Meta-Interpretive learning from 3D shapes

A thesis submitted in partial fulfilment of the
requirements for the award of the degree

Bachelor of Engineering (Mechatronics)

from

University of Wollongong

by

Jared L. Bellingham

School of Electrical, Computer and Telecommunications Engineering

October, 2019

Supervisor: A/Prof David Stirling

Abstract

As society begins to rely more heavily on robotics and autonomous systems the field of Computer Vision is an increasingly relevant branch of computer science. Image classification is an important area of computer vision prevalent in many modern applications. Most recent image classification systems rely on statistical machine learning. However these systems depend heavily on large data-sets for training and do not provide reasoning for their decisions. Multiple alternative systems, both statistical and not, have been proposed to address these issues. One such system is Logical Vision, which aims learn symbolic descriptors of images given a low number of training examples. Logical Vision has shown success in classifying 2D polygons, describing natural phenomenon such as the reflection of light and detecting real-world objects. However, all implementations have been done using 2D images. This work aims to explore the feasibility of incorporating depth information within the Logical Vision framework. In particular, it is shown that valid symbolic representations can be extracted from mid-level features of RGB-D images.

A experimental framework for comparing the performance of the proposed system is presented. It aims to compare three systems, which differs by the type of inputs used as training data and test data. These inputs are 2D RGB, depth (point cloud) and a combination of RGB and depth data. The goal of the investigation is to prove that the performance of an inductive image classifying system can be improved with the combination of RGB and depth information in comparison to systems that only use one of the two. Since a complete system was not developed by the conclusion of ECTE451, this proposed experiment was not completed. The results of pre-processing outputs and robustness and initial Meta-interpretive learning are presented to show that a symbolic learning system that incorporates both RGB and depth information is feasible and worth investigating further in ECTE458.

Statement of Originality

I, Jared L. Bellingham, declare that this thesis, submitted as part of the requirements for the award of Bachelor of Engineering, in the School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications or assessment at any other academic institution.

As author of this thesis, I also hereby grant, subject to any prior confidentially agreements, SECTE permission, to use, distribute, publish, exhibit, record, digitize broadcast, reproduce and archive this work for the purposes of further research and teaching.

(strike out that which does not apply)

This thesis;

~~IS~~

IS NOT

subject to a prior confidentially agreement.

Signature:	[illegible]
Print Name:	JARED L. BELLINGHAM
Student ID Number:	5104701
Date:	20/10/2019

Contents

List of Tables

List of Figures

Abbreviations and Symbols

I	Image
E	Example
B	Background Knowledge
H	Hypothesis

LV	Logical Vision
MIL	Meta-Interpretive Learner
SML	Statistical Machine Learner
DNN	Deep Neural Network
CNN	Convolutional Neural Network
CAD	Computer Aided-Design

Chapter 1

Introduction

Computer vision is the process of extracting or inferring information from an image. This could be viewed as a simple task, as humans have the ability to do it from an early age. However, computer vision is still in its infancy and current systems still have many limitations. Image classification falls under the broader area of computer vision. It is the process of putting an object found in a digital image into a class (for instance the recognition of pedestrians and other vehicles by an autonomous driving system.

The most common image classification systems are based on statistic machine learning (SML), with great success coming from learners using deep neural networks (DNN). Training SML systems requires large data-sets, which are often expensive or not practical to colledt [1]. The systems are also treated as a 'black-box' classifier as they cannot easily show reasoning for their decisions. SML systems are also very sensitive to changes in low-level features [2] and it is difficult to include background knowledge.

Due to these limitations SML image classification differs dramatically from human visual perception [3]. This presents many issues as we begin to rely more heavily on computer vision in tasks like vehicle navigation [4]. If a system is unable to provide reasoning for its decisions, especially in the event of unplanned behaviour, it becomes difficult to ensure that the design has been developed to prevent that specific event from happening again. It also creates problems when wanting to place blame on the system in the event of legal action requiring the system or the designers to be held accountable for misbehaviour [5].

Logical vision (LV) ([6]) aims to address some of the issues outlined above. LV is an image classifier which is able to learn visual concepts symbolically. It can be trained with a limited number of images, down to a single example [7]. LV has been shown to be able to classify simple 2D polygons [6] and real-world objects [7] from a small data-set of images.

While great progress has been made in the field of 2D image classification, 3D processing is falling behind the demand of industry applications including robotics and autonomous driving ([8]). Depth information has been successfully used to increase the performance of SML image classifiers [9], [10]. In [11] it is shown that ILP can been used to analysis 3D point cloud data for scene analysis. However, in contrast to LV [6], [7], all sampling of low-level features is done outside the context of ILP. This paper explores the feasibility of including depth information in the LV process with the aim of improving the performance of LV image classification for noisy images. As a first step, feature analysis is done outside the framework of LV, using MATLAB for pre-processing. However future work is proposed to move this implementation within the framework.

Chapter 2

Literature Review

As society begins to rely more heavily on robotics and autonomous systems the field of computer vision (CV) is becoming an increasing relevantly branch of computer science. CV is a subfield of artificial intelligence dealing with the training of computational systems to extract high-level understanding from digital images and videos [12]. CV has been studied since the late 1960's [13]. However, even after many years and a huge amount of research and published work, it is still in its infancy. When compared with the human visual system, CV is only able to complete relatively simple tasks.

Image classification is a vital subset of CV and focuses on the classification of objects found in an image. Extensive research into the use of image classification has been conducted in many different fields including medical examinations [14], autonomous vehicles [15], [16] and manufacturing ([17]). The most common image classification systems use a statistical machine learning approach, with most success coming from deep neural networks and convolutional neural networks (CNN), in particular [18]. These processes work by sampling low-level features. This creates difficulties as the models created are incredibly abstract, and provide little feedback on the decision making process. To achieve reliable results SML systems typically require large data for training [1]. This becomes expensive, reducing both access to quality SML image classifiers and the viability of retraining systems to suit individual needs. These two limitations greatly reduce the applicability of SML images classification to industry. As we give computer systems increased power over decision making, we introduce significant legal implications [19]. SML systems can be treated as a 'black box' decision makers as they cannot easily provide reasoning for their choice [20]. It then becomes hard to place distribute responsibility in the event of malfunction or dangerous behaviour. Moosavi-Dezfool et. al. [2] show that CNN systems are susceptible to small changes in low-level features. This introduces a significant security issue as these systems are integrated more and more into society. In [21] it is shown that SML image recognition systems can be fooled into producing high confidence predictions for unrecognisable images. Both in [22] and [23] it is demonstrated that SML systems can be led to misclassify natural images that have had artificial perturbation down to changing only one pixel. Humans can use both high-level vision and background knowledge to allow for one-shot or even zero-shot learning. It has been shown that most SML computer vision systems have distinct differences to human visual perception [23]. Various SML processes have been proposed which use high-level feature extraction for one- or two-shot learning. These models are typically trained on large bodies of images representing only a subset of classes. The models are then adapted to suit unseen data. Zhang and Saligrama [24] propose a method which attempts to express target data as a combination of seen class proportions. In a similar method [25], the authors uses semantic knowledge to extrapolate to unseen classes, as long as the class is included in the knowledge base. These methods have shown success however they still require large training data-sets and significant supervision.

Inductive logic programming (ILP), introduced by Muggleton in 1991 [26], is a branch of machine learning based on logical programming. Since then, the advancement of ILP has continued with a variety of industrial applications. Extensive research has been done in the field of drug design. Kijsirkul et al. [27] show that ILP techniques can be used to predict the activity of untested drugs. The symbolic nature of the process provides key insights and is one of the advantages of ILP. Protein shape prediction is still considered a significant problem in molecular biology. In [28], it was shown that ILP was able to boost predictions from about 70% to 80%. ILP has also been used to learn visual concepts. Cohn et al. [29] show that ILP can be used to form general relational concepts from pre-symbolised visual data. In this approach existing low-level computer vision techniques where used to extract a symbolic representation of the scene. In [30] a similar technique is used to analyse hand-printed Chinese characters. Once again images were pre-processed to form symbolic representations before ILP was used to perform inductive learning.

