AINT308 - Machine Vision and Behavioural Computing
Coursework 2 Report

Student No. 10613591
School of Engineering,
Computing and Mathematics
University of Plymouth
Plymouth, Devon

***Abstract*—Machine Vision is field of study whose applications are becoming rapidly more prevalent amongst contemporary technology, with advancements having large implications in a wide variety of fields. This report details the use of a popular open-source machine vision library, OpenCV, in two additional real-world applications: Using disparity mapping to estimate distance to an object, and using edge detection in order to detect road markings.**

Keywords:

Machine Vision, OpenCV, Object Tracking, C++

I. Task 4 - Disparity Mapping

For accompanying large figures, see appendix B
For the video demo, click HERE

A. Introduction

This task demonstrates the use of disparity maps in order to estimate the distance from the camera to an object.

B. Solution

An important first step when working with stereo visions systems is to account for distortions in the image. This can be induced by a number of factors, including both distortions caused by the camera itself as well as distortions caused by a human element of the operation of the camera[1].

Without properly calibrating, these distortions and inaccuracies can have a catastrophic effect on the ability for the system to function. This is even more important for applications with safety-critical consequences, such as vision systems for self-driving cars[2] or industrial monitoring equipment[3].

There are multiple methods for performing the calibration. MATLAB's implementation uses Bouguet's method[4][5], whereas Hartley's method[6] also exists as an alternative. *OpenCV's* built-in calibration function can use both of these algorithms[7].

The calibration function uses pairs of chequerboard images, and generates sets of *intrinsic* and *extrinsic* parameters. The *intrinsic* parameters contains the focal length, principal point, and skew coefficient, all of which are distortion sources local to the camera. The extrinsic parameters store the rotation and translation between the two cameras.

The generated calibration data is loaded into the program, and used to distort the images in order to correct the distortion created. With perfect calibration, the two distortions will completely cancel out, leaving a perfect fidelity image. *OpenCV's remap*[8] function is used to perform the correct. The *remap* function takes in an input array as well as up to two maps to perform the re-mapping, in this case the input image and the two generated maps created using the intrinsic and extrinsic parameters, outputting the results to a destination array[9].

To estimate the distance of an object from the camera, the disparity is used. By using the parallax of the two stereo cameras combined with the disparity mapping the approximate distance of the object can be triangulated. This is similar to how the human brain estimates distance using *binocular disparity*[10][11], and is another example of how robotics emulates life in order to mimic functionality. Similar techniques are used by astronomers to measure the distance to stellar objects[12].

Semi-Global Block Matching (SGBM) is used to generate a disparity map. SGBM takes a small region in one image, and searches in nearby locations in the other image for matches. The disparity is the minimum distance needed to find a match. Sum of Absolute Differences (SAD) is used to calculate the similarity[13]. The value of each pixel in the template section is subtracted from the respective pixel in the target section, then these differences are summed. The lower the value, the closer the match. It should be noted that before this operation, the images are converted to greyscale, to minimise the amount of data needed to be processed, as each pixel can be represented as a single 8-bit value.

TABLE I: Intensity Readings At Known Distances

Distance	Single Pixel Avg	3x3 Avg
30	2094	2093
40	1566	1565
50	1245	1245
60	1040	1040
70	896	896
80	784	783
90	703	703
100	632	631
110	576	576
120	514	513
130	480	479
140	432	432
150	414	414

TABLE II: Program results and error rate

Distance	Program Result	Program Error
30	29.8134	0.62%
40	39.8747	0.31%
50	50.1235	-0.25%
60	60.025	-0.04%
70	69.6632	0.48%
80	79.6928	0.38%
90	88.7714	1.37%
100	98.8448	1.16%
110	108.378	1.47%
120	121.556	-1.30%
130	130.175	-0.13%
140	144.356	-3.11%
150	150.747	-0.50%

$$Disparity = \frac{B \bullet f}{Distance} \tag{1}$$

$$B \bullet f = Distance \bullet Disparity \tag{2}$$

To use the *Disparity-Distance* formula, the unknown parameter $B \bullet f$ needed to be derived using the supplied known distances. The pixel $(280, 350)$ was chosen as the approximate centre of the target in the known distance images, and it's intensity extracted. To reduce the likelihood of errors, an average of a 3x3 grid was taken. The results can be seen in Table I. Using equations 1 and 2, it is possible to approximate the value of $B \bullet f = 62426$. Derivation of the values of B and f individually is not necessary for the task.

