Computer Vision Pipeline for Robotic Arm Control

By

Aniekan Umanah (U1873416)

A dissertation presented to the School of Computing and Engineering

University of Huddersfield, UK.

In partial fulfilment of the requirements for the Final Year Project Module (NHP2400)

Academic Year: 2020/2021

By submitting this work I confirm that it is entirely by own and adheres to all the academic integrity guidelines set out by the University.

Abstract

Pick and place tasks are frequently performed by robotic arms. However, the position of the workpieces may change during the process, necessitating the robot locating such an item before planning a trajectory. This paper focuses on the use of computer vision techniques to solve robotic arm orientation of workpiece by developing an image processing pipeline that will enable detection of a set of items and estimation of their positions using a camera coordinating system and a powerful computer vision library. Building on recent trends in the use of neural networks, machine learning, and deep learning, as well as image processing successes, an application was created in MATLAB using a pretrained network, with the goal of transferring learning to a real robot. To train a robotic arm, this method employs 3D simulation.

Acknowledgements

This work was supported by my supervisors, Dr Naeem Main and Nemwel Ariaga (Researcher)

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Description
CNN/CovNet(s)	Convolutional Neural Network(s)
YOLO	You only look once
SSD	Single shot detector
ACF	Aggregate channel feature
VGG	Visual geometry group
R-CNN	Regions with convolutional neural network
ResNet	Residual Neural Network
IoU	Intersection over union
ROI	Region of interest
NMS	Nonmaximum suppression
COP	Centre of projection
FCN	Fully convolutional network
RMSE	Root mean square error
DAG	Direct acyclic graph
fppi	False positives per image
DCNN	Deep convolutional neural network
DOF	Degrees of freedom

1 Introduction

Robots are being increasingly used in different environments which require object manipulation and task execution [1,12], including smart manufacturing, household settings, warehouse logistics, etc [12]. These robots perform complex tasks autonomously and require a reliable perception of the environment and real-time visual feedback. This information is gathered with the aid of various cameras, sensors, and sensing modalities. Under certain situations, finer recognition might be required, such as determining whether a specific object is present in the scene such as our case. An even more precise scene perception, including object detection and pixel-wise semantic segmentation, is required for more intelligent interaction, such as grasping [1,2,8,12] and manipulating real-world objects.

Manipulators (robotic arms) have been in industrial use for decades [7], newer generation of autonomous robots built to sense and respond to their environment, plan their actions and also work alongside humans and perform human activities. The field of robotic arm control has traditionally been approached by hand-crafting modular [7] designs employing computer vision techniques with image processing [8]. The success of deep learning in recent years has spurred the development of many domains that enable end-to-end learning from examples without the need for handcrafted features or complicated priors. Despite these advancements, the technology has yet to be fully integrated into real-time robotic systems, most models only detect stationary workpieces and are not used to control the robot itself [7,12,22]. In this paper, the study of integrating computer vision for robotic arm manipulation and task execution is demonstrated. Firstly, the application of computer vision and deep learning to the task of object perception for pick and place action on a production line is demonstrated and developed into a pipeline. The problem setting; although the environment is structured and uncluttered (belt, workpiece), the positions of the workpiece may change during the process, requiring the robot to pinpoint/estimate the position of the items and plan a trajectory to execute the pick and place action (bin dropping). The pipeline is adaptable to manipulators, which need to detect workpieces, such as in this design.

In contrast to their remarkable success, [8] deep learning methods are difficult and have some limitations. Because of the high dimensionality of the state space, it is very often impractical to generate sufficient training data with real-world experiments. An alternative method is using simulation to train the controller – which is employed in this study. It, therefore, means they have a high degree of complexity and require large amounts of labelled training data to learn the features needed to solve the task and could take weeks to train, making it virtually impossible to train a network model from scratch. To reduce the computation time, we employ a transfer learning approach [2,8,12,13,19,23], in which we take a CovNet pre-trained on a much larger dataset, normally ImageNet, and use it as a feature extractor in the network's initial layers. The first layers extract general, low-level features that are applicable across images — such as edges, corners, patterns, and gradients — and then fine-tune the latter layers to identify specific features within an image for the high-level task of detection and segmentation. With only a small amount of additional training data, we can harness the power of deep neural networks for robotic applications using this method.

The robot must first see the objects in order to detect them. Stereo vision is used to accomplish this by employing two cameras with parallel image planes – Stereo Vision.

1.1 Computer Vision

Computer vision is widely used in a variety of industries and has been incorporated into a variety of devices. In today's world, we can find common vision algorithms implemented in a variety of contexts, including game consoles (Xbox Kinect, PS camera), security and surveillance cameras, medical imaging, and so on. Computer vision operates in a hierarchical manner, identifying features of objects in a series of applications linked together in a pipeline [9]. Approaches for self-supervision have been developed, as have others that use the discriminant graph model [4]. Computer vision has been critical in the field of robotics, improving control by removing uncertainties; and using passive sensors to allow the robot to observe its surroundings. AI improves the manufacturing process by detecting defects that are not easily visible to the naked eye; this defect detection rate can reach 90 percent [9]. These methods are used in the financial sector to detect counterfeit bills and prevent fraud. Computer vision is also extremely useful in the medical field. With the current COVID-19 pandemic, we see that autonomous robots are being used to conduct dangerous tests, protecting researchers from harmful diseases, and providing vital information for vaccine production. Appendix A contains additional examples of the advantages of computer vision. The majority of computer vision solutions are handcrafted modules that include scene segmentation, object recognition, 3D reconstruction, pose estimation, trajectory planning, and so on. This approach, however, may result in information loss between modules, resulting in error accumulation, and. Moreover, due to the requirement for prior knowledge, it is not flexible for learning a variety of tasks [8].

Computer vision has been made more effective with the introduction of machine learning (ML) and the use of deep neural networks, and [2] this has promoted the research of learning-based robotic grasping. A neural network is made up of a series of interconnected nodes. A node is modelled after a neuron in the human brain. Nodes, like neurons, activate once sufficient stimuli are present (input). This activation propagates throughout the network, resulting in a response to the stimuli (output), see Figure 1(b). The networks are usually represented as DAG. Because of their built-in ability to extrapolate new features from the set of features in the training set, deep learning networks reduce the need for explicit, time-consuming feature engineering techniques. They are widely used for difficult problems that require real-time analysis and scale well to classification tasks that frequently require complex computations [9]. Convolutional Neural Networks are excellent examples of deep learning techniques devoted to image processing. However, the performance of deep learning is more dependent on the quality of the datasets than other conventional machine learning methods. The training data set must be large enough to prevent overfitting, which entails a significant amount of manual labour to gather and annotate images. Furthermore, the dataset must cover all conditions, such as inconsistent lighting, shadow, and occlusion to improve robustness [22].

Thus, investigations into traditional pattern recognition pipelines are valuable and could provide a complementary approach to the supervised deep learning method, especially for applications with limited computational capacity as well as available datasets, such as ours. Specific research objectives were to:

1. Create an object perception model using computer vision and deep neural networks to detect workpieces on a conveyor belt.

2. Using stereo vision, calibrate cameras to obtain intrinsic and extrinsic parameters; use these parameters to rectify the stereo system and estimate depth.

3. Combine steps one and two to create a pipeline for robotic arm control, workpiece position estimation, and task execution.

4. Assess the proposed pipeline's performance at various stages, such as detection, depth estimation, and task execution.

1.2 Related works

Recent works on robotic manipulation have taken advantage of deep learning. Deep learning can be introduced into robot manipulation in several ways. James et al. [8] introduce an intermediate reward policy learning using deep Q-learning to encourage efficient learning. It uses an elipson-greedy approach which decides between a random action or the action corresponding to the highest Q-value produced by the DQN. They use positional information to give the agent an intermediate reward based on the exponentially decaying distance from the gripper to the workpiece with an additional reward when the item is grasped, and then a final reward based on the height of the lift. From the simulation results, it was observed that the robotic arm returned to grasp the workpiece if it dropped during the lifting process. While this task is highly impressive, it required some amount of human involvement when implemented in the real world. [2,4] Many other researchers have attempted to use computer vision techniques to solve robot manipulation problems. The methods applied take in an image or point cloud as input. Schwarz et al. [12] developed a deep learning approach which that allows for robotic manipulation in a cluttered scene. The output from the CNN-based network is a segmented image and object bounding box. They combine the posteriors of the detection and segmentation to produce more accurate pixel-wise object recognition location results. They also develop a depth fusion method that fuses three separate sources of depth improves the overall depth measurement enabling a more accurate grasp pose. In the study of [2], the authors propose a novel occlusion-aware CNN object detection in a cluttered scene which outputs occlusion prediction that convert occluded objects as background. They also apply an automatic switching function using the Kalman filter for multiple vision fusion. This improves the accuracy of the pose estimation. Mavrakis et al. [1] explore methods in which a robot can identify an object using inertial parameters. This suggests that the robot is able to estimate the mass, CoM and inertia tensors and using some synthesis algorithm to minimise the collision force by selecting different grasp points on an object. Currently, these methods face some drawbacks, but with further advancements in machine learning, the methods stated in this study no doubt would make robot manipulation smarter and more efficient. Other computer vision, ML, and DL implementations in robotic such as in the study of Gai et al [22]. They developed a vision system that detects crop plants based for weed control which fuses colour and depth images and performs feature-based extraction for classifying the crop plants and weed. Researchers in [5] developed a machine learning technique to detect shaft defects in rotating machinery. The technique can be used to diagnose faults of other rotating machines such as motors and engines. [4] Proposes a detection method of determining the clusters of points based on a combination of modelled graphics processing with fuzzy logic.

The remainder of the paper is organised as follows. The proposed pipeline's overall framework is introduced (section 2). Investigate the methods for detecting the workpiece (section 3). We describe the proposed depth and position estimation methods in (section 4). The experiment results are evaluated and discussed (section 5). (section 6). The paper is finalised (section 7).

2 Brief

2.1 Problem Formulation

The task at hand was clear. A folder containing the relevant data – Extracted from a 3D simulation software - to be used to experiment was provided. The project suggested the use of Python, but any suitable programming language could be used; the idea was to get the job done. To create an application that could estimate the position of the workpiece for pick and place using computer vision and deep learning and test. The data gathered for these tests would be used control an articulated robotic arm to sort the detected items into their respective bins.

The programming language needed to integrate all the tasks; to be simple to use and program. This was important because when developing complex systems low-level programming makes the task difficult and time consuming.

2.2 Problem Solution

MATLAB was considered to be a suitable programming language and was found to fulfil the necessary requirements. MATLAB has a very friendly user interface and graphical programming environment; it has many apps and toolboxes which were very useful to this project. It was possible to accomplish all the work packages and develop a single application.

2.3 Overview of Work Plan

The idea was for a manipulator to accurately estimate workpieces' position on a conveyor belt. The overall pipeline is illustrated in Figure 1. There are mainly four stages which include Object detection, semantic segmentation, camera calibration, depth Estimation, and position estimation for pick and place. When the images from the left and right camera are seen (see Figure 1(d)), the items are detected through object detection and semantic segmentation to produce pixel-wise recognition(see Figure 1(b)). An algorithm was developed to calculate the disparity using feature matching (see Figure 1(e))to get an average disparity to calculate the depth (z) value as well as the x and y positions. The features within the bounding box area from the detection are extracted from the left and right corresponding images and matched.

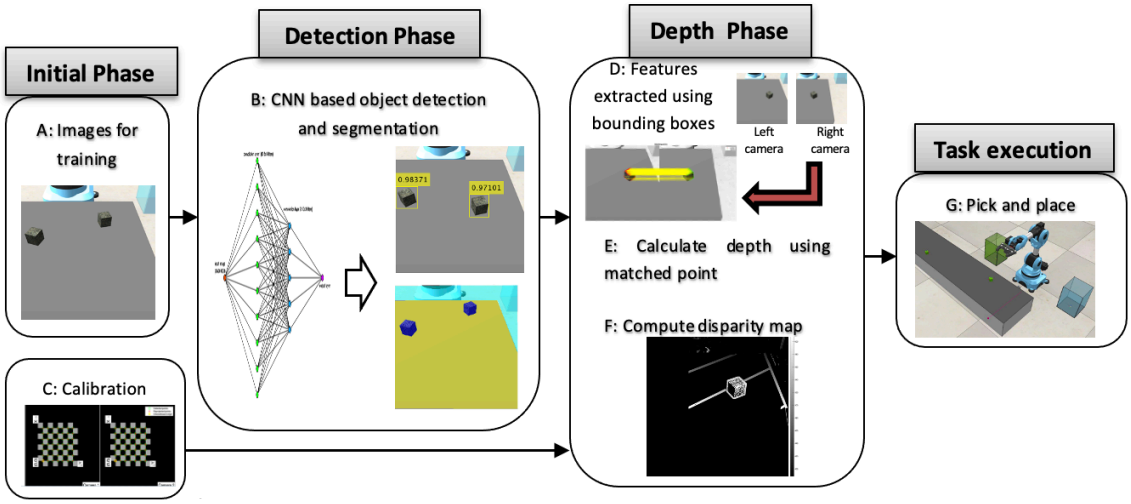


Figure 1 An overview of the proposed pipeline

3 Detection

Similar to [12], two independent methods are explored which can allow the robot to identify items around its vicinity. The first method is object detection; uses bounding boxes to highlight instances of objects in the image from each class for each detection. The second one is semantic segmentation; obtains pixel-level segmentation – category labelling - of the image. Table 1 summarizes each method.

