Underwater Visual Perception System

Gerard Duff
Intelligent Embedded Systems
Mälardalen University
GerardDuff@Gmail.com

Debajyoti Nag
Intelligent Embedded Systems
Mälardalen University
dave0908@gmail.com

ABSTRACT

This report documents the work completed by the vision system software team of project Naiad at Mälerdalens Högskola in Västerås, Sweden. The authors were M.SC. students in the final year of an Intelligent Embedded Systems course. The team's goal was to create the vision system software for an Autonomous Underwater Vehicle (AUV) as part of the "Project in Advanced Embedded Systems" course, to enable it to participate in the RoboSub and SAUC-E competitions. The software was developed to effectively search and locate items with a known description or some specified attribute. It can also map a pair of stereo images to a point cloud and additionally can estimate the velocity of the AUV using stereo images.
The system was primarily developed using the Ada programming language, to introduce safety critical features to the system. The system was completed to be a real time visual perception unit with object detection and recognition features.

Keywords

OpenCV, Point Cloud Library, Ada, Computer Vision

1. INTRODUCTION

This report covers the work of the vision system software team of Naiad [16], which was the project undertaken as part of the course Project in Advanced Embedded Systems, DVA425, at Mälerdalens Högskola in the fall semester of 2013[1]. The team consisted of two international students in the final year of the Intelligent Embedded Systems program at the university.

[1]Mälerdalens Högskola
School of Innovation, Design and Engineering
Project in Advanced Embedded Systems DVA425

Mikael Ekström, Evaluator
mikael.ekstrom@mdh.se

Lars Asplund, Customer
lars.asplund@mdh.se

The goal of this team was to develop a vision system for an autonomous underwater vehicle, AUV, capable of providing real-time object detection, recognition, and velocity estimation. The system should also be capable of recreating a three dimensional map of an object using stereo vision.

1.1 Problems Faced

One of the challenges faced by all underwater vision systems is the ever-changing lighting conditions under water. Rays of light shining down through the water can affect the image detection system. A system was needed to deal with this light disturbance.

1.2 Hardware and Software

There were two hard requirements of the project for the vision system. The first was that Ada [1] should be the main programming language. This was to introduce safety critical features to the code and make the system robust and safety critical. The other requirement was that the software should be developed for the research platform GIMME-2 [8].

For real-time image capture and perception OpenCV [20] was chosen and interfaced with the GIMME-2 and Ada. Point Cloud Library [22], PCL, was chosen for the three dimensional (3D) reconstruction which was also interfaced to work through Ada. The operating system chosen to develop the software was Ubuntu 12.04.

1.2.1 Software and Component Decisions

A number of software components were chosen for this project. Below is a description of the main components used and the reasons behind choosing each component.

1. GIMME-2
   This was a hard requirement from the customer. A state of the art platform described in detail in the hardware report.
2. Ada
   This was a hard requirement from the customer. Introduces safety critical characteristics to the vision software system.
3. OpenCV
   This seemed to be the obvious choice after a research

was conducted on the state of the art technologies, and it was found that this library provides easy access to quite a few of the methods intended to be used in the system.

4. Point Cloud Library
   The main decision to make regarding three dimensional (3D) reconstruction was to choose between OpenCV and Point Cloud Library (PCL) to carry out the three dimensional reconstruction. Both methods have advantages and disadvantages.

   OpenCV was already installed and operating on the vision system and using it would mean time saved on installing extra software. However, to use OpenCV the system needs to be trained to recognise objects and this introduces restrictions to the objects that can be detected by the system.

   Point Cloud Library does not have to be trained to reconstruct the images received from the stereo vision system. However, it does require introducing a new software library component to the system, binding to Ada and adapting the makefile to handle the new library.

   It was decided that the greater flexibility given by Point Cloud Library would warrant the extra work needed to implement it into the system.

1.2.2 OpenCV with Ada
Interfacing OpenCV with Ada would be a problem. Extensive research did not yield any instances of the current version of OpenCV (2.4.7) having support for Ada. Bindings were found for an older version of OpenCV (2.2) [19] but as OpenCV has undergone significant changes since then, most notably the method in which image data is handled, these could not be used.

1.2.3 PCL with Ada
As Point Cloud Library is a relatively new software [25], its support for different languages is limited. Much like OpenCV a method to incorporate PCL to Ada was needed.

