This is one of the methods used to obtain **disparity**, . A higher correlation coefficient value indicates a better match of the feature region [2]. The equation is defined by

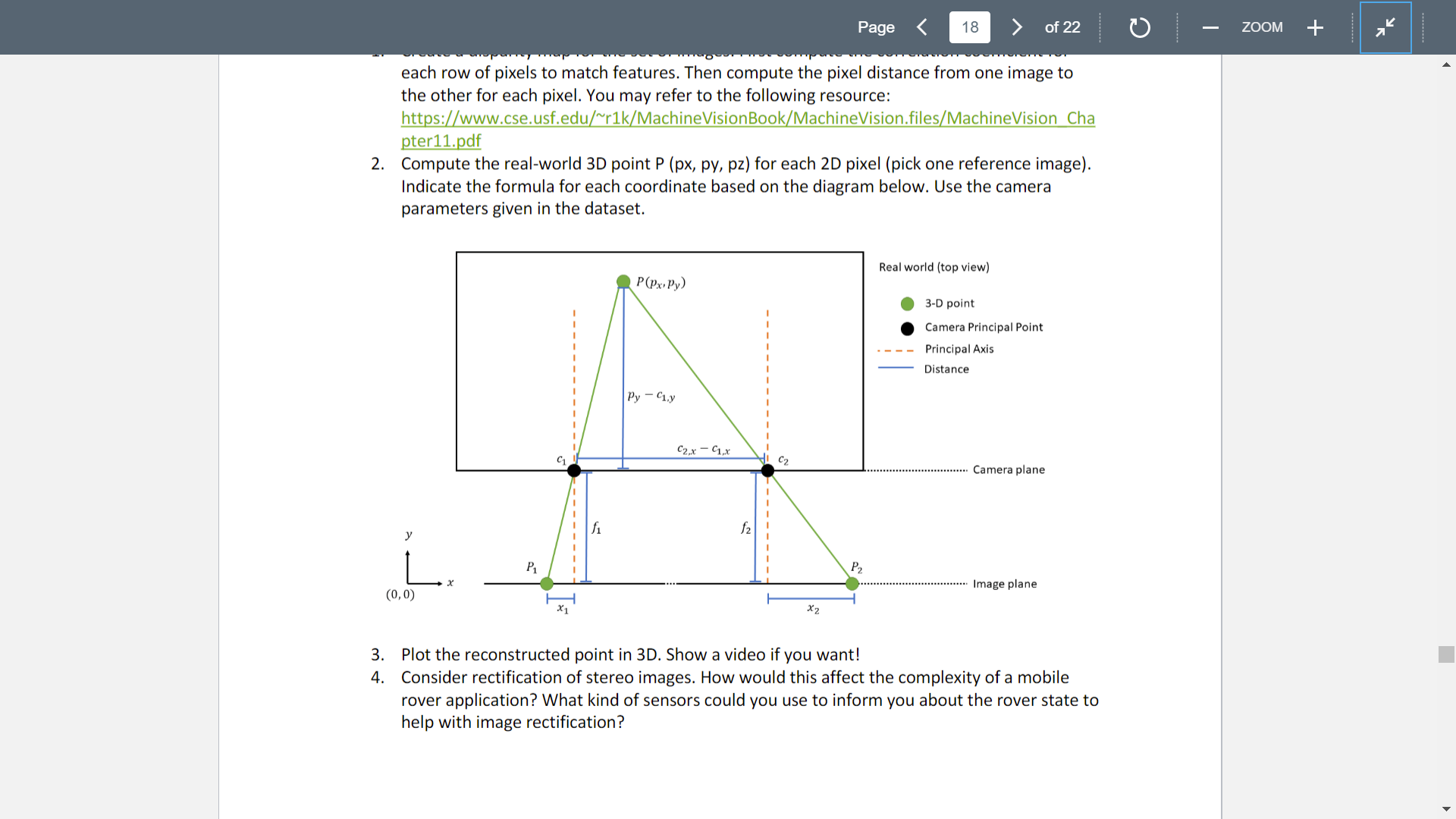
(**AE.1.1**)

where are windows centered around each feature and are the average intensity in a given window. This is a modified equation from [2] that utilizes assumption **A.1** and **A.5**.

The disparity is then found using the following equation, where is the max possible value for :

(**AE.1.2**)

**AE.2** Depth Calculation

The following is a derivation of the formula to calculate the depth. The y axis represents depth in this situation.

