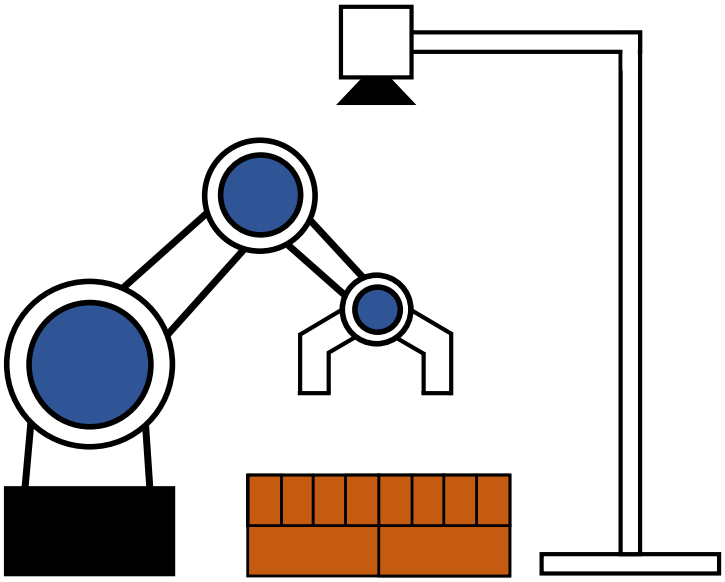
University of Southern Denmark



Bachelor Thesis

Deep Learning Aided Pose Estimation for Robotic Brick Picking

Mads Alber Kuhlmann-Jørgensen
makuh17@student.sdu.dk

Jakob Grøftehauge Rasmussen
jakra17@student.sdu.dk

Handover Date: 01-06-2020

Supervisor: Anders Glent Buch

Abstract

This project aims to examine different approaches to computer vision based object detection and orientation estimation, in the context of automating brick work. More specifically the depalletising task. For detecting bricks, four different solutions are proposed in the project, two methods based on traditional computer vision and two based on the deep learning detector RetinaNet. The traditional methods, which are considered as baselines, are based around edge maps found using the Canny edge detection algorithm. The first method uses the edge map for line extraction utilising the Hough transform. The other is a variation of the chamfer matching algorithm. RetinaNet does not offer orientation estimation, thus the two deep learning based methods aim to adapt RetinaNet to accommodate this. The first deep learning method adds an extra sub-network, for regressing the orientation angle, to RetinaNet. For the second method, the model RotinaNet is proposed, which features changes to allow prediction of tight fitting oriented bounding boxes. For evaluation and model training purposes, two different data sets are produced, and each split into a training and a validation partition. The deep learning methods are trained on the training sets, and all methods are evaluated on the validation sets. The proposed RotinaNet is found to yield the best performance out of the proposed methods, even showing a significant increase in detection performance over vanilla RetinaNet, for the given application.

Contents

1 Introduction

Bricks have traditionally played a large role in European architecture and construction. Denmark in particular has a proud heritage of brick usage in the construction industry, and they continue to be widely used. Not only are bricks pleasant to the eye, but they are also fully recyclable, which is in line with the increasing focus on sustainability. They do come with downsides however, as much manual work, adjustment and handling is required due to the large amount of bricks needed in the building process.

This project is carried out in collaboration with the company Odico A/S. Odico develops robotic solutions for the construction industry and are working on solutions to help automate the use and processing of bricks. Applications could vary from assisted assembly of precast brick panels, to fully autonomous bricklaying. Whatever the final application may be, the first step in the process is picking the bricks from the pallets they ship on. Therefore, this project will revolve around the application of autonomous robotic brick depalletising. The general idea is to have a pallet of bricks delivered somewhat arbitrarily in front of a robot which, aided by a computer vision system, will remove the bricks from the pallet one by one. Exactly what happens to the brick after removal is not of particular relevance for this project. The project is carried out as a computer vision study, where several solutions for detecting bricks are examined.

Throughout the report, references to literature will be indicated by square parentheses, such as [1], containing the associated index in the list of references. For some works, the index will be accompanied by page and/or chapter numbers. The source code developed is available from the associated GitHub repository [1].

2 Delimitation

As introduced, the main focus of the is project is to examine different solutions of detecting palletised bricks for picking. The actual picking, and associated robot control, is taken into consideration, but not explored. The bricks used for the project are provided by the brick manufacturer Strøjer Tegl and carry the default Danish brick dimensions, which are 228 mm × 108 mm × 54 mm [2]. The bricks ship stacked on a pallet as shown in figure 1. As seen in the figure, the bricks are stacked in five layers, of alternating direction, each of 16 bricks. To keep the application scope as wide as possible, Odico wishes for the solution to be able to pick a single brick at a time. This means that the computer vision solution must be able to detect the location of each individual brick, thus making it a multi-object detection problem. For the intended gripping procedure, the centre points and orientation around the vertical axis of the bricks are of particular importance. For this reason, it is essential that the developed solution can predict both orientation and location of each individual brick.

Odico has determined the robot and gripper to be considered, although only the latter is of significant importance to the scope of this project. The gripper is assumed to consist of two suction cups [3] with an outer diameter of 37 mm placed with the centre points 120 mm apart. Hardware related to the computer vision solution can be chosen to fit the need of the project. The Intel Realsense

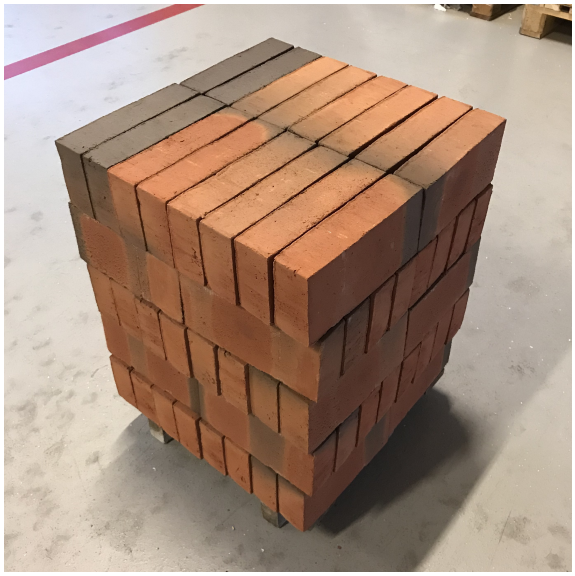


Figure 1: A pallet of bricks stacked as they are delivered from the factory.

D415 depth camera [4] has been selected as the sensor for the solution. There are several key motivations for this choice. The first one is that having depth information, in addition to regular colour information, will greatly ease the process finding the spatial location of the bricks, based on the simple pinhole camera model [5, chap. 2.1]. Another benefit is access to an extensive library for interfacing the camera. For the scope of the project, it is assumed that the factory calibration of the camera is sufficient. The associated intrinsics are easily available through the camera API.

The field of computer vision has existed for more than 50 years [5, chap. 1], and several different methods for object detection have been developed throughout this time. Recently, deep learning methods, based on deep convolutional neural networks, have dominated the research scene as hardware has improved and data become abundant. For this reason, it is desired to explore the potential of deep learning in this project. It is recognised however, that due to the relative simplicity of the object to be detected, traditional computer vision methods may also yield good results. In this paper, traditional methods refers to methods commonly used for detection before deep learning became dominant. The project will explore both deep learning and traditional computer vision based approaches to object detection, treating the traditional vision based methods as a baseline for comparison.

Based on training data, deep convolutional networks can be trained to extract relevant features from input images. This allows for detecting and classifying objects in the images. Several approaches have been proposed for multi-object detection. These are typically divided into two categories, one-stage detectors and two-stage detectors. Two-stage detectors divide the detection and classification process into two steps. First regions of interest are found, and based on these the classification and detection regression is performed. One-stage detectors do not utilise this division. Two stage detectors are typically more accurate, however one-stage detectors are usually faster [6]. For this project it has been decided to base the deep learning solutions on a one-stage network structure referred to as RetinaNet proposed in [7]. More specifically, a keras implementation of the

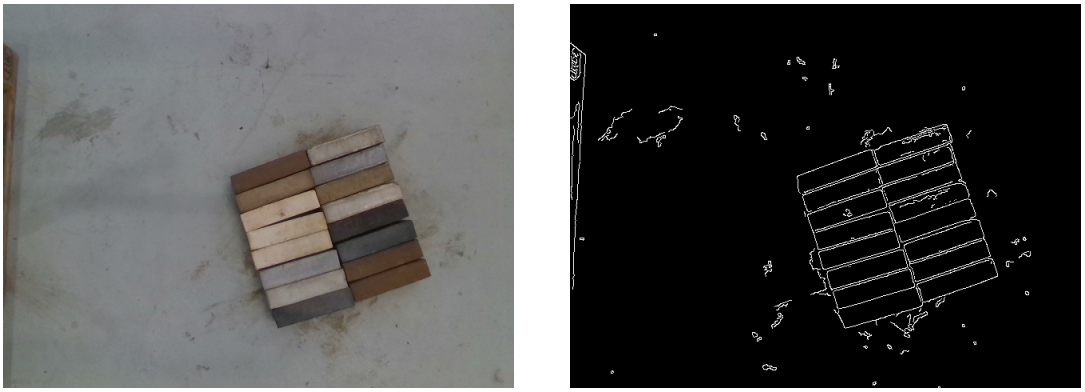
RetinaNet[8], developed by the Dutch company Fizyr, will be used as a starting point. RetinaNet is a detector which predicts axis aligned bounding boxes, and the default structure does not offer a way to predict the orientation of an object. For this reason adaptions accommodating this shortcoming will have to be developed.

Traditional computer vision methods usually involves a sequence of processing steps to extract relevant features from the input images. Based in these features, post processing algorithms should be able to determine where objects are to be detected. This process implies a lot of manual tuning, and places a lot of responsibility on the designer in regards to extracting the correct features for the application at hand. One example of extracting features is through the use of Haar classifiers. Haar-like features are found by subtracting rectangular regions of an image, yielding either a high or low response depending on the underlying image. Each Haar-like feature can be considered a weak classifier, however they can be combined in a cascade to yield a strong classifier. This is proposed in the Viola-Jones paper [9] as method for face detection. The method works well for faces, as they contain a lot of distinct features, such as eyes, brows, etc. The bricks considered in this project have limited amounts of distinct features and therefore the method has been deemed ill-fitting and will not be examined further.

Another traditional method for detecting objects is by the use of template matching, where a template is searched for within an image. This is done by determining regions of minimal pixelwise deviation between the template and the image. The template usually contains a perfect image of the object. The vanilla template matching method, as described in [5, chap. 8.1] searches for the template within an RGB or a grayscale image. This though poses a problem if the object searched for can have different nuances or colours, as it is then required to have a template for each possible colour and nuance, in order for the algorithm to work as intended. Additionally, the possibility of changing lighting conditions may exaggerate this further. This raises the execution time significantly. For this reason, the vanilla template matching algorithm will not be considered for this project.

A way to overcome the issues of multiple colours and nuances, is to consider an edge map of the image instead of the colour image itself. A common detection method utilising the edge map is what is referred to as chamfer matching. Chamfer matching is a variation of template matching, searching for matches between the edge image and an edge template. Another common use of the edge map is to extract lines from an image. The Hough transform [5, chap. 6.3] is commonly used for this purpose. Due to the simple shape of the bricks, it is thought that it may be possible to combine multiple lines and determine the positions of the bricks based on the lines defined by their sides. It is expected that the methods utilising the edge map will perform well, as preliminary test images have shown promising and distinct edge maps. This is visible from figure 2b.

For the task of detection and orientation prediction, two traditional detection methods will be examined, namely chamfer matching and the line based detection algorithm previously described. These two methods will be used as baseline for comparing results obtained by deep learning solutions based on RetinaNet. To test the performance of the solutions, data is required. Odico does not have any data available, and thus the responsibility of producing the needed data lies within the scope of the project. Although the project is carried out as a computer vision study, and no phys-



(a) Sample Image. (b) Edge Map.

Figure 2: Sample image and the corresponding edge map of bricks.

ical tests will be conducted, requirements, which the developed solutions can be checked against, have been formulated based on the desired application. Requirements on position precision are based on the expected gripper design, and timing requirements are based on the acknowledgement that the robot will move physical objects at least several centimetres between pickups.

