Danmarks Tekniske Universitet

DTU

Final Project

31392 Perception for Autonomous Systems

Group 9

Amir Abdul-Rahim Sabbah	s125014
Anders Helbo	s144020
Eduard Maximilian Fiedler	s210134
Jacob August Xander Ottensten	s213820
Mouadh Sadani	s220726

May, 2022

Table of Contents

1 Introduction

As industry technologies advance, it becomes more important to integrate them in more dynamic environments. The scenario with which the data is presented is one where tracking and identification of moving objects on a conveyor belt is to be performed. This is to be fulfilled in 3-dimensional space and despite occlusions. The provided data includes a sequence of images taken from a stereo camera setup with a calibration pattern and objects being conveyed with and without occlusions.

In this report, the method for the applied approach will be presented (Sections 2- 4) and the performance will be discussed (Section 5). The result can be seen here (viewing at 2x speed recommended): `https://youtu.be/AFXVpCuljCg`.

2 Stereo Calibration and Rectification

The goal in this section is to transform the images in such a way as to allow triangulation of 3D position of points, based upon their position in the left and right camera images. To do this, two things are required: the images need to be undistorted and they need to be stereo rectified. Removing distortion makes sure that lines that are straight in the real world are also going to be straight on the images and that relative distances between points are not going to be stretched in some parts of the images. Stereo-rectification transforms the images to achieve rectified epipolar geometry with epipoles at infinity, creating a virtual camera angle where the camera image planes are perfectly square to each other.

First step is to calibrate the individual cameras and determine a camera matrix and the image distortion for each camera. This is done using images of a known calibration pattern; in this case a checkerboard pattern (see Figure 1). The corners in the checkerboard are saved and used the generate the intrinsic camera matrix and distortion matrix using the 'calibratecamera()' function from OpenCV. These matrices can then be used to generate a new scaled camera matrix using 'getOptimalNewCameraMatrix()'. Now the image can be undistorted as seen in Figure 2. Here it is clear that unlike Figure 1 the edge of the table is no longer curved, however, some of the image has been cropped away so that only the nicely undistorted bit remains. It can also be seen that the images are not quite aligned.

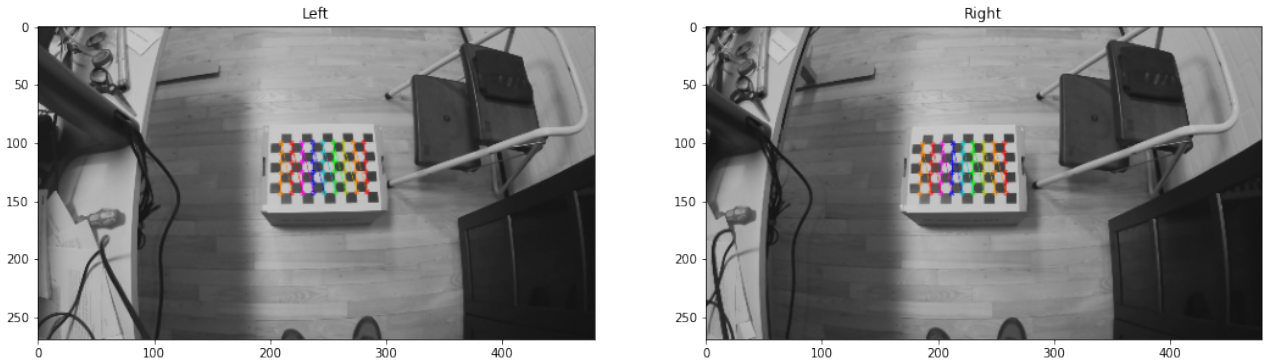

Figure 1: Using calibration images with known pattern to find corresponding points in images.

