Perception for Autonomous System Final Project Report

Group 6:
Manxi Lin - s192230
Peixuan Wang - s192176
Caining Liu - s192146
Max Prüstel - s200268

January 3, 2021

Contents

1 Introduction

1.1 Background and Report Structure

With the development of artificial intelligence technology, more and more state of arts are applied in industries. Standing on industrial big data, data-driven product classification and quality monitoring based on machine learning are getting a wider and wider application. Industry 4.0 is showing its power to our era.

This project aims to emulate one such scenario, automated quality control, which, in industry, should be carried out by an automated system to maintain high-quality control standards and cope with large volumes of objects to evaluate. According to the project description, we can divide the project into 3 tasks, calibration and rectification, object detection and tracking in 3D as well as classification. For each task, we developed more than one solutions and managed to pick up the optimal one or two. In the end, the selected methods are integrated to form our final solution.

Section 2 focuses about camera calibration and image rectification, a crucial step before stereo matching and 3D reconstruction during 3D tracking. We did this task on two different platforms, OpenCV for python and with the MATLAB Stereo Calibration toolbox. The epipolar lines of the rectified images are drawn as the criteria judging the quality of the achieved result.

Section 3 clarifies our work on object detection and tracking. To simplify the task, we split 3D tracking task to 2 steps, 2D tracking followed by 3D projection. Three algorithms based on optical flow and two related with Background Subtractor are developed, the performance of which are compared. As a result, Background Subtractor is deemed to be our final solution. In 3D projection part, stereo matching and triangulation are analysed. Considering the scenario where the target is under occlusion, we introduce 2 history-based methods and Kalman filter, each of which has its own merits.

Classification with machine learning model is introduced in section 4. Following the line of thinking in pattern recognization, we develop 2 classifiers, using SVM and CNN respectively. The result shows both of them are qualified.

As a conclusion, section 5 summarize the whole report and present our final solutions. We put our results on Youtube and attach the links.

1.2 Index: invented terms in the report

- Optical Pyramid algorithm, OP
- Optical Sliding Window algorithm, OSW
- Find the Larget Contour algorithm, FLC
- Given Conveyor algorithm, GC
- Classical Motion Model, CMM

1.3 Index: other important abbreviations

- SOF: Sparse Optical Flow
- BS: Background Subtractor
- KF: Kalman Filter

2 Calibration and Rectification

For this project we have stereo images to accomplish 3D tracking. From these we have to create 3D environments later on. Before this can happen we have to make sure that the two images are transformed so that we can precisely correlate one feature from the left camera with one from the right camera. This requires a two step process. First we have to gather information about the two cameras on their own and then about their relation in space and thus calibrate them. Secondly we have to rectify all images, meaning that we have to perform a transform that aligns all epipolar lines in both the left and the right image, which, when successful, allows us to seamlessly transform point coordinates from one image into the other and into 3D.

2.1 Camera Calibration

Each camera and lens have their individual distortions. To allow us to understand the two cameras that provide images for our tracking we have to calculate the camera matrix. This step was carried out solely in OpenCv and made use of the 50 checkerboard calibration images for both left and right provided.
First we generated feature points using the `cv2.findChessboardCorners` on the 6 by 9 internal corners. With this done for all images `cv2.calibrateCamera` returns a first camera matrix that is specific to each individual camera. This contains all intrinsic camera parameters, but we still have to define one cameras relationship with the other one. For this we can utilize the `cv2.stereoCalibrate` function that provides a rotation and translation matrix between the two. In the end we receive the following camera matrices:

$$C_l = \begin{bmatrix} 707.5615 & 0 & 0 \\ 0 & 707.5559 & 0 \\ 624.1762 & 372.2979 & 1 \end{bmatrix} \tag{1}$$

$$C_r = \begin{bmatrix} 704.4922 & 0 & 0 \\ 0 & 704.6913 & 0 \\ 652.0960 & 374.6756 & 1 \end{bmatrix} \tag{2}$$

As the two cameras can be assumed sufficiently planar only a translation matrix is needed for the relation between the two, which can be expressed as the following:

$$T = \begin{bmatrix} -120.0307 & -0.2427 & -0.4453 \end{bmatrix} \tag{3}$$

Whilst we know about the cameras still have to undistort the images, using `cv2.undistort` in conjunction with `cv2.getOptimalNewCameraMatrix`. In the method using matlab this is unnecessary as it is covered by the rectification step.