C. Limitations

D. Further Improvements

E. Conclusion

II. Task 2 - Object Tracking

For accompanying large figures, see appendix C
For the video demo, click HERE

The second task involves tracking a pendulum and using the observed displacement to calculate the pendulum's angular deviation from its vertical rest position.

TABLE III: Error values of different components.

Component	NMSD Error
U4	0.000642183
C70	0.000428351
U2	0.000984659
L2	0.000689957
Q1	0.000671107
C97	0.000465317
C87	0.000600116
U1	0.00133448
U13 (Missing)	0.0435289
U13 (Present)	1.52745e-07
L8	0.00050902

A. Introduction

B. Solution

$$angle = x * (180/\pi) \tag{3}$$

C. Limitations

D. Further Improvements

E. Conclusion

Appendix

References

[1] R. Koch A. Woods T. Docherty. “Image Distortions in Stereoscopic Video Systems”. In: *Stereoscopic Displays and Applications IV* (1993). URL: http://www.andrewwoods3d.com/spie93pa.html.

[2] Bosch. *Self-driving cars: lights, camera, action!* 2022. URL: https://www.bosch.com/stories/mpc3-self-driving-car-camera/.

[3] Intel. *What Is Machine Vision?* 2022. URL: https://www.intel.com/content/www/us/en/manufacturing/what-is-machine-vision.html.

[4] Matlab. *What Is Camera Calibration?* 2022. URL: https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[5] Jean-yves Bouguet and Pietro Perona. “Camera Calibration from Points and Lines in Dual-Space Geometry”. In: (Oct. 1998).

[6] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge books online. Cambridge University Press, 2003. ISBN: 9780521540513. URL: https://books.google.co.uk/books?id=si3R3Pfa98QC.

[7] *Learning OpenCV: Computer Vision with the OpenCV Library*. O’Reilly Media, 2008.

[8] OpenCV. *Remapping*. 2022. URL: https://docs.opencv.org/3.4/d1/da0/tutorial_remap.html.

[9] OpenCV. *Geometric Image Transformations*. 2022. URL: https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#gab75ef31ce5cdfb5c44b6da5f3b908ea4.

[10] Joseph S. Lappin. “What is binocular disparity?” In: *Frontiers in Psychology* 5 (2014). ISSN: 1664-1078. DOI: 10.3389/fpsyg.2014.00870. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2014.00870.

[11] M.E. Berryhill, C. Hoelscher, and T.F. Shipley. “Spatial Perception”. In: *Encyclopedia of Human Behavior (Second Edition)*. Ed. by V.S. Ramachandran. Second Edition. San Diego: Academic Press, 2012, pp. 525–530. ISBN: 978-0-08-096180-4. DOI: https://doi.org/10.1016/B978-0-12-375000-6.00342-6. URL: https://www.sciencedirect.com/science/article/pii/B9780123750006003426.

[12] T. Pultarova J. Lucas. *What Is Parallax?* 2018. URL: https://www.space.com/30417-parallax.html.

[13] Chris McCormick. *Stereo Vision Tutorial - Part 1*. 2014. URL: http://mccormickml.com/2014/01/10/stereo-vision-tutorial-part-i/.

TABLE IV: Full results of the intensity testing

Distance	Validation	Error	Program Result	Program Error
30	29.82616046	0.58%	29.8134	0.62%
40	39.8889162	0.28%	39.8747	0.31%
50	50.14148903	-0.28%	50.1235	-0.25%
60	60.02514793	-0.04%	60.025	-0.04%
70	69.6720467	0.47%	69.6632	0.48%
80	79.72688869	0.34%	79.6928	0.38%
90	88.79964985	1.33%	88.7714	1.37%
100	98.93209801	1.07%	98.8448	1.16%
110	108.3787393	1.47%	108.378	1.47%
120	121.6884091	-1.41%	121.556	-1.30%
130	130.3259997	-0.25%	130.175	-0.13%
140	144.5049858	-3.22%	144.356	-3.11%
150	150.7878112	-0.53%	150.747	-0.50%

A. Github Repository

For the full code, please see the linked github repository HERE.

B. Task 4 Figures

C. Task 5 Figures

D. Code Printouts

The following pages contain complete printouts of the code used to implement the solutions outlined above, for reference.