Stereo calibration can be done using the points found in the calibration images as well as the intrinsic camera matrices and the distortion matrices for both cameras. The 'stereoCalibrate' command produces a few different matrices including the essential matrix, the fundamental matrix and the rotation and translation matrices. The translation and rotation matrices can now be used along with the intrinsic camera and distortion matrices to calculate a rectification transform and a projection matrix

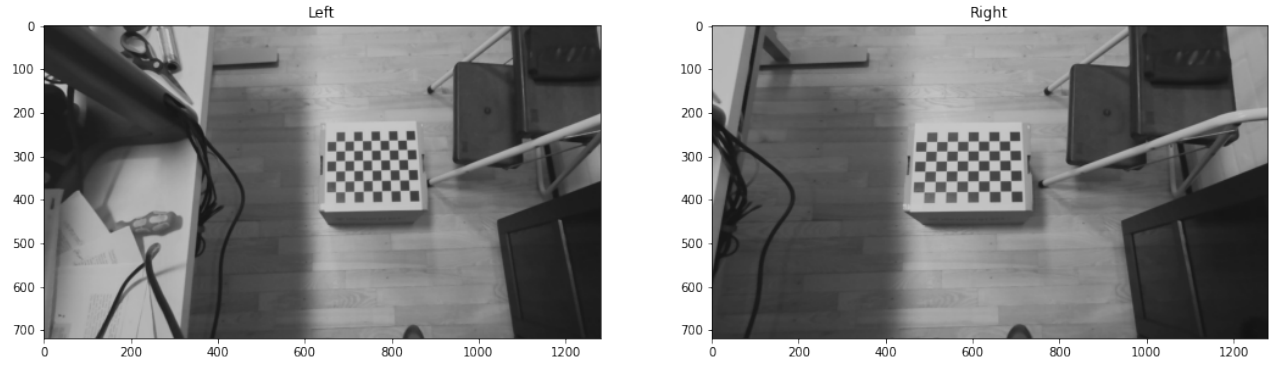

Figure 2: Undistorted images cropped to only show the square part of the image.

for each of the cameras using 'stereoRectify'. The rectification transform transforms points in the unrectified image into the new rectified coordinate system. The projection matrix then projects points given in this new coordinate system in to the rectified image. These can, together with the intrinsic and distortion matrix, be used to create a stereo map for each camera using 'initUndistortRectifyMap'. Once a stereo map has been found for both cameras, images can be translated into rectified stereo images using 'remap'.

In Figure 3 it can be seen that the images are much better aligned than before. This is most easily seen by looking at the box in the images in Figure 2 and Figure 3. In Figure 3 the boxes are more similar in size than in figure 2. We can also see that the images have not been cropped as close in the stereo rectified images, this is not an effect of the process, but rather it is a compromise to avoid losing to much of the right side of the conveyor belt in the video.

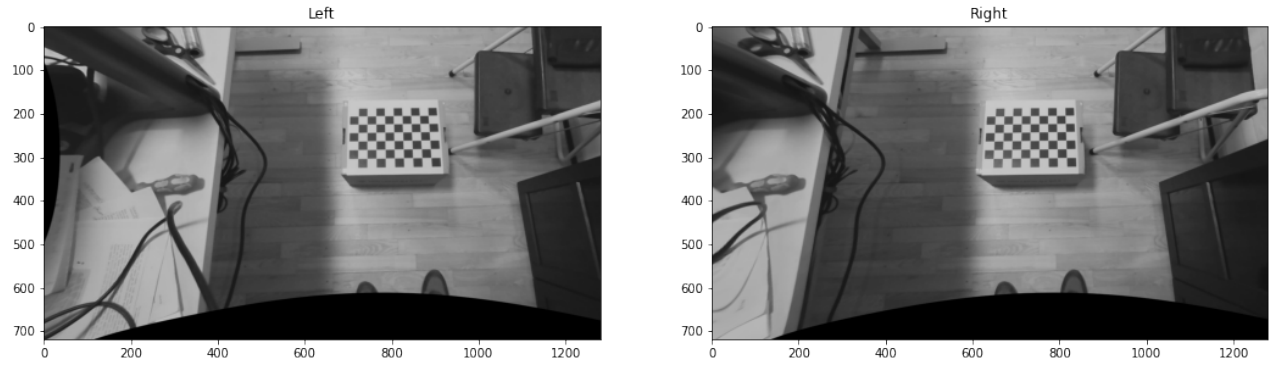

Figure 3: Stereo rectified images. a bit of black has been allowed to stay in the image in order to keep more of the right side of the conveyor belt in frame in the videos.

