Danmarks Tekniske Universitet

DTU

Real-time object tracking and classification project report

Authors

Hassan Ismail Abdalla - s210783
Henrik Fjellheim - s217066
Lukas Malek - s212074
Martin Miksik - s212075
Rebeca Gonzalez Revilla - s212490

Results

> VIDEO

> CODE

May 23, 2022

Contents

1 Introduction

This is the report of the final project for **31392** *Perception for autonomous systems*. The task of the project was to track and classify an array of object moving across a conveyor belt. Additional challenges to the project was having to un-distort and rectify the inpuf footage, as well as dealing with occlusion of the objects. The material footage was stereo imagery captured by Mynteye[1].

This report will cover how the final result was achieved as well as some interesting results/experiments not included in the final result. The final result includes an online algorithm to un-distort & rectify images, track objects moving across the image in three dimensions, an online kalman filter and a pre-trained neural network to classify the objects.

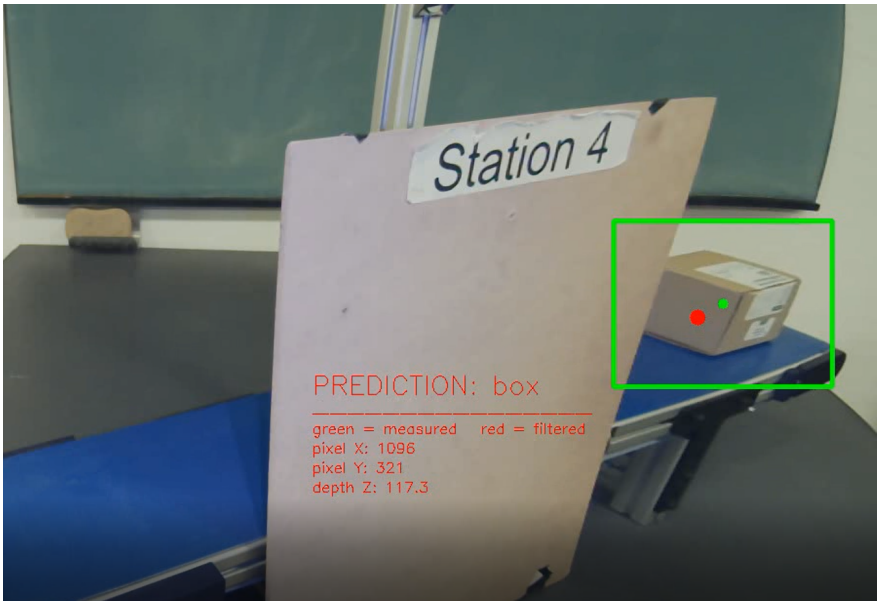


Figure 1: A crop out of the final result

2 Image preprocessing

2.1 Calibration

The first step is to calibrate the camera. In other words, the intrinsic camera matrix and distortion coefficients for both lenses must be determined. To do that, calibration images with a checkerboard pattern was used, where the size of each square pattern is known to be 33.6 x 33.6mm. To find the corners on the checkerboard function cv2.findChessboardCorners() was used. The pixels at which the corner was found are multiplied by the size of the square. In this way, the camera intrinsic matrix in mm is found. In the next step, cv2.calibrateCamera() is used. The output from the function call is a camera

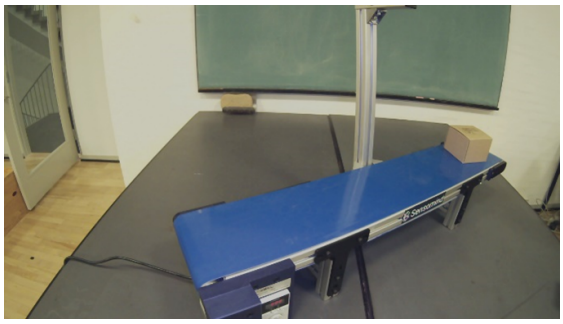
matrix $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ and distortion coefficients, later used to rectify and un-distort the images.