The two methods use transfer learning which leverages on CNNs pre-trained networks (usually trained by an expert). There are 99 images available for training which is not a sufficient dataset size to train a CNN from scratch. The CovNets have been trained on a large classification dataset and are merely modify the final layers to work in the specific domain.

Table 1 Overview of detection methods

Method	Description	Pros/Cons	Estimated Parameters
Object Detection	Classifies the patches of an image into different object classes.	+ Easier to implement + Works well with smaller objects – Coarse estimate	Bounding box, object confidence score
Semantic segmentation	Classifies the pixels in an image into their corresponding classes.	+ Precise localization – Preparing training dataset is time-consuming – Higher chance of misclassification	Segmented image

3.1 Object Detection

Recent advancements in deep learning-based computer vision models have made the development of object detection applications easier than ever. Besides significant performance improvements, they also have the ability to leverage transfer learning to lessen the need for large datasets [14]. This has seen a shift toward end-to-end solutions [2,8,14]. Several algorithms exist and can be used for object detection such as Faster R-CNN, YOLO, SSD, ACF, VGG, etc.

YOLO is a single stage object detector and considered to be one of the fastest real-time object detection algorithms as compared to two stage deep learning detectors [9, 15]. To this end, the YOLOv2 algorithm [13] was adopted. Figure 2 shows the general network architecture. The network mainly consists of two subnetworks – A fixed feature extraction network followed by a detection network. The feature extraction network is a ResNet-50 pre-trained on the ImageNet dataset. The detector subnetwork is a smaller CNN with 9 layers specific to YOLOv2. The layers after 'activation_40_relu' in the ResNet are replaced with the YOLOv2 layers. This feature layer outputs the feature maps that are downsampled by a factor of 16. This gives a good trade-off between spatial resolutions and the strength of extracted features. The input of the network is from the images taken by the stereo cameras. The output is the bounding box of the workpiece and the objectness score. The YOLO v2 uses anchor boxes to detect the classes of the objects in the image by predicting the IoU, anchor box offsets, and class probability for each anchor box [13]. The analysed network had a total of 150 layers.

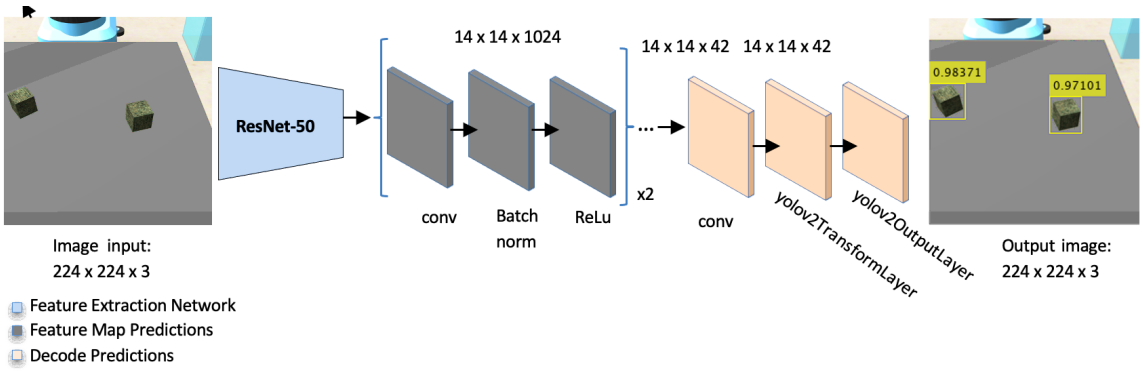


Figure 2 Network Architecture of the yolov2ObjectDetector

3.1.1 Estimating Anchor Boxes

The network uses Predefined bounding boxes of a certain height and width located in the yolov2TransformLayer Figure 2. They are used to capture the scale and aspect ratio of the specific object class(one class in this case), usually selected based on the size of objects in the training data [16]. The training data for estimation had to be resized to the required input size to account for the resizing of the training data; this was done using the transform function. 9 anchors were used for this experiment; this predicts 9 anchor boxes in each grid cell and selects the box with the highest confidence scores using NMS. An IoU (objectness score) would be predicted for each to give a mean score, which shows how confident the system is of the chosen class. Bounding boxes are estimated to be within these areas.

3.1.2 Dataset

Training the detector to spot objects required a labelled dataset. The data set was labelled manually using the Image Labeler app on MATLAB and exported as the ground truth data. This was a bit time consuming as each cube in each image has to be assigned a rectangular ROI label. The ground truth data was then used to create the training data. The training dataset was split into three parts; 75% for training, 10% for validation, and the rest for testing the trained detector. The dataset had to undergo pre-processing and was resized to an input size of [256 256 3]. Data augmentation was applied to the training data to help improve network accuracy by adding variation to the data. This is not applied to the validation and test data to avoid biased evaluation.

3.1.3 Implementation

To train the network, a checkpoint path was set in case the was any power outage or system failure which could interrupt the training. The network used a stochastic gradient descent with momentum (sgdm). The initial learning rate was set 10^{-3} to slow down the learning in the transferred layers, validation data was set to the pre-processed validation data, and max epoch at 50. The validation frequency was set to 5. The mini-batch size was set at 2 observations at each iteration to divide the training data evenly (total = 74 observations). This would give a total of 1850 iteration. Data shuffling was set 'every epoch' to avoid discarding the same data every if the batch size didn't divide evenly. The Execution environment was set to 'auto' and verbose was true. The network was also set to plot the training progress.

3.2 Semantic segmentation

Real-world object manipulation requires precise object localization. Henceforth, an approach involving pixel-level segmentation in relation to robot arm manipulation is studied.

Semantic segmentation is one of the fundamental domains of computer vision, which operates by assigning category labels to each pixel in an image. Current state-of-the-art methods mainly rely on fully convolutional neural network architectures, that are trained by optimizing a per-pixel loss between predictions and ground truth labels [18]. Unlike object detection, this method offers a much more detailed representation of the object. Several networks which can be used for semantic segmentation include DeepLab, FCNs such as SegNet and U-Net [19]. From the different learning architectures proposed, [20] the DeepLab model by Google has stood out to give the most accurate results. To this end, the Deeplab v3+ CNN was adopted. Figure 3 shows the general architecture of the model adopted.

The model consists of a DeepLab v3+ network with weights initialised from a pre-trained ResNet-50 network. The DeepLab model is generally composed of two phases: the **encoding phase** which extracts the essential information from the images using a pretrained network and the **decoding phase** which uses the extracted information to reconstruct the output using appropriate dimensions [20]. The output is a segmented image. The final convolutional layer is adjusted for each task to reflect classes present in the dataset. The analysed network had a total of 206 layers.

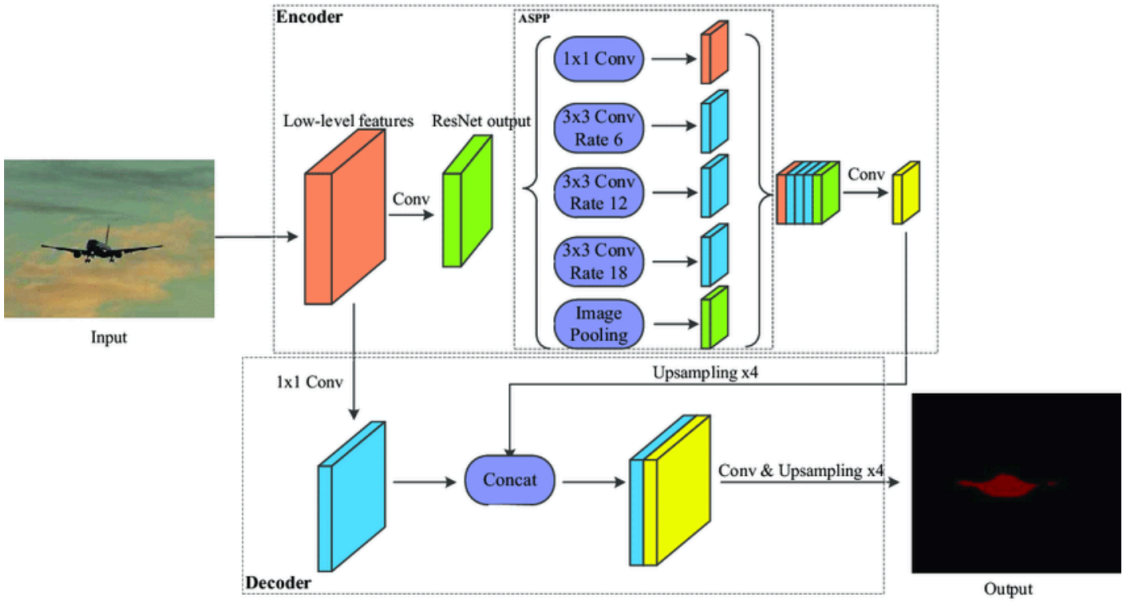


Figure 3 DeepLab v3+ Encoder-Decoder Architecture [21]

The Deeplab v3+ model shown in the figure above is similar to ours. The DeepLab v3+ replaces all max pooling operations with depthwise separable convolutions with striding, adds extra batch normalisation and ReLU activations after 3 x 3 depthwise convolutions, and increases the depth without changing the network structure's entry flow. The encoder has a 16-bit output stride (the ratio of the original image size to the size of the final encoded features). To reduce the number of channels, 1 x 1 convolutions are applied to the low-level features prior to concatenation. Following that, a few

3 x 3 convolutions are used, and the features are upsampled by a factor of four to produce an output that is the same size as the input image.

3.2.1 Balancing Classes

The pixels were divided into three different subsets, and the dataset was analysed to view the distribution of class labels. From this analysis of Figure 4, an imbalance was observed. The scenes have more belt pixels than cube and background because the belt covers more area in the images. This imbalance would bias the learning process to favour the dominant class and would classify every pixel as "belt". To prevent this and improve the training results, the class weights are used to balance the classes using median frequency balancing [19], specifying it using the pixelClassificationLayer and replacing the last layer in the network with this layer.

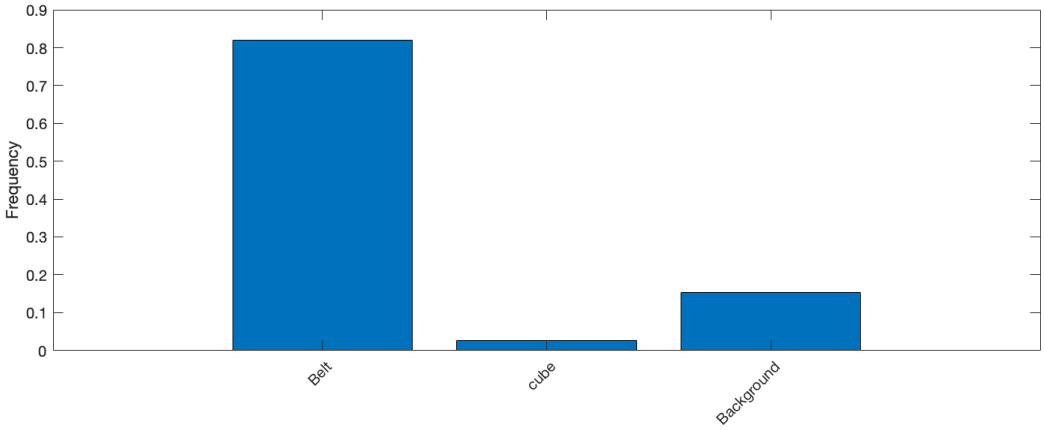


Figure 4 Class distribution of pixels

Table 2 Effects of median frequency balancing

Classes	Frequency	Class weights	Balanced frequency
Belt	0.825	0.187721746036106	0.155
Cube	0.025	5.92833247855132	0.148
Background	0.15	1.00000000000000	0.15

3.2.2 Dataset

Training the network to classify every pixel in the images required pixel labelled data. As in object detection, similar methods were used with the only difference being the label type. This method was exceptionally time-consuming as the whole image had to be assigned pixel ROI labels.

3.2.3 Implementation

The network is finetuned using sgdm. We follow [19] and set the learning rate to 10^{-3}, use L2regularization of 0.0005 to prevent the validation data from overfitting with the training data, a verbose frequency of 2, and shuffle every-epoch. However, the validation frequency is set to 10 iterations, and validation data specified to the pre-processed validation data. The maximum number of epochs is reduced to 30 since the dataset is small and a larger epoch gives a larger iteration which increases training time. The mini batch size is set at 2 to divide data evenly. This would give a total of 1110 iterations. The network environment was set to auto and the training progress was plotted.

4 Stereo Vision

A stereo camera configuration is used in this investigation. The two cameras are focused on the line of sight; this is where a sensor on the belt detects the workpieces' and halts the movement. The cameras then estimate the position of the workpiece and the manipulator then plans a trajectory to perform the pick and place action. This is very important as stereo geometry recovers the shape from the motion between the two different views and gets a sense of depth (see 4.2).

The model is a simple stereo system (coplanar image planes). The cameras are initially calibrated to obtain the extrinsic and intrinsic parameters. The depth is estimated using two independent methods: estimating depth via stereo matching with developed algorithm summarized in 4.3.1. Estimating depth from computing a disparity map.

4.1 Pose Estimation

Two vertically aligned cameras, Figure 5 separated by baseline B, and have a focal length f, observing a scene point O (X, Y, Z) in the image plane would observe in the same optic axes vertically, henceforth the x coordinates can be used to calculate the disparity [17]. To achieve this the cameras are calibrated to fix the lens distortion between the two cameras and rectify the system afterwards. Anil [11] states that all measurements hold error and articulates that these values cannot be recognized without the true values of the quantities being measured; describes calibration in itself as an evaluation method of determining and documenting how much of the equipment is in error with the actual value.