1.2.4 Hardware Platform
The state of the art hardware platform used for the project, the GIMME-2, presented several challenges to the software team. As the platform was state of the art, support for developing software for it was limited.

2. METHOD
The method will now be described. It is split into two sections. Firstly the preparation steps undertaken to integrate all software components with each other is described. Secondly some of the main software modules implemented are described.

2.1 Preparation
Before the modules for the visual perception system could be developed, the software modules needed to be interfaced to each other. The following subsections describes the work carried out to prepare all the software modules.

2.1.1 OpenCV with Ada
The first task undertaken in this project was to try to get OpenCV working with Ada. Bindings were found online [19] however these binding were found to be rigidly implemented to an older version of OpenCV and an operating system no longer supported. An alternative solution was required. After much research a method of binding C++ to Ada was found [17] and this method was adapted to bind OpenCV to Ada.

This generated bindings that could be used by including the wrapper file in the Ada project, however linking several projects that used the bindings together, and compiling and running the project in an Interactive Development Environment (IDE) proved troublesome. A Makefile was constructed to link everything together.

2.1.2 PCL with Ada
Much like OpenCV, PCL needed to be converted from C++ to Ada. The same approach as OpenCV was undertaken. The PCL modules required were first developed in C++ and then Ada bindings were generated using the binding commands found [17]. The PCL functions could then be called from Ada. The Makefile was then adjusted to incorporate PCL. This simply required adding include directories to the necessary PCL libraries in the Makefile.

2.1.3 Developing for the GIMME-2
As the GIMME-2 was a state of the art hardware platform, development support was limited. Fortunately, the processor architecture of the BeagleBone Black and the Pandaboard are compatible to that of the GIMME-2. As the Ubuntu operating system was not supported by GIMME-2, it was decided to compile the binaries needed on a Pandaboard statically and then move the binaries to the GIMME-2 to be run.

2.2 Software
The visual and perception system was broken down into separate modules for each vision component needed. This was done to modularize the system, making it easy to add and remove modules if and as they were required.

2.2.1 Stereo Vision
The hardware is potentially capable of stereo vision, taking pictures with its dual camera lenses at the same time, which enables depth perception and motion calculations.

Calibrating the Cameras
As the GIMME-2 was having problems taking two pictures simultaneously, a custom stereo rig was constructed to run and test the calibration software. Calibration and Rectification code was constructed using a chessboard image to calibrate the cameras. This code can be tested on the GIMME-2 when it has the ability to take 10 consecutive stereo images needed to run the calibration software.

Disparity Maps
Since the GIMME-2 flaunts a stereo system, disparity maps can be used for motion and distance estimation.

Disparity maps are images that contain not only x an y axis values, but also the estimated z axis too. OpenCV functionalities were used for this purpose.

2.2.2 *Image Enhancement*

Cleaning up the image was attempted by four methods. Blurring the image, sharpening the image to make features more clear and easier to detect, manipulating image properties, and noise reduction.

Median Filter

The idea behind the Median filter is to search for highly improbable pixel values, and replace them with the median value of the surrounding pixels [4].

The median filter makes a 3x3 sliding window that reduces noise by filtering every pixel of the image. It works by replacing the value at the centre of the window with the median value of the pixels in the window.

If the window N is centred at (i, j), the median filter can be represented by the following equation as shown by Forsyth (2011) [5]:

$$y_{ij} = med(\{x_{uv} | x_{uv} \in N_{ij}\}) \tag{1}$$

Quaternion Switching Filter

The quaternion switching filter [7] is a filter based on quaternions. A quaternion is a vector representation that has one real part and three imaginary parts as Hamilton discovered [9] of the form:

$$Q = w + ix + jy + kz \tag{2}$$

Quaternions are useful in image processing as an RGB pixel can be represented as a purely imaginary quaternion by letting the real part of the vector equal zero. A quaternion with only imaginary parts is known as a right quaternion [10]. The quaternions allow representing a three-value RGB pixel as a single function. Then quaternion unit transform theory can be used to find the pixel chromaticity difference and pixel intensity difference for the pixels in the sliding window.