As the last step in the calibration, we get the excentric parameters R, T, E, and F for the stereo camera by calling cv2.stereoCalibrate()

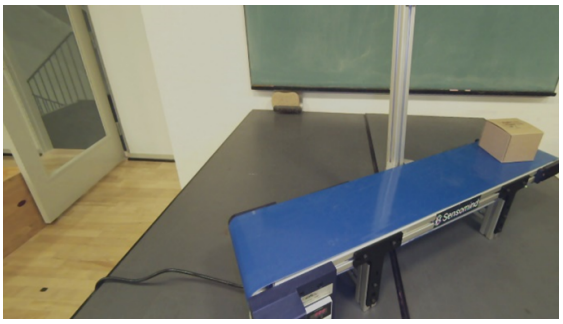
- R = Rotation matrix between camera1 and camera2
- T = translation between camera1 and camera2
- E = Essential matrix
- F = Fundamental matrix

2.2 Un-distortion

While taking pictures with the camera, straight lines are changed to curves because of the lens distortion. Without distortion, the Kalman filters and tracking, in general, would be complicated and nonlinear, as the object would be moving on the curve in the picture, although they are following a straight line in the real world. Un-distortion is the solution. Firstly, the function cv2.getOptimalNewCameraMatrix is used to "prepare" the image using parameter alpha, which helps us adjust the black areas and crop undistorted images. In the next step, cv2.undistort() can be called to undistort the image, and crop it using roi values achieved in the previous step. An example is shown in figures 2a and 2b.



(a) Original Picture



(b) Undistorted Picture

2.3 Rectification and depth map

The idea of the tracking is to track in three dimensions: pixel coordinates (x,y) and depth z. Three methods were considered to find depth of the object.

- Disparity map: Using results of camera calibration to rectify images and create a depth map. Then use detected x,y pixel coordinates to extract depth.
- Triangulation: Use the relative pose of the cameras to triangulate matching features.

- Heuristic: Objects are increasing in size as they move closer.

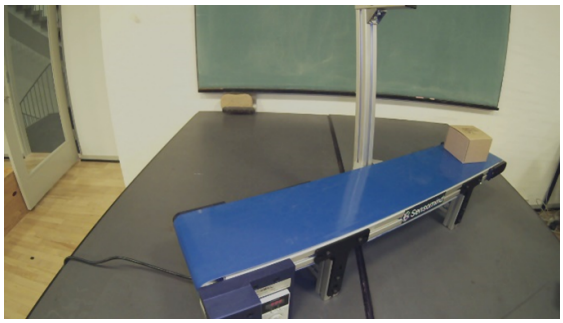
The advantage of the depth map in this scenario is that the environment is stationary, so a single depth map can be calucalted and used for all objects being tracked. For this reason a depth map was chosen.

2.3.1 Uncalibrated Rectification

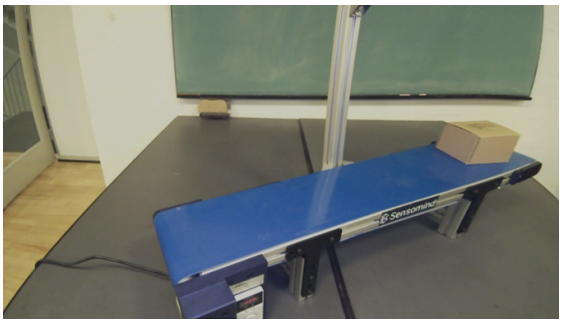
To rectify the images, there are two options in OpenCV. The first option is to use the stereoRectifyUncalibrated(), which needs matched points from both left and right images and a fundamental matrix, but is supposed to work on un-calibrated images. The images should be un-distorted howerver, as the StereoRectifyUncalibrated function heavily depends on epipolar geometry. The matched point are matched using keypoints and SIFT Descriptors from sift.detectAndCompute(), which are matched using BFMatcher().match() and sorted with the best matches in the beginning. The fundamental matrix obtained from calibration was used, and the homogeneous matrices were found using stereoRectifyUncalibrated(). In the last step, the images can be rectified using warpPerspective(). Nevertheless, this approach gave mixed results in this case, as for every image, the rectified image is moved, rotated, and scaled a bit differently, not creating a consistent solution. In addition to the fact that the cameras were already calibrated, it was decided to use Calibrated Rectification instead of un-calibrated, which gives gave a more stable results.

2.3.2 Calibrated Rectification

The Calibrated rectification was more successful than the uncalibrated, providing a good starting point for computing the depth of the image. Firstly, stereoRectify() was run to get the rotation R and projection matrix P of the cameras, later used in the initUndistortRectifyMap function. This function gives a mapping matrices for remapping. The remap() function finally gives us the rectified image as seen in figures 3a and 3b.



(a) Original Picture



(b) Calibrated Rectification

2.3.3 Epipolar Lines

To verify that the rectification was successful, epipolar lines were drawn on the images. Key points were found using swift, points described and matched, and the best/closest

matches were selected. The fundamental matrix F of the rectified images was found by running the findFundamentalMat() function and the lines were received using the computeCorrespondEpilines() function. The results are shown in figure 4.