The calibration and rectification algorithm uses a known object (120 image pairs of a checkerboard) of size [1000 by 1000]mm with [8 by 8] 125mm squares. The points of the board are detected, and 3D world coordinate points are generated. The image points are also computed. Consider a transformation from w to c, the extrinsic parameters which describe the relative pose in 3D space expressed as,

$$\boldsymbol{c}_{\vec{P}} = {}^{c}_{w}\boldsymbol{R} \cdot \boldsymbol{w}_{\vec{P}} + {}^{c}_{w}\boldsymbol{t}, \qquad \textbf{Equation 1}$$

where $w_{\vec{P}}$ is the points in the world frame and $c_{\vec{P}}$ is the points in the camera frame; both [4x1] vectors corresponding to the [x ,y, z,] coordinates. The [3x3] rotation matrix ${}^{c}_{w}R$ is the rotation from world to camera frame and the [3x1] translation vector is the translation from world to the camera frame. The intrinsic parameters which describe the relative pose in the 2D image plane expressed as,

$$\overrightarrow{\boldsymbol{P'}} = \boldsymbol{K} \cdot \boldsymbol{c}_{\vec{P}}, \qquad \textbf{Equation 2}$$

where $\overrightarrow{P'}$ is the [3x1] homogeneous pixels vector in the image plane. The [3x3] intrinsic matrix $\boldsymbol{K}$. The homogeneous coordinates of both parameters expressed as,

$$\boldsymbol{c}_{\vec{P}} = \begin{bmatrix} {}^{c}_{w}\boldsymbol{R}_{3\times3} & {}^{c}_{w}\boldsymbol{t}_{3\times1} \\ \boldsymbol{O}^{T}_{1\times3} & \boldsymbol{1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{1} \end{bmatrix}_{w_{\vec{P}}} \ldots\ldots\ldots \overrightarrow{\boldsymbol{P'}} = \begin{pmatrix} \boldsymbol{z}*\boldsymbol{u} \\ \boldsymbol{z}*\boldsymbol{v} \\ \boldsymbol{z} \end{pmatrix} = \begin{bmatrix} \boldsymbol{f} & \boldsymbol{s} & \boldsymbol{c_x} \\ \boldsymbol{0} & \boldsymbol{\partial f} & \boldsymbol{c_y} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{1} \end{bmatrix}_{c_{\vec{P}}}, \quad \textbf{Equation 3}$$

S is the skew between camera points. c_x and c_y are the offsets (principal points) of u and v. The separating scaling factor between u and v is ∂.

4.1.1 Calibration

The cameras were calibrated to give the extrinsic, intrinsic and lens distortion parameters. The calibration computed 2 distortion parameters, with no skew and optical centre at the centre. This gives the camera parameters as well as the estimation errors which state how accurate the results are. The images were rectified using the parameter to remove the lens distortion. From the computation, only 37 out of the 120 image pairs were considered to give a low reprojection error.

4.2 Depth

Humans are able to perceive depth from two perfectly aligned eyes which also allow for 3D sight. Images viewed from one eye is a little different from the other eye; this creates a difference in perspectives. Viewing objects that are far away doesn't ascertain the same result. This is because the human brain automatically processes the differences and enable us to sense depth [17].

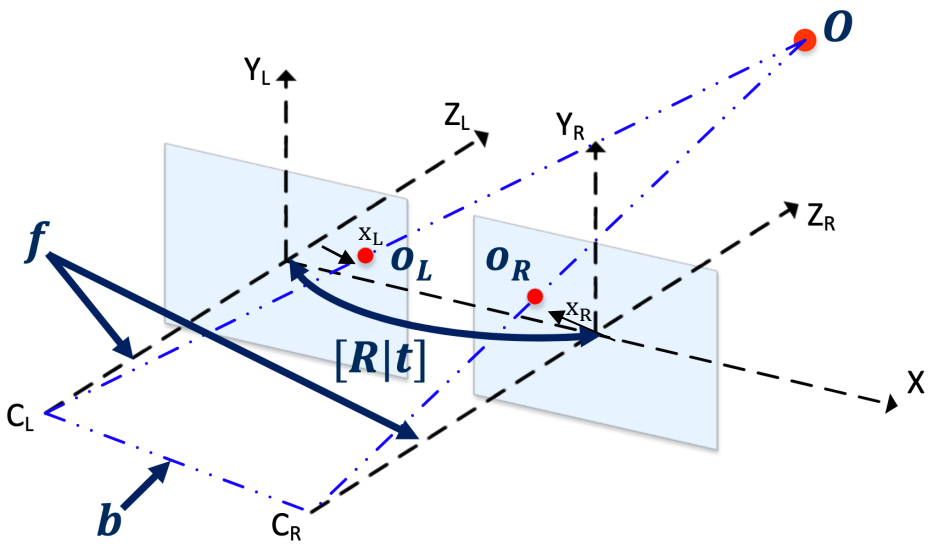


Figure 5 Stereo geometry (Parallel optical axis)[]

Stereo vision can be used to estimate depth. The diagram above shows our setup and is summarized in section 4.1. O is the scene point located at a distance Z from the camera's centre of projection (Z_L and Z_R) and is projected into the left and right cameras. The distance X is positive in the left camera (C_L) and negative in the right camera(C_R) X', points o_L and o_R in the image plane are the distances x_L and negative x_R to the right and left of the optical axes respectively. The diagram contains similar triangles (C_L, O, C_R) and (O_L, O, O_R). Writing the equivalent equations,

$$\frac{B-x_R+x_L}{Z-f} = \frac{B}{Z}, \qquad \textbf{Equation 4}$$

Yields the expression for Z,

$$Z = f\frac{B}{x_L - x_R}, \qquad \text{Equation 5}$$

The difference between the left and right amount is the disparity,

$$(x_L - x_R), \qquad \text{Equation 6}$$

The intra-ocular distance or B which is the translation between the optical centres of the two cameras can be obtained from translation vector T_t within the perspective projection equation, which is **Eq 1** and **Eq 2** combined. Thus giving,

$$\overrightarrow{P'} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = Project[M, \overrightarrow{w_P}], \qquad \text{Equation 7}$$

$$M = K\,T_t\,R_o, \qquad \text{Equation 8}$$

R_o is a [3 x 3] rotational matrix of camera 2 relative to camera 1, Project is a [3 x 4] projection matrix.

4.3 Stereo Matching

Stereo matching is one of the oldest approaches to 3D measurement and is a simpler setup compared to other 3D measurement methods[6]. This method compares the intensity values of pixels that lay on the same epipolar line in blocks of the same size between the two images and select the locations with minimum match cost. We use stereo matching methods in conjunction with each other to obtain two independent sets of results. We develop a feature-based extraction algorithm that uses the corresponding candidate pairs to estimate the depth. From this, we also estimate a disparity range centred around the object of interest and compute a disparity to be used to estimate the depth.

4.3.1 Feature-based matching

The left and right images are from the individual cameras and the detector made in section 3.1 is used as the input to the depth calculating algorithm (**Algorithm 1**). The overall framework is shown in Figure 6 and is outlined in the following steps:

Step 1. Image rectification: This procedure removes the lens distortion between the cameras and corrects the images. This was done using the stereo camera parameters in 4.1.1.

Step 2. Detect ROI (cube): The detector was used to estimate the position of the cubes in the rectified images. Because the input size of the detector is smaller than the actual image, the images had to be resized.

Step 3. Feature-based detection: Detects the local features within ROI of the images which have unique content, use a descriptor to extract the detected features around the specified region. To prevent undue loss of information, the bounding boxes were scaled up and the rectified images from step 1 were used.

Step 4. Matching: Obtain candidate matches between the features and use indices to collect the actual point location from both images.

Step 5. Calculate depth (Z): (**Eq 5**) and was used to calculate the distance of the cubes from the camera's centre of projection. Likewise, the X and Y positions are calculated using the estimated depth value.

The algorithm was then tested for accuracy by calculating the RMSE value. Each step is explained further in the subsequent section.

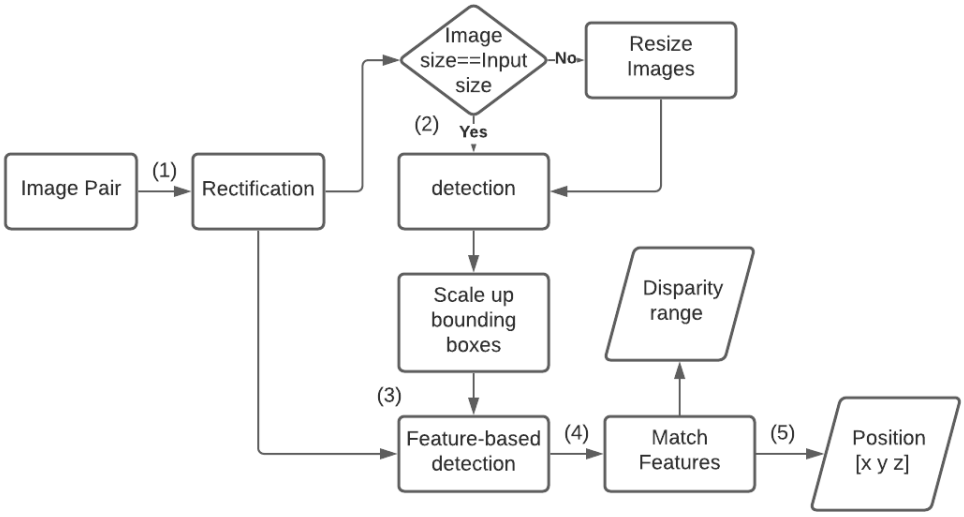


Figure 6 Data flow diagram of the position estimation algorithm

4.3.1.1 Methodology

1. Image rectification – This uses the translation and rotation vectors from the stereo parameters to align the image pairs in the y-axis so that each point observed by the cameras are in the same row in the images from each camera. Because the 2-D stereo correspondence problem is reduced to a 1-D problem, this process is useful for stereo vision.
2. Detect the cube – The detector uses an input size of [256 256 3], and the rectified images are [2033 2032 3]. To be able to detect the cubes in the image pairs, the rectified images had to be resized to the detector input size before the detector is run and outputs the bounding boxes and mean IoU scores. See Figure 13.
3. Feature-based detection – The images are converted to grayscale for 2D analysis. To achieve greater accuracy in the detection, a combination of a blob (KAZE), and a corner (BRISK) detector are used in conjunction with binary (FREAK) and gradient based (KAZE) descriptor. The KAZE detector function constructs a scale space for the given image using nonlinear diffusion. The scale space is then used to detect multiscale corner features, similar to SURF but less noisy. The BRISK detector employs a Binary Robust Invariant Scalable Keypoints algorithm to detect multiscale corner features. Computing binary descriptors are faster but not as precise as gradient-based, while blob detections are more computationally intensive than corner detections [24]. The features are only detected with the regions specified by the bounding boxes. The bounding boxes obtained from the previous step had to be scaled up to match the initial size of the rectified images. The reason for only scaling the boxes and using the rectified image pairs is to minimize the loss of information caused by quantization when the pixel intensities are divided by the decimal scale.

4. Matching – Determine candidate matches by matching FREAK descriptors first, and then KAZE descriptors. To obtain as many feature matches as possible, the max ratio was increased to 0.7 for the FREAK. Obtain the candidate matched point; candidate meaning that some points could be incorrect. That is why two descriptors are used. See Figure 14.
5. Calculate depth (Z) – Using the matched points to calculate the average disparity between the images using the matches which have a zero gradient ($y_L - y_R \cong 0$). This value is then used to estimate the depth using (**Eq 5**). The calculated depth value is then used to estimate the X and Y axis positions. From the disparity calculation, the disparity range is also obtained using the maximum and minimum disparity found in the image. To estimate the X and Y positions, the camera coordinate frame was set using the left optical centre as the reference point. The positions are estimated using the formula,

$$\boldsymbol{X = \frac{Zx_L}{f}, \quad Y = \frac{Zy_L}{f},} \qquad \textbf{Equation 9}$$

Where x_L and y_L are the differences between the principal points (u_0, v_0) and the points on the line (u_L, v_L).

4.3.2 Disparity map

A disparity map is one of the outputs of stereo matching and is can be used to estimate depth. The depth from the disparity map is very important for numerous fields such as virtual reality and in our case 'robotics' [3]. When a disparity map is precise, robot manoeuvrability is improved. Even so, disparity maps play an important role in three-dimensional (3D) reconstruction from input images in the real world. When viewing the contrastive perspectives, the disparity map displays critical 3D information for assigning image pixels to precisely produce the depth of the dissimilar detected objects when examining the opposing viewpoints.

With all of its benefits, depth estimation from disparity maps is one of the most difficult and intricate problems. Many studies have been conducted in recent years to address this issue, and significant progress has been made. [3] suggests an improvement of the disparity refinement stage using adaptive least square plane fitting. Their method proposes multiple point disparity selections on a plane to adaptively group similar disparity values, then calculate the distance of each disparity to ensure they are accurately grouped, and this is understood to improve the accuracy on the plain colour or low texture regions, repetitious pattern, and depth discontinuity area. Schwarz et al. [12] present a simple technique for fusing three separate sources of depth information, thereby improving overall depth measurement accuracy and incorporates this into their training. Such methods would require GPU to reduce computation time. [6] propose a novel stereo matching method that makes use of surface normal data derived from the photometric stereo technique.