The intensity difference and chromaticity difference can be represented as shown by Geng (2012) [7]:

Where
r, g, b: red, green and blue color components with a range between 0 and 255
i, j, k: imaginary coefficients

$$\bar{T}qT = \frac{1}{3}(r+g+b)(i+j+k) - \frac{1}{\sqrt{3}}[(b-g)i + (r-b)j + (g-r)k] \tag{3}$$

$$Tq\bar{T} = \frac{1}{3}(r+g+b)(i+j+k) + \frac{1}{\sqrt{3}}[(b-g)i + (r-b)j + (g-r)k] \tag{4}$$

The intensity difference:

$$d_1(q_1, q_2) = (\frac{1}{2}|[(Tq_1\bar{T} + \bar{T}q_1T) - (Tq_2\bar{T} + \bar{T}q_2T)]) \tag{5}$$

resolves to

$$\frac{1}{3}(i+j+k)(r+g+b) \tag{6}$$

and the chromaticity difference

$$d_2(q_1, q_2) = (\frac{1}{2}|[(Tq_1\bar{T} - \bar{T}q_1T) - (Tq_2\bar{T} - \bar{T}q_2T)]) \tag{7}$$

resolves to

$$\frac{1}{\sqrt{3}}[(b-g)i + (r-b)j + (g-r)k] \tag{8}$$

If the combined greatest difference, $d_1 + d_2$, is greater than a user defined threshold then the pixel is subjected to a Vector Median Filter. If the difference is less than the threshold then no filtering is performed on the pixel.

2.2.3 *Manipulating Image Properties*

Image properties were manipulated in order to enhance certain features of images. This helped improve the performance of the object detection modules.

Image Contrast

This is a basic image manipulation functionality targeted to enhance the image quality if required. It makes use of the OpenCV function *convertTo* using alpha and beta as gain and bias parameters respetively.

Invert Image

In this context, *Invert* refers to the reversal in the nature of the pixel intensities of the image. For instance, an originally pure black input pixel yields a pure white pixel as output. It makes use of the simple form of

$$V_{new} = 255 - V_{old} \tag{9}$$

where V is the value of the pixel in BGR, to invert an image. This method is particularly helpful when working with masks.

In the final implementation, the OpenCV function *addWeighted* was used to achieve the desired functionality, with constants 1.00 and -1.00 for alpha and beta channels respectively.

$$addWeighted(mask, 1.0, src, -1.0, 0, dst)$$

where the mask is a pure white image.

Split Channels

This splits an input image into it's basic color components, so the system could have color-channel filtering without use of any extra hardware.

Gaussian Blur

The Gaussian blur function [6] needed to be implemented for the edge detection modules. A blur basically smooths out the image and makes jagged edges smoother and the image works better with the edge detection modules. Gaussian blur can also be used for noise reduction. The Gaussian kernel G(x, y, σ) can be represented as depicted by Korn (2000): [12]

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma} exp(-\frac{x^2 + y^2}{2\sigma^2}), \tag{10}$$

where the two free co-ordinates are represented by x, y and σ is a parameter.
The Gaussian blur works by using a two dimensional

Gaussian kernel that can be separated into two operations of the one dimensional kernel. This enables very fast implementation of multidimensional Gaussian filtering [12].

Gaussian Sharpen
A Gaussian sharpen image module was implemented. This method was based on the Gaussian blur method as described above. It works by implementing a Gaussian blur on the source image, and then subtracting the Gaussian blurred image from the original image. This results in a sharpening of the image.

Image Fusion
Several situations in image processing require both high spatial and high spectral information in a single image. However, this is not always possible due to design or observational limitations. To overcome these limitations, image fusion techniques are applied.

Naiad's vision system implements single-source image fusion techniques to enhance image clarity and quality. However, this module is not used by default due to cost in terms of execution time. But with time, as the FPGA becomes usable, this module could be easily enabled.

For future, it is being planned to have a multi-focus fusion module in place to work alongside the stereo-vision system to help find objects more efficiently [13, 32, 33].

Enhance Colors
This function was implemented by converting an image to a Hue Saturation Intensity (HSI) image, and then scrolling through each pixel in the image and adding a number to the intensity to increase the intensity of each pixel. This enhanced the colors of the image. This function was particularly useful when detecting red templates. When pictures of the red templates were taken it was observed that in parts of the template the red appeared a little faded. The enhance colors module worked a treat in enhancing the red color which led to better results when the canny and contour modules were run on the template.