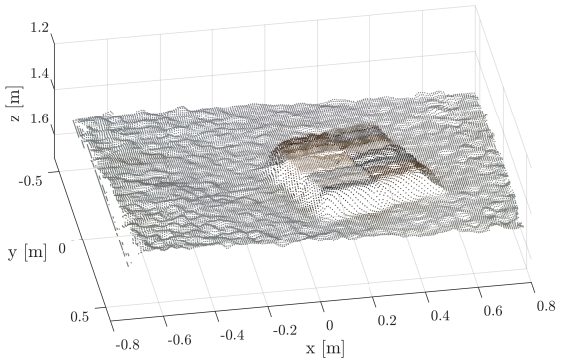
- Computer vision detector solutions have to be developed, from which the centre point and orientation of the bricks can be found.
- One detection cycle should be able to be executed within 1 s on suitable hardware.
- The error on the angle estimate should not contribute to mispositioning of the gripper suction cup centres by more than ±4 mm on average. This is equivalent to an orientation estimate error of 3.8°.
- The predicted brick centre point should not contribute to mispositioning of the suction cups of the gripper by more than ±4 mm on average.

It has been chosen to use openCV for C++ to develope the baseline methods. For building the deep learning solution, Python, utilising Tensorflow and Keras, has been chosen.

3 Data

The Realsense D415 has been selected as the camera to be used for this project, and thus the data considered will be obtained from this camera. The camera supplies a colour image as well as depth map, which can be converted into a point cloud with colour information, as illustrated in figure 3. Using API-instructions [10], the depth map can be aligned to the colour image, thus indicating the depth at each pixel in the colour image. The detection of bricks will merely concern the colour images, although the depth data is useful when determining the spatial position of the objects.

(a) Colour image from Realsense D415.



(b) Colour and depth data from Realsense D415, visualised as a point cloud.

Figure 3: Visualisation of the data available from the Intel Realsense D415 camera. The density of the depth data has been reduced for illustrative purposes.

As introduced, producing data sets, to train and test different detection algorithms, is an important part of the project. To get realistic and representative data, a camera setup, sketched in figure 4, has been constructed replicating what is expected to be used in a finished picking solution. The camera is mounted statically above the bricks, oriented to point straight down, giving a clear top view of the bricks, as is evident from figure 3a. The height has been adjusted to yield a clear view of the bricks and allow clearance for a robot. Due to the temporary nature of the setup, a pallet will be visible in the left side images. The pallet constitutes the base of the camera holding rig.

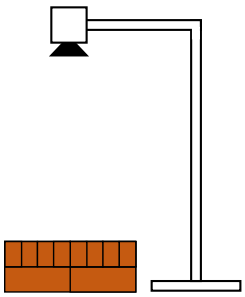
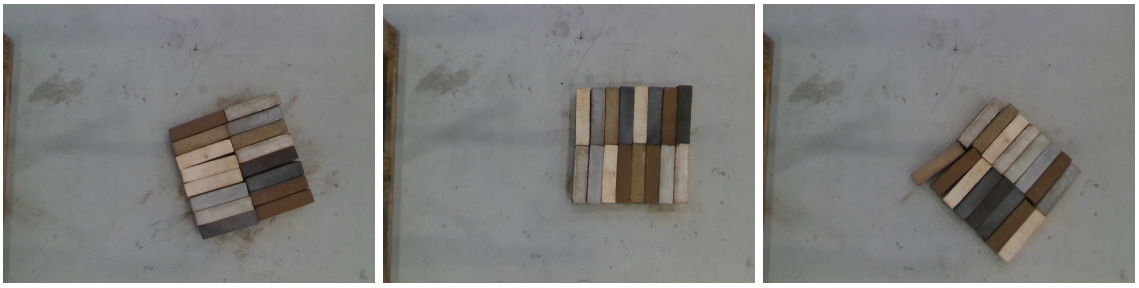


Figure 4: Sketch of the camera setup used to collect image data.

Two data sets, with different characteristics, have been produced and annotated in the process of this project. The first data set, intended as a proof of concept, contains images of 16 bricks of varying colours. The bricks are placed in a single layer directly on the floor, organised in a pattern replicating that of palletised bricks. The bricks are generally placed in the same spot and aligned neatly relative to each other, although some variations have been incorporated. The camera has been adjusted to be 1.52 m above the floor. Figure 5 shows examples from the first set, henceforth referred to as data set 1. Data set 1 contains a total of 192 images at a resolution of 640×480 pixels.

A second data set, referred to as data set 2, has been created, which more accurately depicts the practical scenario. In this data set, bricks are stacked on a pallet exactly as received from the



(a) (b) (c)

Figure 5: Image examples from data set 1.

manufacturer as shown in figure 1. Between each image a brick is removed from the stack to emulate the depalletising process. This means that this data set is distinct from data set 1 in the fact that it features bricks at different distances from the camera, and some bricks may be partially occluded by the layer above. When annotating the images, it has been chosen not to annotate bricks which are partially occluded as these cannot be safely picked. Furthermore, because of the size of the data set, the baseline method described in section 4.3 has been utilised for assisted[1] annotation of the training partition. The bricks are of the type 'Goodwood', and appear in various nuances of red and brown. For this data set, the camera is placed 1.67 m above the floor. Data set 2 contains 495 images at a resolution of 1280×720 pixels, a small selection of which are shown in figure 6.

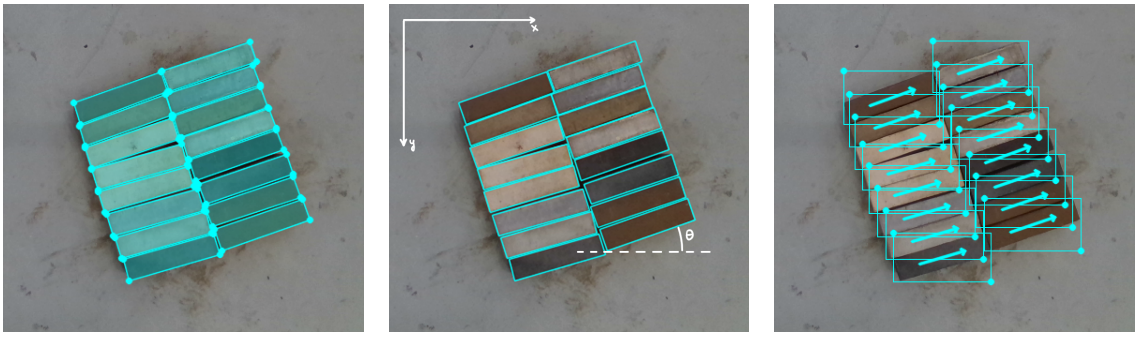
(a) (b) (c)

Figure 6: Image examples from data set 2.

Each data set is randomly divided into a training set and a validation set, using an 80/20 split. The same split is used throughout the project. During training of the deep learning network, only the training set will be used, as the validation set is reserved for performance validation. This is done in an attempt to prevent biasing the performance results, and in an attempt to detect potential overfitting. Since the validation set is typically used for determining hyperparameters, normally a third division, the test set, is used for measuring performance. Due to the limited amount of data available for the project, the test set has been omitted, thus the validation sets will be used for performance evaluation and hyperparameter tuning.

To evaluate the performance of the detectors, and to train the deep learning network, ground truths, or targets for the predictions to be checked against, have to be established. These targets

[1]Initial annotations are created using the baseline method, followed by manual inspection and correction.



(a) Polygons. (b) Rotated rectangles. (c) Axis aligned.

Figure 7: Illustration of annotation and target representations used in the project. 7a shows polygons defined by vertices, 7b shows the minimum area rectangle spanning the vertices, and 7c shows the axis aligned bounding rectangles spanning the minimum area rectangle.

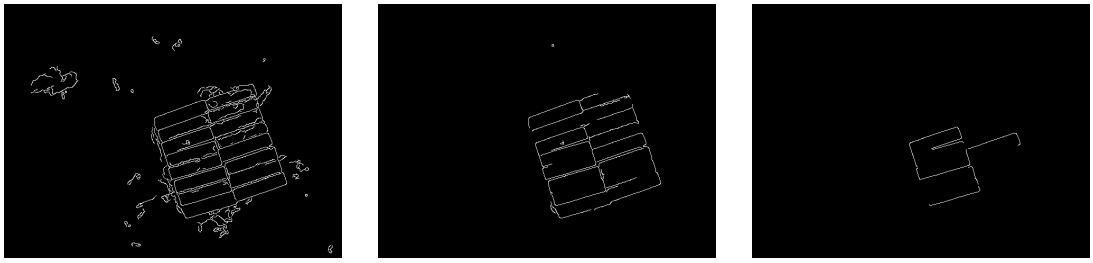
are based on annotations that indicate where bricks are located. Annotations are created using the python package "labelMe" [11], and are defined as polygons covering the brick, spanned by the edges of the brick as illustrated in figure 7a. These annotations are converted to targets for evaluation and training. Two target representations are used in this project, the rotated rectangle representation in figure 7b, and the axis aligned box with angle as seen in figure 7c. In openCV, the rotated rectangle is defined by a centre point, a width, a height and an angle. It is created from the annotations by finding the minimum area rectangle containing the annotation points. The axis aligned box, which is the common target definition for detectors, is defined by two diagonal corner points. These are found as the minimum and maximum x- and y-values of the corner points of the rotated rectangle described before. Each axis aligned box also has an associated angle describing the orientation of the brick inside. The coordinate system definition from openCV, shown in figure 7b has been adopted. Since it does not makes sense to define the direction of a brick, the orientation will be defined as a an angle in the range $\pm 90°$, i.e. the right half of the unit circle.

The 3D-coordinates of the detected objects in relation to the camera can be found using equation 1. Where f_x and f_y are the focal length in pixels with respect to the y- and x-axis. The values x^* and y^* are the distance from the detected point to the image centre point in pixels. The parameter d denotes the depth at the given pixel location, given by the collected depth data.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{x^*}{f_x} \\ \frac{y^*}{f_y} \\ 1 \end{bmatrix} \cdot d \tag{1}$$

4 Baseline Methods

As introduced in section 2, two baseline methods have been developed. This section will provide a detailed description of the line based detection and chamfer matching algorithms. Both methods searches for features in the edge image, thus it has been decided to use a similar preprocessing step for both methods.



(a) Low: 12 High: 53. (b) Low: 51 High: 154. (c) Low: 204 High: 405.

Figure 8: Illustration of the effect of different threshold values in the Canny edge detection algorithm. The edge maps are based on the image shown in figure 5a.

4.1 Preprocessing

To produce the edge map needed for both baseline methods, it has been chosen to use the Canny edge detection algorithm. The Canny algorithm is well-suited since it aims to keep the distance between actual and located edges minimal, i.e. located edges are found at the centre of the edge through non-maximum suppression, which is important for accurate detections. Furthermore, it utilises hysteresis, through double thresholds, to produce strong connected edges. Since the images are originally in colour, the first step is to convert these to grayscale. After conversion to grayscale, the original algorithm dictates that a Gaussian filter should be applied before the image is processed further [5, pp. 95-96]. The Gaussian filter is used in the prepossessing step for the chamfer matching method. For the line based method however, it was discovered, through a series of test, that the use of a median filter yielded a slightly higher performance, therefore it was adopted for this algorithm. In order for the baseline algorithms to work properly the hysteresis thresholds used in the Canny algorithm have to be tuned, image 8a, 8b and 8c illustrates how the thresholds effect the output of the algorithm. It is key that the noise is suppressed while not eliminating the edges originating from the bricks. It is expected that the optimal thresholds are influenced by the background as wells as lighting conditions, furthermore increasing the complexity of finding an optimal threshold. To minimise the number of irrelevant edges in the images the stationary pallet, which can be observed to the left in the images shown in figures 5 and 6, is masked out for all images.

4.2 Line Based Detection

The first baseline method is based around extraction of lines from the edge image. Due to the rectangular nature of the bricks, it is presumed that these can be found based on the four lines marking the sides. This method is split into two main parts, line extraction, utilising the Hough transform, and detection based on the extracted lines. The general idea of this method is based around the assumption that the bricks are presented in a pattern similar to a full layer, as seen in figure 1. This was chosen because it was found that the short edges of singular bricks generally did not have enough edge pixels to consistently yield distinct lines.

