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%

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

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

$$Distance = \frac{B \bullet f}{Disparity} \tag{3}$$

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.

Table II shows the error results at different distances, with the maximum error being a 3.11% is within the 5% margin of error.

To make the guesses on the distance to the target in the unknown image, the same process is repeated. The unknown image pairs are read in, and are greyscaled before using SGBM to produce a disparity map. The retrieved average intensity is used in Equation 3 to produce an estimate for the distance to the target. The results can be seen in Table III.

TABLE III: Distance estimates for unknown image pairs

Image #	Distance Value
0	105.193
1	59.1529
2	114.754
3	151.847
4	49.4572
5	134.862
6	28.9247
7	83.3458

C. Limitations

Given that the system displayed accurate results with the known dataset, and that without the true distances of the unknown dataset performance cannot be conclusively verified, the system performed to specifications. It is difficult to identify limitations given the limited scope of testing, and lack of access to additional testing materials.

D. Further Improvements

This system's performance is directly tied to the quality of it's calibration, so more accurate calibration data would benefit it's performance. This could be done by having less clutter in the calibration images. Ideally, they would be shot on a plain background, with the only difference being the target moving. Additionally, the distance of the target could be measured with a more accurate and precise instrument, to further ensure the quality of the calibration.

One disadvantage of using SGBM is that it is very processing intensive, especially as the size of the area to be searched increases. Performing SGBM on larger images on less powerful hardware may not be possible in real-time. To alleviate this, Field Programmable Gate Arrays (FPGAs) can be used to create bespoke hardware capable of performing the SGBM at high speeds[14].

E. Conclusion

The system performed extremely well, correctly identifying every target distance with a sub 5% error rate. It can be said that the task was a success, despite not knowing the performance when estimating the unknown targets, due to the performance with the testing targets. This demonstrates that disparity and parallax can be used to calculate distance to a target using a simple software solution, and *OpenCV*.

II. Task 5 - Lane Tracking

For accompanying large figures, see appendix C

For the video demo, click HERE

The second task involves using edge detection to correctly identify the current lane in a piece of dashboard camera footage.

A. Introduction

Giving computers the ability to recognise lines has many potential applications, one of the foremost being for self-driving cars and other related applications. Edge detection is one method of letting computers recognise lines, and in this application will be used to recognise lane demarcations on roads, using dashboard camera footage as a testing medium.

B. Solution

After loading in a frame of video data, *OpenCV's* `cvtColour` function is used, to convert the image to greyscale. For this implementation, colour information is not necessary, and while it's loss does reduce the efficacy of edge-detection [15], the solution still performed adequately.

When performing edge detection, it is important to apply filtering to smooth the image, and remove noise. Noise may cause contiguous edges to be picked up as many, smaller edges. To do this, *OpenCV's* `blur` function is used. At it's default settings, `blur` employs a variable sized normalized box filter to smooth an image[16]. 3x3 was chosen, too large a kernel and the more granular details of the image would be lost, whereas too small a kernel would mean the smoothing had very little effect on the image.

During operation, the edge detection would detect false positives in the tree line. This effect is unavoidable, no matter what parameters for the edge detection are used. To avoid this, a rectangular overlay is added to the video feed, blocking the top half of the video. This overlay prevents the upper half of the image from going though the edge detection algorithm, and consequently stops false-positives in the tree-line from being detected.

The solution uses Canny edge detection in order to detect the lines within the target image. *OpenCV* implements Canny edge detection using the `Canny` function[17], which takes in an input and output frame as well as an upper and lower threshold, `upperThreshold` and `lowerThreshold`. Changing the threshold values changes the sensitivity of the edge detection. The final values chosen in the solution are 50 and 100. Different values were experimented with, however the main factor that changed the performance of the solution was the Hough Lines parameters, discussed below.

A Hough Lines transform can be used to detect straight lines. The *OpenCV* implementation allows for two different variations, a *Standard Hough Lines Transform* and a *Probabilistic Hough Lines Transform*. The standard transform allows for detection of straight lines and is highly resistant to noise [18]. The probabilistic transform is more efficient, as it only calculates part of the transform, and is able to handle curved lines and segmented lines better. However, it is less resistant to noise, and with the limited pre-processing performed worse than its standard counterpart. See FIGURE WHATEVER for a comparison between the two variations.

With the vector of lines detected by the Hough Transform, rendering can begin. The solution makes use of *Temporal Smoothing*[19] in order to reduce the amount of jitter between frames. Jitter means the displayed lane boundaries would jump, which is undesirable behaviour and could have negative effects towards the safety of the end user. Temporal Smoothing works by checking the new (current frame) coordinates against the old (previous frame) coordinates. If the change is greater than the threshold, then it is reduced to 10% of the difference, and then the line is rendered. This greatly reduces the amount of jitter, improving the quality of the lane detection.