2.2.4 *Interpreting Image*

After the image preprocessing modules of the vision system have completed, the image processing modules work on feature and object detection. The following modules were implemented to search for features on a given image.

Threshold
The vision system has the ability to threshold the input image to filter for any specified range of the visual spectrum. It works on HSI images, and it was particularly challenging to filter the color red as the hue values see a wrap-around effect for red, meaning that the spectrum begins at red and ends at red.

This problem was solved by working around the easily detected colors. To detect red, firstly yellow and green were filtered separately. Then the orange component was filtered from a bitwise OR-combination of the previous outputs. The resultant output is the red component. The OpenCV function *inRange* is used here.

Canny Edge Detection
The Canny edge detection algorithm [2] was used for detecting edges in the images. Canny states that the optimal detector for step edges would be a convolution with a symmetric Gaussian and directional second spatial derivatives [3]. The Canny algorithm is based on three main objectives [18]:

1. The response to noise is reduced using Gaussian filtering
2. Use non-maximum suppression to obtain good accuracy (only use points from the top of a ridge of edge data, whilst suppressing all others)
3. Eliminate multiple responses to a single edge by providing a single response

The Canny module implemented takes a grey scale input and outputs a binary image where the estimated lines are white on a black background.

Contour Tracing
The contour module used was based on the OpenCV contour module. This module is based on Suzuki and Abe which analyzes the topological structure of binary images by following the detected borders [27]. The contour module searches through the canny input and finds contours on the image. This can then be used in shape or template recognition modules.

Approx Poly
After the contours were found the OpenCV function approx poly was used to approximate the curves found in the contour function. The approxpoly function is based on the Douglas-Peucker algorithm [23]. This was initially used to find shapes in test images supplied to the vision system. If three contours are found the shape is assumed to be a triangle, if four contours are found the shape is assumed to be a square. This was later replaced by OpenCV's shape matching function which provided more accurate results.

Good Features to Track
OpenCV's good features to track module was implemented to use with optical flow. This yielded better results with optical flow than from using optical flow with contours. This module is based on the Shi-Tomasi algorithm [26]. This module uses the algorithm to find the most prominent corners in an image region. The method also shows comparatively better results than the Harris corner algorithm [26].

Optical Flow Tracking
Optical flow tracking was implemented to use as part of velocity estimation in the vision system. Optical flow tracking in OpenCV uses the Lucas-Kanade method and uses a sparse iterative method [21], which computes optical flow for a sparse feature set using features detected in the good features to track module described above. This module was used to find an estimated velocity for the AUV.

2.2.5 *Comprehending Features Found*

When features on the image have been found, the vision system uses the features collected to interpret the image. The following modules were implemented to look for certain shapes and templates.

Hough Circles

OpenCV's hough circles function was implemented to enable the ability to detect circles. The Hough circles module is based on the Hough circle transform as described in Yuen [31].

Hough Lines

The AUV was equipped with line detection capability by implementing OpenCV's Hough lines module. The module is based on Matas (2000) [15]. This can be used to find objects with parallel lines like pipes etc.

Template Matching

OpenCV's *matchShapes* function was used for the object recognition part of the vision system. This function is based on Hu Invariants [11]. The matchShapes module is also scale invariant.

Velocity Estimation

The velocity estimation module implemented was based on OpenCV's optical flow module. It has two modes. First it checks to see if any "large" objects are detected (pipes, barrels etc.) and if they have been detected it uses optical flow to find the average difference of position of the features, and divides by the image capture rate to give an estimated velocity in pixels per second.

The second mode is used if there are no large objects detected. It then uses optical flow to search for particles in the water, or on the floor of the pool/seabed, and estimates the velocity as in the previous mode, except using the positional difference of the small particles. The reason for the different modes is that smaller particles are harder to pick up, and sometimes light in the water can be mistaken for particles and have a negative affect on the velocity estimation. Therefore it is more reliable to estimate optical flow using features of a larger object, and the first mode is preferred.

2.2.6 *Three Dimensional Reconstruction*

Point Cloud Library, PCL, was chosen for the three dimensional reconstruction part of the vision system. Due to timing constraints only a few basic filters and PCL's greedy surface reconstruction algorithm were implemented through Ada.

Converting to Ada.

PCL was converted to Ada in much the same way as OpenCV. Modules were written in C++ and then bound to Ada [17]. The PCL modules could then be called from Ada.