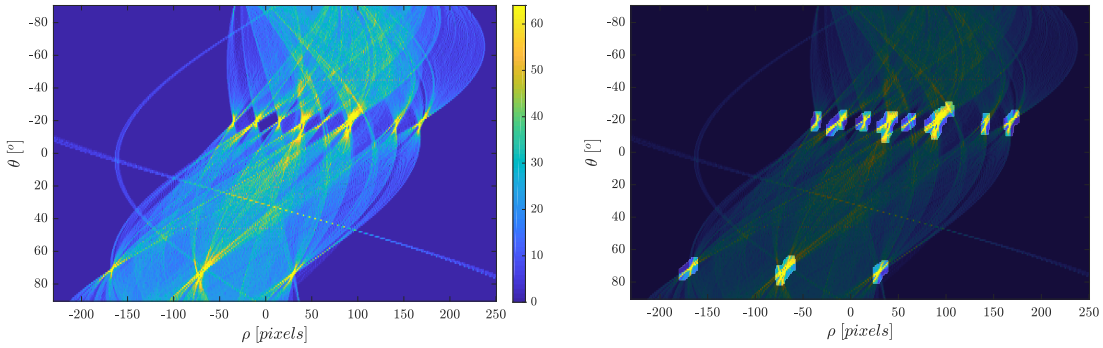
The Hough transform is a widely used technique for extracting lines from an edge image. OpenCV offers an implementation of this algorithm, but the options and information available were deemed

```
edge map ← preprocess image
H ← compute Hough space (ρ, θ).
    M ← normalize and threshold H
        split and shift M
            find clusters in M
            revert shift on cluster points
            lines ← maximum points in H within each cluster
```

Algorithm 1: The algorithm used for extraction lines from edge images.

too limited. For this reason, and since the algorithm is fairly straightforward, it was decided to create a custom implementation. The line extraction algorithm, shown in algorithm 1, works by first computing the Hough space. The Hough space contains an entry for each combination of parameters representing lines that lie within the image. In order to accommodate vertical lines the lines are represented using polar coordinates as seen in equation 2 [5, chap. 6.3.1], where θ represents the angle between the line and image x-axis, and ρ is the orthogonal distance from the centre-point of the image to the line. x and y represent coordinates of edge pixels relative to the image centre. The algorithm iterates through all edge pixels in the edge map. For each edge pixel all possible lines, in a discrete range of θ and ρ, intersecting the point are found and the entry in the Hough space associated with each parameter combination is upvoted. When all edge pixels have been evaluated the resulting Hough space appears as visualised in figure 9a. The parameter combinations in the Hough space which have gained the most votes represent the most distinct lines, i.e. the ones that intersect the most edge pixels.

$$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta) \tag{2}$$



(a) Hough space.

(b) Masked hough space.

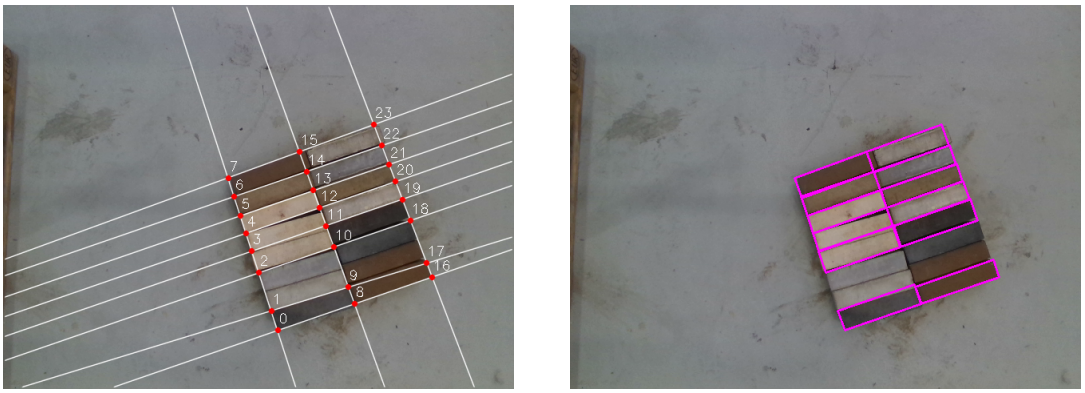
Figure 9: Illustration of the Hough space and masked Hough space. The space has been cropped for visualisation purposes. The results are based on the image shown in figure 5a

To isolate the most distinct lines, a non-maximum suppression, NMS, algorithm is proposed, in which the Hough space is normalised between zero and one, and a binary mask, removing all entries

below the threshold of 0.7, is created. The binary mask is then dilated in order to connect closely located maximum regions. The result of this operation can be seen in figure 9b. Each cluster in the masked Hough space represents a distinct line, whose parameters are determined by the most upvoted parameter combination within the cluster. The NMS procedure used has a flaw in the case of lines that are close to the limits in the parameter θ. These lines can be very similar in the edge image, but are far apart in the Hough space. This means that clusters could be split into two at this exact point. To overcome this issue, if clusters are found near limits, the Hough space is shifted before the dilation.

divide lines into two perpendicular groups, A and B
    sort A and B **by** distance from image origin
C $\leftarrow$ create intersection matrix
for each line, i **in** group A
    for each line, j **in** group B
        $C(i, j) \leftarrow$ intersection between i and j
evaluate each bounding box defined by all 2×2 neighbour point combinations in C

Algorithm 2: Algorithm used for finding predictions based on extracted lines.

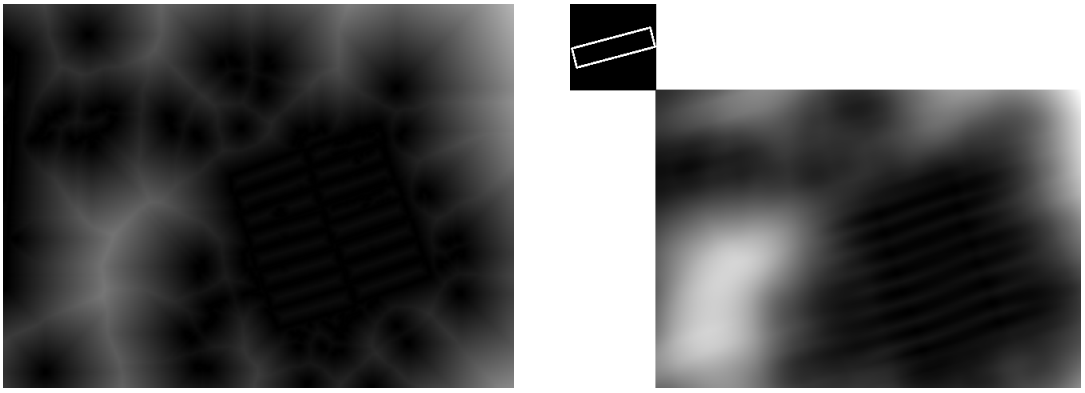


(a) Extracted lines and intersection points.

(b) Predicted Rotated Rectangles.

Figure 10: Illustration of the extracted lines and their intersection points, and the associated predictions.

When the most distinct lines, within the edge image have been located, rectangle predictions are made based on these, as shown in algorithm 2. The predictions are made based on the assumption that that line intersections will mark corners of the bricks. First the lines found are divided into two groups which are roughly perpendicular to each other. Afterwards each group of lines is sorted according to the value of the parameter ρ. Then the first group of lines is iterated through and the intersection with all the lines in the other group are found, yielding the red points seen in figure 10a. These points have a predictable pattern due the how the lines have been sorted. All the points are then stored in a matrix with dimensions equal to the size of each group of lines, keeping the grid pattern seen in figure 10a. Four adjacent points can be fitted with a minimum area



(a) Distance map.

(b) Matching space and associated template.

Figure 11: Distance map found based on figure 5a and a single matching space yielded by a convolution of the distance map with the associated template.

rectangle to indicate a potential prediction. All potential predictions are checked against expected parameters for a brick, to eliminate malformed candidates. The expected parameters are provided by the user and determined for each data set, by examining the targets.

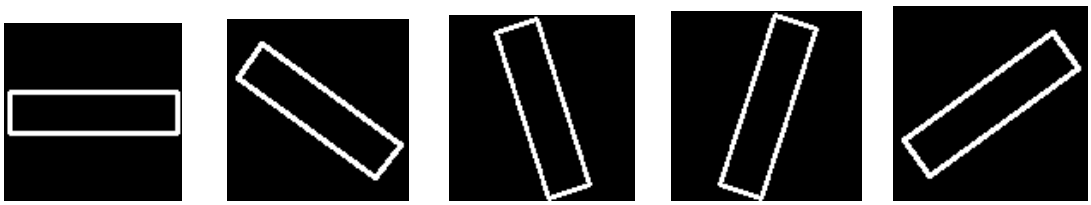
4.3 Chamfer Matching

Chamfer matching utilises a distance transform on the edge map. For each pixel in the edge map, the distance transform returns an estimated distance to the nearest edge pixel [12], yielding a distance map as illustrated in figure 11a. The distance map can be convolved with an edge template resulting in a matching space illustrated in figure 11b. Due to the convolution, the matching space will be reduced in size compared to the distance map, as no padding is used. Each pixel in the matching space represents a score of how well the edge template matches the edge map at the corresponding location. A score of zero means a perfect match and the higher the score, the worse of a match. By finding the local minimums in the matching space, the best match location candidates, for that template can be extracted. Since some local minima may not be a very good match at all, an upper threshold is established to only accept candidates if the matching score is lower [5, chap. 8.2]. The simplest form of chamfer matching will only detect a single template, which means that the method generalises very poorly if the object may vary in rotation and scale. Therefore the method has been adapted to accommodate these issues. Algorithm 3 describes the general idea of the method developed for this project.

As with the simple chamfer matching algorithm, the first two steps involve obtaining an edge map, and performing distance transform on this. However, to accommodate multiple scales and rotations, several templates will have to be used. Traditionally the edge templates are drawn from a specifically prepared image of the object, however due to the simple shape of the bricks, it is possible to easily generate many templates based on a discrete range of rotations and scales as seen at line 5 algorithm 3. It has been decided to use 60 angle steps and 5 scale steps, yielding a total of 300 templates. Examples of the used templates can be seen in figure 12. The templates are modelled based on expected brick dimension ranges supplied by the user. The appropriate

1 edge map ← **preprocess** image
2 distance map ← **distance transform** on edge map
3 **for** θ **in** angles
4 **for** S **in** scales
5 template ← rectangle at angle θ, scale S
6 matching space ← **convolve** template w. distance map
7 match candidates $\overset{+}{\leftarrow}$ local minimum points $<$ threshold
8 matches ← **suppress** overlapping match candidates
9 **return** matches

Algorithm 3: The proposed chamfer matching algorithm.



(a) (b) (c) (d) (e)

Figure 12: Examples of templates at different scales and rotations.

ratios have been found based on the data sets. For the convolution step in algorithm 3 at line 6, the openCV method *matchTemplate()* [12] has been used. This openCV implementation allows for different matching modes. Since templates of smaller scale will contain fewer edge pixels, these will naturally yield lower matching scores. To accommodate this, the normed matching mode, 'CCORR_NORMED', could be used. However, in this project, a custom mode utilising regular convolution and a weighted template image is proposed, and found to yield a better performance. Each template is weighted based on the total number of edge pixels present in that template, as described by equation 3, where T is the original template, and N is the total number of edge pixels in the template. This makes it possible to more fairly compare matching scores across different scales and rotations.

$$T_{weighted} = \frac{100}{N} \cdot T \tag{3}$$

For each template, a convolution is performed and local minima are found, as described above, yielding potential match candidates. Each candidate is defined by the associated matching score and template rectangle at the match location. The list of match candidates will likely contain a fair bit of similar and overlapping candidates, as shown in figure 13. To remove these, "non-minima suppresion" is applied, where all candidates within the list are compared and if the intersection-over-union, IoU, between two candidates exceeds the threshold of 0.15, the candidate with the highest matching score is removed from the list. Because of the template creation procedure, the predictions are represented as rotated rectangles, however for compatibility, these can be converted to the axis aligned representation by fitting an axis aligned box to the rotated rectangle.