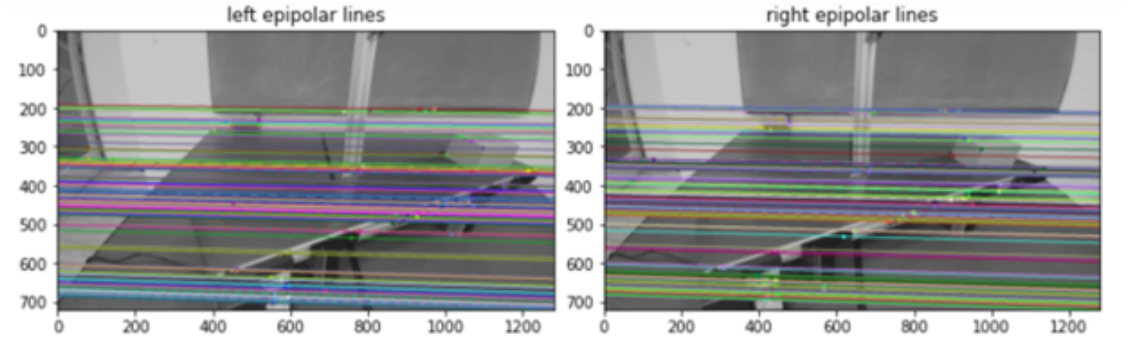


Figure 4: Epipolar lines for Rectified Images

2.3.4 Image Depth / Disparity Map

The stereoBM_create() function was sued to get the stereo image for the left camera. Then, the stereo for the right camera was done by running ximgproc.createRightMatcher(), which was necessary for later, as a wls_filter was used to get the best results possible. After exhaustive parameter tuning of StereoBM_create(), a satisfying disparity map was created.

Lastly ximgproc.createDisparityWLSFilter() was used for the left camera, setLambda and setSigmaColor, and the data was filtered, using the filter() function. The final results with two sets of parameters are seen in figure 5

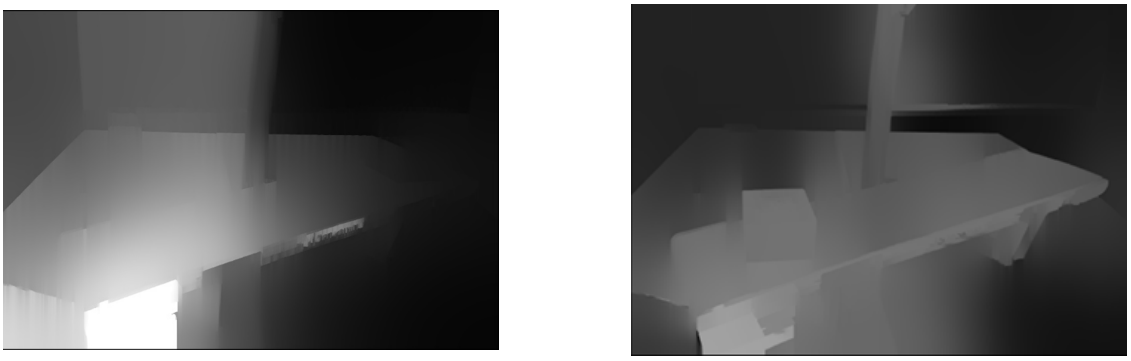


Figure 5: Image Depth with different parameters

3 Tracking

For tracking, two primary methods were proposed and tested. Dense/sparse Optical Flow and contour tracking. Both methods had its pros and cons, but eventually the contour tracking was chosen for its faster performance making it more relevant for an online tracking scenario. The output of the trackers will then be filtered in a Kalman filter reduce noise as well as track occluded objects.

3.1 Dense Optical Flow tracking

The first methods visited was dense optical flow tracking. The chosen Gunnar-Farneback[2] method calculates the magnitude and angle of every scanned pixel's neighborhood that have changed its position between consecutive frames. With that, we can dilate the flow to connect moving pixels optically, find contours, and compute the moving area to prevent tracking outliers and noise. By averaging the angle and velocity into the center of mass, the object's movement and coordinates are obtained. The result can be visually shown by selecting the appropriate hue for the moving pixels, to differentiate pixels based on speed, as show in figure 6.