Down-sampling a Point Cloud

The point clouds to be reconstructed to 3-D must undergo stages before the actual reconstruction can occur. Different filters were implemented to clean the point cloud to obtain better results.
The first PCL module implemented was a down-sampling module [30] . The down-sampling module is based on a voxelized grid approach. The size of the voxel is specified and all points in that voxel are approximated with their centroid.

Remove Outliers

A statistical outlier removal module was constructed to remove outliers from point clouds [28]. The module works by computing the mean distances of every point to their neighbours. Then all points whose mean distances are outside a defined interval are assumed to be outliers and are removed from the point cloud.

Conditional Euclidean Segmentation

This module was based on the conditional euclidean segmentation tutorial on the Point Cloud Library website [22]. The tutorial combines euclidean cluster extraction [24] and region growing segmentation [29]. Euclidean segmentation breaks large point clouds down into smaller, more manageable data units. This improves computational time for the module. The region growing segmentation algorithm then searches for points on the same smooth surface. The segmented point cloud is then plotted using different colors to show the detected object clusters.

Triangulation Mesh

The PCL greedy triangulation mesh algorithm was implemented. This constructs a triangular mesh around the point cloud, effectively turning the point cloud from a group of points to a reconstructed surface. It works by finding a list of points a mesh can be constructed from and expands the list to include suitable points [14].

3. RESULT

The software developed for the vision system using OpenCV was run on the GIMME-2, using datasets as inputs for the modules, as the GIMME-2 wasn't showing consistent and acceptable performance in capturing images at the time.

The PCL software was not tested on GIMME-2 due to timing constraints, but it was tested on the Pandaboard and hence, should run on the GIMME-2 without any problem.

Some images for the tests can be found in the appendix .

4. CONCLUSION

Overall the vision system completed the goals set out at the start of the project. As the GIMME-2 was not functioning reliably throughout the project, with stereo image capture taking up to ten minutes to capture an image from each camera, the software could only be tested using image datasets and not from the GIMME-2's cameras.

4.1 Future work

Computer vision is a vast area and throughout the project the team discovered master theses, and Ph.D theses dedicated to different parts of computer vision so there will always be scope to improve the vision system. With specific regards to project Naiad's vision system there are a number of areas that will require some work to successfully integrate all hardware and software.

4.1.1 Testing

The system can be tested fully using integration, unit tests, system tests and acceptance tests. These would have to be written manually as the test suite AUnit cannot be used with the bindings.

4.1.2 Integrating Disparity Maps

The disparity maps obtained from the GIMME-2 when it functions properly can be integrated into the velocity estimator function. The disparity map can be integrated with the velocity estimator function to receive the estimated distance in centimetres instead of estimated pixels. This could not be implemented as the GIMME-2 image capture was not functioning to a standard that allowed consecutive stereo imagemode 1s to be captured.

4.1.3 Integrating CAN to the System

The Controller Area Network (CAN) controller for the GIMME-2 has been completed by the vision firmware team. This code must be added to the vision software system main file.

4.1.4 Instruction Handling System

A system for handling messages from mission control system must be implemented.

4.1.5 PCL Future Work

PCL was implemented towards the end of the project. Only a few basic filters and a surface reconstruction module was implemented, leaving lots of room for future extension.

Poisson Reconstruction

This implements a "watertight" mesh around the point cloud, meaning that any gaps or holes like windows in the point cloud will be filled in. This may be counteracted by using PCL's concave/convex hull estimator as a "bounding box" to give a more accurate mesh around the point cloud.

Adding Colour to a Mesh

Colour can be added to the reconstructed mesh by using the color intensity values from the original stereo images, and painting the mesh accordingly.

5. REFERENCES

[1] Ada reference manual. `http://www.ada-auth.org/standards/12rm/html/RM-TTL.html`. Accessed 2014-01-14.

[2] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[3] R. Cipolla. *Active Visual Inference of Surface Shape*, volume 1016 of *Lecture Notes in Computer Science*. Springer, 1996.

[4] E. R. Davies. Computer and machine vision theory, algorithms, practicalities, 2012.

[5] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Always learning. Pearson Education, Limited, 2011.

[6] Gaussian blur filter. `http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm`. Accessed 2014-01-07.