3 Object Tracking

To track objects moving on the conveyor belt, they were first isolated from their surroundings. Then, by applying a Kalman filter, the movement of the object was estimated and stabilized such that a prediction of the object's position could be used when it was occluded. Finally, knowing where the object was in both the left and right images, allowed for the extraction of its position in 3-dimensional space.

3.1 Generating a mask on the object

To allow for further analysis of the position of the object, it has to first be isolated from the rest of the environment. For this case, the approach outlined by Thapa et al. [1] was adapted (see Figure 4) to apply a mask that focuses on the object on the conveyor belt.

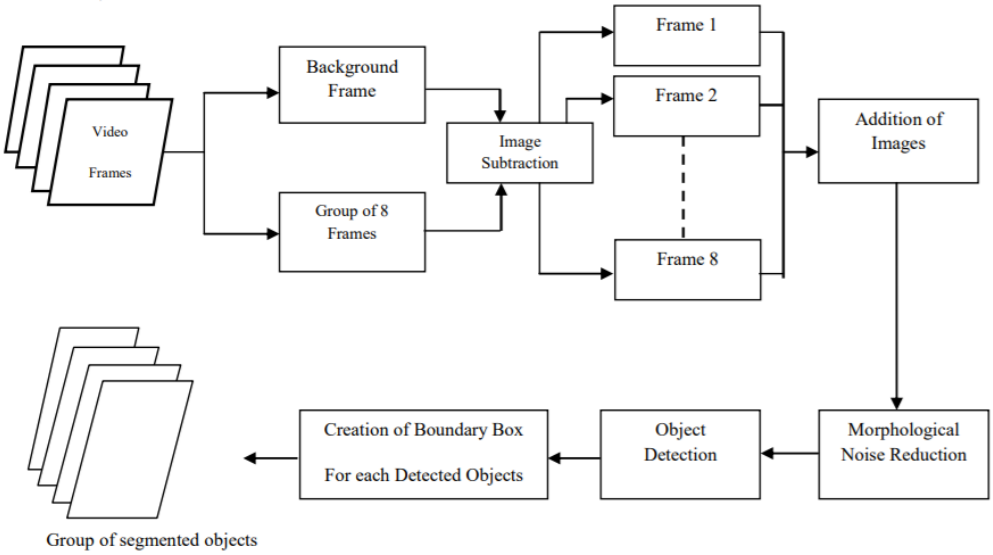

Figure 4: Procedure for segmenting moving objects from videos. Taken from Thapa et al. [1].

This method of segmentation is a simple, yet robust, way of detecting objects based on background subtraction. In background subtraction, the idea is that by having a background model, subtracting this from an altered image with the same background, allows the alteration to be isolated. Thapa et al. [1] have shown that the robustness of a simple background subtraction can be feasibly increased without adding complexity. This is achieved by using a group of eight consecutive frames that are subtracted from the background and then summing these frames, to provide information on the maximum displacement of a moving object within those frames.

For this project, as opposed to Thapa et al. [1], HSV images were used to segment the moving object. It was found that using only gray images caused the black cardboard box to become undetectable. First, 300 randomly selected images from the respective image sets were merged to create median background images. Next, iterating through the left and right image sets, the most recent images were subtracted from their respective backgrounds and saved to a group containing the past eight subtracted frames in the sequence. These were then summed to give a single frame that was then blurred to eliminate some noise, converted to gray, and then thresholded to create a binary image. Then, by applying opening, a mask was obtained for the current frame that isolated the moving object from the image. Furthermore, as subsequent analysis required the location of this mask, the center of the generated blob was extracted.

3.2 Applying a Kalman filter

The Kalman filter optimises the observation of a system based on a model and its measurement over time (see Figure 5). For this project, as the objects had to be tracked despite occlusions, the Kalman filter was required to estimate a position.