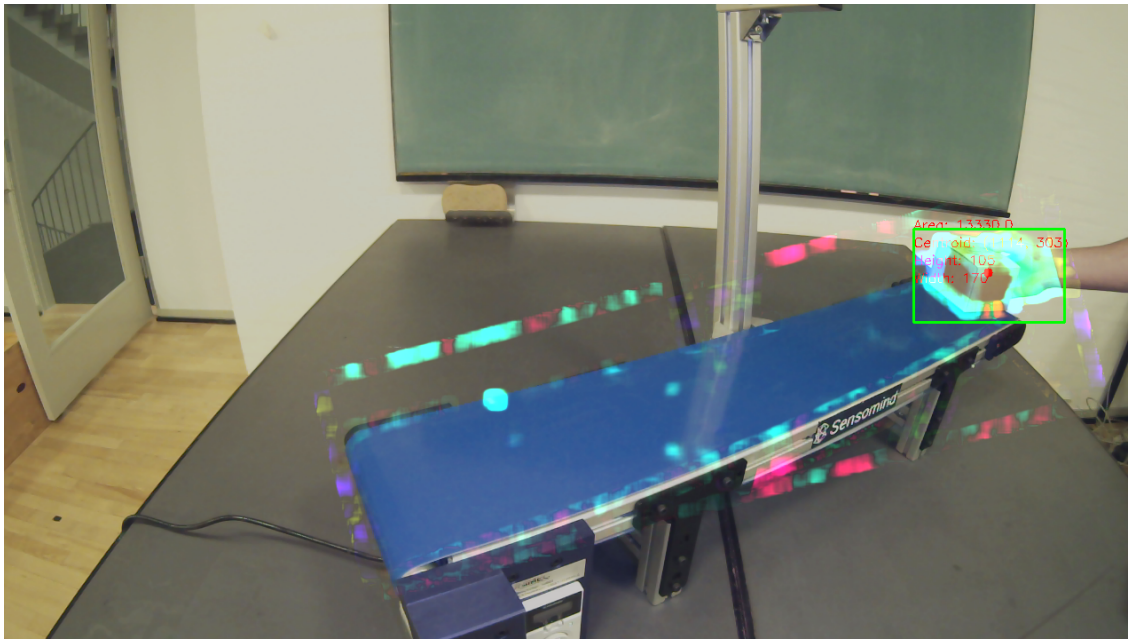


Figure 6: Dense Optical Flow visualization

Problems:
The most significant problem lies in the computational requirements, slowing the process tremendously. Even with further optimization of parameters like scaling, cropping, and reducing the size of the pixel neighborhood used to find polynomial expansion, we would still achieve only a few frames per second. However, with more powerful hardware, this

method would be very suitable for a realistic industrial usage with a need for robustness, precision, and resistance to changing environments and light conditions.

Conclusion:
The singlemost important advantage of optical flow over other methods visited is that speed is directly extracted, and could be used to create a more precise kalman model. This positive however did not outweight the fact that this method is computationally heavy and slow. Also, as the objects being tracked move at constant speeds, measuring speed was not paramount.

3.2 Contour tracking

As described in Section 3.1, the dense optical flow method is robust yet very slow. A very different, and much less computationally heavy method, is tracking by feature selection and image pre-processing to find contours.This is done by using Background subtraction in OpenCV (cv2.createBackgroundSubtractorMOG2() or cv2.createBackgroundSubtractorKNN()), Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene)[3] . Figure 7 shows how background subtraction can produce a mask from a background frame and an incomming "current frame". Figures 8 and 9 also show some examples from the conveyor belt footage.

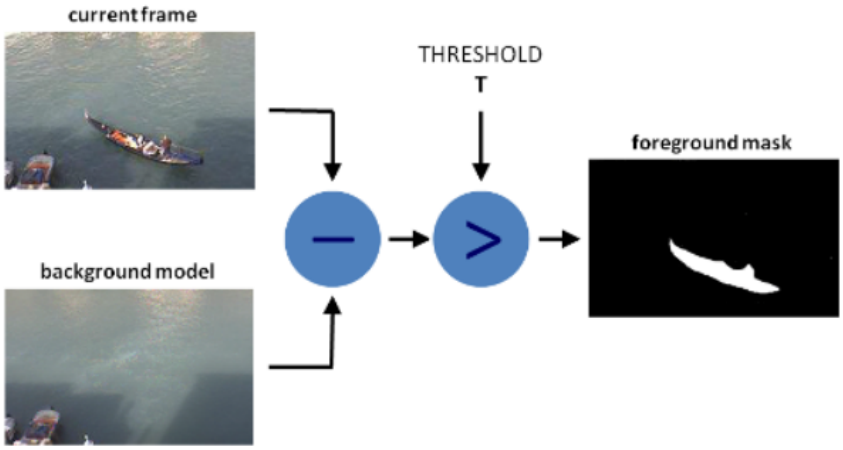


Figure 7: Background subtraction (BS)