Metagol [31], [32], [33] is an application of ILP which is based on meta-interpretive learning (MIL) [26], [34], [35], [36]. MIL facilitates predicate invention through the construction of substitutions into user defined meta-rules (allowable predicate structure).

Logical vision [6] presents an alternative approach to image classification. It is based on an implementation of Metagol and is able to learn visual concepts symbolically. It differs from the uses of ILP in learning visual concepts. LV uses ILP to support the incorporation of background knowledge during scene analysis with regards to high-level relations. In published work LV has been shown to classify simple 2d shapes [6], to recognise real-world objects such as a soccer ball in real application domains such as a RoboCup game [7] and also to invent background knowledge about natural phenomenon such as the interaction of light [7]. In both [6] and [7] it is shown that LV outperforms the compared SML systems when presented with limited training examples. In [6], Dai et al. introduce LV and demonstrates its ability to learn simple polygon shape symbolically using modern ILP techniques. Using both predictive accuracy and F1-score, it is proven that LV outperforms all statistics based learners by a significant margin. This was extended in [7], where LV was improved to facilitate learning from noisy images including those with partially obscured objects. LV outperformed other statistical based systems in determining the direction of out of frame light sources in images of protist and of the moon. Moreover, LV can be used to detect real-world objects, outperforming CART, a SML system, by significant margins when detecting a ball in the RoboCup setting.

While great progress has been made in the field of 2D image classification, 3D understanding is falling behind the demand of industry applications including robotics and autonomous driving. Furthermore, the addition of depth information to image classification systems can improve the accuracy and reliability of the systems [37].

As opposed to RBG image datasets (some exceeding 2.5 million images), there is currently no RGB-D dataset available to train an RGB-D SML system solely from RGB-D data [38]. Most RGB-D image classifiers work in one of two ways, although all must be initially trained using RGB datasets. From there, systems are either fine-tuned with smaller RGB-D image sets or the existing RGB system is used with RGB-D data that has been processed to present the depth information within a standard RGB image. In this way [39] transfers learning from deep CNNs, that have been pre-trained for image classification, by colourising depth information according to the distance a point is from the centre of the image. Recently some

SML systems have been proposed that fuse RGB and depth information in multi-stream CNNs. In both [37] and [40] it is shown that two-stream CNNs can be used as multisensory object recognition systems. Both systems use parallel networks, one RGB stream and one depth stream, which are converged into one fully connected layer towards the end of the network. In both systems the depth CNN is pre-trained with an RGB data-set and then in a similar way to [39], the CNN is feed with RGB-D images rendered as RGB image containing depth data spread across the three colour channels. Schwarz et. al's [37] system is shown to outperform state-of-the-art RGB SML object detectors in terms of accuracy and localisation. Despite the success of these methods, until larger RGB-D models are created the full capabilities of RGB-D SML image classifiers has not been realised.

Farid and Sammut [11] shows that ILP can be used to analyse 3D point cloud data for scene analysis. Pre-processing is used to symbolically represent planer surfaces which are used as examples for an ILP system. The paper also demonstrates that Metagol is capable of predicate invention using 3D point cloud data. However, in contrast to LV [6], [7], all sampling of low-level features is done outside the context of ILP. The presented work aims to extend the previous development of Logical vision with the use of 3D point cloud data. This will involve the analysis of scenes in RGB-D images to perform image classification. Initial implementation will be performed with low-level analysis done in Matlab to extract high-level features used with ILP to learn symbolic descriptions of objects. This is an extension of the work in [11] with the addition of RGB background knowledge and invented predicates. This will be adapted to the Logical Vision (polygons) framework which uses background knowledge to provide a bridge from low-level features to high-level interpretations of objects. Due to its nature this extension of Logical Vision can be implemented purely in Prolog.

Chapter 3

Proposed Framework

The following details the proposed image classification framework for the classification of simple 3d shapes. The framework is based on the inductive logic programming implementation Logical Vision (LV) [6]. An experiment is also proposed with the purpose of comparing the use of RGB data, depth data and the combination of RGB and depth as training examples for the proposed image classifier.

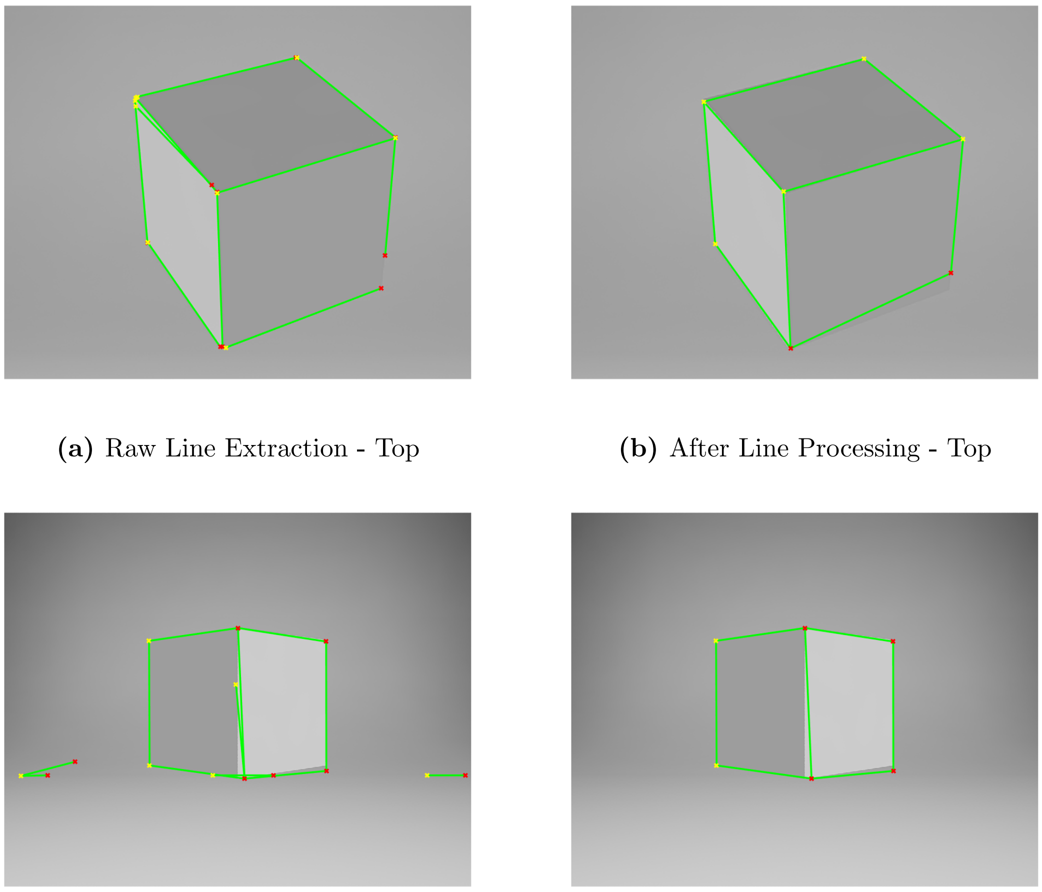
LV is a special case of a Meta-Interpretive learner (MIL) [26], [34], [35], [36]. LV takes background knowledge B and one or more images I to form a set of logical expression H which characterise the positive examples provided. The logical vision framework consists of two distinct parts. The first is mid-level feature extraction which aims to generate symbolic representations of images in the form of logical facts. The second process involves the use of a generalized Meta-Interpretive Learner to generate logical hypothesis of the visual concept. Due to the restricted timeframe for ECTE451 extraction of mid-level features has been done in MATLAB in a heavily supervised way. Features are manually added to Prolog, removing all noise, and Metagol [31], [32], [33] is then used to generate logical representations. This method to perform similar analysis and learning within the framework of LV will be developed in ECTE458. This method will be unsupervised and will thus require a noise tolerant implementation of MIL, such as Metagol$_{\mathrm{NT}}$ [7].