The Kalman filter operates by estimating a model's dynamics and combining this with measurements of the actual model. This is realized by acknowledging the uncertainty of both the estimated system and measurement, treated as Gaussian, to predict the real model's states for the next time

step based on the current time step's information. Applying the Kalman filter equations results in an optimal observation that is more certain than both the estimation and the measurement of the actual model, based on the provided information.

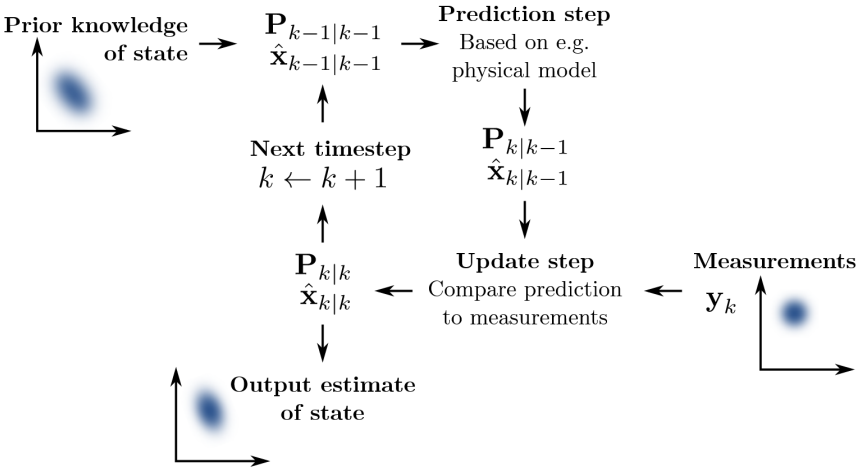

Figure 5: Basic process of the Kalman filter. Taken from Aimonen [2].

For the purpose of this project, the center of the blob created from the mask was measured to determine the object's actual position in xy-coordinates in the image. To estimate this position, in case the object became occluded or the mask briefly unreliable, a Kalman filter was applied to improve object tracking. To do this, the change in the object's position was estimated with the following model:

$$\begin{bmatrix} \ddot{x} \\ \dot{x} \\ \ddot{y} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} + u\,, \tag{1}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}\,, \tag{2}$$

where the input u was used to influence the velocities based on the angle of the conveyor belt, in relation to the camera's perspective and the speed at which it was operating. Setting this input allowed for a better estimation when the object became occluded. Additionally, the initial state of the filter was set to the position at which the objects most likely began on the belt, the initial uncertainty was chosen to be large enough to converge quickly, and the measurement uncertainty was tuned to give good tracking performance. The Kalman filter was initialised every time a tracked object was not within the pixel range at which the conveyor belt was seen in and the update step was ignored when a measurement could not be made. This made it possible to reliably apply the Kalman filter, as objects were placed on the belt when it was empty.

By combining the size of the mask as the object progressed on the conveyor belt and the tracked position of the object, it was possible to generate a bounding box onto the moving object. This was important to define an area threshold, where when 60% of the largest bounding box made by the moving object was lost from occlusion, the Kalman filter would forgo the update such that the predicted state would not lose too much speed.

3.3 Tracking the object in 3-dimensions

Having calibrated the cameras and rectified the images, correspondent points in the images can be used to find their position in space. As the output of the Kalman filter tracked the position of the moving objects in the left and right camera images, assuming this point as correspondent, allows the triangulation of the object into 3-dimensional space.

In OpenCV, the command `triangulatePoints()` can be used to find the projection of the points in either image into 3D space. It requires the projection matrix of either camera, obtained from calibration and rectification, and the measured points of the moving object, obtained from the Kalman filtered tracking. The projection matrix describes where a point in space will appear in the image. By assuming the tracked points are correspondent, their point in 3-dimensional space is assumed to be the same. Therefore, by using a direct linear transformation (DLT) method, the position of the moving object in 3-dimensional space can be estimated [3, pg. 312].