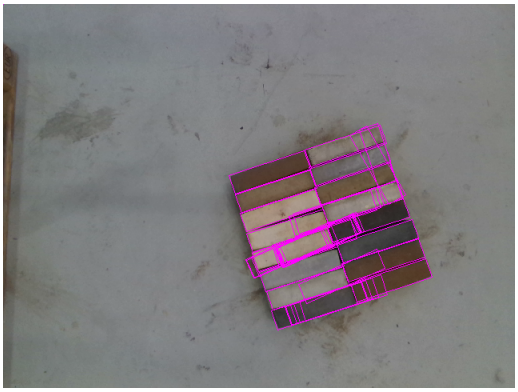


Figure 13: Match candidates before non-minima suppression has been applied.

5 Deep Learning Methods

The deep learning based methods evaluated in this report will be based on the object detector RetinaNet introduced in the paper "Focal Loss for Dense Object Detection" [7]. This section will summarise the key points of the implementation used, mostly referencing the aforementioned paper.

The motivation for the creation of RetinaNet is the introduction of the classification loss function referred to as Focal Loss. Focal loss, addresses the problem of extreme class imbalance between foreground and background classes, and reduces the importance of well classified examples. The focal loss is defined as seen in equation 4 and can be extended to multi-class, although only single-class will be considered in this project. The parameter p_t is defined according to equation 5, where y denotes target class and p specifies the model's estimated probability for the class with label $y = 1$. γ is referred to as the focusing parameter, and determines how the loss should be reduced for relatively well classified examples. The greater the focusing parameter, the more rapid the loss for relatively well classified examples will decrease. α_t is used for addressing the class imbalance of foreground and background examples, and is defined as shown in equation 6. The α parameter is usually determined by the inverse class frequency but can also be tuned using cross-validation.

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{4}$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases} \tag{5}$$

$$\alpha_t = \begin{cases} \alpha & \text{if } y = 1 \\ 1 - \alpha & \text{otherwise} \end{cases} \tag{6}$$

RetinaNet is a single unified network, consisting of a backbone, based on a Feature Pyramid Network, FPN, and two task-specific subnetworks. The backbone is responsible for computing convolutional feature maps over an input image. One subnet is used for classification and the other is used for regression of the object bounding box. The deep learning based methods introduced in this report will all utilise ResNet-50 as part of the backbone. An illustration of the backbone architecture can be seen in figure 14. Here the layers C3-C5 represent the output from the respective residual stages [13]. The feature pyramid is created by performing a convolution on C5 and thereby

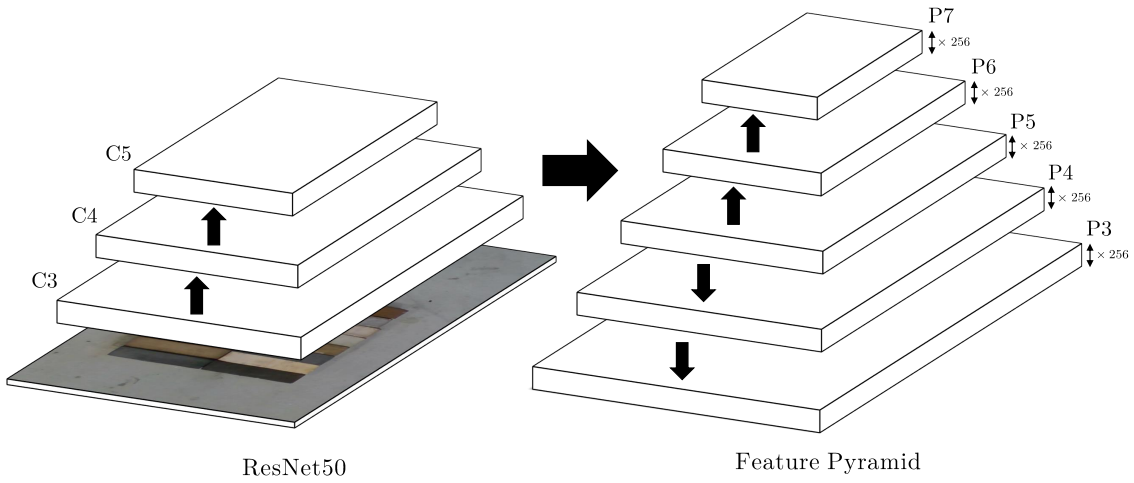


Figure 14: Illustration of the backbone used for this project, consisting of a feature pyramid built upon ResNet50.

creating P5. Layer P4 and P3 are created with a top-down pathway and lateral connections combining the corresponding residual layer in ResNet with a scaled up-version of the above layer in the feature pyramid. Layer P6 is created by convolving P5 and P7 is created by applying the ReLU function to P6 followed by a convolution. All layers in the feature pyramid have 256 channels.

The concept referred to as anchors is vital to the workings of RetinaNet. Anchors are a set of predefined region proposals, based on which prediction is carried out. The default scale of the anchors are related to the feature pyramid level with which they are associated. The smallest anchors are associated with layer P3 and have an area of 32^2 pixels, with respect to the input image. For each above layer the area increases by a factor of 4, yielding areas of 32^2 to 512^2, on pyramid levels P3 to P7, respectively. Anchor positions are tiled across the image in a grid with strides 8 to 128 at P3 to P7 respectively, meaning that each anchor position relates to a "pixel location" in the corresponding feature pyramid layer. At each anchor position, multiple anchors are created, based on a range of aspect ratios and scaling factors. Per default the three aspect ratios of $\{1:2, 1:1, 2:1\}$ are used, as well as the three scaling factors of $\{2^0, 2^{1/3}, 2^{2/3}\}$, resulting in nine anchors per position. Figure 15 visualises the nine different anchors, created at different positions on each pyramid level. The different colours represent different feature pyramid levels. For the sake of visualisation, the anchors are only drawn at a single position but are located in a grid as described previously. Each anchor will be the source of a classification and regression prediction. During training each anchor is assigned a one-hot K length vector of classification targets and a length 4 vector of box regression targets. K represent the number of classes which in the case of this project is one.

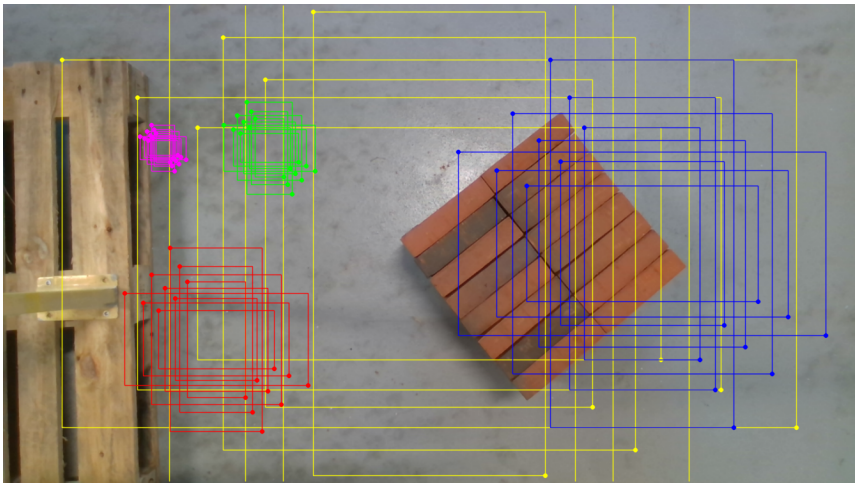


Figure 15: Visualisation of anchors based on the default anchor configuration. Different colours refer to different layers in the feature pyramid. For visualisation purposes, anchors are only drawn at one anchor location on each level.

Both of the sub-networks are built around the structure illustrated in figure 16. Both are fully convolutional networks consisting of five layers. Each of the first four layers applies 256 filters with a kernel size of 3×3, followed by a ReLU activation function. The last layer has $N \cdot A$ filters, where A denotes the number of anchors at each anchor position, in this case nine, and N depends on the sub-network. For the classification sub-network, N is defined by the number of classes, in this case one. For the regression sub-network, N refers to the number of values to be regressed, in this case two corner points meaning four values. The activation function on the last convolutional layer also depends on the sub-network. The output is reshaped for future use.

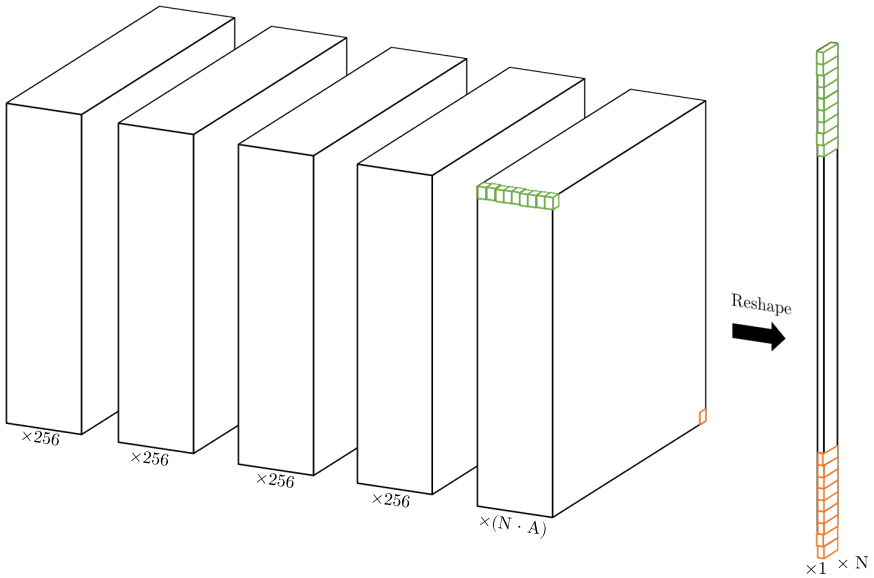


Figure 16: Illustration of the general structure of sub-networks in RetinaNet models introduced in this project.

The classification sub-network predicts a confidence score of each of the K object classes being present at each anchor. The sub-network follows the structure described above. The sub-network is attached at each level in the Feature Pyramid, and weights are shared across all levels. The last convolutional layer uses a sigmoid activation function. After output from each feature pyramid level is obtained the reshaped outputs are stacked on top of each other.

The regression network predicts the object bounding boxes. The sub-network follows the structure introduced in figure 16 and is attached at each feature pyramid level, sharing weights across all levels. The last convolutional layer uses a linear activation function. The box regression network is class-agnostic, meaning box regression is independent of the class prediction. The box regression network does not use the focal loss for training, instead Smooth L_1 [14], described by equation 7, is used as default. Only foreground anchors, described below, are included in the loss, as box regression for anchors not containing objects is irrelevant.

$$\text{Smooth } L_1(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{7}$$

When training RetinaNet, it is crucial to indicate which anchors are foreground, meaning they contain objects of interest, and which are background. Anchors are labelled as foreground and assigned to a target object if the IoU between the two is above 0.5. The anchors are labelled as background if their IoU is between 0 and 0.4. Unassigned anchors with a IoU between 0.4 and 0.5 are ignored during training. If an anchor is assigned to a target object the corresponding entry in the classification target vector is set to one, otherwise it is left at zero. In this report, targets generally refer to the absolute positions in the image coordinate system. However, because of the predefined proposals introduced by anchors, the internal targets generated for box regression are offsets from the points of the specific anchor to the points of the associated absolute target. Furthermore, the values are normalised with respect to the width and height of the anchor. The predictions are converted back to absolute coordinates at inference. Furthermore, at inference, non-maximum suppression is applied such that if box predictions have a higher IoU than 0.5, only the box with the highest confidence in classification is kept.

5.1 Modifications

Section 5 introduced the CNN-architecture of RetinaNet as described in [7], which henceforth will be referred to as 'Vanilla RetinaNet'. This section will describe changes and modifications introduced in this project, which adapts the network to the case at hand, and attempts to improve the performance. Ultimately, this has led to two distinct network designs which build upon Vanilla RetinaNet. The first, which will be referred to as "RetinaNet with Angle" or "RWA", adds orientation prediction to Vanilla RetinaNet. The second, which has been named "RotinaNet", is a modification which allows prediction of rotated rectangles. The required modifications will be explained in the following sections.