[7] X. Geng, X. Hu, and J. Xiao. Quaternion switching filter for impulse noise reduction in color image. *Signal Processing*, 92(1):150 – 162, 2012.

[8] Gimme 2. `http://www.xilinx.com/products/boards-and-kits/1-3BWL4K.htm`. Accessed 2014-01-07.

[9] W. R. Hamilton. On quaternions, or on a new system of imaginaries in algebra. *Philosophical Magazine.*, 25:489–495, 1844. 3.

[10] W. R. Hamilton. Hamilton elements of quaternions. *Article*, 285:310, 1866.

[11] Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8:2:179–187, 1962.

[12] G. Korn and T. Korn. *Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review.* Dover Civil and Mechanical Engineering Series. Dover Publications, 2000.

[13] S. Li, J. T. Kwok, and Y. Wang. Multifocus image fusion using artificial neural networks. *Pattern Recogn. Lett.*, 23:985–997, jun, 2002.

[14] Z. C. Marton, R. B. Rusu, and M. Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 12-17 2009.

[15] J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, CVIU 78 1:119–137, 2000.

[16] Naiad. `http://www.naiad.se/`. Accessed 2014-01-07.

[17] C++ to ada bindings. `http://wiki.ada-dk.org/c_bindings_example`. Accessed 2014-01-07.

[18] M. Nixon and A. S. Aguado. *Feature Extraction & Image Processing, Second Edition.* Academic Press, 2nd edition, 2008.

[19] Ada opencv bindings. `https://code.google.com/p/opencvada/`. Accessed 2014-01-07.

[20] Opencv. `http://opencv.org/`. Accessed 2014-01-07.

[21] Opencv on optical flow. `http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html`. Accessed 2014-01-07.

[22] Point cloud library. `http://pointclouds.org/about/`. Accessed 2014-01-07.

[23] D. D. . T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112âĂŞ122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[24] R. B. Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments.* PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[25] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[26] Shi and C. Tomasi. Good features to track. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 6 1994.

[27] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. CVGIP 30 1:32 – 46, 1985.

[28] Point cloud library, outliers tutorial. `http://pointclouds.org/documentation/tutorials/statistical_outlier.php`. Accessed 2014-01-07.

[29] Point cloud library, region growing tutorial. `http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php#region-growing-segmentation`. Accessed 2014-01-07.

[30] Point cloud library, downsampling tutorial. `http://pointclouds.org/documentation/tutorials/voxel_grid.php`. Accessed 2014-01-07.

[31] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vision Comput.*, 8:71–77, 1990. 1.

[32] H. Zhang, X.-n. Sun, L. Zhao, and L. Liu. Image fusion algorithm using rbf neural networks. *Hybrid Artificial Intelligence Systems*, 5271:417–424, 2008.

[33] S. Zheng. Pixel-level image fusion algorithms for multi-camera imaging system. *The University of Tennessee, Knoxville*, 8 2010.

.