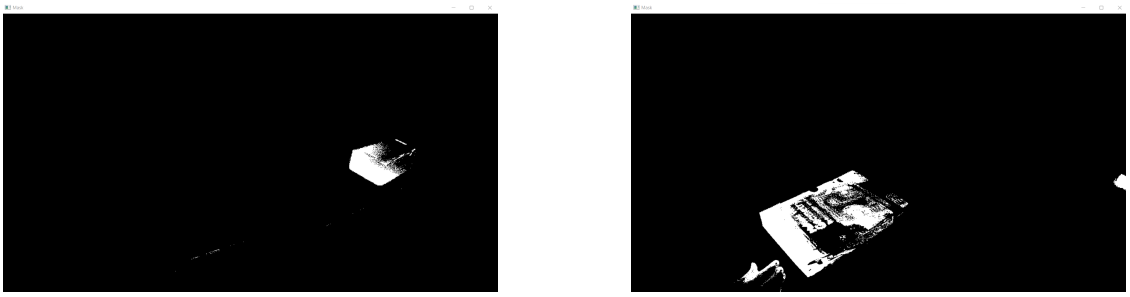


Figure 8: Result after BS

Figure 9: Result after BS

A problem that ofter arises it that the produces mask contains many disconnected dots, instead of an object to track. This is solved by using **dilation** on the image mask. Figure 8 shows an example without dilation, while 9 shows with dilation.

Dilation however did not alleviate another problem; noise. Actually, as can be seen, the dilation would often increase the size of the noise "particles" in the image. The simple solution is to realize that only one object needs to be tracked at the time, so only tracking the the area of those contours with the biggest area. An upper and lower bound was also set to avoid detecting random noise when there are no moving objects in the image (e.g. when object is behind occlusion). Figure 10 shows the effect of dilation on noise.

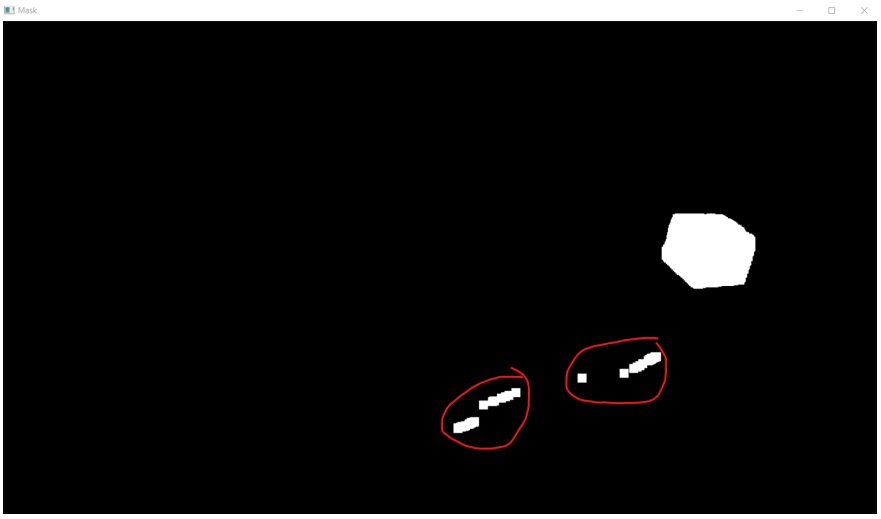


Figure 10: Noises that needed to be eliminated

The openCV function cv2.boundingRect() was used in order to get the coordinates, as well as height and width of the tracked object. Using those coordinates a bounding box can be drawn around each object, and the center of the box was used as the measured location of the object. The final results is shown in figure 11.