2.2 Image Rectification

With all parameters gathered and lens distortion removed from the images, we can now rectify them. This essentially transforms the images so that we can more easily apply depth mapping by bringing identical features onto the same horizontal lines. In our succesful method with matlab we can use the `rectifyStereoImages` function which is applied to both left and right images simultaneously. In OpenCV we used the `cv2.stereoRectify` and remapping functions for this process but the results were unsatisfactory, so that we continued to use the data generated through MATLAB.

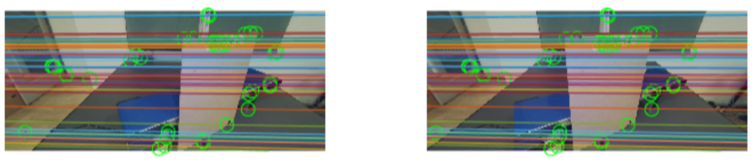

Figure 1: Rectified Stereo Image with parallel epipolar lines

3 Object Detection and Tracking

In this chapter, we will try to clarify a series of different methods we developed for object tracking. Since 2D tracking algorithms are easier to understand, we start with some 2D tracking ideas here. To eliminate ambiguity, in this section, the objects we want to keep tracking are called **targets**. We denote all the moving objects in a frame as **target candidates**. To localize the target, we give the bounding box of a target a name as Region of Interest, also known as **ROI**.
Since all the innovations must be based on reality, in line with the exercise description, we come up with several assumptions and principles below:

Assumption 3.1. *There is at most one target on the conveyor.*

Assumption 3.2. *The trail of a target can be approximated by a straight line.*

Assumption 3.3. *The motion model of the target is a n-order constant motion model. That is, if v represents the velocity of a target, for time t, there exists a non-negative integer n satisfying $\frac{\partial^n v}{\partial t^n} = 0$.*

Assumption 3.4. *The motion of the targets is continuous in space.*

What's more, according to the **real-time** nature of a video, we need to pursue the algorithms with small time complexity.

3.1 2D Tracking

3.1.1 Optical Flow

As an important part of the course, optical flow is the very first idea struck us. The built-in dense optical function in opencv, *calcOpticalFlowFarneback()*, returns 2 values $d_{p,x}$ and $d_{p,y}$ for any pixel point p in an image[3], relating with the displacements in x and y direction respectively. Let's denote the displacement d_p of a pixel point p as the first or second moment of $(d_{p,x}, d_{p,y})$. Calculating displacements for all the pixel points in a frame, we can get a "Displacement Map", standing on which two methods are proposed, illustrated in figure 2. To set a critical constraint for the displacement, we make a new assumption here.

Assumption 3.5. *When the displacement of an object between two frames is less than 1 pixel, the object can be considered static.*

From opencv documentation, a key parameter in Farneback algorithm, **winsize**, the window size, draws our attention[1]. When the window size is larger, the optical flow calculator is more sensitive to an object with a higher speed. Inspired by the Gaussian pyramid, we propose an Optical Pyramid (OP) in this report, whose layers differs in *winsize*. The lower the layer is, the bigger the *winsize* is, and then the faster the object captured is. By setting a proper window size, we can fetch target candidates seperately from a frame. As it can be seen from figure 2, the output of the second layer is exactly what we want.
Another way to process the displacement map is similar to the template matching we learned before since it's using a classical object tracking method, sliding window. Let's call it Optical Sliding Window (OSW) for the moment. After we get the displacement map as stated at the beginning of this subsection, we can set a threshold according to the assumption 3.5 to remove noise points on it. After that, a sliding window is put on the map and we denote a score of a window region as the summation of all the pixel values in it. ROI is defined as the window with the highest score and the score ought to exceed a threshold set before. Once the sliding window find a ROI, in the following frames, according to the assumption 3.4, we only need to search ROI in the neighborhood. To localize the target accurately, we use the center of gravity of ROI instead of geometric centre as the centroid of the object. In the application, inertial coefficient is added to make sure a probability that the ROI remain static, so as to stabilize the tracker.
After we find the first ROI in a video, Sparse Optical Flow(SOF) can be of use. Extracting ORB features in ROI followed by SOF can make the tracking much faster, alleviating the time-consuming problem of dense optical flow in a certain degree. But SOF is very sensitive to noise, and it always tracks loses target especially when the target is small, e.g., a cup, since no feature points to extract.