To compute our disparity map, we use the disparity range obtained in step 5 of the featured-based matching technique. The range is the difference be maximum and minimum disparities from the calculated average disparities using the matched points. The disparity map is then computed using block matching. This method uses the sum of absolute difference (SAD), as the cost function to estimate the displacement between the pixels in the rectified stereo image pair. Pixels are marked for unreliability using a uniqueness threshold value of 25. A block size of 11 is set as the width of the search window. This generates a 2D grayscale image as a disparity map with the same dimensions as the input images. Each value in this output denotes the displacement between conjugate pixels in a

stereo pair image. The average depth value is calculated by combining this image with the binary logical mask of the predicted segmentation ground truth and obtaining the mean value. The Proposed algorithm (**Algorithm 2**):

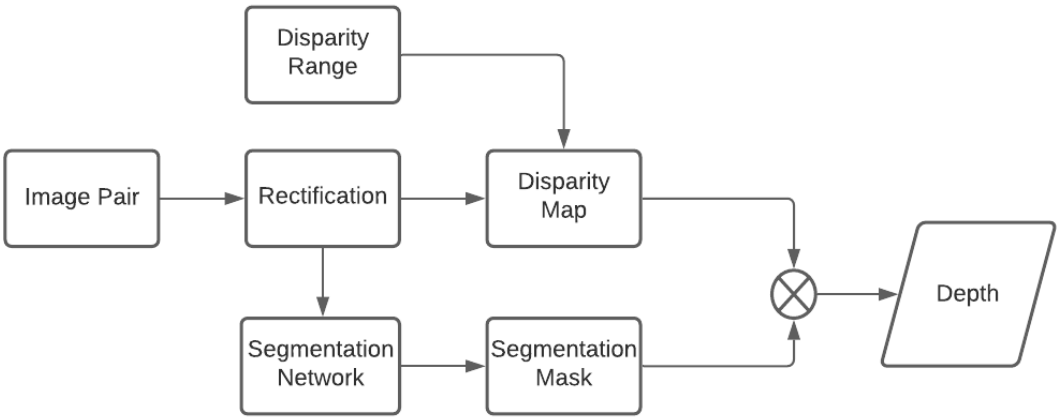


Figure 7 Depth from disparity map algorithm

Summary:

The algorithm is fed the input image pairs, the segmentation network, the detector, and the stereo parameters. It then invokes the position estimation function (**Algorithm 1**), which computes the disparity map using the disparity range and rectified images. The network is then used to segment the left image, and the binary logical mask for the cube is obtained as a result. The mask and the disparity mask are then combined to give the depth by performing element-wise multiplication and obtaining the mean value as the depth.

5 Experimental Results

During the course of this research, all data used was gathered from a particular robotic simulation software; CoppeliaSim. In order to validate the performance of the proposed pipeline, we evaluate the two detection methods for level of accuracy of recognition of the workpiece, evaluate the error estimates from the stereo calibration to analyse the accuracy of the pose estimation, evaluate Algorithm 1 for speed and accuracy for accuracy by comparing our position estimate with the dataset provided and performing a regression analysis. We also evaluate the depth map computed in section 4.3.2 and evaluate the estimated depth using the disparity map and segmented images. Finally, the algorithm is put to the test through simulation.

5.1 Object detection and semantic segmentation

The training and prediction of the DCNNs are implemented on an Intel Iris Plus, 2.3GHz Quad-core Intel Core i5 CPU with 8GB RAM. For evaluation, the pre-processed test datasets were used. Both datasets contained 16 images. This is a very small test set and highpoints the effectiveness of the transfer learning approach. Figure 8 depicts a segmentation and object detection result for an example scene from the dataset.

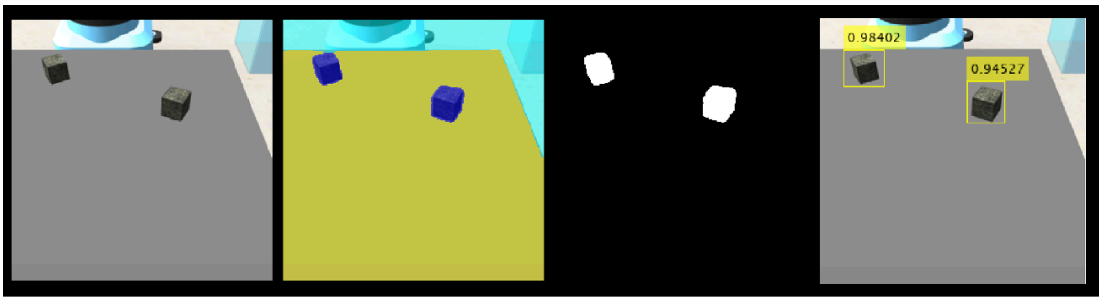


Figure 8 Object detection and segmentation results

Figure 8 left – right: Input RGB image frame[1], predicted segmentation of image[2], the binary logical mask of the segmented image of the workpieces'[3], predicted bounding boxes and mIoU of workpieces'[4].

5.1.1 Object detection

Traditional methods using common metrics such as average precision and log average miss rate are employed. The mean average precision(mAP) provides a single number that incorporates the detector's capacity to make correct classifications (precision) as well as the detector's ability to find all relevant objects (recall). This was done using an initial threshold of 0.7; this determines the extent of overlap of the bounding box around the workpiece given by the YOLOv2 detector over the bounding box of the same workpiece in the ground truth. [15]The precision/recall (PR) curve Figure 9[left], highlights how precise a detector is at varying levels of recall. The ideal precision is 1 at all recall levels. The log average miss rate Figure 9[right] operates opposite to the mAP. Provides a number that incorporates the detectors' ability to make wrong classifications (miss rate), and also find irrelevant objects(fppi). Given an initial threshold value of 0.7.

The detector is evaluated using different threshold values to see how it affects the accuracy of the detection. Table 3 shows the precision and miss rate using the different threshold values and Figure 9 show the plots.

Table 3 Precision and Miss rate for different thresholds

	Threshold		
	0.7	0.75	0.8
mAP	1.00	1.00	0.96
Log average miss rate	0.00	0.00	0.08

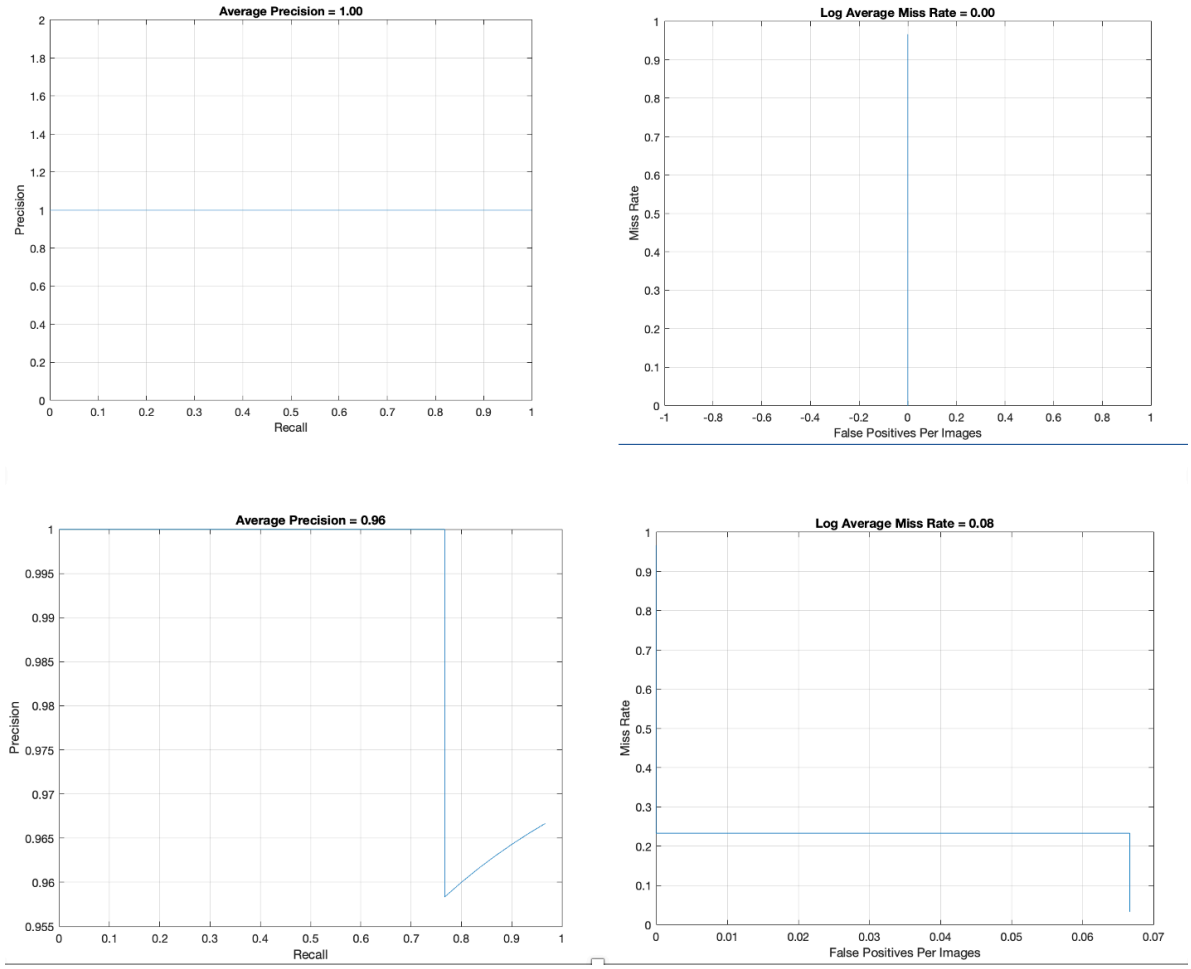


Figure 9 Average precision and Miss rate

5.1.2 Semantic segmentation

The segmentation performance was defined by the 'success rate,' which is defined as the percentage of evaluation images segmented with an intersection over union (IoU) greater than 0.75 (75%) [22], and accuracy greater than 0.9 (90%). The accuracy represents the percentage of correctly identified pixels in each image class defined as,

$$\mathbf{Accuracy} = \frac{\mathbf{TP+TN}}{\mathbf{TP+TN+FP+FN}}, \qquad \textbf{Equation 10}$$

Where TP are the correctly classified pixels, TN are the pixels correctly classified as false, FP are the predicted incorrect pixels classified as correct, FN are the labelled pixels in the ground truth incorrectly classified.

The IoU is a statical measurement of accuracy that penalises false positives (FP). The IoU for each class is the proportion of correctly classified pixels to the total number of pixels assigned by the ground truth and predictor in that class defines as,

$$\mathbf{IoU} = \frac{\mathbf{TP}}{\mathbf{TP+FP+FN}}, \qquad \textbf{Equation 11}$$

The network is run over the pre-processed test dataset using a minibatch size of 4; this returns the results for the test. The predicted labels from the test set are then compared with the ground truth labels. This computation process returns various metrics for the entire test set, for individual classes, and each test image. The dataset metric is first inspected. This metric provides a high-level overview of the network's performance.

Table 4 Dataset Metrics

Global accuracy	Mean accuracy	MeanIoU	WeightedIoU	MeanBF score
0.99256	0.99660	0.94195	0.98587	0.98511

Table 4: The global accuracy ratio is the proportion of correctly classified pixels, regardless of class, to the total number of pixels. The mean accuracy is the average accuracy of all classes in all images. The MeanIoU is the average IoU score of all classes in all images. The average IoU of each class is weighted by the number of pixels in that class to calculate the weighted IoU. The MeanBF score is the average BF score of all classes in all images.

The class metric is then inspected to evaluate the classification accuracy. This shows the impact each class has on the overall performance.

Table 5 Per-class metrics

Classes	Accuracy	IoU	MeanBF score
Belt	0.99111	0.99091	0.99084
Cube	0.99972	0.85212	0.966099
Background	0.99899	0.98282	0.99839

Table 5: Accuracy - The proportion of correctly classified pixels in each class to the total number of pixels in that class based on ground truth **(Eq 10)**. IoU for each class. The MeanBF score for each class is the average BF score of that class across all images.

Note: The contour matching score for the F1 boundary (BF) indicates how well the predicted boundary of each class aligns with the true boundary.

The normalised confusion matrix displayed as a heatmap which shows the percentage count of pixels belonging to the true class versus the prediction, divide by the total number of pixels predicted.