3.0.1 Mid-Level Feature Extraction

Matlab

In the current implementation mid-level feature extraction is done in Matlab. The implementation is shown in C.0.1. All custom functions used are also located in Appendix C.

Lines Images to build and test feature extraction implementation have been taken from 3D renders done in CAD software as well as from images of real objects. Line



(a) Raw Line Extraction - Top

(b) After Line Processing - Top

(c) Raw Line Extraction - Side

(d) After Line Processing - Side

Figure 3.1: Before and After Line Processing

extraction is done using inbuilt MATLAB functions to perform Hough transform (C.0.2). Once raw lines have been extracted the lines are manipulated to join any close endpoints to form a vertex. After joining, all redundant lines are removed. This includes lines which have free end points or lines that appear more then once. This helps eliminate the effect of background noise but can also produce errors. Figure 3.1 shows before and after the functions for line joining (C.0.3) and invalid line removal (C.0.4) are called.

Polygons The extracted line data is in the form of a structure detailing lines with their end points. As detailed above, these lines will have been processed to form one or more polygons. The following, details how the line data is used to extract the polygons. As shown in Figure 3.2, in order to find an unidentified polygon first we must find a point (vertex) which is not found in any already identified polygons.

This is done by the function shown in Appendix C.0.5. During the first iteration this will just be the first point.

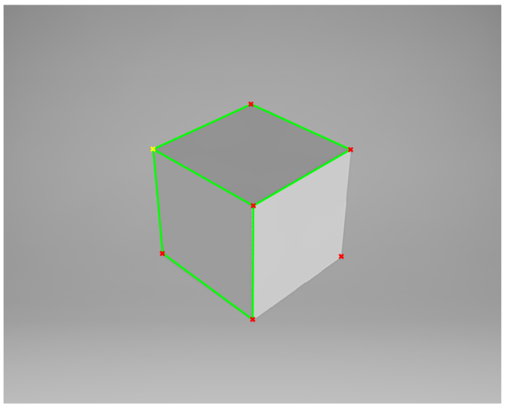
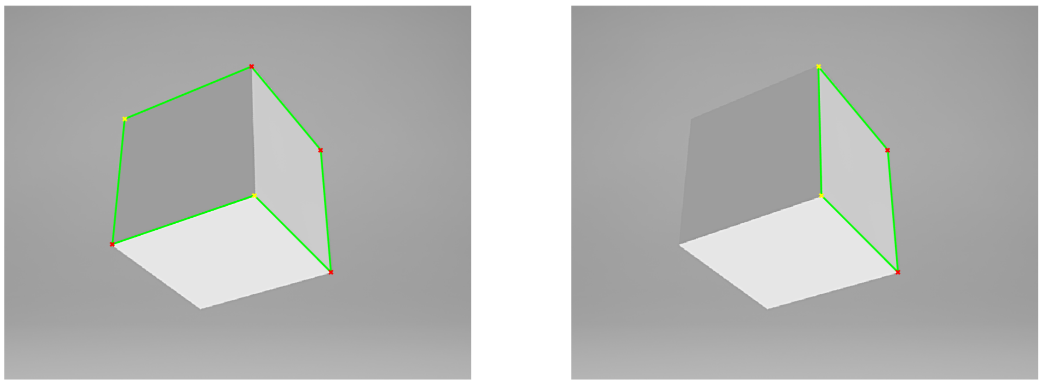


Figure 3.2: Free Point Selection Used to Find All Polygons

Once an unassigned point is found the function found in Appendix C.0.6 is used to generate a possible polygon. This is done by randomly following a path or connected lines until the path comes back to the starting point. Often this does not form a valid polygon; as the speed of the program wasn't a concern for this implementation a more guided method was not developed. However, if this was to be used in a real-time application work could be done to improve speed of this operation. Once a possible polygon has been produced the polygon is validated. A function (C.0.7) determines if the shape contains only one region and if the shape is the simplest possible. Please see Figure 3.3 for examples of a valid and an invalid case. If the polygon is validated it is added to list of polygons.

Once a polygon is determined another unassigned point is deduce from the remaining lines. This continues until all polygons have been extracted. Table 4.1 and 4.2 show examples of the extracted data.

Planes A suitable dataset of RGB-D images of simple 3D shapes couldn't be found during ECTE451, so plane data for MIL was added manually by the user. However, once an RGB-D dataset can be made, the script in Appendix C.0.9 shows how information about planes can be extracted. Depth images used for development have been taken from [38]. The implementation separates point cloud data into different planes, creating individual point clouds for each plane. From this, analysis



(a) Invalid Polygon Containing Extra Line **(b)** Valid Polygon

Figure 3.3: Comparison of Valid and Invalid Polygons

can be performed to determine plane angles and other information. As an example Figure 3.4 shows the original point cloud and Figure 3.5 shows the first two extracted planes. In ECTE458 the analysis of depth data will be further developed to include the calculation of angles between planes and the determination of which plane each polygon lies on.

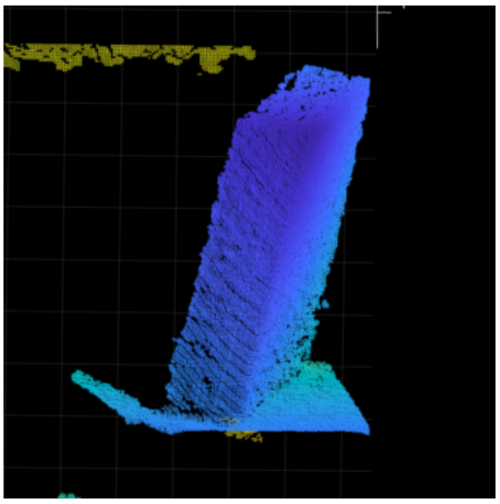
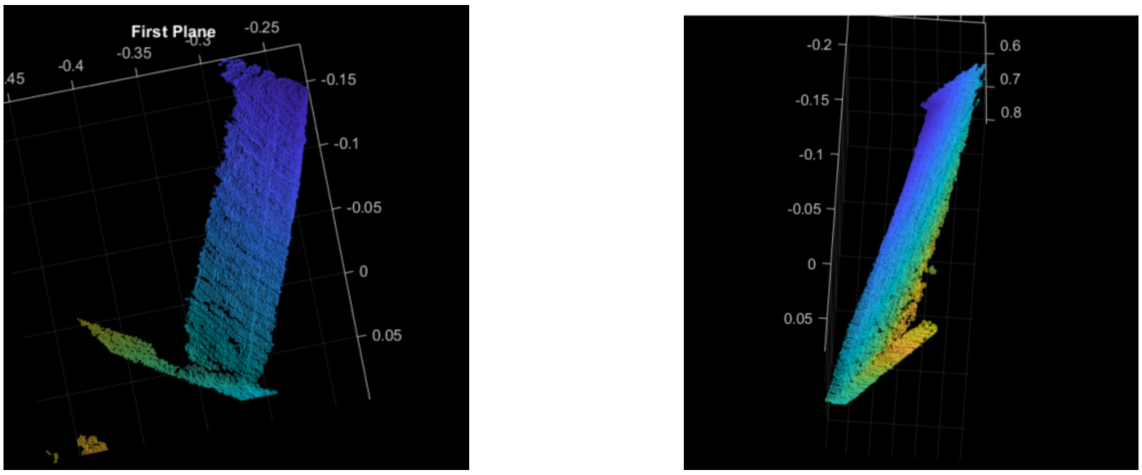


Figure 3.4: Complete Point Cloud of Cereal Box

3.0.2 Logical Vision

In ECTE458 this mid-level feature extraction will be moved within the framework of logical vision. This will be implemented within Prolog with only low-level feature analysis being done by other programs like Matlab or OpenCV. This allows for the extraction of mid-level features to be guided by the background knowledge provide.