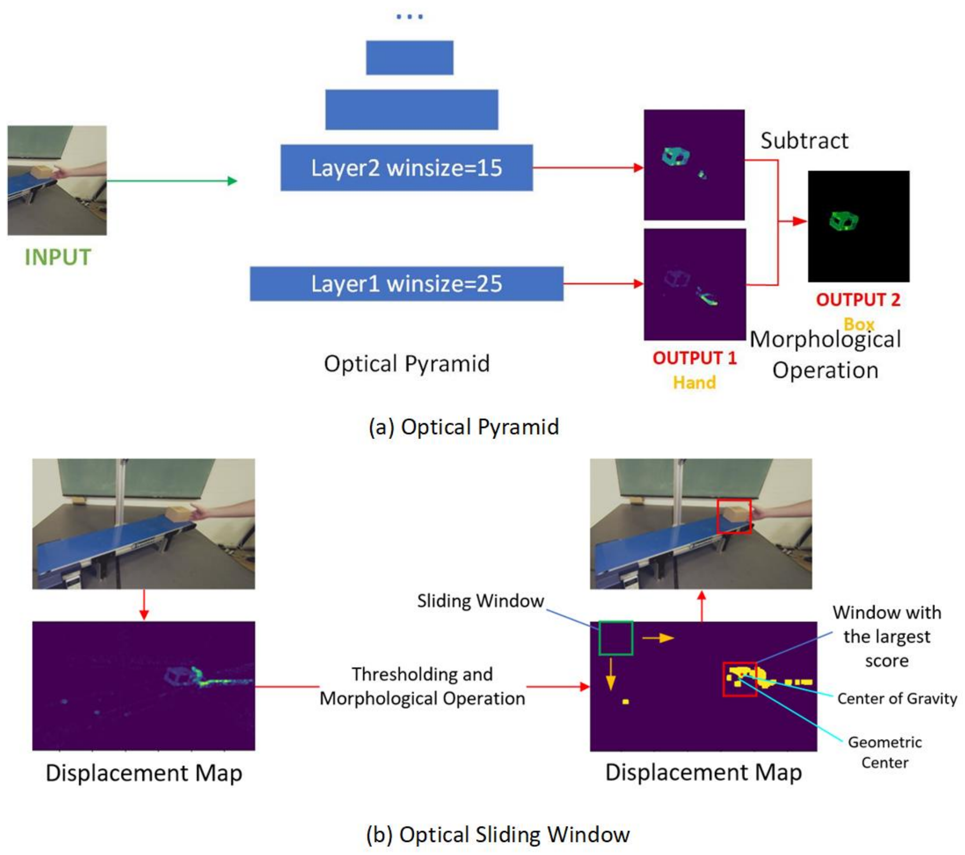

Figure 2: Tracking via dense optical flow

3.1.2 Background Subtractor

Dense optical flow we used in the last subsection takes a long time, is conflict with our real-time principle. Thus Background Subtractor (BS) may be a better choice, which is comparing the difference between the background and foreground of a frame with a threshold. Therefore, the output of BS is a binary image. BS can be based on Gaussian Mixture Model (GOM), K-Nearest Neighbour (KNN) or Bayes (GMG). We choose the simplest one related with KNN without shadow detection. Figure 3 presents the process of our experiment. Since in our video the first 80 frames are all backgrounds, it fits the characteristic of BS that it needs frames to train itself. In morphology, Open operation can keep the shape of the target while close operation helps eliminate noise points, one of which is applied on the binary map BS returns. After that, as it can be seen from figure 3, target candidates are shown in form of blobs on the map, one of which may be the target. What we need to do is to find that blob and bound it with ROI. Two ways are developed to locate the target on the map. One is to Find the Largest Contour (FLC) whose area (the sum of the pixel values inside) is over 2000. The other way is to give the "entry" and "exit" gates of the conveyor manually, we can call it Given Conveyor (GC) for short. The "entry" and "exit" gates are two rectangle regions given by us before processing. According to the assumption 3.1, the target candidate whose center is in the "entry" gate can be seen as the target. Then the tracker will keep eyes on the target until it leaves the "exit" gate. Honestly, to improve the performance, we introduce some heuristic knowledge to judge whether the contour returned by FLC is the target by setting a "entry" gate. In addition, we try to seperate the target from candidates by recalling the assumption 3.4: only when a target candidate observed moving continuously for 10 frames, can it be considered as a target.