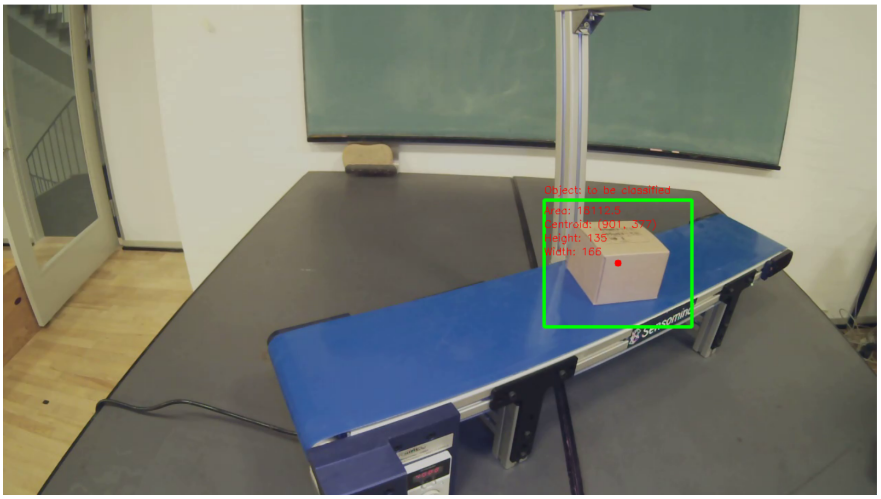


Figure 11: Tracked Object with Area and coordinates

3.3 Kalman filter

In order to reduce noise as well as predict trajectories when no data was available (e.g. when object is occluded) a kalman filter was implemented. Due to the object moving in straight paths across the frame, with more or less constant velocity, a linear kalman filter model is expected to be a good model of the system.

3.3.1 Four state filter

The first approach used was to have four states in the filter; x position, y position as well as x and y velocity. From the tracker described in section 3.2 the x and y pixel position of the tracked object is returned, with a lot of noise. In equation 1 the state vector and state transition matrix is displayed. This model assumes a constant speed in a straigh path trajectory. To achieve a good performance with this simple model, it is important to first **un-distort** the images before tracking (see section 2.2), as otherwise the trajectory would not be straight. It should also be noted that the measurement noise vector was manually tuned to account for measured noise.

$$\vec{x} = \begin{bmatrix} x-pos \\ x-vel \\ y-pos \\ y-vel \end{bmatrix} \mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The results of this filter is shown in figure 12, and shows that the tracking data is too noisy to for the filter to estimate both speed and velocities behind occlusion. This is seen by the fact that the speed is not constant, but seems to be averaged around a constant speed. Note that, excluding outliers, the constant velocity has now been estimated.

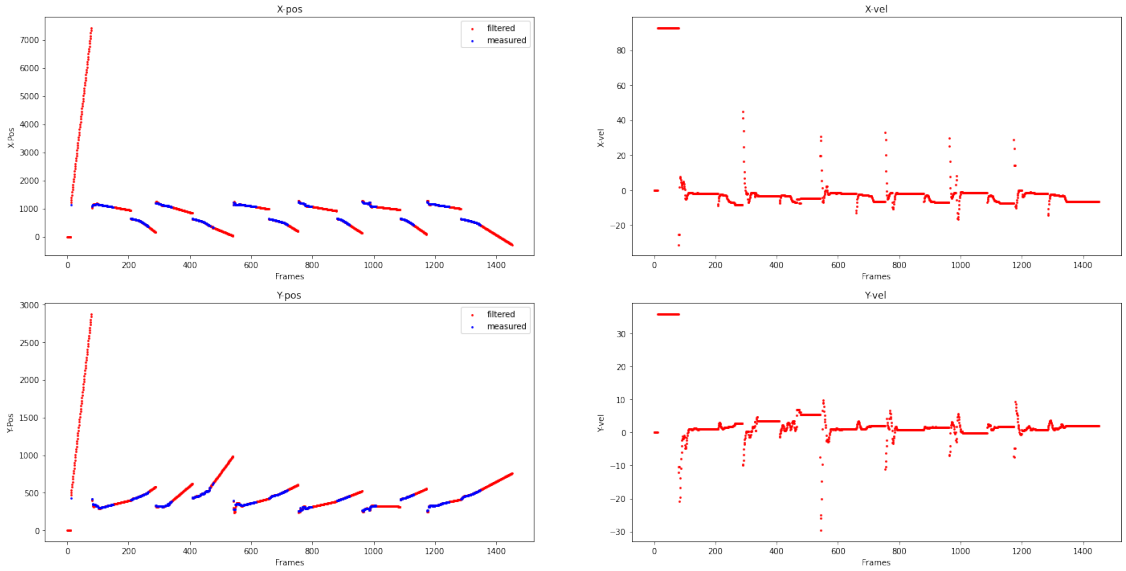


Figure 12: Four state filter results

3.3.2 Two state filter

Using the constant speed estimated in the previous section, a new and simpler kalman filter was implemented. Instead of modelling the speed as a state in the state transition matrix, the speed was modelled as a constant external affect on the system, applied in the

u vector. This new and simple model is shown in equation 2 and the results are shown in figure 13. The constant-speed model works well in this conveyor-belt environment, once the constant speed had been estimated. To achieve these results, the Kalman filter operates with two different **R** - matrices (measurement noise matrices): one R matrices with lowers values is used when the object has been seen to behave according to the model, but if the measurements diverge too much (while diving behind occlusion), the harsher R matrix is put into work, to stop relying too much measurements, and more on the model instead.