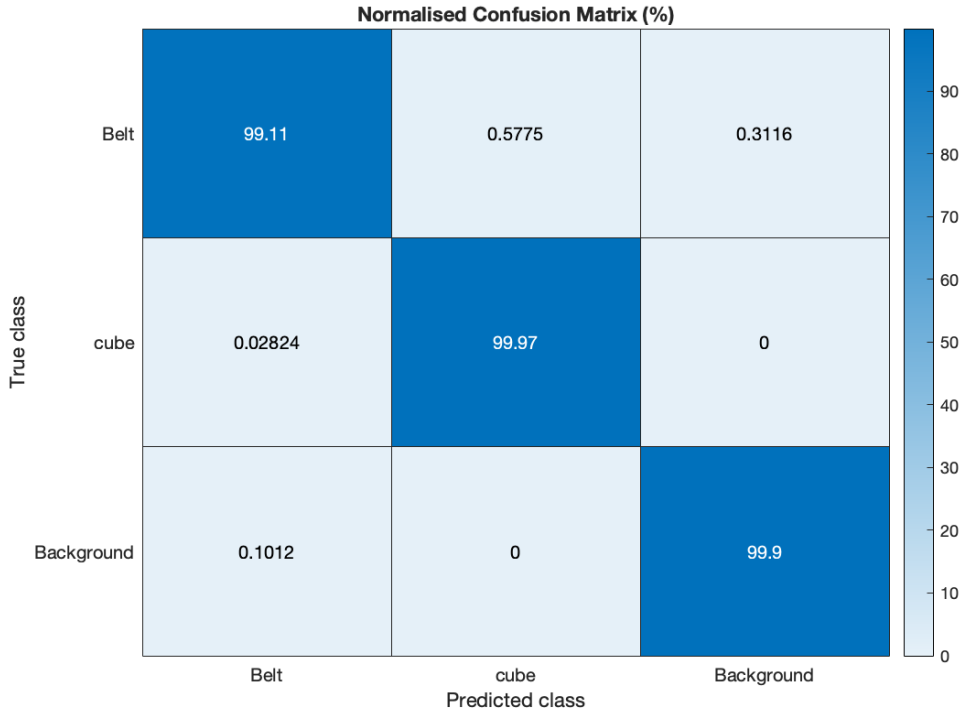


Figure 10 Normalised confusion matrix heatmap

Inspect the average IoU of all three classes in the images.

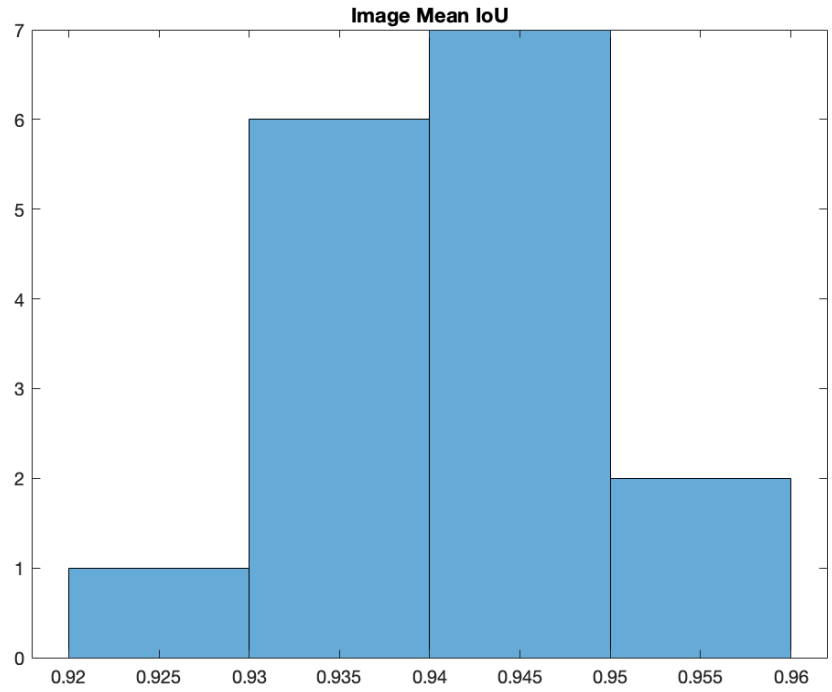


Figure 11 Histogram of per-image intersection over union (IoU)

5.1.3 Evaluation

When compared to the semantic segmentation network, the object detector with object detection algorithm has been shown to have an average precision of 96%, Table 3 and Figure 9. The DeepLab

model was able to classify over 99 percent of the pixels in the image for segmentation. Table 5 shows that, while the dataset's performance is quite high, the class metrics show that the underrepresented 'cube' class is not as well segmented as the 'background' and 'belt' classes. Figure 9 displays this statistic. The heatmap shows the cube pixels having the highest percentage of false positives (0.5775%). Overall, the segmentation algorithm was successful in segmenting pixels in the images (IoU > 80% & accuracy > 90%).

5.2 Camera calibration

The calibration results in the creation of an object that stores the intrinsic and extrinsic parameters of the two cameras as well as their geometric relationship. The geometric relation parameters as shown in **Eq 8** as well as the fundamental matrix, the essential matrix etc. The matrix **K** intrinsic parameters of the cameras obtained by the calibration,

$$\boldsymbol{K} = \begin{bmatrix} f_u & 0 & 0 \\ 0 & f_v & 0 \\ u_0 & v_0 & 1 \end{bmatrix}$$

Where f_u and f_v are the focal length of the horizontal and vertical axis in pixel dimensions, u_0 and v_0 are the principal points.

The translation matrix $\boldsymbol{T_t}$ of camera 2 relative to camera 1,

$$\boldsymbol{T_t} = [-\boldsymbol{B} \quad 0 \quad 0]$$

Where baseline, B is the intra-ocular distance between the two cameras translating from camera 2 to camera 1, hence the negative.

The accuracy of the stereo calibration is evaluated using the visualising obvious errors from the camera extrinsic, the mean reprojection error and examining the estimation errors from the calibration.

Plotting the camera's and calibration pattern's relative positions is a quick way to detect obvious errors. For example, if the pattern is behind the camera or the camera is behind the pattern, or if the pattern is too far or too close to the camera.

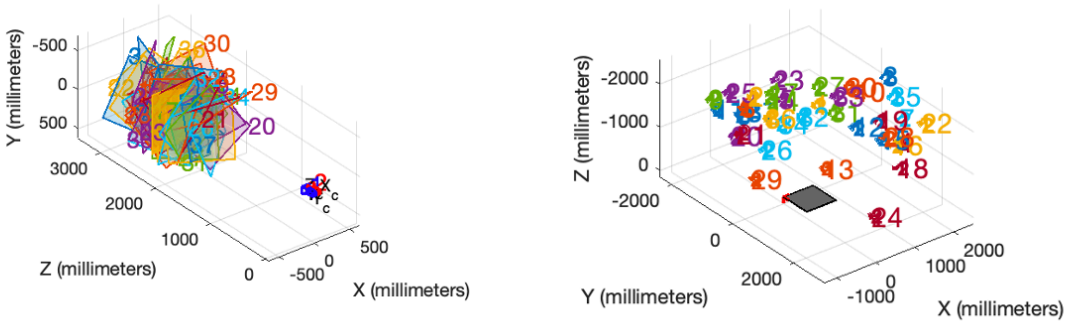


Figure 12 Extrinsic Parameters

Figure 12, [left] calibration pattern locations in the camera's coordinate system, [right] camera positions in the pattern's coordinate system.

The mean reprojection error gives a good estimation of just how accurate the estimated parameters are. This is the average Euclidean distance in pixels between reprojected and detected points across all image pairs []. The closer the re-projection error is to zero, the more accurate the parameters are.

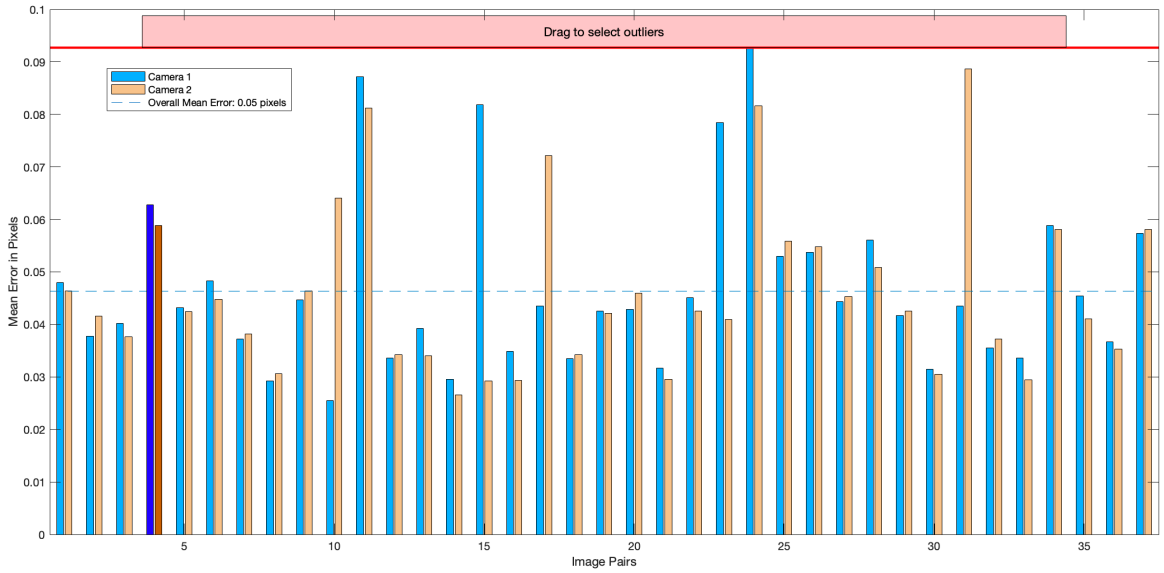


Figure 13 Reprojection Errors

From the extrinsic parameter visualisation, obvious errors could not be detected as image frames and camera frames were clustered. The calibration achieved a mean reprojection error of 0.0463 which was satisfactory. The error number could still be improved by adding more distortion coefficients or removing more image pairs with higher errors.

The estimation errors represent the degree of uncertainty associated with each estimated parameter. This function returns the standard error for the estimated stereo parameter. The standard error (in the same units as the corresponding parameter) that is returned can be used to compute confidence intervals [25]. Appendix A displays results.

5.3 Depth Estimation

Of the two methods used to estimate the positions of the items, only the feature-based matching algorithm was able to give desired values.

5.3.1 Feature-based matching

The algorithm was able to estimate the position [x y z] of the workpieces using the matched feature points. The combined feature detectors were able to detect a fairly large number of points with the ROI but achieved a match rate of about 35%.

The standard deviation of prediction errors (RMSE) was used to evaluate the performance of depth and position estimation, as well as the speed of programme execution. Only 99 of the 100 image pairs allowed the algorithm to estimate the positions of the workpiece. The mistake appeared to have occurred during the feature detection phase. As the error message pointed out the ROI input, it was assumed that the detector failed to detect the cube and output its bounding box. The execution of the function took about 6.174s. Figure 16 show the error comparisons. Some trends were observed

from the compared values; the estimated Y values were greater than the actual Y values, the estimated depth values were smaller than the actual depth value whilst the x values varied. From the regression analysis performed, the RMSE was obtained as 0.0079394 metres; equivalent to 7.94 millimetres.

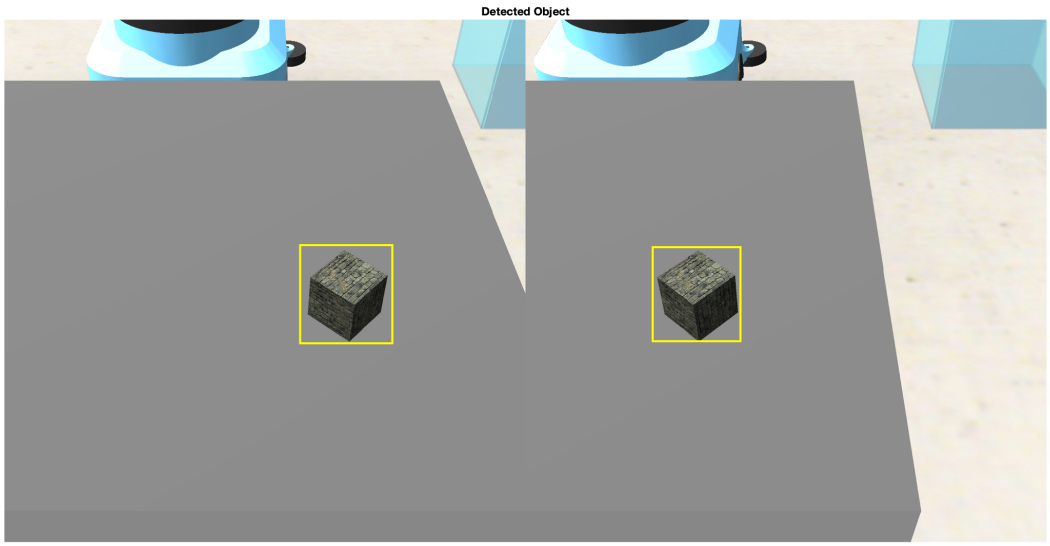


Figure 14 Correctly detected workpiece

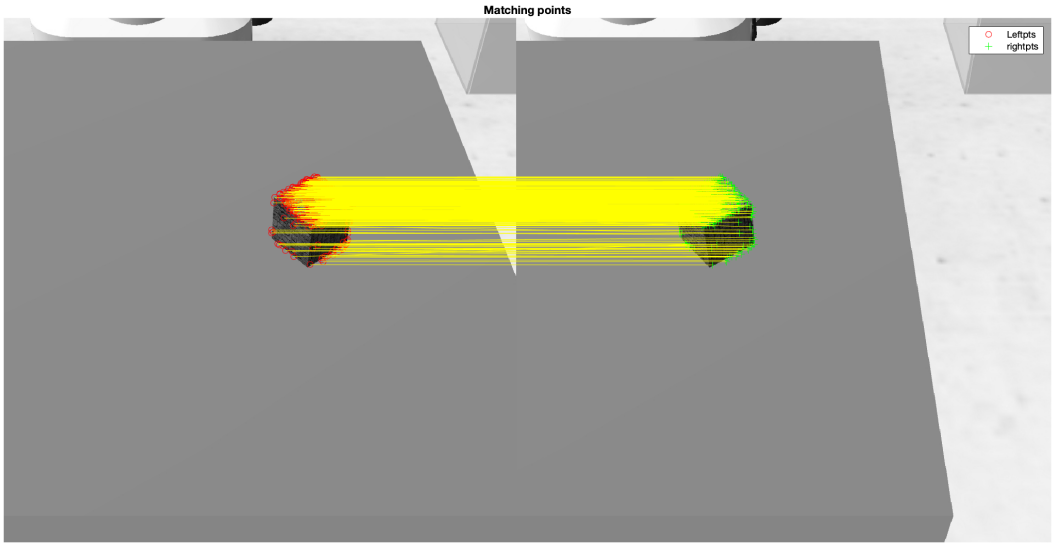


Figure 15 Corresponding matches found

5.3.2 Depth from the disparity map

Unfortunately, this method did not perform how it was expected, but probable causes for the poor performance were identified. Because the disparity range is obtained using the matched points, the disparity map is computed using the max and min value from the points the disparity, but from Figure 15 it can be observed that the corresponding points found do not cover the whole area of the desired object. Just like the matching algorithm, the disparity map computed fails to find all corresponding matches within the range and output a map with few pixels. Because of these missing pixels, the depth estimation is inaccurate as there not enough pixels to obtain a reasonable mean value. Ways to improve the disparity would be explored.

