Universiteit Leiden
The Netherlands

Opleiding Informatica

Object Detection and Rotation Estimation using IntelRealsense depth camera

Dos Santos Cordeiro Phoebe

Supervisors:
Dr. Daan M. Pelt & A.R. Mesquita Fery Antunes

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/06/2025

Abstract

Automated quality control for industrial use has overcome the issue of detecting objects on the conveyor belts, by using neural networks (e.g. YOLO) or reference points on objects (e.g. QR codes), but has yet to accomplish the same for estimating the orientation of the objects on the conveyor belts. While being able to detect objects on the conveyor belts is already a good step in improving the automation of quality control, the lack of orientational information means that not all aspects of the products can be automatically checked. In a similar fashion, the physical twin lab, located at Leiden University, needs to be able to track coloured blocks as well as estimate their orientation. In this thesis several approaches for the estimating the orientation of coloured cubes are explored using an Intel RealSense D455 camera, which provides depth information in conjunction to colour images. Although the use of the depth imaging capabilities of the camera seemed useful, these turned out to not be stable enough to be used with the proposed initial idea of difference between object extremities. Thus, the proposed method for further research is the use of photogrammetry in conjunction with pose estimation techniques and the use of QR codes as the anchor points. This method proved to be the most reliable and least computationally expensive way to track objects and estimate their orientation.

Contents

1 Introduction

For the past ten years there has been a considerable rise in use for object detection algorithms. These algorithms are used in the medical field to assist medical personnel [YY21] by analysing various types of images, computed tomography (CT), X-ray images or magnetic resonance imaging (MRI) among many other variations [YY21]. Outside of their use for static image analysis, object detection algorithms, can also be applied for dynamic image processing. This type of detection makes use of highly efficient one-stage networks [ZLW+21], such as single-shot detection (SSD) [LAE+16] or You Only Look Once (YOLO) [RDGF16]. The use of these efficient networks has allowed for the application of object detection in multiple fields such as: security surveillance [LLT+23], automated cars [YSP+22] and for industrial product lines quality control [BAX19]. Automated inspection for quality control in the food industry is able to assure the quality of products through the use of gas-sensors which can distinguish volatile organic compounds determined by the aroma of different products [KCF23]. However, while AI has proven to be capable at determining the quality of food products based on images [AEP+22, ZZL+19] it faces a key challenge in accurately determining the 3D orientation of the objects it detects. As these objects move on the production line it is for some products important to verify that they are not misaligned in order to assure smooth operation. The physical twin lab at Leiden University aims to find a solution to the problem of orientation estimation by using previously researched object tracking methods [SKJ19] and building on top of these to find a suitable method for the estimation of the orientation of objects, which are coloured cubes in this case, using the Intel RealSense D455 camera. The additional layer of information can help detecting misplacement of objects on the conveyor belt and thus assure smooth operation of the machinery surrounding the conveyor belts. Additionally the rotational information can be used to help in the detection of defects.
This paper aims to explore and compare multiple methods of object detection and orientation estimation in order to answer the following questions:

1. Is it possible to use object detection in conjunction with the depth camera from the D455 camera to accurately detect and estimate the orientation of objects?

2. Is it possible to accurately and reliably detect small QR codes to estimate the orientation thereof?

1.1 Thesis overview

The second section will cover some related work that explains the concepts of the methods used for both the object detection pipeline as well as the orientation estimation. The third section gives an overview of the hardware that was used which includes the Intel RealSense D455 [inta] with its depth imaging capabilities as well as the conveyor belt from the LIACS Twinlabs that was used to perform some of the experiments as it provides for a suitable prototyping environment. The methodology behind the object detection pipeline, which allows for the detection of the small cubes chosen for the experimentation, is also elaborated upon in this section. The fourth section then presents the methods that were theorized and implemented for the orientation estimation. This includes a theoretical approach, which was unable to be implemented due to the limitations of the depth imaging of the chosen camera as well as an implemented method whose robustness and limitations were tested. In the conclusion, the take-aways from this paper are summarized as

well as ideas for further research are given. This work was part of a bachelor thesis at the Leiden Institute of advanced Computer Science (LIACS) with the help of dr. Daan M. Pelt as a supervisor and A.R. Mesquita Fery Antunes as a second supervisor.

2 Related Work

At the core of this paper are two concepts: object detection and orientation estimation.

2.1 Object Detection

Detecting objects within images is a task usually done by convolutional neural networks. The first such network to successfully solve the task of object detection is the CaffeNet [KSH17] architecture, this architecture was however slow due to using a selective search method for the generation of region proposals. As an improvement on this previous method the Fast R-CNN network was developed. This network makes use of a feature map that is calculated and onto which the region proposals given by the selective search are projected [Gir15]. However, this method still suffered from the usage of the selective search which lead to it getting replaced by its next iteration being the Faster R-CNN. The Faster R-CNN network applies the selective search directly to the feature map instead of projecting the proposed regions onto it [RHGS17]. However, even though this made it possible to make more use of the feature map and the information it provides, the speed of these neural networks was still nowhere near where it needed to be for them to be used in real-time analysis. These networks are what we classify under two-shot algorithms [DZW20] as they make use of region proposals before extracting features from the proposed regions.
In order to make the task of object detection faster single-shot or one-shot neural networks were developed. The first of its kind was the Single shot multibox detector (SSD) [LAE+16]. These networks perform their detection procedures for a large number of regions in a single iteration, making them faster than two-shot networks but less accurate. Two other single-shot architectures are the You only look once (YOLO) [RDGF16] and RetinaNet [LGG+17]. The networks from the YOLO architecture will be focused on further as they are central to the content of this paper. At the time of writing, the YOLO architecture boasts twelve neural network versions each improving upon the last iteration [Ultb]. In this paper the fifth iteration of the network is used as it provides satisfactory performance in detection of objects in real-time. Early iterations of the YOLO networks, including YOLOv5, make use of a DarkNet-53 neural network [pjr], which is a convolutional neural network that is used for the recognition of objects. The YOLO architecture also does not make use of the previously mentioned selective region search and instead scales and divides the input image into square regions within which recognition takes place. By subdividing each square into three more rectangles and estimating the presence of an object within these. The rectangles can be of differing sizes, which allows for leaving the area with the highest probability. YOLO also allows for processing individual frames from a video sequence in a single pass, making it a suitable method for use in real-time application.
However, object detection, whether static or dynamic in nature, suffers from the fact that the coordinates of the detected objects or entities are represented within the pixel space of the image. Thus, the detected objects need an extra layer of translation from between pixel space to real life coordinates. In a similar research [AKU+22] using a stereo camera (Intel RealSense D415) to

get the depth information and a YOLOv3 neural network for object detection a method to relate camera pixel location and depth information to real world axes is described.