$$\vec{x} = \begin{bmatrix} x-pos \\ y-pos \end{bmatrix} \vec{u} = \begin{bmatrix} x-vel \\ y-vel \end{bmatrix} \mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

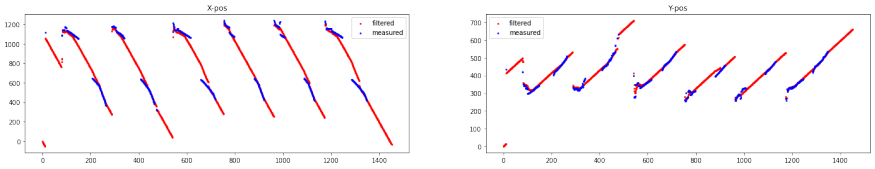


Figure 13: Two state filter results

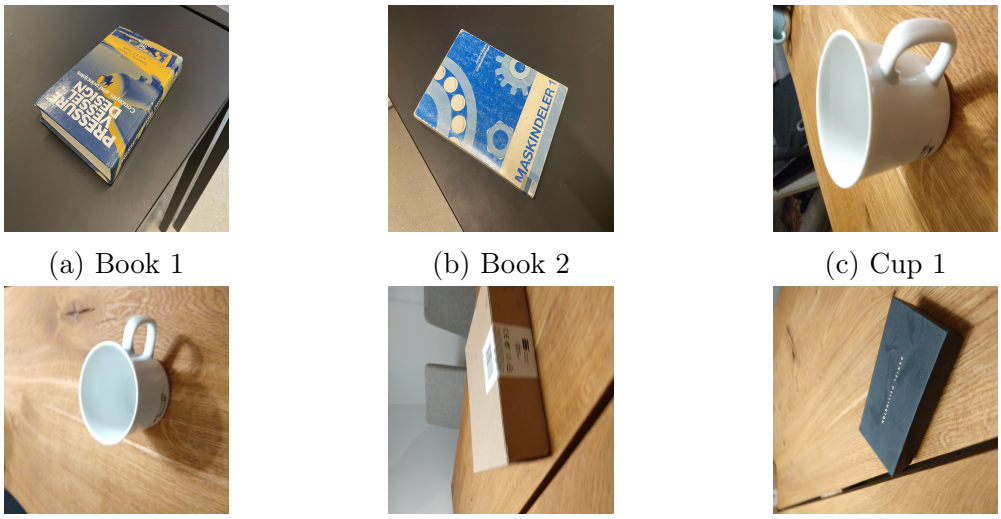
4 Classification

The remaining task is to classify the tracked objects into three categories: cups, books and boxes. The method chosen was neural networks, despite the knowledge that this method is data hungry. Neural networks was chosen as the group wanted to learn more about this exciting field.

4.1 Dataset

A dataset of the relevant three items is needed to train our model. The first approach was searching for already created dataset on the Internet as usually you can find big datasets of images. However, we couldn't find any that fitted our requirements particularly well. The external data-sets were tested with mixed results, and as a result it was decided to also create our own dataset. Pictures of cups, boxes and books were taken from different angles to try and get as many pictures as possible. In the end about 200 image were taken of each object, as seen in figure 14.

Data augmentation was also applied to artificially grow the data-set, by rotating, blurring and scaling the images differently.



(a) Book 1 (b) Book 2 (c) Cup 1

(d) Cup 2 (e) Box 1 (f) Box 2

Figure 14: Some examples of the images in the dataset

4.2 Model

Different classification methods were tested, where only the more interesting will be explained in more detail. All models tested where ANN based, where some operated directly on scaled-down raw image data, others worked on PCA data [4]. The best scores were achieved by the convolutional-neural-network (CNN) models, classifying data in the cropped out bounding boxes, and those models are what is focused on below.

4.2.1 Model 1 - basic CNN

CNN is a powerful neural network approach, to automaticall train convolution layers/kernels to extract key information from images, needed to classify an object [5]. Firstly, a basic CNN was implemented consisting of a couple of convolutional layers (relu activation) with maxpooling, flatten and finally, a dense layer with softmax activation function to get the probabilities of each image belonging to one of the three possible classes. This architecture was a somewhat random selection to test the possibilities offered by CNN.

After training the model on our own raw data, the model was tested on tracked objects. It achieved almost 58% accuracy on test data.