To evaluate the performance of methods, we introduce 2 metric here. Processing speed measures the time complexity of the algorithm while the Lose Target Frequency, given by observation, reflects the accuracy.

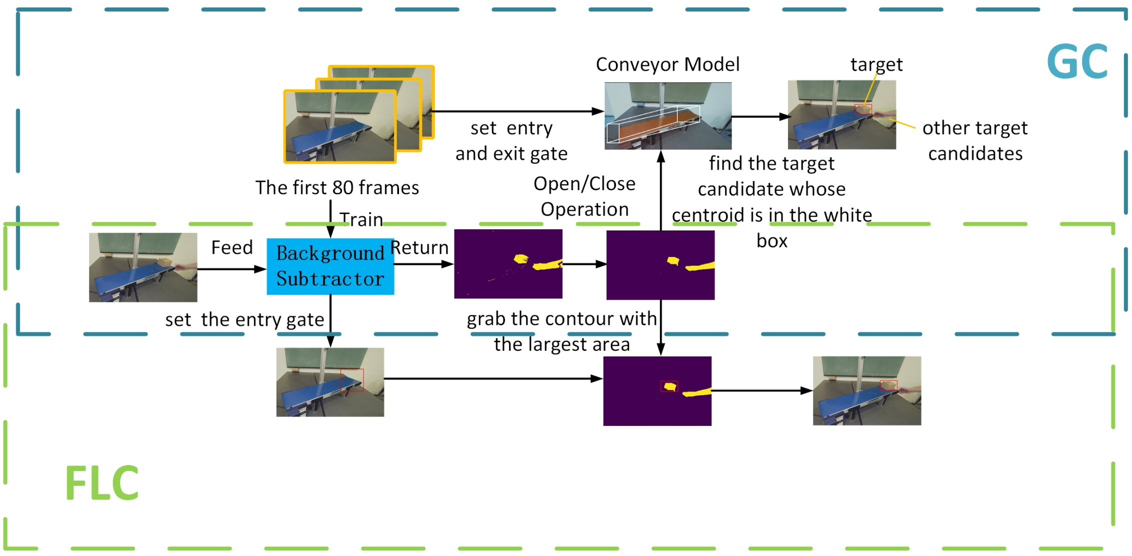

Figure 3: Tracking by background subtractor

Table 1: Performance of different methods

Method	Processing Speed (fps)	Lose Target Frequency
2-Layer OP	1.3	often
OSW	2.1	**seldom**
OSW+SOF	7.1	very often
BS	22.4	**seldom**
BS+SOF	**27.5**	very often

As far as I'm concerned, BS is the best way to solve the problem. We choose BS as our end solution.

3.2 3D Tracking

3.2.1 Stereo Matching

To meet the requirements of this project, 3D tracking is needed. As we rectified the photos in the first section, we implemented the modified H. Hirschmuller algorithm to get the disparity map of each frame. Figure 4 (a) illustrates the result. Since the right camera has a wider view of the conveyor, in this project, the disparity map is right disparity map, different from what we learned in class.

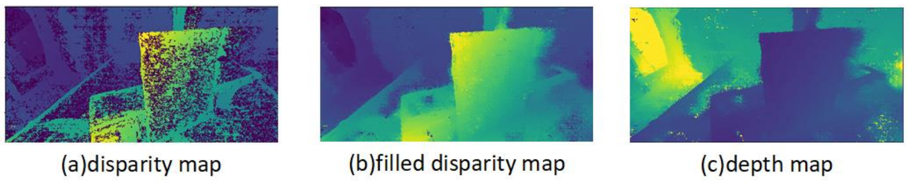

Figure 4: Disparity and Depth Maps

It's known from the figure that there are many noise points in the map, which can be eliminated with the use of average filter. Figure 4 (b) shows the filled disparity map. Integrated map is used here to save time.
From the book, we have an equation for the depth:

$$Z = \frac{f \cdot b}{d} \tag{4}$$

where Z and d are the depth and disparity of a pixel point respectively while f is the focal length of the right camera, b is the baseline length between two cameras. The calculated depth map is shown in Figure 4 (c).
However, it takes a really long time to get the depth map. The time to process one frame is about 17

second. For sure it's not applicable in a video. Thus when we need to track the target, loading the offline-generated depth map may be a better choice. The depth of the target can be seen as that of the centroid of the target.