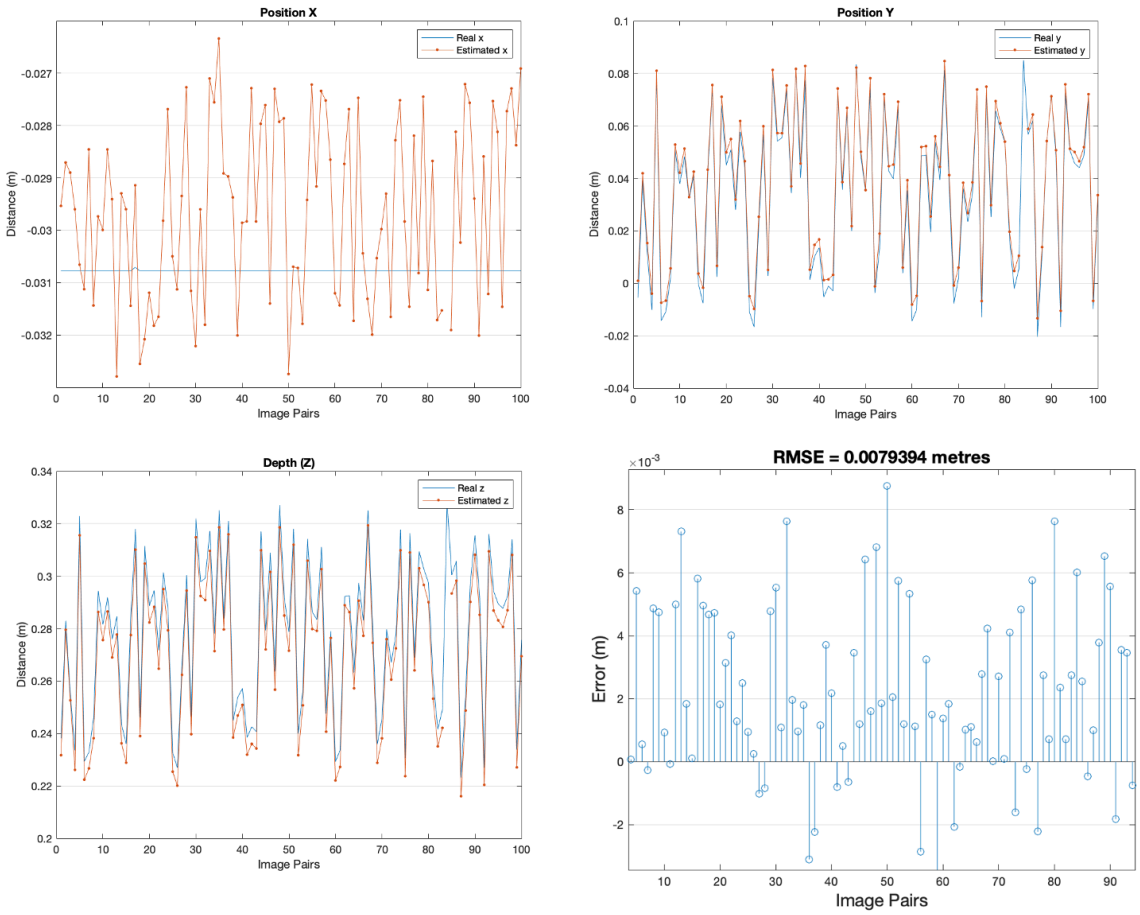


Figure 16 Comparison of real and estimated values as well as regression analysis of Algorithm 1

5.4 Task Execution

In order to investigate direct transfer to a real robot, the simulation environment had to be as close to the real world as much as possible. A 6-DOF NiryoOne robot arm, two camera modules, a conveyor belt, two spotlights, workpiece cubes, and a bin comprise the simulation setup. To reduce complexity, the manipulator is stationary, the joint positions for the six joints are determined ahead of time, and the trajectory to the bin is calculated so that the robot knows where it is without having to necessarily locate it.

The control and computation of task are done on a single computer equipped with an Intel® Core™ i3 CPU (2.50GHz) with 8GB RAM. The code to perform the pick and place task is run on MATLAB and the simulation on CoppeliaSim. The software's are linked using a remote API server. To summarize the operation, the belt is equipped with a proximity sensor that acts as the line of sight for the robot and halts the conveyor belt when an item is detected. The streamed images from the stereo cameras are then processed to detect the item of interest Figure 14 and estimate the position of the item using the algorithm developed in section 4.3.1, the joint values are then obtained from the estimated position and the pick and place action is initiated. The manipulator behaved as expected, moving towards the cube and performing the pick and place action, implying that the transition from simulation to real-world is possible.

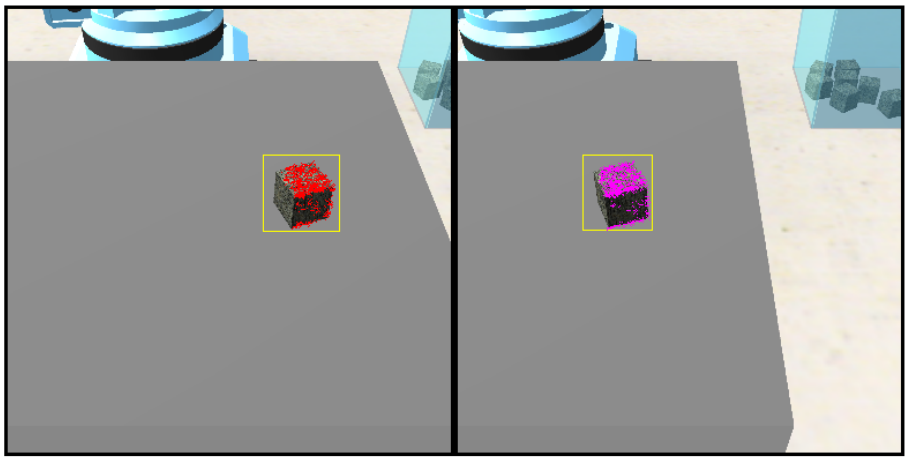


Figure 17 Detected object from stream; left and right images

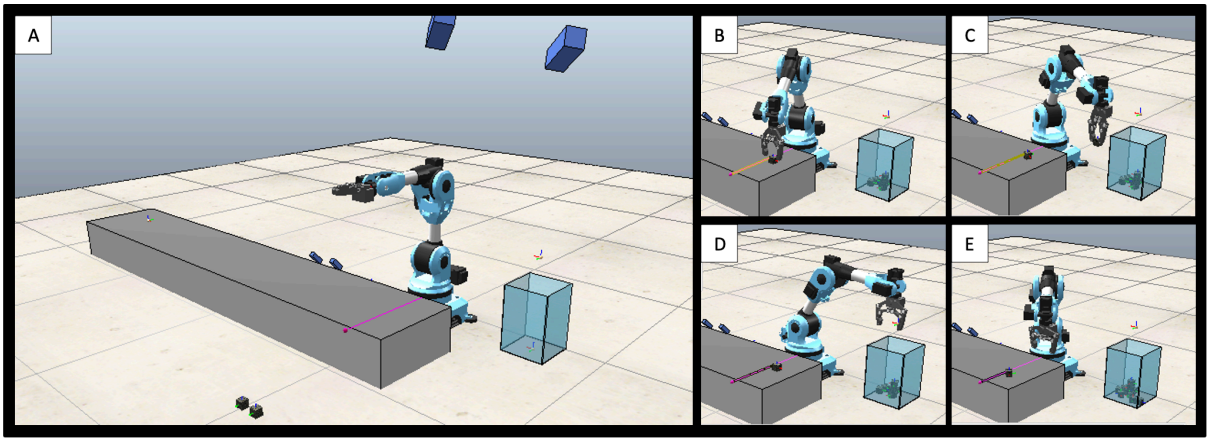


Figure 18 Visualisation of simulation

Figure 14: A real-world setting for a virtual-reality simulation (a). The proximity sensor flashes, indicating that an object has been detected and action is required. The robotic arm recognises the item, estimates its position, and attempts to grasp it (b). The robot arm successfully grasps and lifts the workpiece before performing the place action (c). After dropping, the arm moves to the next target and the process is repeated (e).

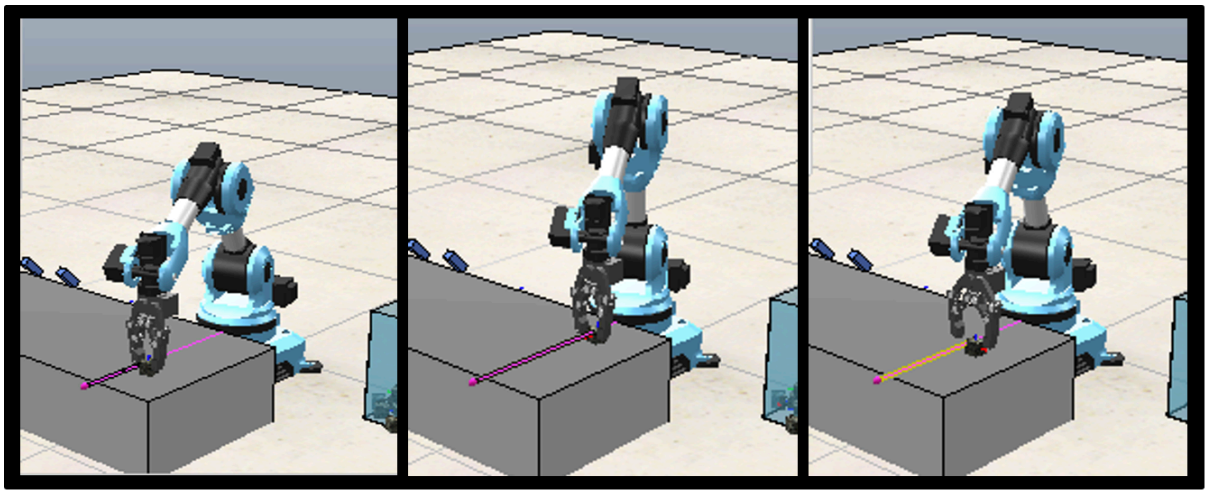


Figure 19 Task execution with workpiece at different positions

6 Discussion

The accuracy of the segmentation model was about 3% higher than that of the object detector. However, the segmentation model does not overlap for the cube class as well as the other classes present in the ground truth data as seen in Table 5. Perhaps using a different weight balancing method might improve the results. The YOLO detector completed 1850 iterations in about 62 minutes which seems like a very long time; However, the detector was able to achieve a very low loss of 0.0087 and RMSE of 0.0932 which were considered acceptable. Initially, the detector was trained using 7 anchor boxes, but this proved insufficient when applying the position estimation algorithm as the network could not detect some of the bounding boxes accurately, hence the change to 9 anchor boxes for more areas of detection. The segmentation network completed 1110 iterations in 84 minutes achieving an accuracy of 99.3%. On the first attempt, the segmentation model failed to properly segment the image; this was due to the class weights being unbalanced, so the network classified almost all pixels as 'belt', hence the use of class weights in section 3.2.1. Overall, both models were able to detect the workpiece accurately.

The featured-based matching approach fared better than the disparity map computation when estimation the position of the item. The computation time was 6.174s which isn't too slow but could be improved. From the profiling results, it was observed that about 59 % of the computation time was spent on the feature extraction. This was hugely taken up by the KAZE extraction; as noted in 4.3.1.1 gradient-based descriptors are more computationally intensive. It was noted that only 35% of the detected point were able to find corresponding matches. Though this number seems low, compared to single detectors the performance was significantly better; Appendix C shows results of single feature detector matching. More matching could be obtained by adjusting some of the parameters of the matching function such as the thresholding and ratio, bearing in mind that this could introduce inaccurate correspondences. At first, the position estimation did not yield plausible results for the x axis, this was due to the coordinate frame being set in the middle of the two cameras and calculation being done with the left camera centre as the reference point. This was later corrected, and the new dataset yielded better results. The algorithm was assumed to have a 99% estimation rate in view that it was able to estimate the positions in 99 out of the 100 images. The RMSE revealed the error to be about 7.94mm which is below a centimetre but could be improved upon. Perhaps following the methods of [12] and combining the posteriors of the segmentation and object detection could yield greater accuracy.

The task execution was successful but was extremely slow. To complete a single pick and place action took 3m 41s. Even at that, the code performed accurately. After 100 episodes the manipulator never once failed to detect the workpiece or drop the item. The robot had also displayed the ability to detect the item at different positions on the belt as seen in Figure 19. Improvements to the execution time could be made. Perhaps a parallel execution environment or the use of a GPU could speed up the process.