(a) First Extracted Plane - Front **(b)** Second Extracted Plane - Side

Figure 3.5: Extracted Planes Shown in Same Orientation as Figure 3.4

This includes descriptions learnt from previous examples. This will continue the work in [6] and [7] and will use the methods of line extraction, polygon formation and curve and ellipse fitting as detailed in these papers. To the best of our knowledge the analysis of plane data within a Prolog context has never been published. This will also be explored with the goal to integrate it with Logical Vision.

3.0.3 Meta-Interpretive Learning

Metagol Due to the timeframe of ECTE451 and the exploratory nature of this stageno framework was developed to automatically pass extracted mid-level features from MATLAB to Prolog. Features have thus been manually symbolised and passed to a MIL system for learning. The MIL system use is Metagol. The script found in Appendix D shows an implementation of Metagol used to learn from one positive cube examples and no negative examples. Given background knowledge B and a set of symbolic examples E, a MIL system like Metagol aims to learn a hypothesis H such that $B, H \models E$.

Noise tolerant Metagol Due to the noisy nature of real-world images a noise tolerant version of Metagol must be implemented. This is based on the version presented in (Noisy images). However, some changes must be developed to include background knowledge regarding planes. Mid-level extraction of noisy images will often result in noisy logic symbols. As described in [7] this may cause the depth-first search in standard Metagol to fail or return hypothesis that only cover a single example. Noise tolerant Metagol has been altered to accept imperfect hypotheses. A noise tolerant version of Metagol acts as a wrapper around Metagol as described about. It allows for the acceptance of false positives and generates possible hypothesis, which are then used on the remaining training data. The most accurate hypothesis is kept as the solution.

Chapter 4

Experimental design

4.0.1 Materials

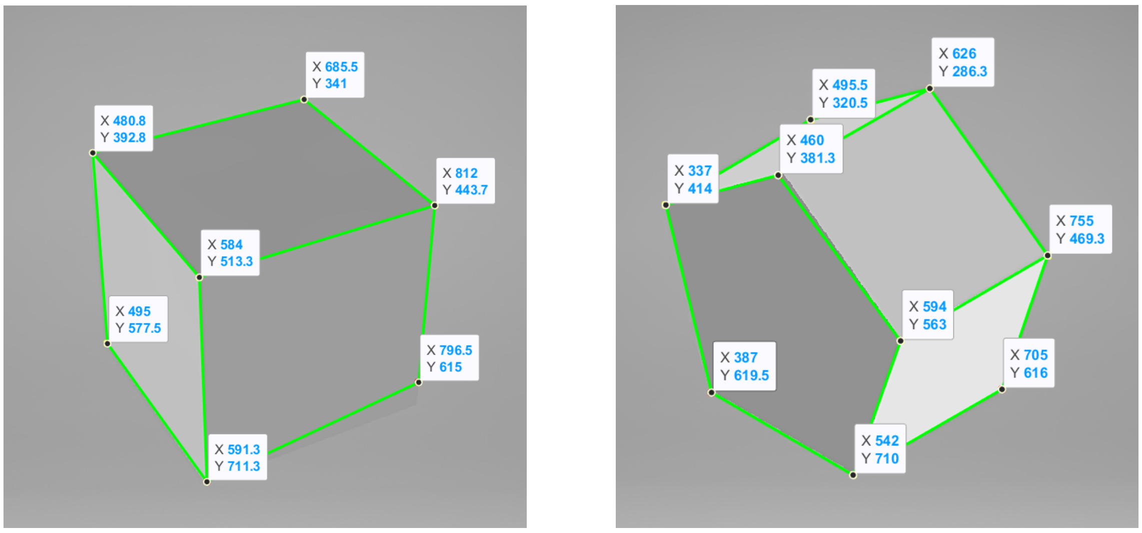
Following similar methods to those used in [6] and [7], 1 labelled RGB-D datasets of simple real-world objects captured with the Intel RealSense depth camera is compiled. It contains images of cubes, square pyramids, triangular prisms and pentagonal prisms. These will be used in 4 learning tasks, one for each class. Each class consists of multiple pictures of each object taken from different angles. The dataset complains 200 images, 50 of each shape. For testing, each class from the dataset will be partitioned, at random, into five-folds, four will be used for training and one for testing.

4.0.2 Methods

The aim is to compare the use of RGB, depth and RGB + depth data as inputs to the proposed meta-interpretive learning system for object recognition. For each learning task 100 images are used for training. This includes 40 positive examples of the class to be learnt and 60 negative examples (20 of each remaining class). Each image is processed to generate symbolic representations of both the 2D and depth data. The three MIL systems to be compared are given specific parts of this data. One receives only the 2D RGB representation, another only the depth representation and the last is given both the 2D and depth information. After training the remaining 10 positive images are used for testing. Similarly to [6], the performance of each system will be compared using the predictive accuracy for each class.

4.0.3 Current results and discussion

The results of pre-processing robustness and initial Meta-interpretive learning are presented to demonstrate the feasibility of this as an image classification technique. MATLAB was successfully used to extract mid-level features from rendered 2D images. Figures 4.1a and 4.1b show example images with successfully extracted lines.



(a) Cube Polygon Extraction **(b)** Pentagonal Prism Polygon Extraction

Figure 4.1: Extracted Lines of Various 3D Shapes

Tables 4.1 and 4.2 give the respective polygon data outputted by MATLAB for the cube and the pentagonal prism. From this information each polygon class, and which polygons connect to which can be deduced.

X1	Y1	X2	Y2
480.3	393.3	685.5	341
685.5	341	812	443.7
812	443.7	584	513.3
584	513.3	480.3	393.3
NaN	NaN	NaN	NaN
591.3	711.3	584	513.3
584	513.3	480.3	393.3
480.3	393.3	495	577.5
495	577.5	591.3	711.3
NaN	NaN	NaN	NaN
796.5	615	591.3	711.3
591.3	711.3	584	513.3
584	513.3	812	443.7
812	443.7	796.5	615
NaN	NaN	NaN	NaN

Table 4.1: Extracted Polygon data from Cube Image

X1	Y1	X2	Y2
460	381.3	594	563
594	563	542	710
542	710	387	619.5
387	619.5	337	414
337	414	460	381.3
NaN	NaN	NaN	NaN
495.5	320.5	337	414
337	414	460	381.3
460	381.3	626	286.3
626	286.3	495.5	320.5
NaN	NaN	NaN	NaN
755	469.3	626	286.3
626	286.3	460	381.3
460	381.3	594	563
594	563	755	469.3
NaN	NaN	NaN	NaN
705	616	755	469.3
755	469.3	594	563
594	563	542	710
542	710	705	616
NaN	NaN	NaN	NaN

Table 4.2: Extracted Polygon data from Pentagonal Prism Image

Extraction of features from noisy images had limited success with many unpredicatable results. As the Hough transform extracts any straight lines, background noise greatly effected performance. Further image pre-processing prior to the Hough extraction would be required to improve the system reliability when dealing with noisy images. Figure 4.2 shows an example of attempted line extraction on a noisy image.

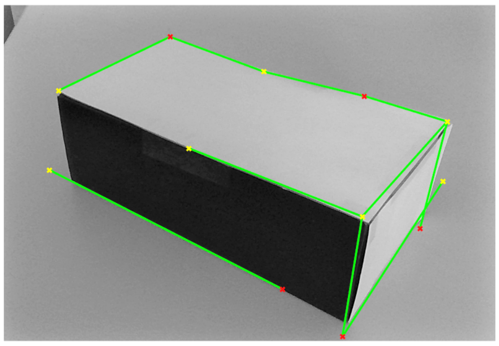


Figure 4.2: Attempted Mid-Level Feature Extraction from Noisy Image

It is also demonstrated how plane information can be extracted from depth images and how this information can be combined with the 2D information found. Figures 3.4 and 3.5 show a complete point cloud image along with two separated planes. Further work is required to extract usable information from these planes. This would include a method to determine the angle between planes as well as method to determine which polygons lie on which plane.