3.2.2 Triangulation

Triangulation is another way to track the target in 3D. Compared with stereo matching, it's much faster ,for which it fits the real-time characteristic we want. Even though the built-in triangulation function in opencv only need one point to work, to reduce re-projection error, more points the better. When processing small targets, as stated before, there is no feasible feature points found in ROI, thus we use the centoid of the target as the input. For other targets, ORB feature points are extracted from ROI, matched, then triangulated. Note that we should make sure the Z got from triangulation positive, since the conveyor is in front of the two cameras.

3.3 Motion Prediction

In the previous two subsections, we developed several different methods to track the targets in 3D. Considering the scenario that objects are under occlusion, we need motion prediction to keep tracking. In our project, when implementing GC, if the target on the conveyor disappear in the frame without leaving the "exit" gate, it will be deemed under occlusion. When implementing FLC, once the area of ROI shrinks to less than 20%, it's thought obscured.

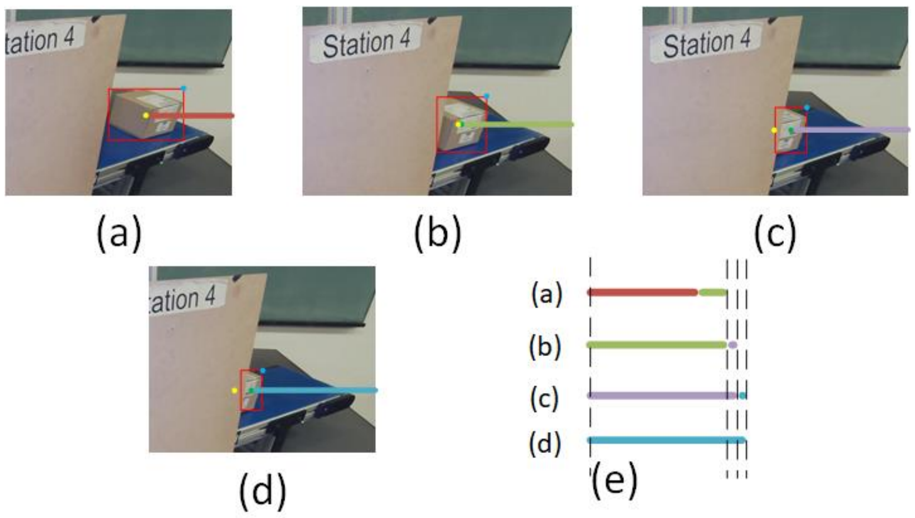

Figure 5: The process of object entering occlusion

3.3.1 Kalman Filter

Due to the assumption 3.2, Kalman Filter(KF) should perform better than Hidden Markov Model. Even though KF is able to track a target in 3D, thanks to the triangulation we introduced before, in this project, KF can ignore the depth of the target. When applying KF in prediction, we assume our model a 1-order constant motion model, also known as the Constant Acceleration (CA) model. Nothing do we know until the target appears. Therefore, KF is initialized with a high uncertainty.
However, even with high uncertainty, it takes time for KF to stabilize. Two ways are designed to handle the situation. One is to introduce priori knowledge to give KF an approximate initial position. In the other way, the coordinates between two consecutive frames are linearly interpolated to generate 100 points, with which we are able to update KF for 100 times in a frame to accelerate the convergence.
There is still one thing remaining to be done that since the target is bounded in ROI, we need a point whose coordinates can be seen as the measurement of KF. At first we tried to make use of the centroid of ROI. Figure 5 (a) to (b) illustrates the process for a target being obscured. The yellow points are the man-marked box centroid while the green ones are centroids of ROI. The colored straight lines are used for indicating the change of the target's horizontal position, which is shown in figure 5 (e). We can draw a conclusion that when the target is getting out of our sight, the area of ROI is decreasing

sharply, resulting in a tendency of the ROI centroid moving away from the real target centroid. Thus, the measured velocity by KF becomes slower than the real value. The longer the target is, the greater the degree is. Regarding feature points as the measurements is another idea for the problem, whereas it has the same drawbacks with SOP, that is, few features for small targets and noise-sensitive. As a final solution, we take the right upper corner of ROI, the blue points in figure 5, because it's the latest point on the target to be obscured. It's a pity that a good blue point relies on a fabulous morphological operation in FLC. For these reasons, our KF's performance is not that perfect.
During application, we build independent Kalman filters for the left and right camera output. The predicted 2D coordinates from the two KF are then put into triangulation part to get estimated 3D coordinates.