4.2.2 Model 2 - basic CNN on augmented data

In this case, the same network as in 4.2.1 was used. The difference between these two methods is data it was trained on. Our own dataset was quite small, and different degrees of augmented data was added to bolster the size. In the end, the dataset was increased by a factor of three in size, from a total of 672 to 2016 images.

The accuracy increased up to 61% by increasing the size of the data-set but not changing any other parameter of the model. It was discovered that adding too much augmented data would reduce the performance, which probably can be contributed to over-fitting, as augmented data is pretty much the same data "under-the-hood".

4.2.3 Model 3 - pretrained model

From the available pretrained models in Keras, the pre-trained MobileNet was also tested. The architecture of this model consists of 55 layers. It uses depthwise separable convolution instead of the standard convolution in 4.2.1 and 4.2.2. The main difference is instead of a single 3x3 convolution layer followed by the batch norm and ReLU, Mobile Nets split the convolution into a 3x3 depth-wise conv and a 1x1 pointwise conv [6].

The results obtained for this model were almost the same as the ones obtained in 4.2.2 with a test accuracy of 60%.

4.2.4 Model chosen

After training and testing different models, we have seen that data augmentation increase the performance though not by much. Moreover, it can be seen how complexity is not giving better results. Most of the pre-trained models have more than 20 layers which also make them slower at training. Some of this models took even 2-3 hours to train and the end the accuracy obtained was around 30-40%. In the case of the MobileNet, it was one of the fastest ones (20 mins) as well as the one with the best outcome. However, it could not increase the accuracy obtained by our simple CNN model trained with augmented data. For this reason, we thought that it was best to keep it simple and faster as the result was almost the same.The final model chosen for the project is Model 2. Although, the benefits of using augmented data were not much, it was not damaging any other aspect.

A visual example of the predictions made by the model are shown in Figure 15. Only six predictions are represented which gives us an idea of which could be the problems that our network is having when classifying. In this image is not that clear, but if we analyze the whole prediction for the test set, we can see how the model is struggling to differentiate mostly between boxes and books.

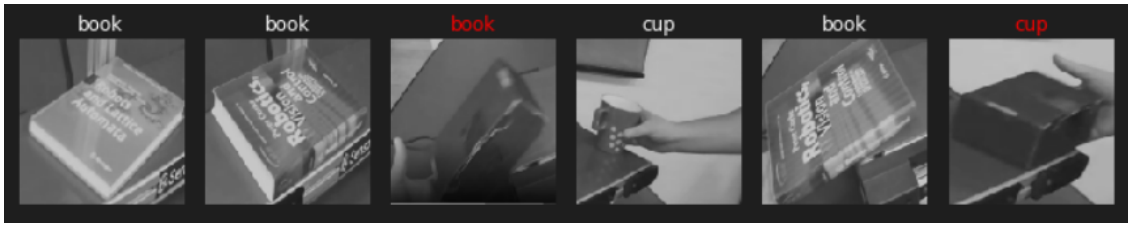


Figure 15: Model predictions

4.2.5 Dealing with low precision and high variance

It was discovered that the predictions of our chosen model had a lot of variance from frame to frame, but by averaging the predictions over time the results could be somehow improved. Despite this, it will be noticed that in the final result the classifications is not always spot on.

5 Conclusion

Throughout the project many different methods were tested and experimented with. In the end however, methods were chosen not only based on precision but also on computational performance, as the team really wanted to make an algorithm that can be run "online"/"real-time" rather than something super-slow running on stored images. Therefor our finished method gets a stream input images, un-distorts and rectifies them, tracks objects and predicts all in close-to real time speed.

Running the final project requires running the file "final_project/FULL_PROJECT/main.py", after properly downloading loading the model from "final_project/FULL_PROJECT/cnn_model_3" and placing the raw input videos in the same folder.

An example can be seen here: VIDEO

List of Figures

References

[1] "Stereo sensor product." https://www.mynteye.com/products/mynt-eye-d-order. Accessed: 2022-01-14.

[2] G. Farneback, "Very high accuracy velocity estimation using orientation tensors, parametric motion, and simultaneous segmentation of the motion field," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 1, pp. 171–177 vol.1, 2001.

[3] "Background subtraction (bs)." https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html#:~:text=Compatibility,scene.

[4] "Principle component analysis (pca)." https://www.keboola.com/blog/pca-machine-learning.

[5] "Convolutional neural networks (cnn)." https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/.

[6] "Keras pretrained mobilenet." https://keras.io/api/applications/mobilenet/.