This method was applied for the project, as it provided a fast estimation based on previously extracted information, over generating a disparity map at every frame. The command `triangulatePoints()` outputted a position vector with a scaling factor and mapped the points to a coordinate system where the x-axis pointed to the right, the y-axis pointed downwards, and the z-axis pointed into the image. The moving objects were tracked with respect to the perspective of the camera onto them, which was most likely tilted downwards at the belt. This explains why the y-position was trending downwards as the objects moved closer, despite the objects being level on the conveyor belt. The true mapping into 3-dimensional space, assuming the conveyor belt was flat, would not be verifiable as no true measurements were provided.

The result of the mask combined with the Kalman filtered tracking and positional estimation are shown in Figure 6.

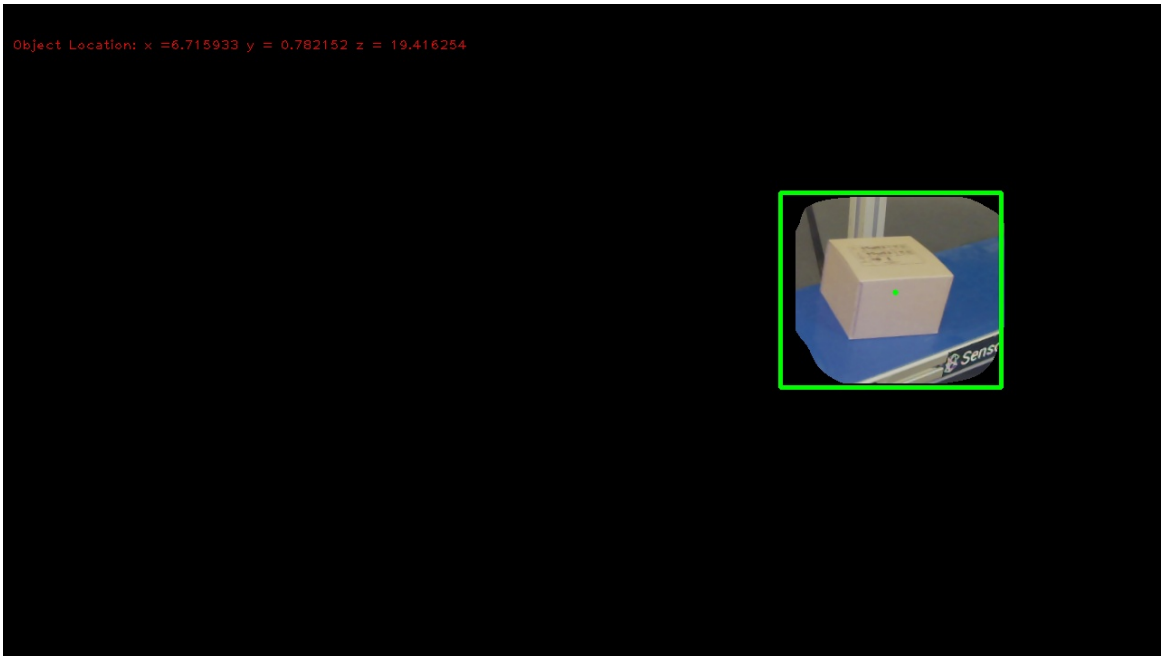

Figure 6: Result of the object tracking procedure used for the tracking of the objects moving on the belt. Shown is an image from the left camera, where the brown cardboard box was moving on the conveyor belt. Displayed is the generated mask onto the image, the Kalman tracked point (green dot) and calculated bounding box (green box), as well as the estimated position in 3D (top left).

4 Classification

Classification is a type of Supervised Machine Learning, where the goal is to determine a target value from a set of classes, depending on some input. Typically, the input is a vector which gets mapped to the output class using a *learned* inference function. The learning in this case constitutes analyzing the set of input vectors, and finding the class boundaries in a (possibly) multidimensional feature space. It is supervised, because the training data is labeled, and thus we have a preconceived notion of how this data should be separated and categorized. The idea is that, by showing the algorithm a small subset of the things we are trying to classify, it can learn a mapping that generalizes to cases it has not seen before. Intuitively, we are trying to maximize separation of the classes in the feature space, such that there, for each sample, seen or unseen, exists a strong sense of belonging (i.e. to some class).