6. APPENDIX A

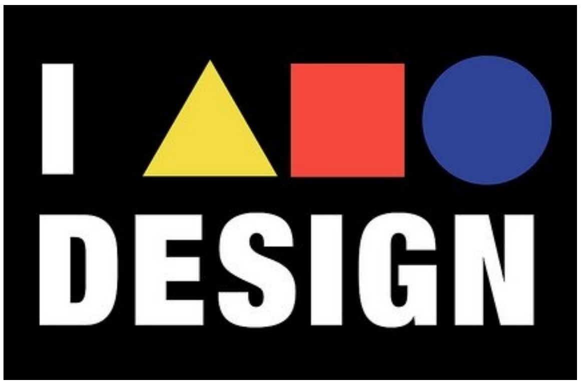

Figure 1: Sample Input

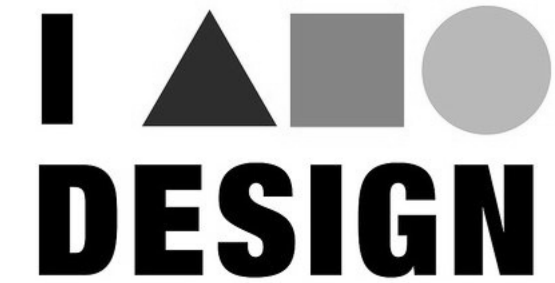

Figure 2: Invert Output

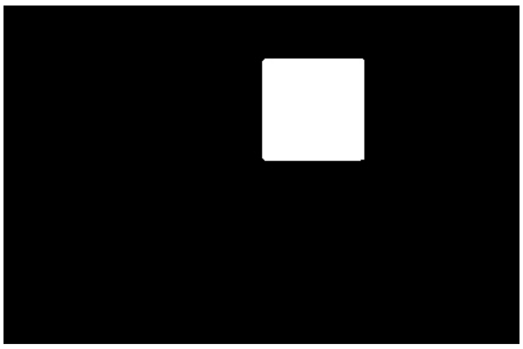

Figure 3: Sample Output for Red Threshold Masking

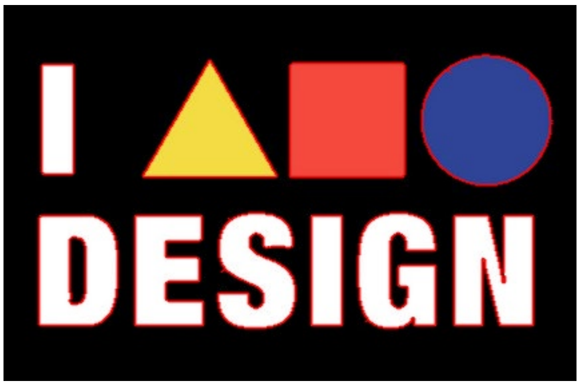

Figure 4: Sample Output for Contour Tracing

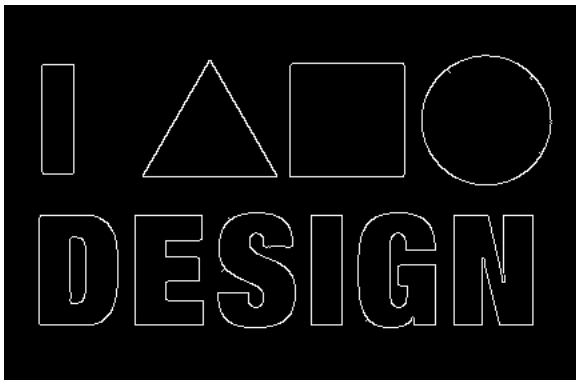

Figure 5: Sample Output for Canny Edge Detection

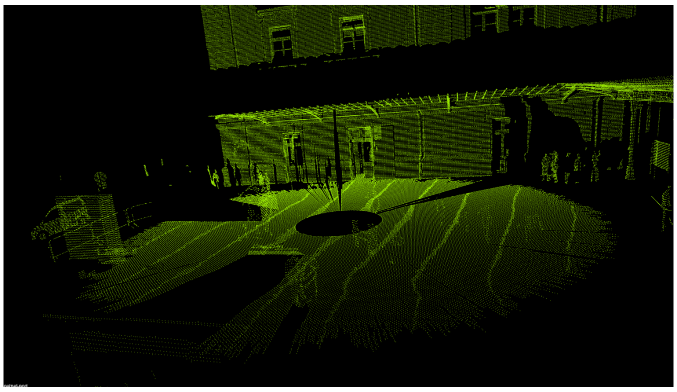