Using similar triangles:

Figure 3: Binocular Vision System from assignment Manual [16]

(**AE.2.1)**

**(P) Process**

Python Libraries

OpenCV and NumPy were the primary libraries used for reading the image and doing matrix calculations in order to find the disparity map. Cvkit was the library used to create the 3D model and Time was also imported to compare runtimes.

**(P.D)** Disparity Map Creation and Depth Calculation

**P.D.1** Both images were obtained by using OpenCV. They were then resized and changed to grayscale by using cv2.resize() and cv2.cvtColor() respectively. The values given in the calib.txt were also reduced by the same factor used to resize the image.

**P.D.2** The average intensity of each point for both images was taken by obtaining the average of the sum of the grayscale values over a window S using np.sum() and then mapped into a numpy.array the same size of my resized images.

* 1. The size of S that I used were {3x3, 5x5, 7x7, 9x9, 11x11} measured in pixels

**P.D.3** Only the area that held features was determined for both images. These boundaries were applied to the resized image and the average intensity for both images.

1. Left camera boundaries:
2. Right camera boundaries:

**P.D.4** From the previous step I obtained: . These values are then plugged into equation **AE.1.1** to obtain the correlation values

**P.D.5** Equation **AE.1.2** was then used to obtain the disparity map. This map was then normalized and visualized using cv2.normalize() and cv2.imshow() respectively.

**P.D.6** The depth values for each (x,y) point and associated disparity map given in step **P.D.5** was calculated using equation **AE.2.1** with the baseline and focal length adjusted for the scaling factor

**(P.M)** 3D Model Creation

**P.M.1** After the disparity map was created in step **P.D.5**, this disparity map was converted into a .pfm file using a function available on github made by *chpatrick* [4]. This was done so the disparity map could be 3D modelled with use of cvkit.

1. The given function flips the image on the x axis when saving as a .pfm file so the original disparity map was flipped using cv2.flip() for proper alignment

**P.M.2** To add color to the 3D model, the original image (Figure 2) had to be resized so its dimensions matched my .pfm file. In addition, the calib.txt file had to updated with the associated scaling factor of 0.3 to accurately create the model.

**P.M.3** The 3D model was created by double clicking my .pfm file with the required associated files (an image file and the calib.txt file) and a video was taken

**(D) Discussion**

**D.1** Disparity Map Optimization

**A picture containing food

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA picture containing food

Description automatically generatedA screenshot of a tree

Description automatically generated**When creating the disparity map, 5 different window sizes for S were chosen: {3x3, 5x5, 7x7, 9x9, 11x11}. It was determined that a 9x9 window resulted in the most accurate disparity map, with the least amount of noise. However, a large amount of noise was still present so to refine my disparity map, I compared three different gradient filters (Laplacian operator, Sobel operator on x, and Sobel operator on y) on the image, between step **P.D.1** and **P.D.2**, with inspiration taken from [5]. This was done because in a binocular stereovision system, the depth is limited by the disparity gradient as opposed to the absolute disparity values [14]. As a result, applying a gradient function to the pictures before calculating the disparity would result in a disparity map that more accurately represents the depth. Those three operators were chosen because they were readily available on OpenCV, preventing additional code from being written which would increase runtime. The Laplacian operator significantly decreased the amount of noise for the disparity map, which can be seen in Figure 4, and was the chosen disparity map for my 3D model.

Figure 4: (top left) original disparity map (top right) Laplacian operator (bottom left) Sobel operator on x (bottom right) Sobel operator on y. These disparity maps were obtained using a 9x9 window for S and a resizing factor of 0.3. As can be seen, the Laplacian operator resulted in a much smoother disparity map with a decreased amount of noise.

**D.2** Runtime

One of the most difficult problems to address with stereovision, is the amount of resources required to create an accurate disparity map. Specifically, stereovision is a time intensive process and majority of research on stereovision currently is focused on improving the speed of the associated algorithms [6] [7] [8]. To create a more accurate disparity map, a greater amount of time is required, and this relationship is nonlinear using my algorithm **AE.1**, which can be seen in Figure 5. In reality, assumption **A.3** does not hold, so I had to choose a balance of runtime and accuracy of my disparity map. I ultimately decided on a scaling factor of 0.3, which brought the dimensions of the image to 606x873 and an average runtime of 576.46 seconds. When the Laplacian operator was used, a negligible increase in runtime was discovered (587.89 seconds), indicating that applying the Laplacian gradient operator does not sacrifice a significant amount of resources for a much greater quality in disparity map creation.