This project was initially started on Python, but it was later moved to MATLAB. This was due to the calibration dataset not being compatible with the software. Attempts will still be made to write this in the Python programming language. MATLAB, on the other hand, proved to be adequate for the task. With very little prior knowledge of computer vision and deep learning, this project was undertaken.

7 Conclusion

In this paper, we proposed a computer vision pipeline that allows for a manipulator to detect workpieces on a conveyor belt. We explored two independent methods of detecting the items. Both methods achieve high detection accuracy 96% for the YOLO object detector and 99.11% for the DeepLab segmentation model. We also explored two way in which the positions of the items in the image can be estimated. The calibration had a mean reprojection error of 0.0463 pixels. Unfortunately, only one of the algorithms was able to deliver plausible results. **Algorithm 1** was able to achieve an estimation RMSE of 7.94mm. The robot was able to estimate the various positions of the workpiece execute that pick and place task without any error. From the results of the experiment, we can conclude that:

- Computation of disparity maps is still a very challenging field and requires more research
- Object detection and semantic segmentation are effective methods that can be used for object recognition in computer vision.
- It is possible for a robot to accurately estimate the position of items within an image to execute a task using computer vision algorithms.

Key achievements include:

- Ability to develop an image processing pipeline.
- Successfully developing a position estimation algorithm.

The framework of the proposed estimation algorithm can be extended to different objects using a specified detector and stereo calibration parameters.

Related codes to the estimation algorithm are available online at:

https://github.com/aniekanBane/stereo-geometry.

Further investigations into the disparity map computation and the combination of detectors posteriors would be the future direction for improvements.

8 References

[1] Mavrakis, N., & Stolkin, R. (2020). Estimation and exploitation of objects ' inertial parameters in robotic grasping and manipulation: A survey. Robotics and Autonomous Systems, 124, 103374. (https://doi.org/10.1016/j.robot.2019.103374)

[2] Liu, W., Hu, J., & Wang, W. (2020). A novel camera fusion method based on switching scheme and occlusion-aware object detection for real-time robotic grasping. Journal of Intelligent & Robotic Systems, 100(3-4), 791. (https://doi.org/10.1007/s10846-020-01236-7)

[3] Hamzah, R. A., & Ibrahim, H. (2020). Improvement of disparity map refinement stage using adaptive least square plane fitting technique. Electronics Letters, 56(18), 918-920. (https://doi.org/10.1049/el.2020.1067)

[4] Woźniak, M., & Połap, D. (2018). Object detection and recognition via clustered features. Neurocomputing (Amsterdam), 320, 76-84. (https://doi.org/10.1016/j.neucom.2018.09.003)

[5] Lee, Y. E., Kim, B., Bae, J., & Kim, K. C. (2021). Misalignment detection of a rotating machine shaft using a support vector machine learning algorithm. International Journal of Precision Engineering and Manufacturing, 22(3), 409-416. (https://doi.org/10.1007/s12541-020-00462-1)

[6] Song, E., Kim, S., & Chang, M. (2020). Novel stereo-matching method utilizing surface normal data. International Journal of Precision Engineering and Manufacturing, 21(8), 1437-1445. (https://doi.org/10.1007/s12541-020-00350-8)

[7] Szabó, R., Gontean, A. (2013). Full 3d robotic arm control with stereo cameras made in LabVIEW. 30th federated conference on computer science and information systems (Fedcsis). pp 37-42. (https://annals-csis.org/proceedings/2013/pliks/52.pdf)

[8] James, S., johns, E. (2016). 3d simulation for robotic arm control with deep q-learning. retrieved from: (https://arxiv.org/pdf/1609.03759.pdf)

[9] How to Do Deep Learning With SAS®. (2019). Retrieved from: (https://www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/deep-learning-with-sas-109610.pdf)

[10] Gilewski, J. (2019). Modular image processing pipeline using OpenCV and python generators. Retrieved from: (https://medium.com/deepvisionguru/modular-image-processing-pipeline-using-opencv-and-python-generators-9edca3ccb696)

[11] Anil. (2018). metrology. IntechOpen. retrieved from: (https://www.intechopen.com/books/metrology/introductory-chapter-metrology)

[12] Schwarz, M., Milan, A., Periyasamy, A. S., & Behnke, S. (2018). RGB-D object detection and semantic segmentation for autonomous manipulation in clutter. *The International Journal of Robotics Research*, *37*(4–5), 437–451. (https://doi.org/10.1177/0278364917713117)

[13] MATLAB. (2021). Object Detection using YOLOv2 deep learning. retrieved from: (https://uk.mathworks.com/help/vision/ug/train-an-object-detector-using-you-only-look-once.html)

[14] Hulstaert, L. (2018). Going deep into object detection. retrieved from: (https://towardsdatascience.com/going-deep-into-object-detection-bed442d92b34?gi=d23176487019)

[15] MATLAB. (2021). Getting started with YOLOv2. Retrieved from:

[16] MATLAB. (2021). Anchor boxes for object detection. retrieved from:

[17] Yasin eser, A. (2020). The depth I: stereo calibration and rectification. retrieve from: (https://python.plainenglish.io/the-depth-i-stereo-calibration-and-rectification-24da7b0fb1e0)

[18] Chen, Y., Dapogny, A., & Cord, M. (2020). SEMEDA: Enhancing segmentation precision with semantic edge aware loss. *Pattern Recognition, 108*, 107557. https://doi.org/10.1016/j.patcog.2020.107557

[19] eddins, S. (2018) Semantic segmentation using deep learning.

[20] PAL, S. (2019). semantic segmentation: introduction to the deep learning technique behind google pixel's CAMERA! retrieved from: (https://www.analyticsvidhya.com/blog/2019/tutorial-semantic/segmentation-google-deeplab/)

[21] Zhang, Sanxing & Ma, Zhenhuan & Zhang, Gang & Lei, Tao & Zhang, Rui & Cui, Yi. (2020). Semantic Image Segmentation with Deep Convolutional Neural Networks and Quick Shift. Symmetry. 12. 427. 10.3390/sym12030427.

[22] Gai, J., Tang, L., & Steward, B. L. (2020). Automated crop plant detection based on the fusion of colour and depth images for robotic weed control. *Journal of Field Robotics, 37*(1), 35-52. https://doi.org/10.1002/rob.21897

[23] CS231n. (2017). Convolutional neural networks for visual recognition. retrieved from: (https://cs231n.github.io/transfer-learning/)

[24] MATLAB. (2021). extract features

[25] MATLAB. (2021). estimate camera parameters.

9 Appendix A

Standard Errors of Estimated Stereo Camera Parameters

Camera 1 Intrinsics

Focal length (pixels): [2472.2443 +/- 0.0928 2472.1777 +/- 0.0901]

Principal point (pixels): [1024.3966 +/- 0.0710 1024.5489 +/- 0.0690]

Radial distortion: [-0.0000 +/- 0.0004 0.0004 +/- 0.0037]

Camera 1 Extrinsics

Rotation vectors:

[0.0001 +/- 0.0001 0.0000 +/- 0.0001 -1.5708 +/- 0.0000]

[-0.3140 +/- 0.0001 -0.5167 +/- 0.0001 -2.1199 +/- 0.0000]

[-0.3172 +/- 0.0001 0.5443 +/- 0.0001 0.5835 +/- 0.0000]

[0.6128 +/- 0.0000 1.1095 +/- 0.0000 -1.8262 +/- 0.0000]

[0.4017 +/- 0.0000 0.2447 +/- 0.0001 0.9902 +/- 0.0000]

[1.0085 +/- 0.0000 0.4322 +/- 0.0000 0.8134 +/- 0.0000]

[-0.5448 +/- 0.0000 0.9234 +/- 0.0000 0.5623 +/- 0.0000]

[0.8255 +/- 0.0000 0.1303 +/- 0.0000 1.3405 +/- 0.0000]

[0.6124 +/- 0.0000 -0.6468 +/- 0.0000 -0.7236 +/- 0.0000]

[-0.0910 +/- 0.0001 0.3903 +/- 0.0001 -0.2822 +/- 0.0000]

[0.8653 +/- 0.0000 -1.0396 +/- 0.0000 -1.3536 +/- 0.0000]

[-0.2564 +/- 0.0001 0.4635 +/- 0.0001 -1.3314 +/- 0.0000]

[-0.2760 +/- 0.0000 -0.7824 +/- 0.0000 1.4832 +/- 0.0000]

[0.4836 +/- 0.0000 0.7454 +/- 0.0000 1.4768 +/- 0.0000]

[-0.8276 +/- 0.0000 -0.1985 +/- 0.0000 1.7759 +/- 0.0000]

[-0.6369 +/- 0.0000 0.7331 +/- 0.0000 0.1212 +/- 0.0000]

[0.2159 +/- 0.0001 -0.2757 +/- 0.0001 -0.5463 +/- 0.0000]

[-0.9410 +/- 0.0000 0.3873 +/- 0.0000 -0.2107 +/- 0.0000]

[0.2687 +/- 0.0001 0.7851 +/- 0.0001 -2.0487 +/- 0.0000]

[1.0201 +/- 0.0000 0.6813 +/- 0.0000 2.0400 +/- 0.0000]

[0.5464 +/- 0.0000 -0.6070 +/- 0.0000 -0.1739 +/- 0.0000]

[-0.3480 +/- 0.0000 0.8497 +/- 0.0000 -0.8603 +/- 0.0000]

[0.4921 +/- 0.0000 -0.2071 +/- 0.0001 -0.5308 +/- 0.0000]

[-0.8539 +/- 0.0000 -1.1109 +/- 0.0000 1.4035 +/- 0.0000]

[0.4018 +/- 0.0001 -0.5793 +/- 0.0001 -0.4388 +/- 0.0000]

[-0.2030 +/- 0.0000 -0.6366 +/- 0.0000 -0.3327 +/- 0.0000]

[0.5114 +/- 0.0001 -0.1617 +/- 0.0001 -2.0048 +/- 0.0000]

[-0.5604 +/- 0.0000 0.4453 +/- 0.0000 -0.4790 +/- 0.0000]

[0.6733 +/- 0.0000 -0.8563 +/- 0.0000 1.2704 +/- 0.0000]

[-0.1704 +/- 0.0001 0.3088 +/- 0.0001 1.4026 +/- 0.0000]

[0.2732 +/- 0.0001 0.1647 +/- 0.0001 -2.1279 +/- 0.0000]

[0.0014 +/- 0.0001 -0.2190 +/- 0.0001 0.8556 +/- 0.0000]

[-0.0177 +/- 0.0001 0.1193 +/- 0.0001 -1.0786 +/- 0.0000]

[0.2523 +/- 0.0001 -0.2305 +/- 0.0001 1.5546 +/- 0.0000]

[0.3060 +/- 0.0001 0.6491 +/- 0.0001 -2.0792 +/- 0.0000]

[0.1844 +/- 0.0001 -0.4658 +/- 0.0001 0.6153 +/- 0.0000]

[0.8679 +/- 0.0000 -0.7302 +/- 0.0000 -1.3148 +/- 0.0000]

Translation vectors (millimetres):

[-488.8933 +/- 0.0649 356.1638 +/- 0.0630 2259.2034 +/- 0.1014]

[-176.6366 +/- 0.0747 557.7898 +/- 0.0723 2582.4908 +/- 0.1272]

[-106.3028 +/- 0.0908 -357.2386 +/- 0.0886 3163.5157 +/- 0.1110]

[-15.1019 +/- 0.0728 108.9893 +/- 0.0716 2558.0473 +/- 0.0844]

[215.9453 +/- 0.0577 -405.0651 +/- 0.0559 1986.9981 +/- 0.0878]

[-79.4812 +/- 0.0593 -225.1597 +/- 0.0580 2064.5996 +/- 0.0923]

[-6.1568 +/- 0.0807 -173.0286 +/- 0.0793 2816.0461 +/- 0.0917]

[205.5509 +/- 0.0568 -493.8714 +/- 0.0563 1970.9892 +/- 0.0947]

[-234.4591 +/- 0.0724 -126.0314 +/- 0.0706 2496.2956 +/- 0.1180]

[-337.9269 +/- 0.0774 -225.1057 +/- 0.0746 2698.0052 +/- 0.0965]

[-93.0807 +/- 0.0731 233.0472 +/- 0.0704 2522.2798 +/- 0.1159]

[-371.6224 +/- 0.0738 102.7704 +/- 0.0714 2575.8950 +/- 0.0925]

[362.8619 +/- 0.0654 -318.3349 +/- 0.0637 2293.4399 +/- 0.0854]

[203.4585 +/- 0.0574 -262.1489 +/- 0.0552 1988.4226 +/- 0.0847]

[413.4008 +/- 0.0730 -40.5516 +/- 0.0706 2546.0676 +/- 0.0844]

[-150.9664 +/- 0.0704 -165.1104 +/- 0.0688 2461.2057 +/- 0.0775]

[-411.2700 +/- 0.0683 -142.4915 +/- 0.0658 2333.6878 +/- 0.1081]

[-323.9650 +/- 0.0845 19.4339 +/- 0.0823 2943.5988 +/- 0.0990]

[-114.2614 +/- 0.0769 205.9993 +/- 0.0756 2700.8274 +/- 0.0941]

[390.8584 +/- 0.0464 12.7366 +/- 0.0452 1610.6573 +/- 0.0812]