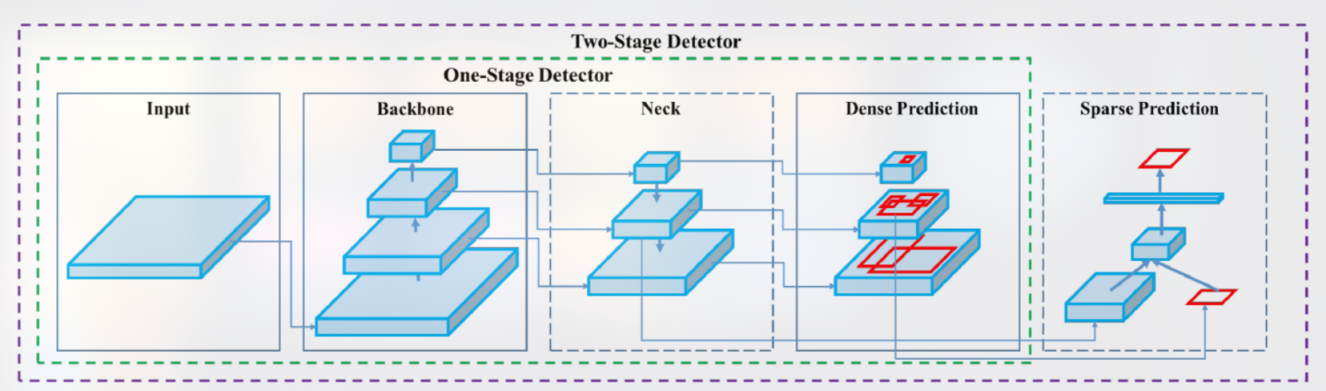
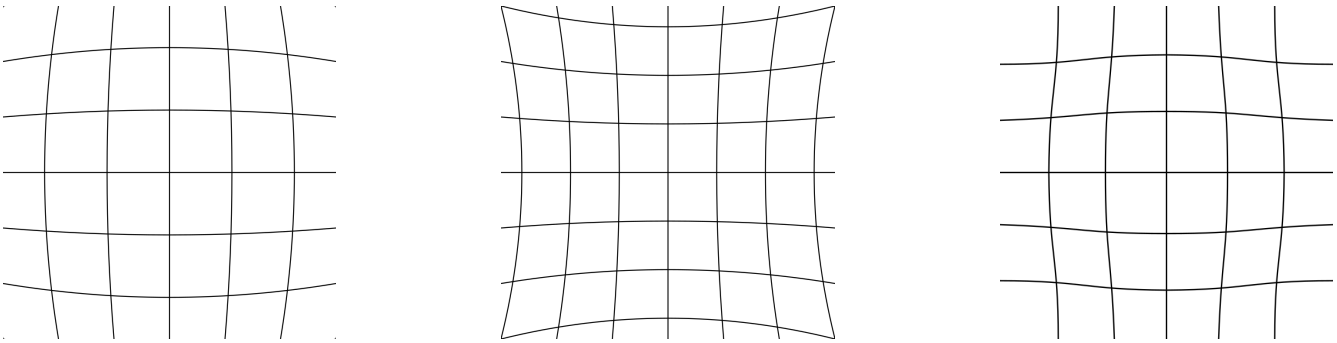


Figure 1: Schema demonstrating the process of detecting objects for one-stage and two-stage neural networks [BWL20]

2.2 Object orientation

In terms of object orientation we take a look at the field of pose estimation and photogrammetry. The orientation of objects is a concept that has been long used with Augmented Reality (AR) [MUS16], where a virtual object is correlated to the real world through a camera in order to give the impression that it is in fact real. AR technology is most notably used in conjunction with QR codes or ArUco codes, the latter of which was specifically designed for AR applications [ACPT22, CFA+11]. At the core of these technologies is the translation of a 3-dimensional object into a 2-dimensional view. For this Perspective-n-Point (PnP) is particularly interesting in our case. This method makes use of known points within an image [QL99] to correlate them to the pixel space. In order to correlate an object with known points from its world coordinates to their projection within an image, the intrinsic parameters from the camera used to capture the image are necessary. These parameters are the camera's distortion coefficients as well as the camera matrix. The reason behind the need for these parameters is to correct any kind of distortion that the camera lens might have introduced to the image.
Most cameras make use of a lens in order to allow for sharp pictures. The lens of a camera determines its focal point and the refraction that light undergoes when passing through the lens. Refraction is the process of bending the light as it passes through the lens and the light waves speed is altered [Pen00]. The point at which all the light waves meet, before reaching a mirror or an imager [Fos93] depending on the age of the camera, is the focal point of a camera. While lenses are essential to the working of a camera, they are the cause of radial distortion. Radial distortion comes in three types, which depends on the type of lens used: barrel distortion, pincushion distortion and moustache distortion. [vW]. Barrel distortion occurs when a fisheye lens is used as these take hemispherical views in order to map an infinitely wide object plane into an image using this kind of distortion. Zoom lenses also cause barrel distortion in the middle of the lens's focal length range and the distortion amplifies at the wide-angle end of the range. Pincushion distortion occurs due to the use of a convex lens, which causes the image to be magnified the further away it is from the centre.

And lastly, moustache distortion is a mixture of both previous types of radial distortion, which occurs less frequently and causes horizontal elements from the image to bend in the shape of a handlebar moustache.



(a) Barrel distortion (b) Pincushion distortion (c) Mustache distortion

Figure 2: Different types of distortion, images by WolfWings - Own work, Public Domain, url

To correct the radial distortion that lenses and other radial components of cameras introduce, the Brown-Conrady distortion model [Bro66] can be used. This model also corrects for tangential distortion which can be caused by the misalignment of camera components. Correcting the distortion in images is paramount to achieving good results when solving the PnP equation, as distortion leads to the points of interest being misplaced resulting in inaccurate projections from the 3D plane to the image.