In this project, we worked with image classification. The final model should be able to classify unseen images into 3 object classes: cups, books and boxes. We train a Linear Discriminant Analysis classifier.

4.1 Data gathering and preprocessing

We created our own dataset of images that we captured ourselves. We took 600 images of different books, boxes and cups, at many different angles. Our data preprocessing pipeline was as follows: 1) background removal 2) resizing and 3) edge detection. We describe these steps below.

It quickly became apparent that the background of the training images would become a problem in our setup. It is easy for us humans to distinguish the object from the background, but that is a difficult task for computers. This meant that the trained algorithm could end up extracting a lot of information from the background, when in fact we wanted to ignore it. That is why we chose to remove it altogether. We did this by hand. We also shrank all images down to a size of 128×128. The purpose of this was two-fold: 1) in order to decrease computational load and 2) to ensure all input data are of the same dimensionality.

Figure 7: Training images examples

Using the pictures with no background, we detected the edges of the objects using the Canny algorithm in OpenCV: `cv2.Canny()`. This gave us accurate edge representations of the objects. Some examples can be seen in Figure 7.

4.2 Linear Discriminant Analysis

LDA is an interesting method in Machine Learning, seeing as it can be used for both dimensionality reduction and classification. We reduce the dimensionality to just 2 (`n_components` in the code below). This enables visualisation of the representations of the learned classes in a 2-dimensional space (see Figure 8). This is not possible with black-box deep neural network techniques, although they might be more accurate. LDA is closely related to Principal Component Analysis (PCA), another form of dimensionality reduction. But LDA differs because it considers the labels of the input data, and tries to maximise the separation of the classes, whereas PCA does not regard the labels. We trained the LDA classifier on the binary edge representations of the training images, using the following parameters:

```
LinearDiscriminantAnalysis(n_components=2, solver='eigen', shrinkage=0.1)
```

4.3 Inference and results

The test input to our model was images of objects (cups, books and boxes). As described in the previous sections, we detect the objects in the image, and create a bounding box around them. These bounding boxes are then used to crop a subset of the pixels out of the whole image. We use these smaller images as input to the preprocessing pipeline. Since we cannot remove the background by hand at inference time, we simply omit this step and go directly to the resizing and edge detection. Once the edges are detected, it is relatively easy to perform inference using the `predict()` method. Mathematically, LDA assigns the sample x to the class whose mean is the closest in terms of Mahalanobis distance, while also accounting for the class prior probabilities [4].

In Figure 8, we visualize the separation of classes performed by the LDA model. It transforms the input data (consisting of $128 \times 128 = 16384$ dimensions) down to just 2 dimenesions using the learned coefficients/weights. We can see that the model is extremely effective at separating the training data, the dark points in Figure 8. The lighter points are the testing images, which scatter inbetween the reference points. We can see how especially the blue points, representing images of books, are mostly in their own region. There is a bit of a clash though in the lower left of the plot, where the boxes and cups overlap. This means that the model is more uncertain when classifying those.