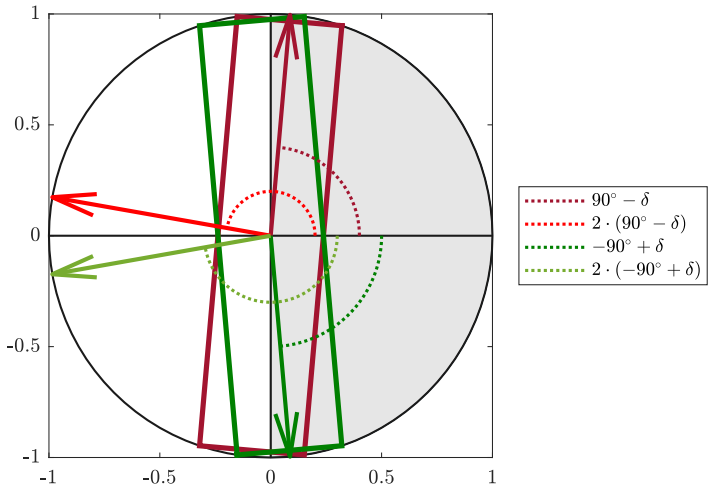


Figure 17: Visualisation of the singularity arising from the brick orientation only being defined in the right half of the unit circle, and different ways to represent the angle in an attempt to accommodate the issue. The rectangles represent bricks at each side of the singularity. The darker arrows visualise the Cartesian Half representation and the light arrows represent the Cartesian Full representation.

5.1.1 RetinaNet with Angle

Vanilla RetinaNet does not fulfil the requirements of the application, as it does not offer orientation estimation. RetinaNet with Angle aims to rectify this issue while keeping deviations from Vanilla RetinaNet minimal. The way RetinaNet is structured, the orientation estimation can easily be added by introducing an extra sub-network to regress the angle of the object. Thus the key addition to RWA, is an added regression sub-network. To keep things simple, it has been chosen to use the same sub-network structure as implemented for the box regression sub-network introduced previously, shown in figure 16. The only difference is the number of output values per anchor, which will be either 1 or 2, instead of 4, depending on the way the orientation angle is represented.

There are several ways which the orientation of the bricks can be represented. Since it does not make sense to define a direction of the bricks, their orientation will be within a range of $\pm 90°$. As is illustrated by figure 17, bricks with orientations of $\theta_1 = 90° - \delta$ and $\theta_2 = -90° + \delta$ will look very similar in the image. Regressing the orientation as a single scalar value may not be ideal, as there exists a form of singularity at the aforementioned points, where a very slight change in the orientation of the brick will result in a very large difference in the target value.

Instead of representing the orientation as a single scalar value, the angle, θ, can be mapped to a Cartesian representation as $\theta_C = \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix}^T$. One would expect this representation to perform better, since the singularity is avoided in the first element, as $\cos(90°) = \cos(-90°) = 0$. The issue still exists for the second element however. To completely avoid the potential singularity, the angle can be mapped to a Cartesian representation spanning the entire unit circle by multiplying the angle by two. This leads to the three representations presented in table 1. For the last two, the inverse mapping will obviously have to be performed at inference time.

Representation	Mapping	Range
Regular Angle	$\theta_R = \theta$	$[-90°; 90°]$
Cartesian Half	$\theta_{CH} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$	$\begin{pmatrix} [0;1] \\ [-1;1] \end{pmatrix}$
Cartesian Full	$\theta_{CF} = \begin{pmatrix} \cos 2\theta \\ \sin 2\theta \end{pmatrix}$	$\begin{pmatrix} [-1;1] \\ [-1;1] \end{pmatrix}$

Table 1: The tested representations of the orientation angle.

A series of tests have been conducted to determine which representation to use going forward. These tests also examine the effects of three different loss functions for the regression; L_1, L_2 and Smooth L_1, thus giving a total of nine different combinations. For each combination six models have been trained. The mean and standard deviation of the test performance across these six are reported. These tests are performed on data set 1, meaning the models are trained on the 162 images in the training partition and evaluated on the 40 images in the validation partition. Each model is trained for 25 epochs with a batch size of 1, using visual data augmentation, but no transformational augmentation except a flip over the vertical axis, with a probability of 0.5. The results are summarised in table 2. The F1 score, which relates to a combination of the classification and box regression, is an average over several IoU-thresholds for detection acceptance. The angle error is evaluated based on the true positive detections at a threshold of 0.5. The evaluation process and metrics are more thoroughly described in section 6.1.

Orientation Rep.	Loss	Average F1	Angle Error [°]
Regular Angle	L_1	0.886 ± 0.006	2.164 ± 1.100
	L_2	0.883 ± 0.006	2.425 ± 0.829
	Smooth L_1	0.888 ± 0.004	1.802 ± 0.666
Cartesian Half	L_1	0.892 ± 0.003	1.291 ± 0.370
	L_2	0.881 ± 0.006	1.853 ± 0.551
	Smooth L_1	0.887 ± 0.009	1.570 ± 0.496
Cartesian Full	L_1	0.890 ± 0.004	1.003 ± 0.299
	L_2	0.880 ± 0.005	1.254 ± 0.174
	Smooth L_1	0.888 ± 0.005	0.939 ± 0.136

Table 2: Summary of the results from the tests for determining the best orientation representation.

From table 2 it can be seen that the tests quite closely match the expectations for the different orientation representations. The "Regular Angle" representation has the worst performance by a large margin, and the "Cartesian Full" representation outperforms "Cartesian Half" somewhat convincingly. Furthermore, it is found that the L_1 Smooth loss yields the best performance for estimating the orientation. For this reason, Cartesian Full and Smooth L_1 are used going forward. There is no clear indication as to which loss function yields the better box regressions, and thus it is chosen to keep the original, which is Smooth L_1.

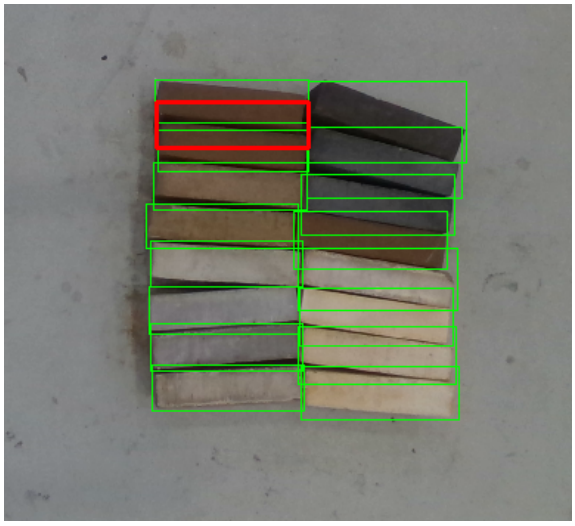
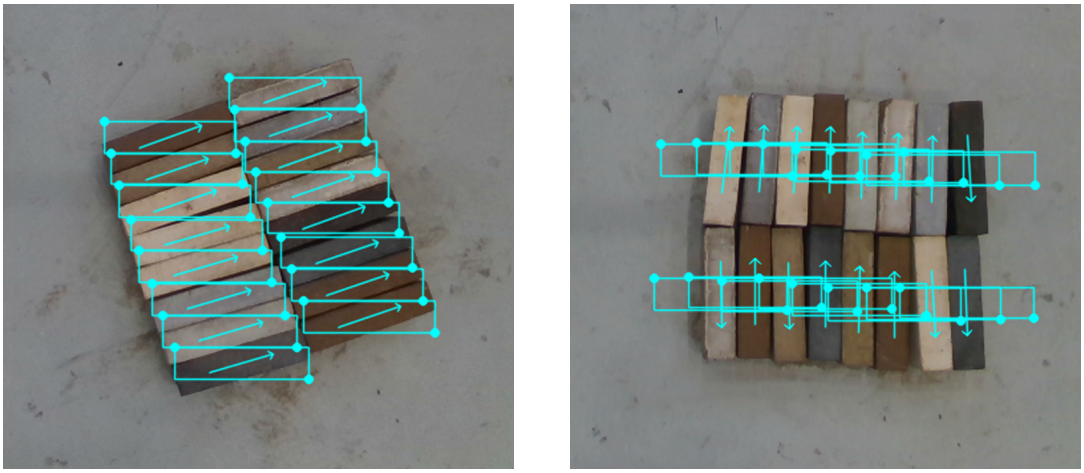


Figure 18: Detections yielded by RetinaNet with Angle, showing a red false positive bounding box, overlapping the green true positives. This can likely be eliminated by a stricter NMS.

5.1.2 RotinaNet

RWA, described above, merely adds an angle estimate to the existing output from Vanilla RetinaNet. This means that the inference output will be in the form of an axis aligned bounding box coupled with an orientation angle. This is good enough to fulfil the minimum requirement of finding the orientation and centre point of the brick. It is theorised, however, that better performance can be achieved by predicting tight fitting rotated rectangles rather than axis aligned boxes. This theory is motivated largely by the NMS algorithm used for the chamfer matching detector described in section 4.3. This algorithm is very strict, using a very low IoU threshold, since it is known that the rigid bricks will not overlap. Having this sort of NMS would likely reduce the occurrence of some specific false positives, depicted on figure 18, which have been observed. Naturally, to take advantage of a strict NMS, axis aligned bounding boxes cannot be used, as these will inherently overlap quite significantly when the bricks are not parallel to the image frame. Furthermore, it is theorised that combining box and orientation regression in the same sub-network may give more accurate predictions. For this reason, the rotated rectangle based detector, "RotinaNet" is proposed in this report, which introduces two major changes to Vanilla RetinaNet. These are adapting the box regression sub-network to include the angle regression and changing the target definition for training.

Adapting the box regression sub-network is straightforward. As the Cartesian Full angle representation is used, it is merely a case of adding two extra outputs per anchor, thus increasing it from four to six. The tricky part is defining the new targets. One option would be to define the targets as the openCV definition of a rotated rectangle, introduced in section 3, which is a centre point, width, height and an angle. To properly implement this, it would be needed to redefine the anchors, and since these are quite vital to the implementation, this would be a sizeable change. In order to reuse the original anchor definitions, the new targets will be defined as illustrated on figure 19. The target for the rotated rectangle will thus be described by an axis aligned rectangle, which contains information about the width, height and centre point of the brick, and an orientation.



(a) (b)

Figure 19: Representation of the targets used for RotinaNet, shown at different brick orientations.

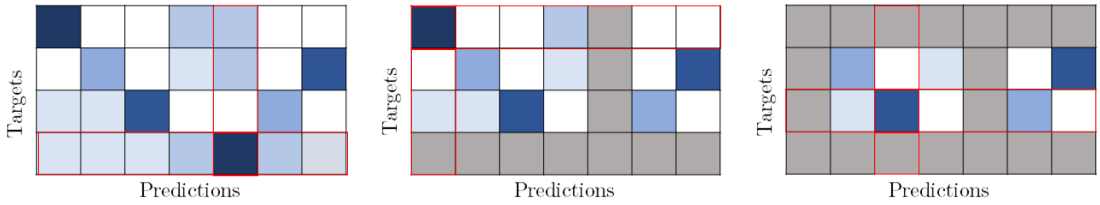
This new definition of the targets has some implications on the assignment of anchors, which, as described above, is vital for the performance of RetinaNet. As described, anchors are assigned to specific targets, or background, based solely on how much the targets and anchors overlap. As seen from figure 19b, assignment of anchors can become ambiguous when the bricks are close to vertical, as significant overlaps between the target boxes exist. To overcome this issue, a different assignment condition has been introduced, the normalised centre point distance, which is computed as shown in equation 8

$$c_{\text{dist}} = \frac{\sqrt{(a_x - t_x)^2 + (a_y - t_y)^2}}{t_h}, \tag{8}$$

where a_x, a_y, t_x and t_y are the x and y coordinates for the centre points of the anchor and target respectively, t_h is the height of the target box. Dividing by t_h ensures equal weight of the centre point distance across scales. The IoU condition is also kept, meaning that an anchor is assigned to the target which yields the lowest normalised centre point distance, and the anchor is labelled as foreground if the centre point distance is below 0.4 and the overlap is above a threshold of 0.5. Furthermore, since all targets will now adhere to a landscape format, it does not make sense to include anchors with width to height ratios smaller than one. The anchor ratio definitions have been changed to reflect this.