3 Methods and Hardware

To apply the methodology in practice, the hardware used must be defined. This is because the hardware, especially the selection of the camera, has significant effects on the end result. The camera used is the Intel RealSense D455 and is further described in section 3.1. To run the neural network as described in section 3.3 a Desktop, with a NVIDIA GeForce RTX 4090 and a AMD Ryzen 9 7950X3D CPU was used.

3.1 Depth Camera

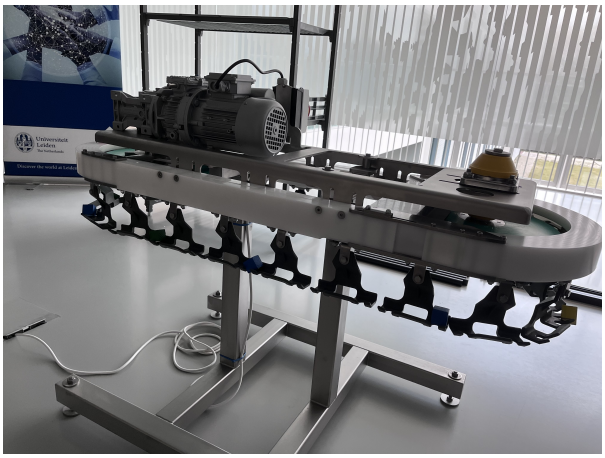
The camera that was used is an Intel RealSense D455, which is a newer model compared to the one used in the aforementioned study [AKU+22]. According to intel this newer model features the same capabilities as the older model's but with better performance at longer range [inta]. The important feature of this camera is its depth imaging. To get depth information the camera makes use of two imagers, which are spaced away from each-other by a known distance on the same plane [Intb]. By correlating the pixels from both images the depth is calculated for each pixel using the known distance between the two imagers and the range to the pixels.
The depth images can be ran at 60 frames per second (fps) at a resolution of 640 by 480. the camera does allow for a depth resolution of up to 1280 by 720 and a max of 90 fps, but it is not possible to reach both of these upper limits at the same time. At its maximal resolution the camera is able to

do at most 30 fps, while to reach 90 fps the camera needs to be operating at a resolution of 640 by 360 at most.
Our choice in resolution and framerate strikes a balance between the maximum resolution and fps the camera is able to output. To be of note is also the fact that to use higher resolutions and framerates the camera needs to be connected to using a USB 3.0 connection as otherwise the data throughput is not high enough to handle all the data transmitted by the camera. A fully detailed list of all possible configurations can be found within the data sheet provided by intel.

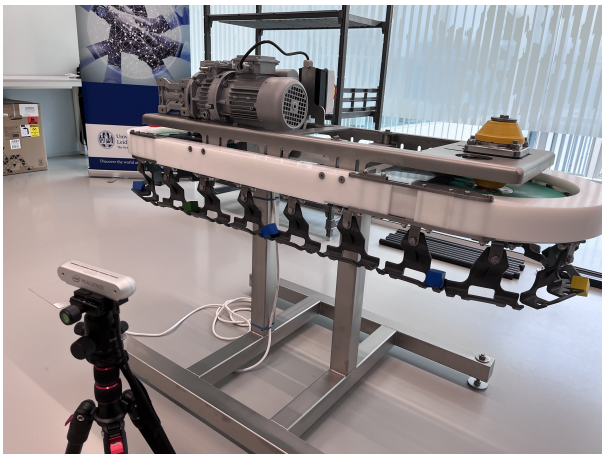
Figure 3: Intel RealSense D455 Depth and Colour Camera

3.2 Mini conveyor belt

A small conveyor belt used to test the different methods and implementations described in further sections. A picture of the conveyor belt can be seen in figure 4a. This conveyor belt allows for the simulation of realistic application for the object tracking and orientation estimation methods. The conveyor belt's speed can be adjusted with a valve-like controller that can be found on the motor located at the top of the conveyor belt.



(a) Conveyor belt without the camera



(b) Conveyor belt with the camera

Figure 4: Small conveyor belt from the LIACS TwinLabs

3.3 Object Detection

Taking inspiration from the study on detecting apples using a similar camera [AKU+22] the YOLOv5 [Joc20] architecture from ultralytics was chosen for the object detection pipeline. Being a one-shot neural network architecture made for use in video feed object detection makes it ideal for usage in a quality control environment. This architecture makes use of a three part system consisting of the backbone, the neck and the head [Joc20, Ulta]. The backbone is the main part of the neural network containing the convolutional layers, the neck of connects the backbone of the network to the head by making use of Spatial Pyramid Pooling - Fast (SPPF) [HZRS15] and Path Aggregation Network (PAN) [LQQ+18] to remove the fixed size needed by images and object instance segmentation respectively. Finally, the head is the part responsible for the final predictions given by the model.
To detect specifically the coloured cubes that were used for these experiments, the first step was to train the neural network on different images of coloured cubes. For this a mixture of a dataset provided by Jakob Sluf [Slo23], which was then further processed with some data augmentations to have a bigger dataset, and a home-made dataset were used. The data augmentations performed on these images include:

- Adding noise
- Removing random parts of the image (effectively creating black squares)
- Rotating the images

For the home-made dataset, images of the cubes on the conveyor belt were taken periodically as the camera was pointed at the conveyor belt. The images were then processed algorithmically to label the location of the cubes. This is done by thresholding images for each of the cube colours in the Hue, Saturation and Value (HSV) space, which gives us a binarized mask. The lower and upper bounds for Red, Green and Blue (RBG) that were used are as follows:

- Red: (170, 50, 50) - (180, 255, 255)
- Green: (40, 100, 50) - (80, 255, 255)
- Blue: (100, 150, 50) - (140, 255, 255)

After thresholding the images OpenCV's *connectedComponentsWithStats* function is used to clean the mask to only contain the cubes. With this final mask the label information can be computed by obtaining the normalized centre coordinates, height and width of the remaining shapes.
Making use of this mixed data set the YOLOv5 model is trained to be able to recognize the cubes in real-time and categorize their colour. By using a threshold of 45% for the prediction confidence cubes are reliably detected in most lighting settings, without detecting other cubic objects as cubes. To be of note is that a confidence threshold of 50% can be used in bright lighting for even better accuracy, however the model struggles with blue coloured cubes in darker lighting, resulting in the use of a threshold of 45% to increase the reliability of the model for different environments.