Figure 6: Sample Input for Conditional Euclidean Segmentation

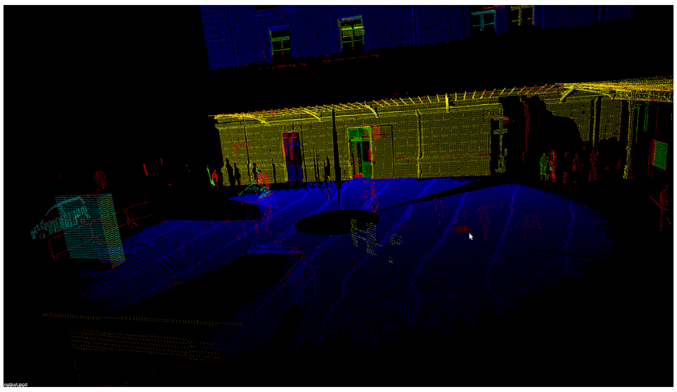

Figure 7: Sample Output for Conditional Euclidean Segmentation

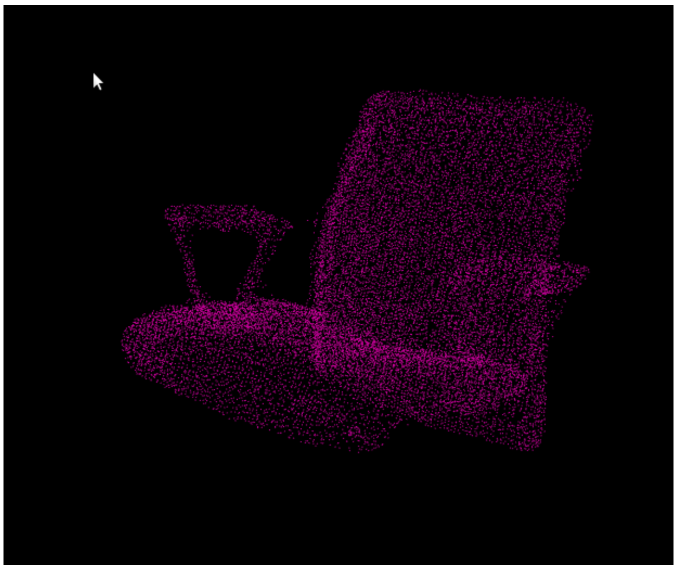

Figure 8: Sample Input for Surface Reconstruction

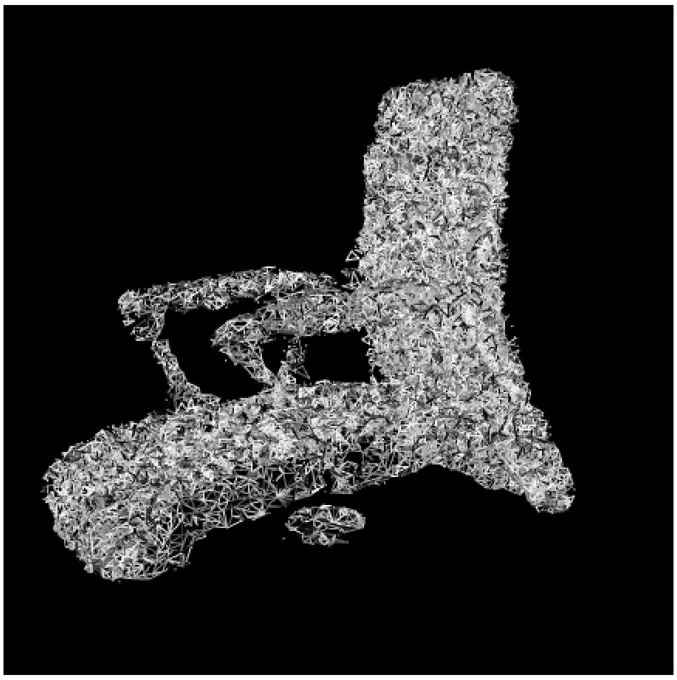

Figure 9: Sample Output for Greedy Triangulation Algorithm

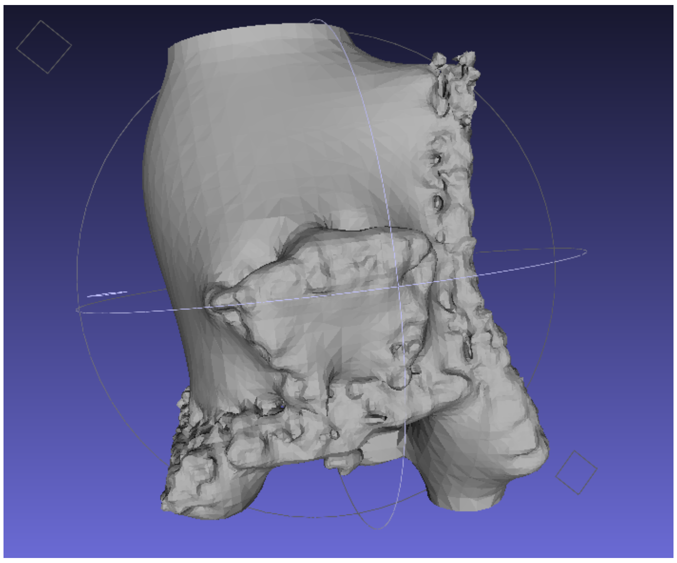

Figure 10: Sample Output for Poisson Algorithm