From this extracted data, symbolic representations can be formed to be used in a MIL system. Due to incomplete feature extraction methods, symbolic features were documented manually for use in the MIL process. Example logical representations are seen below;

$square(A).$

$triangle(B).$

$squ\_points(A,\ [467, 417],\ [642, 516],\ [441, 716],\ [483, 600]).$

$tri\_points(B,\ [784, 600],\ [538, 735],\ [664, 425]).$

$no\_sides(A,\ 4).$

$plane(Ap_1).$

$joins(A,\ B).$

$on\_plane(A,\ Ap_1).$

$angle\_plane(Ap_1,\ Bp_2,\ '90').$

$square(A) :-$

$\qquad sides(A,\ 4).$

$triangle(B) :-$

$\qquad sides(A,\ 3).$

No successful results were obtained by using the MIL system Metagol for learning. It is believed this is due to the choice of Metarules. A example Prolog script is shown in Appendix D. This attempts to learn a description of a cube, from one positive example. Further work is needed to produce appropriate Metarules. When looking at related work such as [6], [7] and [11], it is believed that once these are found Metagol will be able to successfully learn valid logical hypothesis from the mid-level representations presented.

Chapter 5

Conclusion

This paper present a framework for comparing the performance of three simple 3d shape image classifiers, all Meta-Interpretive Learning system. The three systems differ by the type of inputs used as training data and test data. These inputs are 2d RGB, depth (point cloud) and a combination of RGB and depth data. The aim of the investigation is to show that the performance of an inductive image classifying system can be improved with the combination of RGB and depth information when compared to systems that only use one of the two. As a complete system has not been completed, this proposed experiment was not completed. The results of pre-processing outputs and robustness and initial Meta-interpretive learning are presented to show the feasibility of the framework.

Valid symbolic representations of 2D mid-level features could be extracted using image processing techniques in MATLAB. These features included the extraction of individual polygons and the information describing them, as well as the interaction between polygons such as joining edges. The feasibility of MATLAB to extract depth data was also explored. It was shown that individual planes can be separated from a point cloud. However, a method for extracting a symbolic representation of planes wasn't found. This will be explored further in ECTE458 to determine a method to find the angle between planes and to determine which plane each polygon lies on.

A MIL system utilising Metagol was also presented. To test its function symbolic mid-level features describing 2d and 3d characteristics of a positive (cube) were given with the aim to induce a hypothesis to describe a cube. However, no successful induction could be performed. This is believed to be due to inadequate Metarules which dictate the allowable forms of the hypothesis. Further work is needed to produce appropriate Metarules.

In summary, based on the information available from previous work and on the work presented here it is believed that a logical image classifying system that uses the combination of both RGB and depth information can be fully developed. It is hypothesised that this system would outperform a similar system that analysed only one of these two data types.

From here, the implementation of MATLAB pre-processing will be furthered. This will include extending the plane processing to extract plane angles and to locate the orientation of polygons on each plane. Further work must be performed to derive appropriate Metarules for use in the Meta-Interpretive Learning process. After that the pre-processing can be moved into the framework of Logical Vision, using Prolog instead of MATLAB and allowing background knowledge to guide feature extraction.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, “Slic superpixels compared to state-of-the-art superpixel methods,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, 05 2012.

[2] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] H. G. Barrow and J. M. Tenenbaum, “Interpreting line drawings as three-dimensional surfaces,” *Artif. Intell.*, vol. 17, no. 1-3, pp. 75–116, Aug. 1981. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(81)90021-7

[4] J. Janai, F. Güney, A. Behl, and A. Geiger, “Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art,” *arXiv e-prints*, Apr. 2017.

[5] B. Assunção Ribeiro, H. Coelho, A. E. Ferreira, and J. Branquinho, “Legal implications of autonomous vehicles: What we know so far and what’s left to work on,” in *Progress in Artificial Intelligence*, P. Moura Oliveira, P. Novais, and L. P. Reis, Eds. Cham: Springer International Publishing, 2019, pp. 287–298.

[6] W. zhou Dai, S. H. Muggleton, and Z. hua Zhou, “Logical vision: Meta-interpretive learning for simple geometrical concepts,” in *Late breaking paper proceedings of the 25th international conference on inductive logic programming.* CEUR, 2015, pp. 1–16.

[7] S. Muggleton, W.-Z. Dai, C. Sammut, A. Tamaddoni-Nezhad, J. Wen, and Z.-H. Zhou, “Meta-interpretive learning from noisy images,” *Machine Learning*, vol. 107, no. 7, pp. 1097–1118, Jul 2018. [Online]. Available: https://doi.org/10.1007/s10994-018-5710-8

[8] J. K. Aggarwal, *Multisensor fusion for computer vision.* Springer Science & Business Media, 2013, vol. 99.

[9] T. Ophoff, K. Van Beeck, and T. Goedemé, “Exploring rgb+depth fusion for real-time object detection,” *Sensors*, vol. 19, p. 866, 02 2019.

[10] A. Eitel, J. T. Springenberg, L. Spinello, M. A. Riedmiller, and W. Burgard, “Multimodal deep learning for robust RGB-D object recognition,” *CoRR*, vol. abs/1507.06821, 2015. [Online]. Available: http://arxiv.org/abs/1507.06821

[11] R. Farid and C. Sammut, “Plane-based object categorisation using relational learning,” *Machine Learning*, vol. 94, no. 1, pp. 3–23, 2014.

[12] D. H. Ballard and C. M. Brown, *Computer Vision*, 1st ed. Prentice Hall Professional Technical Reference, 1982.

[13] S. Papert, “The summer vision project,” *Technical Report Memo AIM-100*, 07 1966.

[14] T. F. Cootes and C. J. Taylor, “Statistical models of appearance for medical image analysis and computer vision,” in *Medical Imaging 2001: Image Processing*, M. Sonka and K. M. Hanson, Eds., vol. 4322, International Society for Optics and Photonics. SPIE, 2001, pp. 236 – 248. [Online]. Available: https://doi.org/10.1117/12.431093

[15] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive Deep Driving: Model Predictive Control with a CNN Cost Model," *arXiv e-prints*, Jul. 2017.

[16] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] T. Brosnan and D.-W. Sun, "Improving quality inspection of food products by computer vision—-a review," *Journal of food engineering*, vol. 61, no. 1, pp. 3–16, 2004.

[18] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015. [Online]. Available: https://doi.org/10.1007/s10462-012-9356-9

[19] F. Doshi-Velez, M. Kortz, R. Budish, C. Bavitz, S. Gershman, D. O'Brien, S. Schieber, J. Waldo, D. Weinberger, and A. Wood, "Accountability of AI Under the Law: The Role of Explanation," *arXiv e-prints*, Nov. 2017.

[20] S. Schocken and G. Ariav, "Neural networks for decision support: Problems and opportunities," *Decis. Support Syst.*, vol. 11, no. 5, pp. 393–414, Jun. 1994. [Online]. Available: http://dx.doi.org/10.1016/0167-9236(94)90015-9

[21] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[22] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, Oct 2019.

[23] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv e-prints*, Dec. 2013.