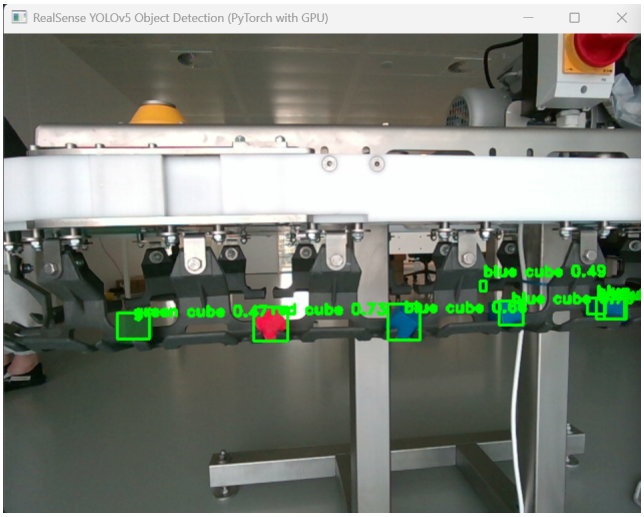


Figure 5: Cube detection using the YOLOv5 architecture and a confidence threshold of 45%. The green rectangle indicate the predicted pixel area occupied by the cubes and the text indicates the colour of the cube as well as the confidence in the prediction.

4 Experiments

To start experimenting with the Intel RealSense camera and its capabilities as a depth camera, the first step was to set up and train a model for the object detection as described in section 3.3.

4.1 Depth difference between edges

The initial idea to solve the estimation of the 3D orientation of cubes was to use the object detection pipeline as described in section 3.3 to get the location of the cubes. The location of the cubes is then looked at the corresponding pixels of the depth image to get the depth information of the detected cubes. To be able to correlate both images they are captured at the same time, same resolution and framerate. These exact parameters are mentioned at the end of section 3.1
By taking the depth information of the cubes, the orientation can be estimated by taking four depth points of the cubes. The four depth points are used in the following pairs:

- Leftmost point and rightmost point
- Topmost point and bottommost point

These points are used in pairs in order to calculate the difference between the extremities of the detected cube area. Since the cube detection is not perfect the area classified as cube might include some background and objects that are not the cube we can't just use a single point for the extremities. Instead we have to use an area of points around the centre of the extremities, which is then averaged out, while removing dead pixels if there are any in order to keep the values reliable. In addition to using an area of pixels we also move the centre of this area towards the centre of the cube detection, which further reduces the likelihood of including depth pixels that are not part of the actual cube. The depth average is calculated as follows:

$$P_{avg} = \frac{\sum_{i=1}^{n} p_i}{n} \tag{1}$$

where n is the amount of depth pixels within our extremity area, p_i is the i-th point within the extremity area.
The horizontal difference is calculated as follows:

$$H_{diff} = L_{avg} - R_{avg} \tag{2}$$

where L_{avg} and R_{avg} are the left and right depth averages calculated using the equation 1.
Using the value of H_{diff} the following conclusions can be made on the horizontal orientation of an object:

- 0 : no difference in depth between left and right means horizontal axis is flat.
- > 0 : left area has a higher value indicating that it is further away, meaning that the right side is tilted forwards.
- < 0 : right area has a higher value indicating that it is further away, meaning that the left side is tilted forwards.

Since the horizontal tilt is not enough to estimate the 3D orientation we also need a second axis, which is where the topmost and bottommost points are used to calculate the vertical difference. The vertical difference is calculated as follows:

$$V_{diff} = T_{avg} - B_{avg} \tag{3}$$

where T_{avg} and B_{avg} are the top and bottom depth averages calculated using the equation 1.
Using the value of V_{diff} the following conclusions can be made on the vertical orientation of an object:

- 0 : no difference between top and bottom means the vertical axis is flat.
- > 0 : top area has a higher value indicating that it is further away, meaning that the bottom is tilted forwards.
- < 0 : bottom area has a higher value indicating that it is further away, meaning that the top is tilted forwards.

By making use of the conclusion drawn from both H_{diff} and V_{diff} an estimation of the 3D orientation can be made. This estimation is however, still prone to error, an example for this would be the cube position illustrated in figure 6. In this situation the cube would be estimated to be flat, which would not be correct. To circumvent this issue a fifth area of points is introduced at the centre of the detection. By using this extra depth area a difference between the centre and the extremities can be calculated, which allows to determine when the cubes are in the specific orientations that would not be properly estimated with just H_{diff} and V_{diff}

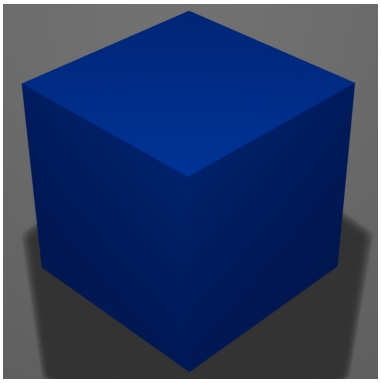


Figure 6: Render of a coloured cube with one of the corners pointing towards the camera.