[-230.5125 +/- 0.0483 -261.9150 +/- 0.0473 1667.2606 +/- 0.0826]

[-202.6438 +/- 0.0985 124.9858 +/- 0.0949 3404.5728 +/- 0.1171]

[-614.0406 +/- 0.0671 50.4090 +/- 0.0646 2301.8594 +/- 0.1043]

[77.5874 +/- 0.0894 -197.1773 +/- 0.0864 3096.0320 +/- 0.1066]

[-500.7292 +/- 0.0750 -252.0236 +/- 0.0723 2565.8401 +/- 0.1219]

[-314.8590 +/- 0.0530 -110.7994 +/- 0.0511 1827.2515 +/- 0.0796]

[-113.6923 +/- 0.0650 532.9572 +/- 0.0624 2259.3928 +/- 0.0935]

[-525.1949 +/- 0.0775 43.6418 +/- 0.0749 2694.4301 +/- 0.0919]

[472.7202 +/- 0.0481 -363.1211 +/- 0.0482 1681.6073 +/- 0.0804]

[457.6801 +/- 0.0721 -607.4536 +/- 0.0701 2519.1301 +/- 0.0976]

[-211.0834 +/- 0.0587 482.2081 +/- 0.0561 2065.2462 +/- 0.0827]

[12.4805 +/- 0.0611 -498.5069 +/- 0.0594 2114.5059 +/- 0.0934]

[-640.6864 +/- 0.0708 227.3993 +/- 0.0692 2495.7829 +/- 0.1016]

[240.1102 +/- 0.0546 -285.4988 +/- 0.0539 1888.7218 +/- 0.0858]

[-61.9430 +/- 0.0902 461.4501 +/- 0.0875 3151.4026 +/- 0.1140]

[93.4313 +/- 0.0722 -676.7940 +/- 0.0714 2501.9871 +/- 0.1128]

[-147.6475 +/- 0.0579 242.4697 +/- 0.0556 1997.7613 +/- 0.0909]

Camera 2 Intrinsics

Focal length (pixels): [2472.2141 +/- 0.0926 2472.1423 +/- 0.0897]

Principal point (pixels):[1024.5042 +/- 0.0732 1024.3854 +/- 0.0689]

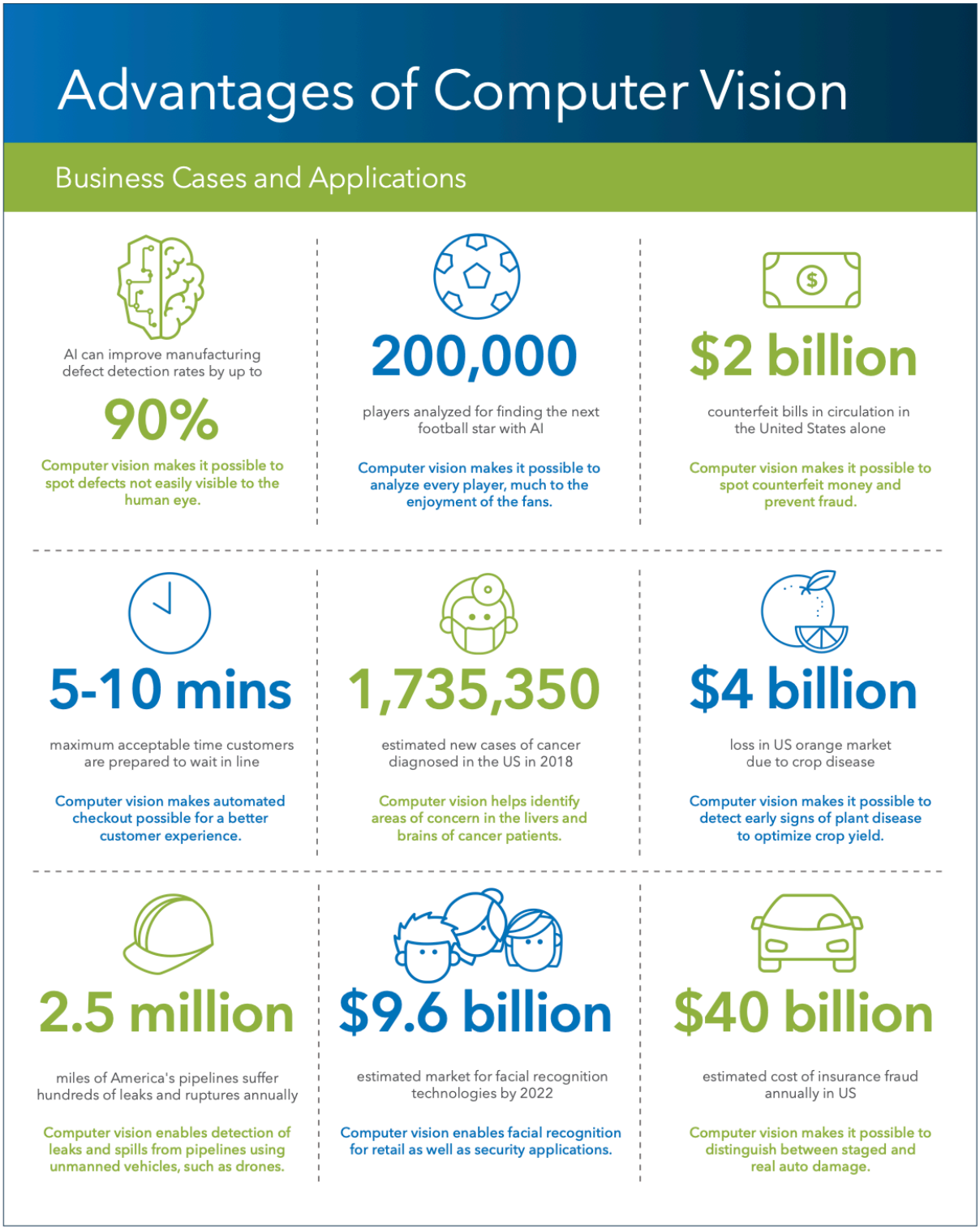
Radial distortion: [0.0001 +/- 0.0004 -0.0029 +/- 0.0034]

Position and Orientation of Camera 2 Relative to Camera 1

Rotation of camera 2: [-0.0001 +/- 0.0000 -0.0000 +/- 0.0000 -0.0000 +/- 0.0000]

Translation of camera 2 (millimetres):[-63.5032 +/- 0.0090 -0.0122 +/- 0.0089 -0.0401 +/- 0.0625]

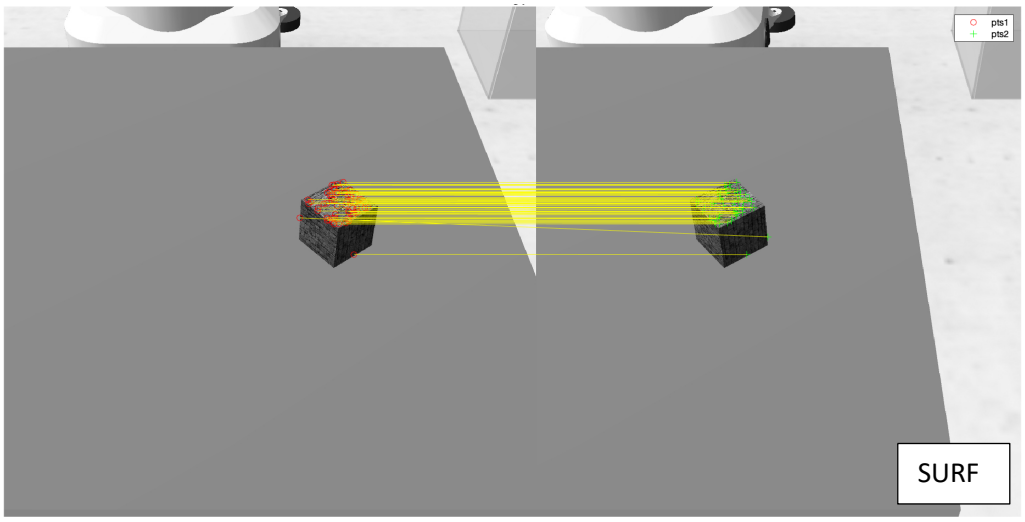
10 Appendix B

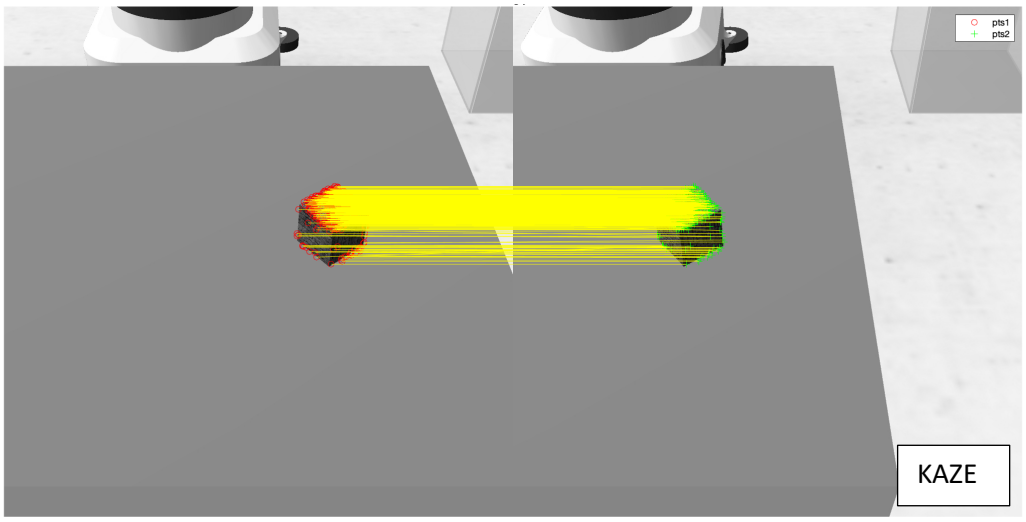


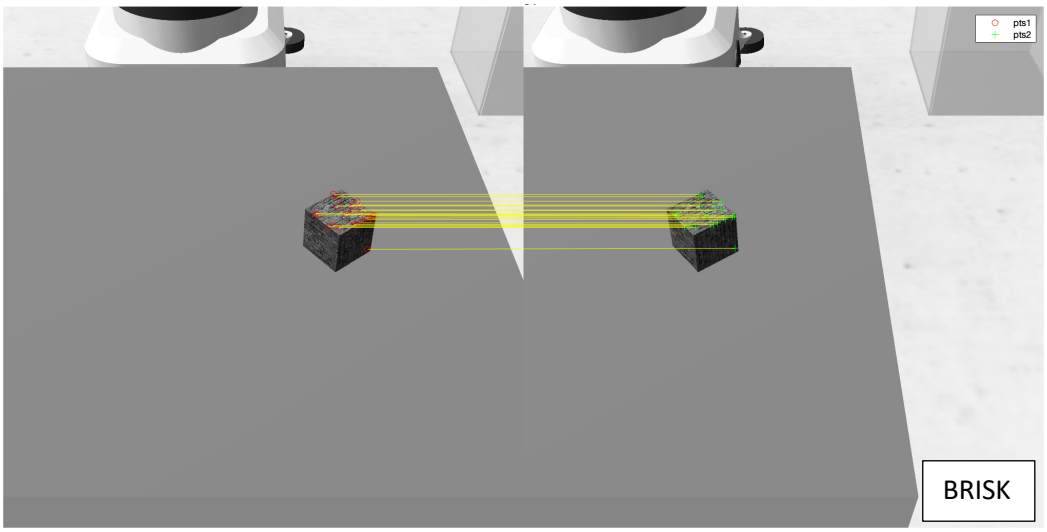
[9]

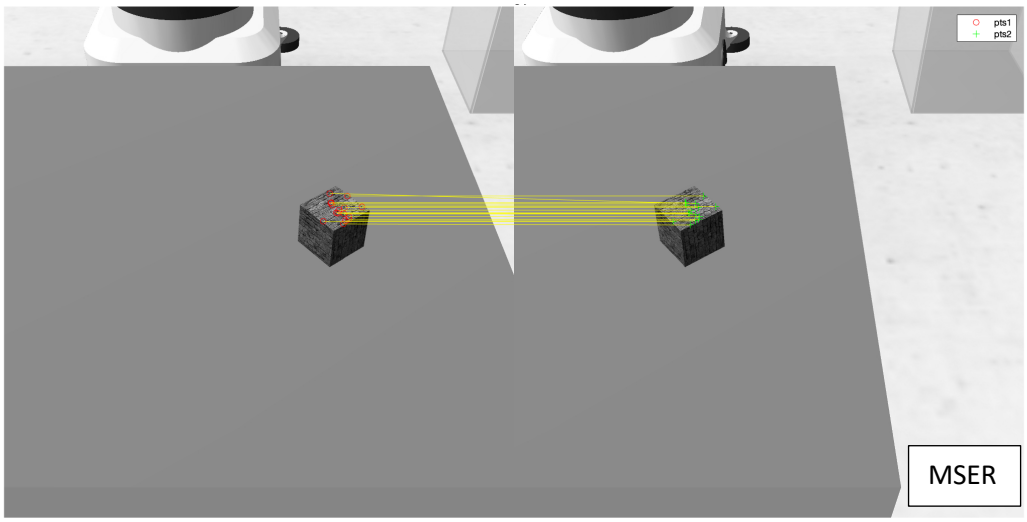
11 Appendix C

Figures showing the different feature detection and extraction methods applied to the detection algorithm and their corresponding matches.








pts1
pts2
MSER