6 Evaluation of Methods

This section aims to evaluate the performance of the different methods introduced in the report. The tests are conducted in order to determine which solution has the best prerequisite for fulfilling the requirements specified in section 2. For the evaluation, a specific process and specific metrics have been determined, which are described in section 6.1. The parameters used for the different methods are summarised in section 6.2.



(a) First pair. (b) Second pair. (c) Third pair.

Figure 20: Illustration of the prediction/target matching process. The colouration indicates the IoU between the respective prediction and target. Darker colours indicate higher IoU. When a pair has been found, the prediction and target are removed from the pool, indicated by the grey colour. The matching process stops when the pool is empty, or when the pair with the highest IoU falls below the required threshold.

6.1 Evaluation Process

To determine the performance of the solutions presented in this project, a general evaluation process has been designed. First, a common process of pairing predictions with targets, which is used across all performance tests described in the report, are introduced. The assignment of target and prediction pairs is based on the assumption that one target can only be paired with no more than a single prediction. This implies that there can be at most as many true positives as there are targets. If there are more than one possible candidate for a target, the prediction with the highest IoU is selected. In practise the matching is done by calculating the IoU between all possible pairs of predictions and targets. The pair with the highest IoU is then located and extracted. The pair is then removed from the pool of possible prediction and target pairs. All combinations where either the target or the prediction are included, are removed as well, as illustrated on figure 20. The process repeats itself until no more annotations are available or the highest IoU in the pool is below the decided threshold.

Based on the requirements, three metrics have been chosen for measuring the performance of the developed methods. To determine detection accuracy, based on IoU between detections and targets, the F1-score has been adopted. For evaluating orientation prediction, the angle error metric is introduced. Lastly, to determine how well the brick centre points are determined, the centre offset metric is used. The F1-score used is inspired by the COCO primary evaluation challenge metric [15], and it is calculated as an average F1-score across the range of IoU thresholds as shown in equation 9. The F1-score for a single IoU threshold can be calculated as seen in equation 10.

$$\text{Average F1} = \frac{1}{10} \sum_{i \in I} F1^{\text{IoU}=i}, \quad I = \{0.5, 0.55, ...0.95\} \tag{9}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{10}$$

The angle error is calculated using the formula seen in equation 11,

$$\text{Angle Error} = \frac{1}{N} \sum_{i=1}^{N} \min \left(|p_{i,\theta} - t_{i,\theta}|, |p_{i,\theta} - t_{i,\theta} + 180^\circ|, |p_{i,\theta} - t_{i,\theta} - 180^\circ| \right), \tag{11}$$

where N is the total number of true positive prediction/target pairs in the validation set. $p_{i,\theta}$ and $t_{i,\theta}$ denote the predicted and target angle value for the i^{th} prediction/target pair. The minimum function is used to eliminate artefacts introduced by the singularity due to the angle only being defined in the right half plane.

The centre point offset is found as an average of the euclidean distance between centre points of the prediction and target box pairs across the validation set, converted from pixels to millimetres using the depth and focal length. The centre point offset is thus calculated as given in equation 12

$$\text{Centre Offset} = \frac{1}{N}\sum_{i=1}^{N}\left(\sqrt{(p_{i,x}-t_{i,x})^2+(p_{i,y}-t_{i,y})^2}\right)\cdot\frac{d}{f}, \tag{12}$$

where N is the total number of true positive prediction/target pairs in the validation set. $p_{i,x}$, $p_{i,y}$, $t_{i,x}$ and $t_{i,y}$ denotes the x and y-coordinates of the prediction and target centre points for the i^{th} prediction/target pair. f is the average focal length of the camera in x and y. d is the depth of the objects relative to the camera. To ensure a pessimistic estimate, d has been chosen as the height of the camera above the floor.

The the angle error and the centre point offset are only evaluated for predictions labelled as true positive, as these are the only predictions where it is possible to compare with a target. As opposed to the average F1-score, these are not calculated as an average over a range of IoU thresholds, instead the results reported are from the lowest IoU-threshold of 0.5. This has been chosen as it is expected that this value will generally be more pessimistic than averaged across multiple IoU-thresholds.

6.2 Training and Tuning

This section aims to establish the parameters which are used in obtaining the evaluation results. The parameters are chosen with the objective of producing the best performance. To ensure the best performance for the baseline methods, all customisable parameters of the algorithms will have to be tuned to the specific data set. Even for the relatively simple algorithms presented in this report, the search space becomes very vast. For this reason, tuning has been limited to the parameters which are expected to have the highest impact.

For the line based detection method, the quality of the edge map is essential. Parameters that have been deemed the most important to the quality of the edge map are the smoothing filter and the thresholds used for the hysteresis in the Canny algorithm. Different hysteresis thresholds have been explored via a grid search. The grid search has been performed using Gaussian filters of kernel size 3×3 and 5×5 as well as with a 5×5 median filter. The median filter yielded the best results and the associated grid search for data set 1, evaluated as axis aligned boxes, is visualised in figure 21. The grid search is repeated at a higher resolution around the best parameters for increased performance. The best performing parameters for both data sets and target representations, are presented in table 3.

The chamfer matching method has been found to be more robust towards noise in the edge map, thus the exact thresholds used for the Canny algorithm are not as vital as for the line detection

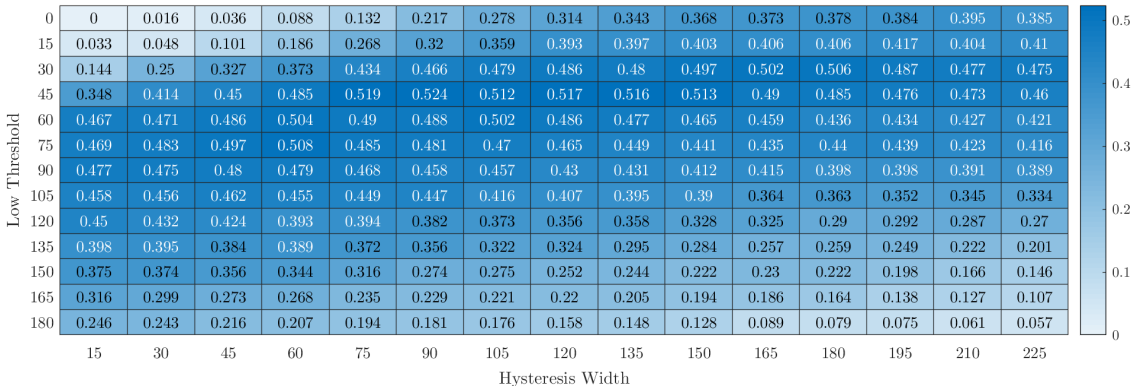


Figure 21: Visualisation of the grid search for the best Canny hysteresis thresholds for data set 1, evaluated as axis aligned boxes. The vertical axis indicates the lower threshold and the horizontal axis shows the hysteresis width. The scores presented are the average F1 scores.

Data set	Target Rep.	Low Thresh.	Hyst. Width
1	Axis Aligned	48	84
	Rotated Rectangle	48	123
2	Axis Aligned	30	135
	Rotated Rectangle	30	135

Table 3: Summary of the best parameters for the line based detection method.

algorithm. Through experimentation and examination of edge maps it was found that using a low threshold of 25 and a hysteresis width of 45 yielded good results. For data set 1, a 3×3 Gaussian filter is used for smoothing in the Canny algorithm, while using a 5×5 Gaussian filter yielded better results for the higher resolution images of data set 2. The template matching mode and matching score thresholds are expected to be important parameters. The matching modes, of the utilised openCV method *matchTemplate()*, "CCORR", "CCORR_NORMED", and the weighted template mode introduced in section 4.3, have been examined in an appropriate range of matching scores, finding that the weighted template mode yielded the best performance. Figure 22 shows the performance across a range of matching score thresholds for the chamfer matching method, using the weighted template mode. Table 4 summarises the results for both data sets and target representations.

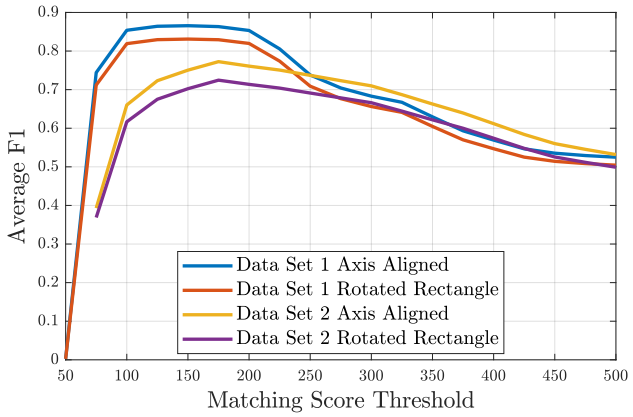


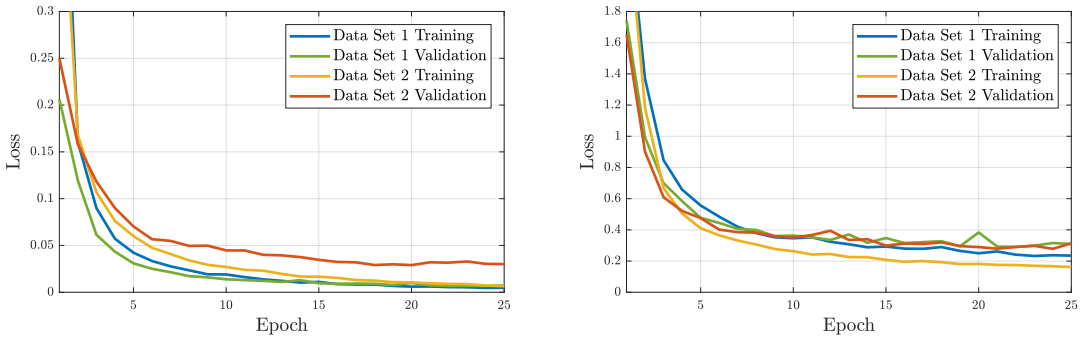
Figure 22: Visualisation of the search for the best matching score threshold for the chamfer matching algorithm. The figure shows results for the weighted template matching mode.

Data set	Target Rep.	Matching Score Thresh.
1	Axis Aligned	150
	Rotated Rectangle	175
2	Axis Aligned	175
	Rotated Rectangle	175

Table 4: Summary of the best parameters for the chamfer matching method.

The training process of a deep neural network also involves a large amount of hyperparameters subject to tuning and experimentation. Again the search space becomes incredibly vast, furthermore, since training is required, the time it takes to evaluate a single set of hyperparameters is much higher than for the baseline methods. For this reason, experimenting with different hyperparameters for training has not been prioritised for the project. Instead, most of the suggested values from the implementation used have been adopted. The implementation provides a script to be used in training, "train.py", which has been utilised. The script implements the Adam-optimiser with a learning rate of 10^{-5}, which reduces on plateaus. Visual data augmentation is used, but no transformational augmentation except a flip over the vertical axis with a probability 0.5. Training has been carried out over 25 epochs using a batch size of one. At each epoch, the training set is cycled once. Each model is trained based on an initial set of pre-trained weights[2] provided by the creator of the implementation [16]. Figure 23 shows an example of learning graphs from RotinaNet training on the two data sets. For both data sets, the losses converge. It is worth noting that for data set 2 there is a noticeable gap between the training loss and the validation loss, especially for the classification sub-network, indicating a slight tendency to overfit the training data. It is theorised that this may be influenced by the way the data set has been created and partitioned. This will be addressed further in section 7. The deep learning models supply a confidence score for each predicted bounding box. At inference, this score can be used to remove unlikely candidates, typically yielding higher detection precision. Through experimentation it was found that using a

[2]Model weights are based on a ResNet-50 backbone and trained on COCO (resnet50_coco_best_v2.1.0.h5)



(a) Classification Loss

(b) Regression Loss

Figure 23: Loss graphs obtained during training of RotinaNet on both data set 1 and data set 2.

confidence score threshold of 0.9 for Vanilla RetinaNet and RetinaNet with Angle yielded the best performance. Due to the stricter NMS utilised for RotinaNet, a lower threshold has been found to yield better performance. A threshold of 0.8 is used for RotinaNet.