4.2 Performance of the object detection

To be able to make use of the method as described in section 4.1 the object detection necessary to correlate pixel coordinates to the relevant depth values needs to be accurate. To this end the accuracy of the YOLO model trained on our cube dataset was measured using average precision (AP), for this we need to calculate the precision and recall of our predictions, which are calculated as follows:

$$\text{precision} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{5}$$

where TP are true positives, FP are the false positives and FN are the false negatives. TP and FP are determined using the Intersection over Union (IoU), which is given by:

$$\text{IoU} = \frac{A_{\text{truth}} \cap A_{\text{est}}}{A_{\text{truth}} \cup A_{\text{est}}} \tag{6}$$

where A_{truth} is the bounding box area defined as ground truth and A_{est} is the bounding box estimated by the neural network. AP is then given by:

$$AP = \int_0^1 P(R)dR \tag{7}$$

where $P(R)$ is the precision as a function of recall and the integral gives us the value for the area under the curve of that function. We interpolate this curve to 11 points in the same way as was introduced in the 2007 PASCAL VOC [EVGW+10] challenge to calculate the AP, which is given by:

$$AP = \frac{1}{11} \sum_{r \in [0, 0.1, \ldots, 1]} p_{interp}(r) \tag{8}$$

where the precision at each recall level r is interpolated by taking the maximum precision measured for a method for which the corresponding recall exceeds r [EVGW+10]:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{9}$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$ We use a threshold of 0.50, where any score above the threshold is considered a TP and below is FP. FN are missclassifications or missing detections in a given frame. For our tests two variables were tested at different values: the distance of the camera (in cm) and the amount of cubes. The distance of the camera allows us to measure how much screen space the cubes need to occupy in order for the detection to work accurately. The camera was placed at three different distances: 10cm, 15cm and 20cm. As for the amount of cubes, we tested one to three simultaneous cubes. The measurements were made on 60 images taken from the cameras live feed.

Amount of Cubes	Camera distance	Precision	Recall	AP
1	10cm	98.3	100	99.8
	15cm	96.6	96.6	99.5
	20cm	95.1	96.6	99.5
2	10cm	98.3	100	99.8
	15cm	94.8	96.5	97.2
	20cm	94.5	91.2	96.9
3	10cm	98.8	96.6	99.7
	15cm	95.9	94.2	96.9
	20cm	95.2	92.4	96.5

Through our tests we find that at any distance between 10cm and 20cm our YOLO network is capable of detecting between 1 and 3 cubes at a satisfactory AP above 96%. We estimate that these results apply to up to 5 simultaneous cubes. To be of note is that at higher distances the Precision might suffer from the network mistakenly classifying background objects as cubes. The Recall also suffers at greater distances, due to the fact that the cubes we have used are relatively small objects.

4.3 Limitations of the Depth Camera

The depth camera as described in section 3.1 is unfortunately not suited to make use of the method for orientation estimation as described in the previous section. This is mainly due to the fact that the depth camera is unable to get the entire cube's depth information as can be seen in figure 7. Due to the way that the camera calculates depth, which requires correlating pixels between the right and left imager, it has dead pixel areas, these are areas where the camera is unable to calculate the depth. The reason behind the occurrence of the dead pixels at the edges of objects is occlusion. When only one of the two imagers can see around the corner this causes the camera to be unable to calculate the depth, which in turn results in dead pixels. This makes it impossible to get an accurate reading of the needed areas of the cubes in order to calculate both H_{diff} and V_{diff}.
This is not the only issue that presents itself when trying to apply the initial idea to estimate the object's orientation. Due to the fact that the chosen object for these experiments is a small-sized cube, the fluctuations of the depth camera readings make the information unusable. The difference between a tilted cubes' edges is small enough, 1mm to 5mm difference between centre pointing outwards and corners, that these fluctuations make it so that the cube would seemingly constantly be changing orientation, when in reality it has not moved.
During these experiments the depth camera was used at a resolution of 640 by 480 to match the resolution of the object detection pipeline so that the pixels could be cross-referenced in order to find the points of interest from the cubes. This lower resolution than the camera is capable of

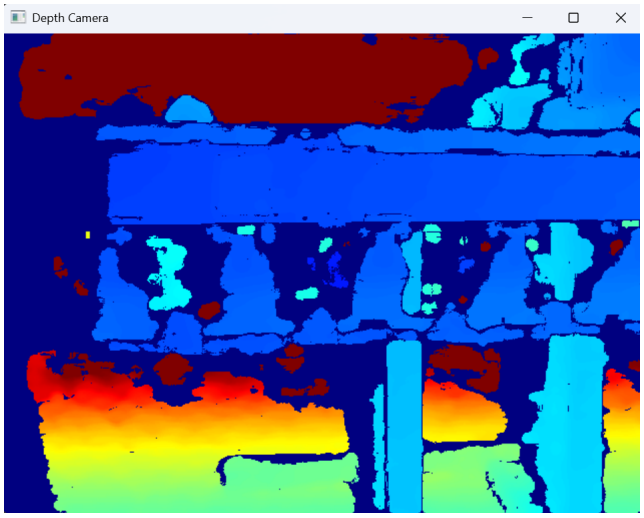


Figure 7: Depth camera output of the same frame as seen in Figure 5. The colour on the image indicates a relative distance, where the colder the colour the closer the object is to the camera. Dark blue indicates dead pixels with no value.

running was used due to two reasons, which are, as mentioned, to match the resolution of the object detection and due to the lack of a USB 3.0 cable. The USB 2.1 cable that was used in conjunction with the camera is unable to run at higher performance due to the lack of transfer speed of the older USB standard. Using a higher resolution could improve the accuracy of the depth image, however would likely not improve the fluctuations of the readings nor get rid of the dead pixels. While the lack of precision means that the method as described in section 4.1 cannot be implemented with this specific camera, it could potentially be implemented with a more accurate depth camera such as a light detection and ranging (LiDar) camera as these have shown promising results in other fields, such as transportation and 3D reconstruction [GB19, Kul22].

4.4 Usage of light difference

Due to the undesirable results of the previous method as described in section 4.1 the next idea was to make use of the difference between colour intensity caused by lighting. This method would make use of the fact that the parts of the cube facing away from a light source would receive less light and thus appear darker. This would mean that the colour would be less bright from the cameras perspective. This could then be used to calculate the difference in colour intensity similarly to how the depth difference was calculated. However, this idea was quickly abandoned due to its fragility, due to the fact that while the lighting of a room can be controlled to some extent it is hard to keep it constant.

4.5 Photogrammetry with QR codes

Another approach is to make use of a photogrammetry implementation by Temuge Batpurev [Bat]. This makes use of a QR coordinate system that is then translated to a camera coordinate system. The first step to make use of this implementation is to get the intrinsic camera parameters.