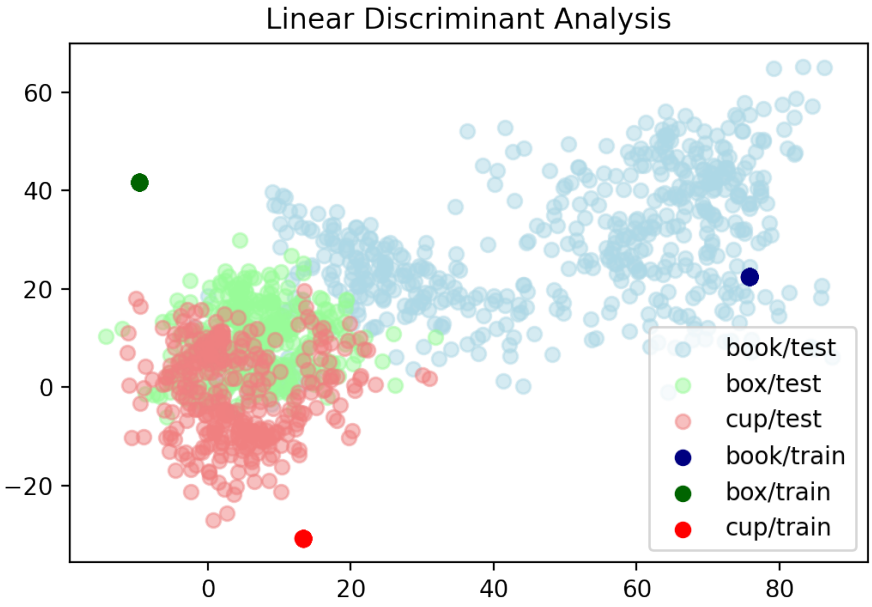

Figure 8: Projection of training and test data using the linear discriminants to maximise separation between classes.

Overall, our model achieves an overall test accuracy of 68%. When showing the label in the video, we use a "voting" system on top of the raw predictions. Instead of just showing the predicted class at each frame, which may momentarily change, we show the label which the classifier guessed most frequently in the first 14 frames of its tracked state. This extra step prevents noise from outlier classification errors, and enables the class label to be shown even when the object is occluded. This compound analysis exploits the *context* available in time continual video. Context is an interesting and distinct aspect of *video* image classification, which we discuss in Section 5.3. The voting step can be seen as violating the unseen nature of the test set, but that is why we report the overall accuracy as well which is 68%. We allow this inference exclusive step because it is separate from training.

5 Discussion and Conclusion

5.1 Improving tracking accuracy

As mentioned, the tracked points using the Kalman filter were not a reliable correspondence. This was especially exacerbated when the objects came out of the occlusion. As a result, triangulating these points does not produce a reliable 3-dimensional tracking of the object.

A better approach would be within the bounding boxes, if it was possible to remove the object from the background reliably, to use features of the object that could be matched between the left and right images. This would provide better correspondences of the object in both images as it moved through space, giving more certainty that a correct triangulation is being obtained.

In an industry setting, if tracking were vital despite occlusions, an additional camera could be implemented from another viewpoint. This would reduce the possibility of an occlusion blocking the view of cameras pointing in the same direction and would also improve the ability to track the object via sensor fusion.

5.2 Representation of data in training

The concept of *garbage in, garbage out* is common in Machine Learning. If the data used for training is not good, then the inference accuracy will never be good either. But what constitutes good training data? In many cases [5][6][7], good training data means *a lot* of training data — number of samples being many hundred thousand. If our primary objective is to maximize the models ability to generalize, then having a lot of diverse training samples is of utmost importance.

Our access to data was limited to the open source libraries available online, and our own ability to take photographs of the relevant objects. While these were both viable approaches, stitching together 100's of thousands of examples was not possible. Seeing as we had limited access to data, our approach instead relied on creating a representation of the data which was simpler and more general. We did this by finding the edges using the Canny algorithm. The intuition being, as noted in [8], that even small children can recognize objects from simple line drawings.

This idea of breaking an image down into smaller parts (edges and line segments) is heavily utilised in state-of-the-art Convolutional Neural Networks (CNNs). In such deep models, the kernels filter the image and initially find vertical or horizontal edges. Intuitively, deeper layers will combine these edges into line segments, curves and shapes. The model will finally learn how certain shapes combine to create the objects in the image [9]. As such, our approach can be viewed as a *poor man's CNN*.

To improve our classification model, a few steps can be identified. Firstly, improving our training data by either gathering more samples, or augmenting the ones we already have, or both. For example, a cup is still a cup if it is flipped along its vertical axis. Secondly, improving the object detection, such that the bounding boxes are even tighter. Advanced techniques such as image segmentation could even be considered, in order to isolate the objects of interest from the background.

5.3 Classifying objects in video

In standard image classification, the best option would always be to classify the object into the class that it most likely belongs, based on the pixels in the image. Indeed, doing anything else would be insane. This is due to the lack of contextual information. When classifying objects in a video on the other hand, we have a large stream of very similar images (the *context*), and we can exploit this knowledge to vastly improve the performance of the model [10]. We call this step *voting*.