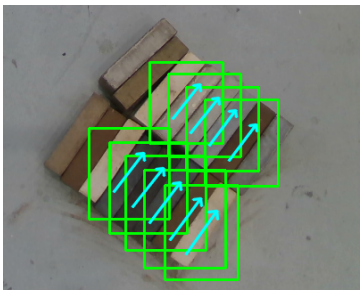
6.3 Test Results

Based on the evaluation procedure, tuning and training described above, the key results obtained during this project are summarised below. As introduced, some methods support only the axis aligned target representation. Since the rotated rectangle representation can easily be converted into the axis aligned representation, as described in section 3, all methods are evaluated based on the axis aligned representation. The results are summarised for data set 1 and data set 2 in tables 5 and 6 respectively. In addition to the four methods developed for the project, Vanilla RetinaNet has been included for comparison. As this method does not provide an angle prediction, the associated column is not relevant. Examples of predictions can be seen in figure 24 and 25. Since training of deep learning models entails inert randomness, each model has been trained six times to report the average performance and standard deviation. The methods which support the rotated rectangle representation are also evaluated based on this representation. The results for the two data sets are summarised in tables 7 and 8, and examples of predictions are shown in figures 26 and 27.

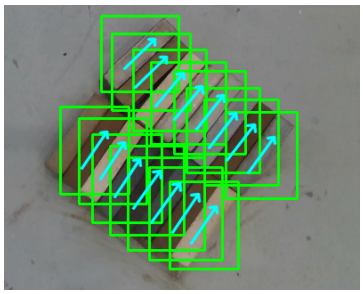
The deep learning methods are executed on an NVIDIA Quadro P5000 GPU. The baseline methods are not optimised for GPU execution and are executed on a consumer grade laptop CPU, Intel i7-6650.

Method	Average F1	Angle Error [°]	Centre Offset [mm]	Time/Img [ms]
Line Based Detection	0.527	1.852	7.932	17.32
Chamfer Matching	0.866	1.035	3.593	3271.88
Vanilla RetinaNet	0.882±0.005	-	3.292±0.106	179.9±2.1
RetinaNet w. Angle	0.888±0.005	0.939±0.136	3.376±0.071	197.6±1.8
RotinaNet	0.891±0.008	1.016±0.071	2.796±0.227	183.1±1.1

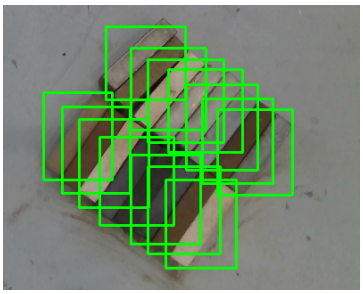
Table 5: Test results of the introduced methods evaluated on data set 1, using the axis aligned bounding box representation.



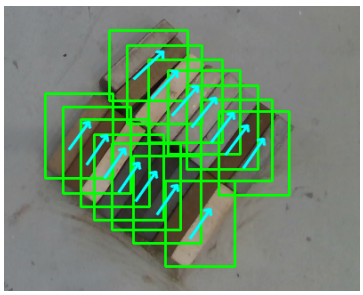
(a) Line Based Detection.



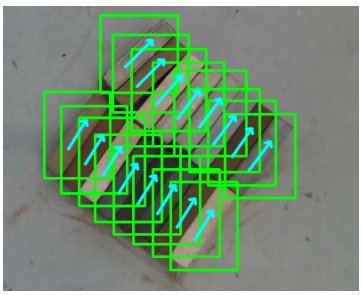
(b) Chamfer Matching.



(c) Vanilla RetinanNet.



(d) RetinaNet w. Angle.

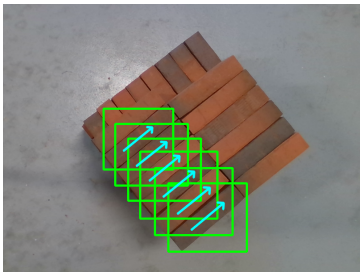


(e) RotinaNet.

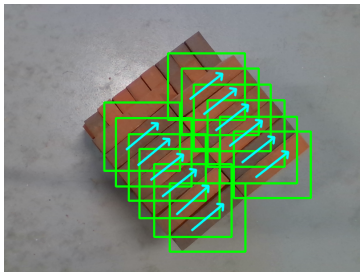
Figure 24: Example of predictions using the methods presented in table 5.

Method	Average F1	Angle Error [°]	Centre Offset [mm]	Time/Img. [ms]
Line Based Detection	0.490	0.993	7.930	37.0
Chamfer Matching	0.773	0.913	4.174	5740.8
Vanilla RetinaNet	0.907±0.006	-	4.583±0.218	151.3±1.0
RetinaNet w. Angle	0.904±0.010	0.928±0.122	4.631±0.311	166.7±0.7
RotinaNet	0.928±0.005	0.881±0.077	3.118±0.159	153.2±0.8

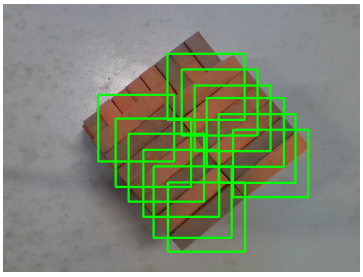
Table 6: Test results of the introduced methods evaluated on data set 2, using the axis aligned bounding box representation.



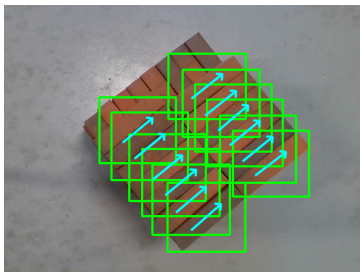
(a) Line Based Detection.



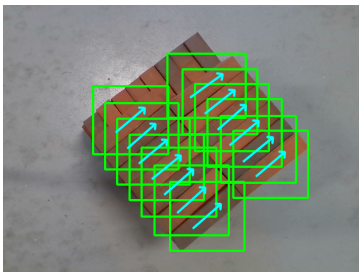
(b) Chamfer Matching.



(c) Vanilla RetianNet.



(d) RetinaNet w. Angle.

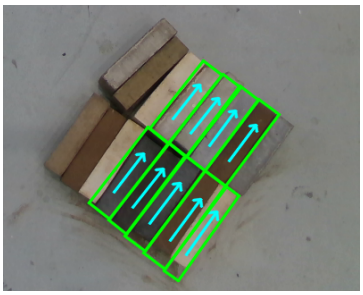


(e) RotinaNet.

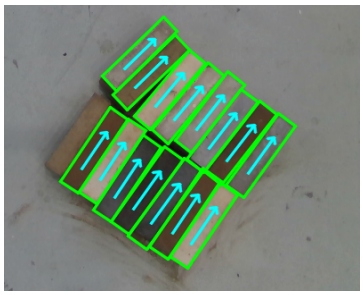
Figure 25: Example of predictions using the methods presented in table 6.

Method	Average F1	Angle Error [°]	Centre Offset [mm]	Time/Img [ms]
Line Based Detection	0.481	1.492	6.896	24.7
Chamfer Matching	0.816	1.051	3.976	1693.9
RotinaNet	0.865±0.008	1.017±0.071	2.786±0.209	167.1±31.7

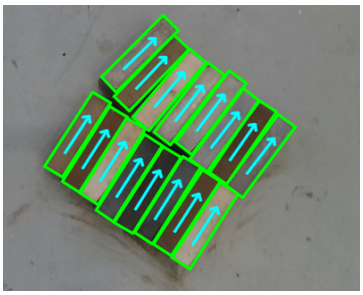
Table 7: Test results of the introduced methods evaluated on data set 1, using the rotated rectangle representation.



(a) Line Based Detection.



(b) Chamfer Matching.

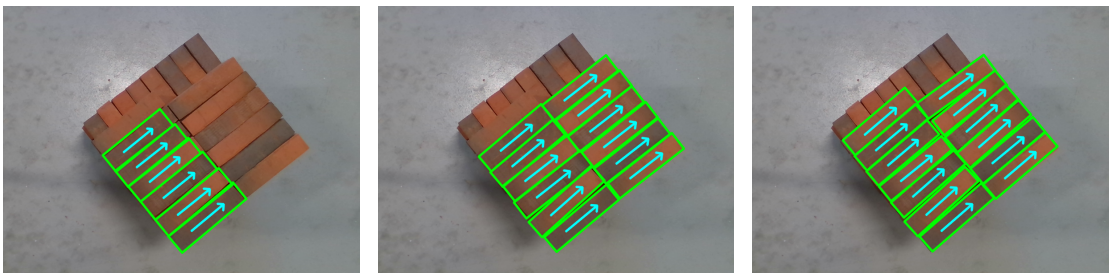


(c) RotinaNet.

Figure 26: Example of predictions using the methods presented in table 7.

Method	Average F1	Angle Error [°]	Centre Offset [mm]	Time/Img [ms]
Line Based Detection	0.433	0.861	8.157	42.6
Chamfer Matching	0.725	0.912	4.251	6778.9
RotinaNet	0.866±0.008	0.881±0.077	3.118±0.159	152.3±1.2

Table 8: Test results of the introduced methods evaluated on data set 2, using the rotated rectangle representation.



(a) Line Based Detection. (b) Chamfer Matching. (c) RotinaNet.

Figure 27: Example of predictions using the methods presented in table 8.

7 Discussion

This project aimed to examine different approaches to computer vision based object detection, in the specific context of automating brick work. More specifically the depalletising task. Two baseline methods, based on traditional computer vision, have been developed, as well as two methods based on deep learning, more specifically the RetinaNet structure. The results presented in section 6 aim to provide a comparison between the different methods.

From the results, it appears that the better performing methods are those based on deep learning, moreover, the proposed RotinaNet shows the best results of all. The baseline method using line based detection clearly shows the worst performance across both of the data sets produced. The deep learning methods perform the best, and very similarly on data set 1, and chamfer matching is not far behind. For data set 2, the deep learning methods have improved performance, whereas the chamfer matching method has quite a significant decrease in performance. A possible explanation for this behaviour is addressed in the sections below.

Considering the requirements introduced in section 2, only the performance yielded by RotinaNet is deemed satisfactory across all tests, based on the test metrics used. The hardest requirement to fulfil appears to be the centre offset. It can be argued however, that the pessimistic approximation used may be too strict, especially for the case of data set 2, where the actual depth of the objects may be much less than the distance from the camera to the floor. Based on this, it can be argued that RetinaNet with Angle and chamfer matching may be viable as well. Chamfer matching is however the only method which does not meet the required execution time, although it is expected that this can be optimised significantly, which will be addressed below. Based on the specific

data sets however, the deep learning methods, of which RotinaNet has the best performance, are preferable over chamfer matching due to the better detection accuracy.

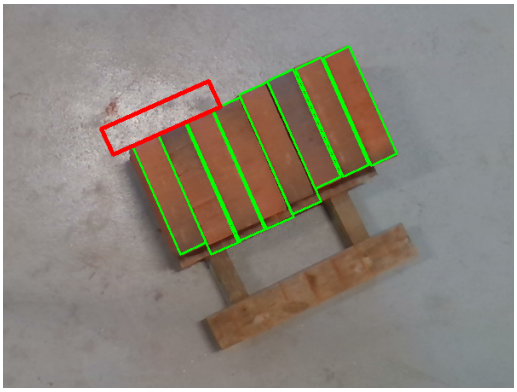
7.1 Baseline Methods

Both baseline methods introduced are based on a similar preprocessing step which produces the edge map to be utilised. This step is vital, however it features some vulnerabilities, especially towards changes in lighting conditions. Changes in lighting conditions may greatly impact the condition of the edge map, ultimately affecting the predictions. The parameters can be tuned to a given setting, but adapting to a change in the environment may require human intervention.