4.5.1 Intrinsic camera parameters

There are two main intrinsic camera parameters that are needed to make use of the QR code implementation. These two parameters are the distortion coefficients and the camera matrix. To get both of these parameters a calibration process is ran. To run the calibration a chessboard pattern is used, this pattern can be from any source but since the methods that are used to run the calibration are from the python OpenCV library it is recommended to use the chessboard pattern provided on the github repository from OpenCV.
After printing out this pattern images need to be taken using the camera with the chessboard in the frame. This requires about 5 images at the very minimum, but roughly 10 to 15 are recommended. OpenCV's built-in functions can then be used to find the chessboard on the images. We first make use of OpenCV's function to find a chessboard pattern within an image. If this function successfully finds the chessboard it returns the locations of the corners of the chessboard pattern, which can be seen on figure 8. We then attempt to improve the detection of the corners, once again

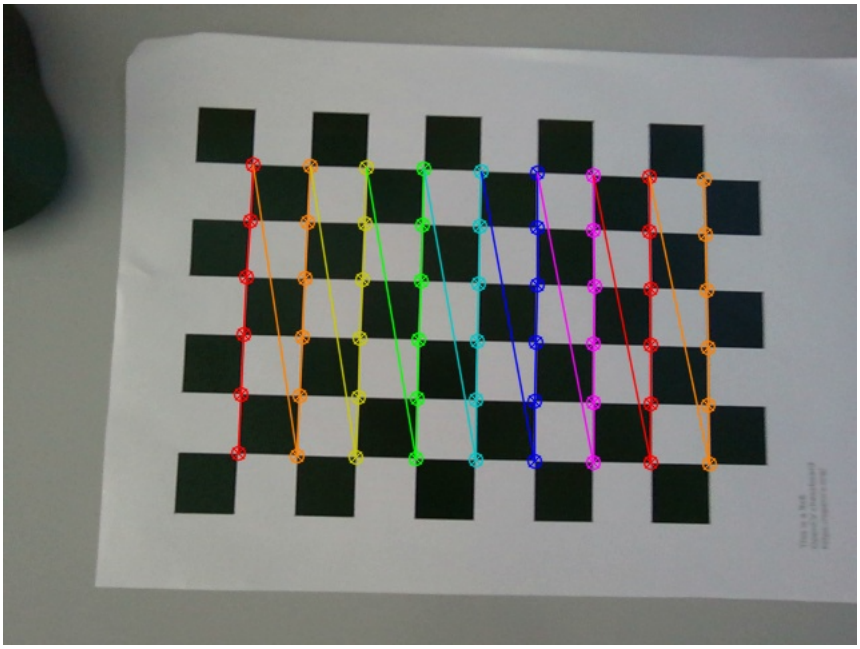


Figure 8: Detected chessboard pattern corners connected by lines indicating their position in the grid coordinates. The red corner at the bottom left is the origin point and the end point is the orange point at the top right, creating a 2-dimensional coordinate system

with OpenCV's built-in methods, before using these corners to use OpenCV's camera calibration. OpenCV's camera calibration makes use of the corners coordinates to then calibrate the camera and return the camera matrix, distortion coefficients, rotations per frame and translations per frame. On top of these the function also provides an root mean squared re-projection error (RMSE), which should be below 2 and ideally below 1 for the best results. The re-projection error
The reason that we require both the distortion coefficients and the confusion matrix is that cameras introduce distortion to the images. This distortion comes in two types: radial distortion and tangential distortion.
Radial distortion is what causes straight lines to be curved on images and becomes larger the

further away these straight lines are from the centre of the camera. Radial distortion, as discussed previously, can come in three subtypes of distortion [vW]: barrel distortion, pincushion distortion and mustache distortion. The type of distortion depends on the lens used by the camera. Radial distortion can generally be represented as follows:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{10}$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{11}$$

where x and y are pixel coordinates, k_n is the n^{th} radial distortion coefficient and r is the Euclidean distance between the distorted image point and the distortion centre [dVLG08]. To be of note is that the radial coefficient k will typically be negative for barrel distortion and positive for pincushion distortion.
Tangential distortion also occurs because the camera lens is not perfectly aligned to the image. This type of distortion causes parts of the image to look closer than they are in reality. Tangential distortion can be represented as follows:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{12}$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{13}$$

where p_n is the tangential distortion coefficient.
Using this information we can get the distortion coefficients which are given by:

$$\text{Distortion coefficients} = (k_1, k_2, p_1, p_2, k_3) \tag{14}$$

In addition to the distortion coefficient we also need the camera matrix, which represents intrinsic parameters. These parameters are unique to each camera and are composed of the focal length (f_x, f_y) and optical centres (c_x, c_y) of a camera. With these two parameters we can create a camera matrix that is then used to remove distortion caused by the cameras specific lens. Since the focal length and optical centres of each camera is unique, their camera matrix is also unique, but can be reused on images taken by the same camera. This matrix is represented by:

$$\text{camera matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

4.5.2 QR code orientation

The first step in determining the orientation of a QR code is to detect the QR code in a picture. For this we make use of OpenCV's built-in function to detect QR codes. This function returns two values: the first value is a boolean indicating if a QR code was found and the second value is either empty if no QR code was found or it is a list containing the pixel coordinates of the four corners of a QR code as shown in Figure 9 By making use of the four corners a coordinate system can be defined for the QR code. This coordinate system uses the corner labelled as 1 in figure 9 as the origin for the coordinates. We define the x-axis to be pointing from corner 1 to corner 4 and our the y-axis to be pointing from corner 1 to corner 2. The reason these axes are defined as such is so that the z-axis is pointing outwards from the QR code. To create the coordinate system for the

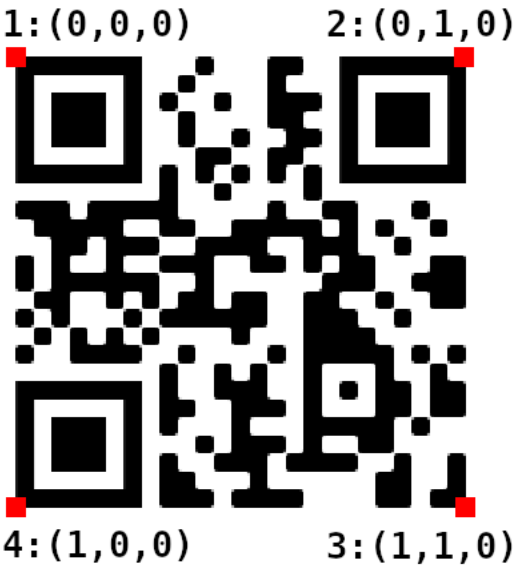


Figure 9: Corners of a detected QR code with their respective coordinates within the defined coordinate system.