Figure 5: Runtime for steps **P.D.4** and **P.D.5** versus scaling factor. When modelled, a polynomial of order three was created for the runtime. This makes sense given that my algorithm **AE.1** incorporated three for loops, indicating a time complexity of O(n3).

**D.3** Rectification for Mobile Rover

The main idea behind stereo rectification is that two images that are at an angle displacement need to be projected onto a similar plane such that the two images align and are parallel, to some degree, on the new plane [9]. This is done by multiplying each image by a transformation matrix, which can be thought of as a rotation to the camera to obtain a properly aligned image [9]. Without rectification, creating a disparity map would be a 2D problem, increasing the complexity and runtime [9]. With a mobile rover, the primary issue with rectification would be the increased time required to rectify the image. Calculations must be performed for each pixel in both images to obtain the transformation matrix [10], and iteration over each pixel could be a lengthy process, depending on the chosen resolution size.

**D.3.1** Sensors for Mobile Rover Rectification

A possible sensor that could be used in aid for a mobile rover image rectification is a gyroscope. A gyroscope can determine orientation, by measuring the angular velocity [11]. A research paper from Stanford [12] details a method of using a gyroscope to stabilize a video in real time. In lower end cameras, such as the raspberry pi camera our group was using, rows of pixels are sequentially sent as output rather than all pixels being sent at the same time to create the image. This results in a different time window for each row of pixels. The algorithm proposed in this paper addresses this issue by using a rolling shutter correction, after calibrating the gyroscope to the video camera. This creates a more stable video, which would experience lower amounts of “shaking” (vertical and horizontal displacement). In stereovision, image rectification is simplified when the two images have none or minimal vertical displacement and use of this algorithm could help create two images that are aligned horizontally. This would help preserve accuracy and decrease the runtime required to create a disparity map while in real time.

Bibliography

[1] Nair, D. (2018, November 5). A Guide to Stereovision and 3D Imaging. Retrieved from https://www.techbriefs.com/component/content/article/tb/features/articles/14925?start=1

[2] University of Florida. (n.d.). *Machine Vision*.

[3] Nesic, N., Westling, P., Wang, X., Kitajima, Y., Krathwohl, G., & Scharstein, D. (2013). 2014 Stereo datasets with ground truth. Retrieved from http://vision.middlebury.edu/stereo/data/scenes2014/

[4] chpatrick, 262588213843476. (2014). Python PFM <-> Numpy library. Retrieved from https://gist.github.com/chpatrick/8935738

[5] Danilo Alves de Lima, Giovani Bernardes Vitor, Alessandro Corrêa Victorino, Janito Vaqueiro Ferreira. A disparity map refinement to enhance weakly-textured urban environment data. IEEE International Conference on Advanced Robotics (ICAR 2013), Nov 2013, Montevideo, Uruguay. pp.1-6. ffhal00936041f

[6] Hirschmüller, H., Innocent, P.R. & Garibaldi, J. Real-Time Correlation-Based Stereo Vision with Reduced Border Errors. *International Journal of Computer Vision* 47, 229–246 (2002). https://doi.org/10.1023/A:1014554110407

[7] Gonzalez, R.C. & Cancelas, J.A. & Alvarez, Juan & Fernandez, J.A. & Enguita, Jose. (1999). Fast stereo vision algorithm for robotic applications. 1. 97 - 104 vol.1. 10.1109/ETFA.1999.815343.

[8] Di Stefano, Luigi & Marchionni, Massimiliano & Mattoccia, Stefano. (2004). A fast area-based stereo matching algorithm. Image and Vision Computing. 22. 983-1005. 10.1016/j.imavis.2004.03.009.

[9] Fusiello, Andrea. (1999). Tutorial on Rectification of Stereo Images.

[10] Scientific Computing and Imaging Institute. (2012). Image Rectification (Stereo). Retrieved from http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/CS6320-CV-F2012-Rectification.pdf

[11] Goodrich, R. (2018, May 31). Accelerometer vs. Gyroscope: What's the Difference? Retrieved from <https://www.livescience.com/40103-accelerometer-vs-gyroscope.html>

[12] Karpenko, A., & Jacobs, D. (2011). CTSR 2011-03 Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes.

[13] Miscellaneous Image Transformations¶. (n.d.). Retrieved from https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous\_transformations.html#void cvtColor(InputArray src, OutputArray dst, int code, int dstCn)

[14] Mckee, S. P., & Verghese, P. (2002). Stereo transparency and the disparity gradient limit. *Vision Research*, *42*(16), 1963–1977. doi: 10.1016/s0042-6989(02)00073-1

[15] ESC204 – Skill Building Assignment\_V2