[24] Z. Zhang and V. Saligrama, "Zero-shot learning via semantic similarity embedding," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[25] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1410–1418. [Online]. Available: http://papers.nips.cc/paper/3650-zero-shot-learning-with-semantic-output-codes.pdf

[26] S. Muggleton, "Inductive logic programming," *New Generation Computing*, vol. 8, no. 4, pp. 295–318, Feb 1991. [Online]. Available: https://doi.org/10.1007/BF03037089

[27] B. Kijsirkul, M. Numao, and M. Shimura, "Efficient learning of logic programs with non-determinate, non-discriminating literals." *Proc. the 8th Int. Workshop on Machine Learning*, pp. 417–421, 1991. [Online]. Available: https://ci.nii.ac.jp/naid/10006400221/en/

[28] S. Muggleton, R. D. King, and M. J. Stenberg, "Protein secondary structure prediction using logic-based machine learning," *Protein Engineering, Design and Selection*, vol. 5, no. 7, pp. 647–657, 10 1992. [Online]. Available: https://doi.org/10.1093/protein/5.7.647

[29] A. G. Cohn, D. C. Hogg, B. Bennett, V. Devin, A. Galata, D. R. Magee, C. Needham, and

P. Santos, *Cognitive Vision: Integrating Symbolic Qualitative Representations with Computer Vision.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 221–246. [Online]. Available: https://doi.org/10.1007/11414353_14

[30] A. Amin, C. Sammut, and K. Sum, "Learning to recognize hand-printed chinese characters using inductive logic programming," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 10, no. 07, pp. 829–847, 1996.

[31] S. H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad, "Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited," *Machine Learning*, vol. 100, no. 1, pp. 49–73, 2015.

[32] A. Cropper and S. H. Muggleton, "Learning higher-order logic programs through abstraction and invention." in *IJCAI*, 2016, pp. 1418–1424.

[33] ——, "Metagol system," https://github.com/metagol/metagol, 2016. [Online]. Available: https://github.com/metagol/metagol

[34] S. H. Muggleton, D. Lin, J. Chen, and A. Tamaddoni-Nezhad, "Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement," in *International Conference on Inductive Logic Programming.* Springer, 2013, pp. 1–17.

[35] S. H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad, "Meta-interpretive learning: application to grammatical inference," *Machine learning*, vol. 94, no. 1, pp. 25–49, 2014.

[36] A. Cropper and S. H. Muggleton, "Logical minimisation of meta-rules within meta-interpretive learning," in *Inductive Logic Programming.* Springer, 2015, pp. 62–75.

[37] T. Ophoff, K. Van Beeck, and T. Goedemé, "Exploring rgb+ depth fusion for real-time object detection," *Sensors*, vol. 19, no. 4, p. 866, 2019.

[38] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 567–576.

[39] M. Schwarz, H. Schulz, and S. Behnke, "Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features," in *2015 IEEE international conference on robotics and automation (ICRA).* IEEE, 2015, pp. 1329–1335.

[40] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust rgb-d object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2015, pp. 681–687.

Appendix A

Project plan

University of Wollongong

SCHOOL OF ELECTRICAL, COMPUTER AND TELECOMMUNICATIONS ENGINEERING
ECTE451 PROJECT PROPOSAL FORM

1. Candidate Details	
Name: Jared Bellingham	**Student No: 5104701**
Supervisor: A/Prof David Stirling	
Title of Project: Meta-Interpretive learning from 3D shapes	

Brief Overview:

Computer vision is the process of extracting or inferring information from an image. This could be seen as a simple task as humans can do this from an early age, however, computer vision is still in its infancy and currently systems still have many limitations.
Image classification falls under computer vision. It is the process of putting an object found in a digital image into a class.

Current image classification systems are based on statistic machine learning. These systems have some limitations and as such, the development of alternative systems is worthwhile.
Logical vision (LV) is a computer vision system that has been used for image classification and object detection. This project aims to expand the use of logical vision as an image classification tool from simple 2d shapes to 3d object both simple noise-free rendered images and noisy real-world images.

An implementation of LV will be developed that is able to classify 3d objects into their shape given limited training images. The use of 2D, 3D and the combination of both will be explored. It is hypothesis that systems which use a combination of both data types will outperform a system using just one.

2. Project Description: (Expand to one page maximum)

The most common image classification systems are based on statistic machine learning (SML), with great success coming from learners using deep neural networks (DNN). Training SML systems requires large datasets which sometimes is expensive or not practical. The systems are also treated as a ‘black-box’ classifier as they cannot easily show reasoning for their decisions. SML systems are also very sensitive to changes in low-level features and it is difficult to include background knowledge.

Due to these limitations SML image classification differs dramatically from human image classification. This alone presents many issues as we begin to rely more heavily on computer vision in tasks like vehicle navigation. If a system is unable to provide reasoning for its decisions, especially in the event of unplanned behaviour, it becomes difficult to ensure the design has been changed to prevent that specific behaviour from happening again.

Logical vision (LV) (Dai WZ et. Al. 2015) aims to address some of the issues above. LV is an image classifier which is able to learn visual concepts symbolically. It can also be trained with limited images, down to a single example. In published work LV has been shown to be able to classify simple 2d polygons from a small dataset of images. This project aims to extend this to the classification of simple 3d shapes. It will be compared to other current state-of-the-art SML systems including convolutional neural networks.
The aim of the project is to show that, for small datasets, a LV approach is valid for 3d object classification and it is more accurate when using both 2D and 3D data compared to just one type.

3. Project Plan: (Two pages maximum)

An implementation of LV will be developed. The system is implemented in Prolog and will be run in SWI-Prolog. The system is split into two stages, feature extraction and hypothesis induction.

The first stage is feature extraction. This will be split into polygon extraction and then polygon combination to form objects. Polygon extraction is a process of low-level extraction guided by mid-level conjectures. The program first discovers lines, these lines are then connected to allow shape extraction. From this, joined polygons can be extracted as objects. The extraction of simple polygons is outlined by Dai WZ et. Al. 2015 (ref 1). A similar implementation will be extended to allow for 3d object extraction.

After mid-level feature extraction Meta-interpretive learning (MIL) (Muggleton et. al. 2015) namely $\text{Metagol}_{\text{LogicVision}}$ (Dai WZ et. Al. 2015) will be used to abduce hypothesises to explain the presented object.
The implementation of $\text{Metagol}_{\text{LogicVision}}$ from Dai WZ et. Al. 2015 will need to be modified to include some specific background knowledge needed for 3d object extraction.

This will be split into two parts. Firstly, the classification of individual sides and their orientation. Then the analysis of how shadow gradients on each side relate to the sides orientation to other sides and also the light source.

Implementation will be initially developed for computer generated 2d renders of 3d objects. This will include various angles of specific simple 3d objects including cubes, spheres, cones, and cylinders. These images will include an out of frame light source and thus each shape will have surfaces with varied light intensities.
These images will form databases for training and testing.

The LV system developed will be compared to current SML systems. These systems will likely come from a popular toolbox like VLfeat. This follows a similar methodology to Muggleton S. et. Al. 2018 (ref 2) and Dai WZ et. Al. 2015 (ref 1).

If implementation on rendered images is successful, the program will be extended for use with noisy images. Likely, this will be implemented in ECTE458. This will have to include a more robust object extraction stage capable of handling classification noise and attribute noise. The MIL implementation will also need to be changed. The noise tolerant version of Metagol, $\text{Metagol}_{\text{NT}}$ (Muggleton S. et. Al. 2018) will be modified to suit 3d object classification.

This program will be trained and tested using real-world images of simple 3d objects. Objects may have various illumination, be partly hidden, in previously unencountered orientations or be images with reduced quality. A robust system would be negligibly affected by these noise factors.

Again, this system would be compared to noise robust current SML systems focusing on the accuracy of image classification.

Timeframe (Milestone)

ECTE451

- Week 5 - Have complete rendered images dataset
- Week 6 – Have a polygon extraction program developed that is able to locate and extract individual polygons from a 3d shape. It must also detail edges of joining faces.
- Week 7 – Have Metagol implantations that can, from a logical representation of joined polygons, abduce hypothesises that classify the shape
- Week 9 – Complete implementation of LV
- Week 10 – Completed testing of LVs system of test dataset

By then end of ECTE451 a complete working LV implementation will be developed that is able to classify rendered images of 3d shapes given limited training images. The use of 2D, 3D and the combination of both will be explored. It is hypothesis that systems which use a combination of both data types will outperform a system using just one.