QR code the 4 corners are given their own 3 value tuple indicating x, y and z position within the coordinate system defined for the QR code.
With this information, it is possible to turn the 2-dimensional pixel coordinates from the detected QR code into 3-dimensional axes by making use of OpenCV's Perspective-n-Point (PnP) [MUS16] solver. This solver gives us a rotation vector and a translation vector, which can then be used to reproject the axes from the QR code's coordinate system to camera pixel values, which allows us to display them on a 2D image or video feed.

4.5.3 Ideal configuration and limitations

By making use of this implementation and printing out small QR codes that were then stuck to the cubes, as seen in figure 10, the robustness and the limitations of this method were tested. Since this method relies on finding the QR codes to get an orientation, this means that we cannot use Intersection over Union (IoU) for our calculation of the average precision, we can however use a similar threshholding method to determine true positives from false positives. For this, instead of determining a ground truth bounding box for each frame, a ground truth for the three axes that should be projected from the QR code are defined. The difference between the defined axes and the estimated axes can then be used as follows:

$$\sum_{i=0}^{3} |(x_{true}, y_{true})_i - (x_{est}, y_{est})_i| \tag{16}$$

where $(x_{true}, y_{true})_i$ represents the ground truth i^{th} point on the image and $(x_{est}, y_{est})_i$ is the equivalent estimation of that point. The three axes are constructed by four points and we can use the sum of the absolute value of the difference of each of these points to know by how much the estimation is off. While this is not a perfect way of measuring this it allows us to have a comparable

metric to our object detection accuracy evaluation. For this use case we use a threshold value of 80. This value allows each x and y to be off by about 10 pixels before the orientation estimation is classified as a false positive. With this the average precision can be calculated. To test the robustness of this method the same parameters as described in section 4.2 were tested. However, for this method smaller distances were used for the camera as the QR code needs to occupy at least a certain percentage of the screen space for the program to be able to solve the PnP.

Amount of Cubes	Camera distance	Precision	Recall	AP
1	5cm	98.3	100	99.8
	9cm	96.7	49.1	45.1
	10cm	0	0	0
2	5cm	95.2	86.9	80.6
	9cm	98.1	43.7	36.1
	10cm	0	0	0
3	5cm	92.0	26.1	25.8
	9cm	90.9	5.6	8.3
	10cm	0	0	0

Through these tests the limitations in the amount of simultaneous QR codes that can be read as well as the needed size the QR codes need to be were deduced as these were the main factors contributing to the robustness of the QR code reading. This implementation of the QR code

Figure 10: Cube with a QR code stuck on top. The content of the QR code does not impact the use of this method and can be anything. In this case reading the QR code returns the message "I am a cube"

orientation, which is a slightly modified version of the original [Bat] in order to work with the Intel RealSense D455 and with the newer version of OpenCV, is able to accurately scan one to two QR codes at the same time and get their axes in real time at sixty frames per second. Lowering, the frames per second to thirty improves the robustness slightly as the script gets more time to calculate the axes and solving the PnP in between frames, however the limit remains the same.

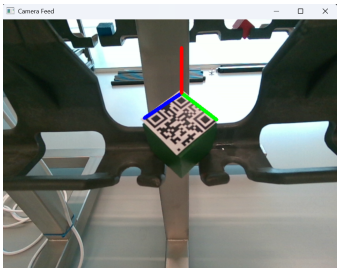


Figure 11: Picture of a QR code on a cube with the respective axes indicating the orientation of the cube. The red axis indicates the z axis, the blue line indicates the x axis and the green line indicates the y axis.

Increasing the amount of QR codes results in a decrease in detections although small, it does decrease the recall. On the matter of the size needed by the QR codes, this is mostly due to the amount of space or pixels that these need to cover in the cameras view. Without enough pixels the script is unable to find the QR code or solve the PnP, which means that the orientation cannot be read. Thus, on the used camera resolution of 640 by 480 the QR code's axes were most consistent when the QR code is taking up between 4.4% to 70% of the image, meaning that it is in our case between 1cm to 5cm away from the camera. 4.4% of the screen equates to roughly a 110x110 pixel area at a resolution of 640x480. When the QR code takes up more screen space, there are either parts of it out of frame or it is pointing straight at the camera, which in the first case results in an unreadable QR code and in the second case results in the PnP solver not being consistent with the z-axis and it changing constantly instead of pointing towards the screen. When the QR code covers between 1.7% and 4.4% of the image, which for our QR code is a distance of 9cm to 5cm, the reading of the QR code becomes unstable, due to the lower amount of pixels. This results in flickering readings due to the inability to find or read the QR code from the camera feed, causing the discrepancy between the recall and precision, as when the QR code is able to be detected the orientation estimation is accurate, however in about half the frames the QR code is unable to be solved for PnP, resulting in the big amount of false negatives. At less than 1.7% of the cameras image, which is an area of about 75 by 70 pixels, the QR code cannot be found and read, except for one or two frames within a second. This means that ideally the QR codes should be at a distance or size that makes them take up between 4.4% to 70% of the image for the readings to be usable. Using a higher resolution for the camera output, for example the maximum resolution that the Intel RealSense D455 can output which is 1280 by 720, decreases the lower bound for these thresholds as the image contains more pixels making it possible to either use smaller QR codes or to have the QR codes be further away.

5 Conclusions and Further Research

Concluding, we can determine that the best method for object detection and orientation estimation using the Intel RealSense D455 camera is a method not involving the use of the depth imaging that the camera provides. While the use thereof in conjunction with a neural network for object detection is interesting as it would be able to handle more objects simultaneously, the lack of precision and full coverage of the image due to dead pixels make it hard, respectively impossible to use for small

object orientation estimation. The depth cameras capabilities could prove sufficient for bigger objects as the noise from the camera would have less impact. On the other hand, using methods for pose estimation with just a standard video feed are more compelling. These are computationally less expensive, compared to the use of a neural network especially, and provide good precision for orientation estimation. This method however suffers from the struggle to detect multiple codes simultaneously at a reliable rate. It also requires the camera to be a lot closer to the objects due to needed size of the QR code, which is not necessarily an issue, but does need to be considered.
Further research into the usage of the axes calculated through the QR code coordinate system, described in section 4.5.2, for practical use is needed. In particular how to determine if an object is oriented inappropriately, through the usage of a fixed world origin point decided by for example a QR code placed at a fixed location which can be used for this purpose. To distinguish the QR code that has been determined as the world origin point the content of the QR code could be used, since QR codes are able to contain information and can be easily read from if detected. Another option would be to use just the axes from the QR codes themselves and based on the direction of these within the camera pixel space determine if the object is oriented in a problematic way or not. This could potentially be a more viable way of applying this implementation as it would only rely on the QR codes themselves and doesn't need to read and interpret the contents of the QR codes, which introduce more delay in the processing.