Equations 4 and 5 are both equations that represent lines, non-parametric and parametric respectively. In the parametric equation, ρ represents the perpendicular distance from the line to the origin, and θ is the angle between the perpendicular and the horizontal axis, measured counter-clockwise[20]. Representing lines parametrically avoids undefined computational behaviour present in the standard model when lines are completely vertical or horizontal.

$$y = mx + c \tag{4}$$

$$\rho = x\cos\theta + y\cos\theta \tag{5}$$

The solution only renders lines that meet one of the following constrains:

$\rho \leq 1$
OR
$\rho \geq 2.3$

This only renders lines that fall close to the vertical, of which the majority of the lane demarcations do. This effectively eliminates most of the false positive results, such as lines detected on passing cars, as well as the lower edge of the overlay rectangle.

Once the Temporal Smoothing has occurred, the calculated coordinates are pushed into the `corners` vector, that stores the edges of the polygon to be drawn showing the correct lane. `topMid` and `btmMid` are the endpoints of the centre line, calculated using the corners of the boundary polygon.

To draw the translucent boundary polygon on to the video frame two *OpenCV* functions are used: `fillPoly` and `addWeighted`. `fillPoly` uses an array of points in order to fill an area with a specified colour[21]. A new matrix `overlay` is created, and the current frame is copied into it using `copyTo`. `copyTo` copies a matrix's data to another, invoking the `create` method in order to properly reallocate the size of the destination matrix[22]. `fillPoly` draws a green polygon between the points specified in the `corners` onto the overlay frame.

$$g(x) = (1-\alpha)f_0(x) + \alpha f_1(x) \tag{6}$$

TABLE IV: 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

`addWeighted` performs a linear blend between two frames, following Equation 6. f_0 and f_1 are the video frames, α is the transparency. As the transparency must equal 1, the α of the other image will equal $(1-\alpha)$. In areas where $f_0 = f_1$, there will be no change in the image.

$$\begin{aligned} f_0 =& f_1 \\ g(x) =& (1-\alpha)f_0(x) + \alpha f_0(x) \\ g(x) =& f_0 \end{aligned} \tag{7}$$

For areas where $f_0 \neq f_1$, the result will be a blend between the two images as per the ratio of the α values. This will be the area where the green rectangle was drawn.

This process will continue for every frame of the loaded video, only stopping when the video does.

When testing on the provided video - dashcam footage taken from a road in Crikvenica, Croatia - the solution performs well. However, it is not perfect, and several frames of the video cause the detected area to jump erratically to the other lane. Due to the use of a standard Hough line transform, the system cannot recognise dashed lane demarcations with any significant degree of accuracy. However,

C. Limitations

In it's current state, the solution would not be fit for purpose. High-risk applications such as self-driving cars, or anything to do with automotive applications, have very narrow margins of error. Additionally, when undefined behaviour occurs, the consequences can include serious injury up to and including, death.

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*. Jan. 2014. URL: http://mccormickml.com/2014/01/10/stereo-vision-tutorial-part-i/.

[14] MathWorks. *Stereo Disparity Using Semi-Global Block Matching*. 2022. URL: https://ww2.mathworks.cn/help/visionhdl/ug/stereoscopic-disparity.html.

[15] Shaveta Malik and Tapas Kumar. "Comparative Analysis of Edge Detection between Gray Scale and Color Image". In: *Communications on Applied Electronics* 5 (May 2016), pp. 38–43. DOI: 10.5120/cae2016652230.
[16] OpenCV. *Image Filtering*. 2022. URL: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37.
[17] Zhao Xu, Xu Baojie, and Wu Guoxin. "Canny edge detection based on Open CV". In: *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*. 2017, pp. 53–56. DOI: 10.1109/ICEMI.2017.8265710.
[18] Thuy Tuong Nguyen, Xuan Dai Pham, and Jae Wook Jeon. "An improvement of the Standard Hough Transform to detect line segments". In: *2008 IEEE International Conference on Industrial Technology*. Apr. 2008, pp. 1–6. DOI: 10.1109/ICIT.2008.4608701.
[19] Liam Bowers. *Temporal Smoothing to Remove Jitter in Detected Lane Lines*. 2017. URL: https://medium.com/@liamondrop/temporal-smoothing-to-remove-jitter-in-detected-lane-lines-d1430cb5c106.
[20] OpenCV. *Hough Line Transform*. 2022. URL: http://man.hubwiz.com/docset/OpenCV.docset/Contents/Resources/Documents/d3/de6/tutorial_js_houghlines.html.
[21] OpenCV. *Drawing Functions*. 2022. URL: https://docs.opencv.org/4.x/d6/d6e/group__imgproc__draw.html#ga311160e71d37e3b795324d097cb3a7dc.
[22] OpenCV. *cv::Mat Class Reference*. 2022. URL: https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html#a33fd5d125b4c302b0c9aa86980791a77.

TABLE V: 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.