4. Resources Required: (Expand to a half a page maximum)

This statement should identify any materials (software/hardware) or access to infrastructure required to complete the project.

Limited resources will be required.

If the project is able to be successfully implemented on CG objects a valid webcam or camera will be needed.

If required access to a systems incorporating multicore CPU and/or GPU hardware may be needed for rendering and training.

Student Signature
Declaration by the student: I have understood the feedback provided to me by the supervisor.

	Signature	**Date**
Student Name:		

Note: the typical over all page count should not exceed 15 pages

A marked assessment rubric will be appended once completed

Appendix B

Logbook Summary Signature Sheets

A Logbook Summary Signature Sheet

SCHOOL OF ELECTRICAL, COMPUTER AND TELECOMMUNICATIONS ENGINEERING
ECTE451 : Logbook Summary Signature Sheet

Week No.	Date	Comments, if applicable	Student's Signature	Supervisor's Signature
1	01/08/19		[illegible]	[illegible]
2	08/08/19		[illegible]	[illegible]
3	15/08/19		[illegible]	[illegible]
4	23/08/19	Separate meeting on Friday 23rd	[illegible]	[illegible]
5	29/08/19		[illegible]	[illegible]
6	5/9/19		[illegible]	[illegible]
7	13/9/19		[illegible]	[illegible]
8	19/9/19	email direction	[illegible]	[illegible]
9	26/9/19		[illegible]	[illegible]
/	3/10/19	Mid session break - No meeting	[illegible]	[illegible]
10	10/10/19		[illegible]	[illegible]

A Logbook Summary Signature Sheet

SCHOOL OF ELECTRICAL, COMPUTER AND TELECOMMUNICATIONS ENGINEERING
ECTE451 : Logbook Summary Signature Sheet

Week No.	Date	Comments, if applicable	Student's Signature	Supervisor's Signature
11	17/10/19		[illegible]	[illegible]

Appendix C

MATLAB Scripts and Functions

C.0.1 Main - Polygon Analysis

```
img = 'block2.jpg';
% Perform Hough Transform and return line extracted
[lines, image] = houghTransform(img, 2, 0.5, 13, 50, 70);
% Join close points to form exact vertices
linesJoined = joinVertices(lines, 50);
% Remove invalid lines which contain free ends (not fixed in vertex)
linesRemJoined = removeRedundant(linesJoined);
polygons = [];
% Extract ununssigned vertex (point) from object
[lineIndex, startPoint] = unassignedPoint(linesRemJoined, polygons);
% Continues until all polygons have been found (no unassigned vertices)
while startPoint ~= 0
    % Randomly generate line path as possible polygon
    polygon = generatePolygon(linesRemJoined, lineIndex);
    % Validate polygon and check it is in simplest form
    validity = validatePolygon(polygon, linesRemJoined);
    % Store polygon data
    if validity == 1
        n = nan(1,4);
        polygons = [polygons ; polygon ; n];
    end
    % Generate next free vertex (point) if it exists
    [lineIndex, startPoint] = unassignedPoint(linesRemJoined, polygons);
    if startPoint == 2
        linesRemJoined = swapEndPoints(linesRemJoined, lineIndex);
    end
    % Continues until 'unassignedPoint' returns '0' as start point
end
```

C.0.2 Hough Transform

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% This function performs haugh transform initial pre-processing then
% generates haugh transform and identifies lines
% Lines are returned in structure array

function [lines, image] = houghTransform(img, rad, sharp, N, FillGap,
   MinLength)
   % Read in image and convert to grayscale
   RGB = imread(img);
   A  = rgb2gray(RGB);
   image = imsharpen(A,'Radius',rad,'Amount',sharp);
   BW = edge(image,'canny');
   % Calculate hough transform
   [H,theta,rho] = hough(BW);
   P = houghpeaks(H,N,'threshold',ceil(0.05*max(H(:))));
   x = theta(P(:,2));
   y = rho(P(:,1));
   % Generate lines
   lines =
      houghlines(BW,theta,rho,P,'FillGap',FillGap,'MinLength',MinLength);
end
```

C.0.3 Join Vertices

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% This function takes standard lines structure and joins lines whos ends
% are close together to form vertices.

function linesJoined = joinVertices(lines, range)
   % create easy to work with array
   a = vertcat(lines([1:length(lines)]).point1);
   b = vertcat(lines([1:length(lines)]).point2);
   points = [a;b];
   for l = 1:length(points)
```

```
        % find points that are withing Euclidean distance 'range' from
        Idx = rangesearch(points,points(1,:), range);
        d = Idx{1};
        num = length(d);
        % Calculate centre between two close points
        X = 0; Y = 0;
        for c = 1:num
        X = points(d(c),1) + X;
        Y = points(d(c),2) + Y;
        end
        Xav = X/num;
        Yav = Y/num;
        % modify points
        for c = 1:num
        points(d(c),:) = [Xav, Yav]
        end
    end
    % Edit lines structure to reflect new line end points
    half = (length(points)/2);
    p1 = points(1:half,:)
    p2 = points((half+1):end,:)
    linesJoined = lines;
    for l = 1:length(linesJoined)
        linesJoined(l).point1 = p1(l,:)
        p1(l,:)
        linesJoined(l).point1
        linesJoined(l).point2 = p2(l,:)
        p2(l,:)
        linesJoined(l).point2
    end
end
```

C.0.4 Remove Redundant Lines

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
```

```
% This function remove redundant lines from shape
% These are duplicate lines, or lines that have free ends (therefore do
    not
% create a valid polygon

function linesJoined = removeRedundant(linesJoined)
    % If two lines are equal remove one
    j = 1; k = 1;
    while j <= length(linesJoined)
        k = 1;
        while k <= length(linesJoined)
            if j ~= k
                a = isequal(linesJoined(j).point1, linesJoined(k).point1)
                b = isequal(linesJoined(j).point2, linesJoined(k).point2)
                c = isequal(linesJoined(j).point1, linesJoined(k).point2)
                d = isequal(linesJoined(j).point2, linesJoined(k).point1)
                if (a == 1 && b == 1) || (c == 1 && d == 1)
                    linesJoined(k) = [];
                    j = 0; k = 1; break
                end
            end
            k = k + 1;
        end
        j = j + 1;
    end
    % Changes formate to simple array
    a = vertcat(linesJoined([1:length(linesJoined)]).point1);
    b = vertcat(linesJoined([1:length(linesJoined)]).point2);
    points = [a;b];
    len = 1;
    % Sets all invalid points to origin, Implementation assumes no image
    % will be cropped to the corner of a shape.
    while len <= length(points)
        x = sum(points(:,1)==points(len,1));
        y = sum(points(:,2)==points(len,2));
        % Check if any points only occur once (lines with free end)
        if (x == 1 || y == 1) && (points(len,1) ~= 0 && points(len,2) ~= 0)
            points(len,:) = [0,0];
            % remove also other endpoint of free line
            h = length(points)/2;
            if len <= h
                points(len+h,:) = [0,0];
            else
                points(len-h,:) = [0,0];
            end
            len = 0;
        end
        len = len + 1
    end
    % transfer infomation to line structure
    half = (length(points)/2);
    p1 = points(1:half,:)
    p2 = points((half+1):end,:)
    for l = 1:length(linesJoined)
        linesJoined(l).point1 = p1(l,:);
        linesJoined(l).point2 = p2(l,:);
    end
    % Remove all invalid points
    len = 1;
    while len <= length(linesJoined)
        if (linesJoined(len).point1(1,1) == 0 && ...
                linesJoined(len).point1(1,2) == 0) || ...
                (linesJoined(len).point2(1,1) == 0 && ...
                linesJoined(len).point2(1,2) == 0)
            linesJoined(len) = [];
            len = 0;
        end
        len = len + 1;
    end
end
```

C.0.5 Extract Unassigned Point

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% This function returns a point1 oe point2 in lines which has not yet been
% place in a polyggon.
% lineIndex shows what line to pick, pt gives the end, point1 or point2
% Inputs lines (all lines) and polyg (already assigned lines)

function [lineIndex, pt] = unassignedPoint(lines, polyg)
    % Looks through all possible lines
    for countLines = 1:length(lines)
        diffent = 0;
        for countpolyg = 1:length(polyg)
            point1 = lines(countLines).point1;
            check1 = [polyg(countpolyg, 1), polyg(countpolyg, 2)];
            check2 = [polyg(countpolyg, 3), polyg(countpolyg, 4)];
            % Checks to see if point1 exists in a polyggon already
            a = isequal(point1, check1);
            b = isequal(point1, check2);
            % Counting how many times point is not assigned
            if a == 0 && b == 0
                diffent = diffent +1;
            end
        end
        % If point1 wasnt equal to any of the previous polyggon points
        if diffent == length(polygg)
            lineIndex = countLines ;  % returns index to unassigned point
            pt = 1;
            return
        end
        % Checks for point2's if no point1's were unassigned
        diffent = 0;
        for countpolyg = 1:length(polygg)
            point2 = lines(countLines).point2;
            check1 = [polygg(countpolyg, 1), polygg(countpolyg, 2)];
            check2 = [polygg(countpolyg, 3), polygg(countpolyg, 4)];
            a = isequal(point2, check1);
            b = isequal(point2, check2);
            if a == 0 && b == 0;
                diffent = diffent +1;
            end
        end
        if diffent == length(polyg)
            lineIndex = countLines;  % returns index to unassigned point
            pt = 2;
            return
        end
    end
    % Returns zeros to show no unassigned points
    lineIndex = 0;
    pt = 0;
end
```