References

[ACPT22] Fabio Arena, Mario Collotta, Giovanni Pau, and Francesco Termine. An overview of augmented reality, 2022.

[AEP+22] J. Andrew, Jennifer Eunice, Daniela Elena Popescu, M. Kalpana Chowdary, and Jude Hemanth. Deep learning-based leaf disease detection in crops using images for agricultural applications. *Agronomy*, 12, 2022.

[AKU+22] Nikita Andriyanov, Ilshat Khasanshin, Daniil Utkin, Timur Gataullin, Stefan Ignar, Vyacheslav Shumaev, and Vladimir Soloviev. Intelligent system for estimation of the spatial position of apples based on yolov3 and real sense depth camera d415. *Symmetry*, 14(1), 2022.

[Bat] Temuge Batpurev. Orientation of QR code using OpenCV — temugeb.github.io. https://temugeb.github.io/python/computer_vision/2021/06/15/QR-Code_Orientation.html. [Accessed 29-05-2025].

[BAX19] Mahdi Bahaghighat, Leila Akbari, and Qin Xin. A machine learning-based approach for counting blister cards within drug packages. *IEEE Access*, 7, 2019.

[Bro66] DC Brown. Decentering distortion of lenses. *Photometric Engineering*, 32, 1966.

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[CFA+11] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*, 51, 2011.

[dVLG08] Jason P. de Villiers, F. Wilhelm Leuschner, and Ronelle Geldenhuys. Centi-pixel accurate real-time inverse distortion correction. In *Optomechatronic Technologies 2008*, volume 7266, 2008.

[DZW20] Lixuan Du, Rongyu Zhang, and Xiaotian Wang. Overview of two-stage object detection algorithms. In *Journal of Physics: Conference Series*, volume 1544, 2020.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[Fos93] Eric R. Fossum. Active pixel sensors: are CCDs dinosaurs? In Morley M. Blouke, editor, *Charge-Coupled Devices and Solid State Optical Sensors III*, volume 1900, pages 2 – 14. International Society for Optics and Photonics, SPIE, 1993.

[GB19] Suliman A. Gargoum and Karim El Basyouny. A literature synthesis of lidar applications in transportation: feature extraction and geometric assessments of highways. *GIScience and Remote Sensing*, 56, 2019.

[Gir15] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37, 2015.

[inta] intel. Introducing the Intel® RealSense™ Depth Camera D455 — intelrealsense.com. https://www.intelrealsense.com/depth-camera-d455/. [Accessed 01-06-2025].

[Intb] Intel. Stereo depth cameras for mobile phones — dev.intelrealsense.com. https://dev.intelrealsense.com/docs/stereo-depth-cameras-for-phones.

[Joc20] Glenn Jocher. Yolo v5 by ultralytics, 2020.

[KCF23] Abhishek Kumar, Mickael Castro, and Jean François Feller. Review on sensor array-based analytical technologies for quality control of food and beverages, 2023.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60, 2017.

[Kul22] Marek Kulawiak. A cost-effective method for reconstructing city-building 3d models from sparse lidar point clouds. *Remote Sensing*, 14, 2022.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, 2016.

[LGG+17] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-October, 2017.

[LLT+23] Yuzhao Liu, Wan Li, Li Tan, Xiaokai Huang, Hongtao Zhang, and Xujie Jiang. Db-yolov5: A uav object detection model based on dual backbone network for security surveillance. *Electronics (Switzerland)*, 12, 2023.

[LQQ+18] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[MUS16] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. Pose estimation for augmented reality: A hands-on survey, 2016.

[Pen00] J. B. Pendry. Negative refraction makes a perfect lens. *Physical Review Letters*, 85, 2000.

[pjr] pjreddie. GitHub - pjreddie/darknet: Convolutional Neural Networks — github.com. https://github.com/pjreddie/darknet. [Accessed 05-05-2025].

[QL99] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21, 1999.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, 2016.

[RHGS17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 2017.

[SKJ19] Zahra Soleimanitaleb, Mohammad Ali Keyvanrad, and Ali Jafari. Object tracking methods:a review. In *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 282–288, 2019.

[Slo23] Jakub Slof. red,green,blue cube detection dataset. https://universe.roboflow.com/jakub-slof/red-green-blue-cube-detection, aug 2023. visited on 2025-05-13.

[Ulta] Ultralytics. Architecture Summary — docs.ultralytics.com. https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure.

[Ultb] Ultralytics. YOLO12 — docs.ultralytics.com. https://docs.ultralytics.com/models/yolo12/#where-can-i-find-usage-examples-and-more-detailed-documentation- [Accessed 01-06-2025].

[vW] PA van Walree. Distortion (archived). https://web.archive.org/web/20090129134205/http://toothwalker.org/optics/distortion.html. [Accessed 29-05-2025].

[YSP+22] Yogitha, S. J. Subhashini, Dharshana Pandiyan, Shivini Sampath, and Surabhi Sunil. Review on lane and object detection for accident prevention in automated cars. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Smart Energy, ICAIS 2022*, 2022.

[YY21] Ruixin Yang and Yingyan Yu. Artificial convolutional neural network in object detection and semantic segmentation for medical imaging analysis, 2021.

[ZLW+21] Yifan Zhang, Xu Li, Feiyue Wang, Baoguo Wei, and Lixin Li. A comprehensive review of one-stage networks for object detection. In *2021 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–6, 2021.

[ZZL+19] Lei Zhou, Chu Zhang, Fei Liu, Zhengjun Qiu, and Yong He. Application of deep learning in food: A review, 2019.