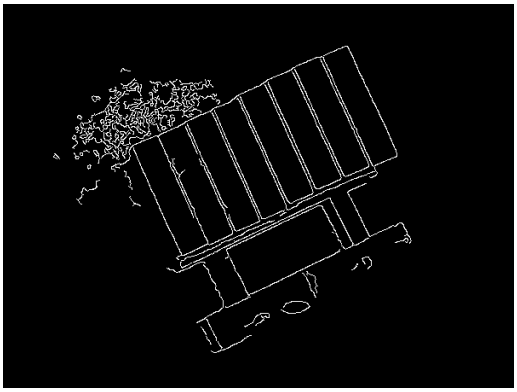
Since the stacked bricks generally feature distinct edges, it was believed that a detection method based around extraction of lines could perform well. There are however several limitations to the proposed line based detection method, which can contribute to the poor performance observed. One limitation comes from the fact the method assumes that the bricks will appear in a grid-like pattern, and that they are all roughly parallel to each other. In the data sets provided, the bricks are mostly arranged in a grid, however, data set 2 contains many examples where the bricks are not all parallel, since bricks from the underlying layer will be perpendicular to the upper layer. This means that lines originating from bricks on the underlying layer can intersect lines from the upper layer, yielding line intersections that are not at all related the position of brick corners. This problem of "phantom" corners is a big shortcoming of the method, as they can also arise from false line detections based on noise in the edge map. Thus making the method vulnerable to noise. As introduced in section 4.2, the grid pattern had to be assumed since the short brick edges did not yield enough edge pixels to consistently extract distinct lines from singular bricks. This means that several bricks will have to be placed next to each other in a grid to yield detections, and thus single bricks are unlikely to be found. It is possible however that this issue could be accommodated by first finding the strong long edges and afterwards searching for the weaker short edges perpendicular to these. This would allow a lower threshold to be used which would be required to find the weak edges. The method utilises predefined rules about the bricks to rule out ill fitting candidates. These rules were specifically tailored to each data set. This means that given a data set with higher variance, most specifically in scale, the rules will have to be less strict, and the average F1-score is expected to decrease as more false positives predictions are introduced.

Although the chamfer matching algorithm is more robust towards noise than the line detection based algorithm, it is still vulnerable to some occurrences of noise. If dense regions of high frequency exist in the image, this causes dense regions of edge pixels in the edge map, as seen in figure 28b. This region will carry a low distance score and thus also a low matching score, leading to false detections as seen in figure 28a. There are a couple of possible ways to accommodate this issue. First, the amount of high frequency noise can be reduced through further examination of filters. Secondly, it may be possible to adapt the templates such that the presence of edge pixels outside the expected areas in the template are penalised. However, no obvious way to adapt the template, without introducing other implications, has been found.

As mentioned in section 4.3, the chamfer matching algorithm relies on a predetermined definition of the expected scale range of bricks to be detected. As the range of scales, which bricks can



(a) Predicted rotated rectangles.



(b) Edge map.

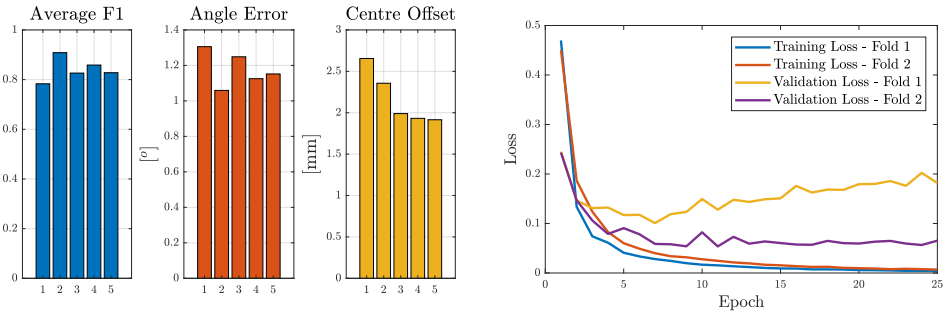
Figure 28: False detection, highlighted in red, caused by dense regions of noise in the edge map. The images have been cropped for illustrative purposes.

appear in, increases, a trade-off between precision and speed has to be determined. By increasing the number of scale steps, thus increasing the number of templates used, detection accuracy can be prioritised, however it comes at the cost of increased execution time. Conversely, using fewer templates will decrease execution time, but likely reduce detection accuracy.

As evident from the results presented in section 6, the evaluation time for the chamfer matching method is much higher than any of the other methods. It is expected that this can be reduced very significantly, if the method is optimised. The most time consuming part of the algorithm, is the series of matching convolutions, which can be parallelised for GPU execution. A convolution using 300 206×206 filters, equivalent to 300 of the largest templates used for data set 2, has been shown to be executed in less than 100 ms on the NVIDIA Quadro P5000. The remaining functionality of the algorithm is timed on the Intel i7-6650 CPU and can be executed in less than 20 ms. Adding to an estimated total of 120 ms, which is well within the requirement.

7.2 Deep Learning Based Methods

Deep learning approaches carry the inert characteristic that their performance heavily rely on similarities between the training set and the data set on which the method is evaluated. This means producing high quality, representative training data may be seen as the most important task in developing a deep learning based detector. It can be argued that the way especially data set 2 has been created and split, may give preference to the deep learning methods over the baselines, thus giving slightly unfair results. The data is collected by removing, or adding, a single brick to the stack between each image. This means that between two images, the only real difference is the presence of a single brick. The rest of the motive remains mostly identical. Since the images for the validation set are picked at random, this means that there will be incredibly low variance between the data in the training set and in the validation set, which is ideal for the neural network to perform well. An argument can be made that this performance cannot be expected to carry over into a practical application, since the images presented will not be as similar as in the case presented. To examine this, a 5-fold split is proposed, where the images are



(a) Results from the 5-fold split.

(b) Classification loss for training and validation for fold 1 and 2.

Figure 29: Results from the 5-fold split, where the data is split based on the collection sequence, introduced as an alternative to the original random split.

partitioned based on the sequence in which they were obtained, instead of at random. The first fold has the first 80 % of the sequence as training set and the last 20 % as the validation set. The second fold has the first 60 % and the last 20 % as training partition, with the remaining 20 % reserved for validation, and so on. It is important to note that the results presented thus far are not rejected, since a random data split is generally regarded as an optimal choice. The 5-fold split is simply introduced to give a different perspective on the problem. Evaluations on the new splits show a somewhat significant reduction in performance, as seen from figure 29a, and a greater tendency for the network to overfit the training data, as illustrated by figure 29b. Fold 1, which shows the worst performance, also shows quite significant signs of overfitting from the classification loss. Fold 2, which achieves the best detection performance of the five, shows much less of this tendency. The reduced performance, compared to the random split, is somewhat expected, since the similarities between the training and validation sets have been reduced significantly, and the training set is relatively small relative to the complexity of the neural network. It is believed that this tendency to overfit can be reduced by introducing transformation augmentation to the data during training, and possibly by freezing the weights of the network backbone, essentially reducing the complexity of the trainable network. Ideally, the data should have been collected in a way that incorporates more variance and randomness, yielding a less systematic data set. A solution would be to remove all bricks from the pallet and restack between images, although this would have been both cumbersome and infeasible time-wise.

RotinaNet, a modified model based on RetinaNet, has been introduced in this report. The model gives a considerable boost in performance to the state of the art results obtained by the original RetinaNet model. The changes are made for this specific application and the performance boost is only verified on the introduced data sets. It should therefore be noticed that the increase in performance is not guarantied to generalise to other applications. It is however thought to be applicable to scenarios similar to the one introduced, where the objects can be seen as rigid rotated rectangles. RotinaNet is designed in a way that require minimal modifications of the RetinaNet model in order to support rotated rectangles. It was decided to represent the the rotated bounding box with an axis aligned bounding boxes plus an angle, as described in section 5.1.2. With this in mind it is plausible that more efficient and better performing methods of defining the targets

and anchors exist. It is expected that a more well designed model and representation of targets potentially could cause an additional increase in performance.

7.3 Perspective and Future Work

The project has been carried out as a computer vision study, however it is intended for the solution to eventually be used as part of a physical brick depalletising robot cell. Although the results obtained in this project, especially from RotinaNet, are promising, it cannot be said how well they transfer into practical applicability. To confirm this, the next step would be to carry out a series of picking and depalletising tests involving the robot. Due to the possible issues with data set 2, introduced previously, it may be necessary to create a new or expanded training set for the practical tests to achieve comparable performance. In addition to testing how well the detection methods generalise to practical use, the physical tests would expose how well the spatial position of the objects can be determined using the depth data captured by the Intel Realsense D415.

This project has introduced RotinaNet, an adaptation of the existing RetinaNet model to achieve a partial pose estimation of bricks. A reasonable future advancement for the developed method would be to introduce full pose estimation of the brick. Obtaining a full pose estimation would open the possibilities of more general application where the bricks are not neatly positioned, but could be placed on uneven terrain or dumped in a pile. An obvious application of this would be robotic assistance in cleaning up construction or demolition sites. The RetinaNet architecture has obviously not been developed with the intention of such adaptations, and may be too rigid for such extensive modifications. Instead, it would be interesting to explore the potential of using CenterNet, introduced in the article "Objects as Points" [17], as it shows great promise for customisability and state of the art performance.

8 Conclusion

Four different computer vision methods for detecting palletised bricks have been introduced. Two methods are based on modern deep learning based techniques, and the other two, which are treated as baselines, are based on traditional computer vision methods. The first baseline method is based on finding intersections between distinct lines in the input image. The distinct lines are extracted from an edge image, utilising the Hough transform. The second baseline method is also based around the edge map, but utilises the chamfer matching algorithm for finding the presence of a template in the edge map.

The two deep learning based methods are adaptions to the existing one-stage detector RetinaNet. The adaptions are motivated by the need for an orientation estimate of the bricks. The first model, referred to as RetinaNet with Angle, adds an extra sub-network to the existing two of RetinaNet. The new sub-network regresses the angle of the objects. The second model, RotinaNet, combines box and angle regression into a single sub-network predicting rotated rectangles.

Two different data sets, each divided into a training and validation partition, have been introduced and produced to serve as basis for training and test of the developed methods. All four methods

have been evaluated on the validation partitions and checked against requirements based on the prospects of the practical application of robotic brick picking. The line based detection method had the worst performance of all, achieving an average F1 score of 0.527 on data set 1 and 0.490 on data set 2. Chamfer matching outperforms the line based detection method reaching an average F1 score of 0.866 on data set 1 and 0.773 on data set 2. RetinaNet with Angle achieves an average F1 score of 0.888 on data set 1 and 0.904 on data set 2. The RotinaNet is the best performing method across both data sets, reaching an average F1 score of 0.891 on data set 1 and 0.928 on data set 2. For the application at hand, the RotinaNet model has yielded and increase in detection performance of up to 2.1 p.p on data set 2, compared to Vanilla RetinaNet. Through the tests conducted, RotinaNet was the only method found to meet all requirements stated in section 2. It cannot be guaranteed however, whether the performance can be transferred directly to a practical application.

References

[1] M. A. Kuhlmann-Jørgensen and J. G. Rasmussen, "Brick-Pose-Estimation," https://github.com/JakobGroeftehauge/Brick-Pose-Estimation, 2020.

[2] Strøjer Tegl, *Strøjer Tegl - Bricks*, 2020. [Online]. Available: https://strojertegl.dk/mursten/

[3] "Piab BX35P Suction Cups," https://www.piab.com/Products/suction-cups/shape/multibellows/bx--2bellows-duraflex10110mm-7366cb58/0106604/#specs, 2020.

[4] Intel, *Realsense D415*, 2020. [Online]. Available: https://www.intelrealsense.com/depth-camera-d415/

[5] K. Dawson-Howe, *A Practical Introduction to Computer Vision with OpenCV*. Wiley, 2014.

[6] P. Soviany and R. T. Ionescu, "Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction," 2018.

[7] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[8] Fizyr, *Keras RetinaNet*. [Online]. Available: https://github.com/fizyr/keras-retinanet/tree/a81b313eedbfd2ab849c3aecd6b7bef1f6cf9294

[9] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Comput. Vis. Pattern Recog*, 01 2001.

[10] Intel, *Intel RealSense*. [Online]. Available: https://github.com/IntelRealSense/librealsense

[11] K. Wada, "labelme: Image Polygonal Annotation with Python," https://github.com/wkentaro/labelme, 2016.

[12] *openCV Documenation imgproc*, 2020. [Online]. Available: https://docs.opencv.org/3.4.10/d7/d1b/group__imgproc__misc.html

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[14] R. Girshick, "Fast R-CNN," 2015.

[15] *COCO, Detection Evaluation Metrics*, 2020. [Online]. Available: http://cocodataset.org/#detection-eval

[16] Fizyr, "keras-retinanet: Releases," https://github.com/fizyr/keras-retinanet/releases, October 2019.

[17] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as Points," 2019.