C.0.6 Generate Random Polygon

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% Generate random polygon from lines given starting point.
% polygon form is nx4 where n is the number of lines.
% For each row index 1 and 2 and index 3 and 4 represent point 1 and
    point 2 respectively.
% linesRemJoined is a structure with 1x2 matrixs, point1 and point2
% pt is the index to the starting point as generated by unassignedPoint

function [polygon] = generatePolygon(linesRemJoined, pt)
    % Initiations
    p2 = 0;
    start = pt;
```

```
polygon = [];
% Repeates until finishing point of added line is back at the starting
% point
while p2 ~= linesRemJoined(start).point1
    p1 = linesRemJoined(pt).point1;
    p2 = linesRemJoined(pt).point2;
    random = 360;
    % Pick which route to take, wont pick current line
        for comp = 1:length(linesRemJoined)
        if comp ~= pt
            % Only picks next line that joins to current line
            if linesRemJoined(comp).point1 == p2
                % Unconventionsal way to choose random line
                numb = randi([1 359],1,1);
                if (numb < random)
                    random = numb;
                    next = comp;
                end
            % If chosen line joined from point 2
            elseif linesRemJoined(comp).point2 == p2
                store = linesRemJoined(comp).point2;
                linesRemJoined(comp).point2 =
                    linesRemJoined(comp).point1;
                linesRemJoined(comp).point1 = store;
                numb = randi([1 359],1,1);
                if numb < random
                    random = numb;
                    next = comp;
                end
            end
        end
    end
    % Move to next point and save line to polygon
    pt = next;
```

```
        polygon = [polygon ; p1 , p2];
    end
end
```

C.0.7 Validate Polygon

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% This function is used to validate that generated polygon is of most
% simple form given lines available.

function validity = validatePolygon(polygon, lines)
    % Initiate, returns true if no
    validity = 1;
    % Extract polygon expression
    xv = [polygon(:,1)' , polygon(end,3)]
    yv = [polygon(:,2)' , polygon(end,4)]
    % Test whether shape is a well-defined polygon, matlab function
    p = polyshape(xv,yv,'Simplify',false)
    validity = issimplified(p)
    % for all possible lines in 3D shape
    for lineCount = 1:length(lines)
        % find centre of line
        P1 = lines(lineCount).point1;
        P2 = lines(lineCount).point2;
        % find centre of line
        centre = (P1(:) + P2(:)).'/2;
        % Does point lie within polygon
        % As rounding errors occur leading to false negatives, we
        % test points, 1 unit from centre in all 4 direction
        in1 = inpolygon(centre(1)+1,centre(2),xv,yv);
        in2 = inpolygon(centre(1),centre(2)+1,xv,yv);
        in3 = inpolygon(centre(1)-1,centre(2),xv,yv);
        in4 = inpolygon(centre(1),centre(2)-1,xv,yv);
        in5 = inpolygon(centre(1),centre(2),xv,yv);
        % All points must fall inside the polygon to invalidate
        if in1 == 1 && in2 == 1 && in3 == 1 && in4 == 1 && in5 == 1
            % if yes set marker on
            validity = 0;
            return
        end
    end
end
```

C.0.8 Swap Line End Points

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% This function swaps the end points of a line

function lines = swapEndPoints(lines, index)
    store = lines(index).point2;
    lines(index).point2 = lines(index).point1;
    lines(index).point1 = store;
end
```

C.0.9 Extract Plane Data

```
%%% CODE CREATED BY J. BELLINGHAM, UNIVERSITY OF WOLLONGONG %%%
% Script to perform the extraction of planes from point cloud data

loc = load('cereal_box_1_2_32_loc.txt');
depth = imread('cereal_box_1_2_32_depthcrop.png');
[pcloud, distance] = depthToCloud(depth); % Fuction from: A Large-Scale
% Hierarchical Multi-View RGB-D Object Dataset - Kevin Lai et. al.
ptCloudN = pointCloud(pcloud)
ptCloud = pcdenoise(ptCloudN)
% Set Parameters for plane extraction
maxDistance = 0.01;
referenceVector = [1,0,1];
maxAngularDistance = 15;
% Extract first plane
[model1,inlierIndices,outlierIndices] = pcfitplane(ptCloud,...
    maxDistance,referenceVector,maxAngularDistance);
plane1 = select(ptCloud,inlierIndices);
remainPtCloud = select(ptCloud,outlierIndices);
roi = [-inf,inf;-inf,inf;-inf,inf];
sampleIndices = findPointsInROI(remainPtCloud,roi);
% Extract second plane
[model2,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,...
            maxDistance,'SampleIndices',sampleIndices);
plane2 = select(remainPtCloud,inlierIndices);
remainPtCloud = select(remainPtCloud,outlierIndices);
```

Appendix D

Prolog Scripts

```
:- use_module('../metagol').
%% metagol settings
body_pred(square/1).
body_pred(plane/1).
body_pred(joins/2).
body_pred(on_plane/2).
body_pred(sides/2).
body_pred(angle_plane/3).
%% background knowledge
square(A) :-
    sides(A, 4).
square(c_1).        square(c_2).         square(c_3).
plane(cp_1).        plane(cp_2).         plane(cp_3).
joins(c_1,c_2).     joins(c_1,c_3).      joins(c_2,c_3).
on_plane(c_1, cp_1). on_plane(c_2, cp_2). on_plane(c_3, cp_3).
angle_plane(cp_1, cp_2, '90').
angle_plane(cp_1, cp_3, '90').
angle_plane(cp_2, cp_3, '90').
%% metarules
metarule([P,Q], [P,A,B], [[Q,A,B]]). % identity
metarule([P,Q], [P,A,B], [[Q,B,A]]). % inverse
metarule([P,Q,R], [P,A,B], [[Q,A],[R,A,B]]). % precon
metarule([P,Q,R], [P,A,B], [[Q,A,B],[R,B]]). % postcon
metarule([P,Q,R], [P,A,B], [[Q,A,C],[R,C,B]]). % chain
metarule([P,Q], [P,A,B], [[Q,A,C],[P,C,B]]). % recursion
%% learning task
  Pos = [
    cube(c_1, c_2, c_3)
  ],
  learn(Pos,[]).
```