The voting step only occurs at inference time, because the model was trained as a standard image classification model, but in theory, a model can also be trained on videos, and thus learn the optimal voting strategy. We fix the value of k, that is, the number of past frames to take into account when assigning class probabilities. Other voting schemes can also be implemented. Of course, objects could change — the book could get swapped out with a cup behind the wall. Our method assumes objects placed on the belt remain the same. The voting method could be changed to be a "rolling average" rather than "best initial guess wins all" to account for changing objects.

Other times, objects merely appear to change, due to occlusions or moving slightly or completely out of the frame. But that does not mean they cease to exist. For example, in the case of perception for a self-driving car, it is extremely important to account for such temporarily "vanished" objects, as they could heavily influence the robot control decisions. The positions and velocities of surrounding objects can be estimated and modeled in 3D space around the vehicle. In this project we have built the foundations for such an advanced system. The Kalman filter estimates the objects relative position and velocity, even under occlusions. The classifier can recognize different objects, albeit only a select few at present. The next step would be to track multiple objects at the same time, and then to model these in a virtual 3D environment. Other sensors such as LiDAR could also be helpful in modeling the surroundings, although a certain prominent company[1] thinks otherwise [11].

5.4 Deep Learning for classification

Another approach for classification is Deep Learning. We implemented the YOLOv3 [12] (You Only Look Once) model, which is a deep-learning approach for real-time object detection, to observe its results. It uses a pre-trained set of weights, and is ready for inference (almost) out of the box. We applied YOLO using the online pre-trained dataset called COCO [13]. COCO is an open-source dataset that is used for classification purposes. The problem with COCO is that no class "box" exists within the dataset, so it was only possible to classify the cups and books. YOLO is a more robust approach because the pre-trained model has seen a lot more images during training, and thus has a more general understanding of the classes. Yet, it is limited by the classes it has seen during training. This limitation can be alleviated by training new layers on top of the existing ones, using a new dataset with a potentially totally different set of classes. This approach is called transfer learning, because we take an existing general model and apply it to a different but similar problem, and thus transferring the pre-trained knowledge to a new problem, while also adding our own "flavor" on top.

5.5 Conclusion

The stereo calibration and rectification were done with the help of images of chessboards, allowing an initial calibration of the individual cameras. Then, the setup could be stereo-calibrated and rectified to give undistorted images that provided scanlines for depth estimation. A tracking based on the Kalman filter was then implemented to estimate the object position despite occlusions. As seen in the uploaded result video, this allowed the tracking of objects in 3D space.

The implemented classification method was correct most of the time when analyzing the video frame by frame, but as seen in Figure 8 it was not possible to separate the classes completely from each other due to a lack in training data. The training data was collected by taking a lot of photos of the 3 classes from different angles. If more variation in the objects were used as training data, it is expected that the applied method would give a more robust classification result.

[1] The company in question is Tesla Inc.

References

[1] Gopal Thapa, Kalpana Sharma, and MK Ghose. Moving object detection and segmentation using frame differencing and summing technique. *International Journal of Computer Applications*, 102 (7):20–25, 2014.

[2] Petteri Aimonen. Basic concept of Kalman filtering. *Wikimedia Commons*, 2011.

[3] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[4] Linear and quadratic discriminant analysis. URL `https://scikit-learn.org/stable/modules/lda_qda.html#id1`.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

[6] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/ramesh21a.html`.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL `https://doi.org/10.1145/3065386`.

[8] Richard Szeliski. Computer vision algorithms and applications, 2011. URL `http://dx.doi.org/10.1007/978-1-84882-935-0`.

[9] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001. ISBN 978-0-471-05669-0.

[11] Ben Dickson. Tesla ai chief explains why self-driving cars don't need lidar, Jul 2021. URL `https://venturebeat.com/2021/07/03/tesla-ai-chief-explains-why-self-driving-cars-dont-need-lidar/`.

[12] Yolo: Real-time object detection. URL `https://pjreddie.com/darknet/yolo/`.

[13] coco - dataset. URL `https://cocodataset.org/#home`.