3.3.2 Linear Regression

It's apparent that the Kalman filter predict the target motion based on the velocity and acceleration got from the last frame before being totally obscured. To make our prediction more reliable, we can make use of all the historical coordinates of the target centroid. In this report, we call them **history points**, denoted as P. First, to make the whole thing easier, let's keep the assumption that the motion model is CA and the target only moves in one direction. With the help of difference, we can derive the velocity V and acceleration A for each point in P. A predicted velocity and acceleration are thought as the weighted average of V and A. We denote W_v and W_a as the weight vector of V and A respectively. Then we need to solve the problems that:

$$argmin_{W_v} \|(V - I_v \cdot {W_v}^T \cdot V)^T \cdot (V - I_v \cdot {W_v}^T \cdot V)\|_2 \tag{5}$$

$$argmin_{W_a} \|(V - I_a \cdot {W_a}^T \cdot V)^T \cdot (V - I_a \cdot {W_a}^T \cdot V)\|_2 \tag{6}$$

where I_v and I_a is a vector filled with 1, whose shape is the same with V and A respectively. SVD is used to figure the weights out. Once we get velocity and acceleration, prediction of the target position is achievable. If we apply the prediction in all three directions, we will be able to track the object in 3D under occlusion. Triangulation is not necessary here, even though it's a good choice.

3.3.3 Classical Motion Model

In line with assumption 3.2, we have a novel history-based method: fitting a line l with the history points. That is:

$$argmin_l \|l \cdot \widetilde{P}\|_2^2 \tag{7}$$

where $\widetilde{P}$ is the homogeneous form of the history points. Least square helps us solve the equation. In addition, we regard l as a vector from the first history point to the last one, whose direction is seen as that of the movement of the target under occlusion. To predict the motion, Classical Motion Model (CMM) is introduced. If we denote the number of the history points as time t, according to the assumption 3.3, for a CA model, we have:

$$d = d_0 + v \cdot t + \frac{\partial v}{\partial t} \cdot t^2/2 \tag{8}$$

where d is the displacement of the target. d_0, v and $\frac{\partial v}{\partial t}$ are the displacement, velocity and acceleration of the target when recording start, respectively. d_0 is equal to 0 here. We can easily extend the model to n-order, presented as:

$$d = v \cdot t + \frac{\partial v}{\partial t} \cdot t^2/2 + ... + \frac{\partial^n v}{\partial t^n} \cdot t^{n+1}/n + 1 \tag{9}$$

To clarify the motion model, we split l evenly by n equidistant points. d_i is defined as the distance from the start point of vector l to the recorded position at time $i/n \cdot t$. Then we have:

$$\begin{bmatrix} v \\ \frac{\partial v}{\partial t}/2 \\ ... \\ \frac{\partial^n v}{\partial t^n}/(n+1) \end{bmatrix} = \begin{bmatrix} \frac{t}{n} & (\frac{t}{n})^2 & ... & (\frac{t}{n})^{n+1} \\ \frac{2t}{n} & (\frac{2t}{n})^2 & ... & (\frac{2t}{n})^{n+1} \\ ... & ... & ... & ... \\ t & t^2 & ... & t^{n+1} \end{bmatrix}^{-1} \cdot \begin{bmatrix} d_1 \\ d_2 \\ ... \\ d_n \end{bmatrix} \tag{10}$$

where $[.]^{-1}$ means the pseudo inverse of a matrix. For any given n-order constant motion model, we can simulate and make prediction.

4 Classification

In this task, we need to classify unseen images into the 3 classes(cups, books, and boxes) based on a machine learning system. The training data set should be gathered by ourselves. Testing set is the provided video images.
In order to clarity the below sections clear, we will call each training or testing image as "sample" while the attributes of the image (pixel values) as "feature".

4.1 Training Dataset

The most important thing for a training model is the data set it used to train. We think 2D image is enough to distinguish the three classes and also much more easy to gather than 3D data.

We try three different data sets: 1.self-take pictures(size:300). 2.Online picture(size:300), 3.Cropped images from the provided video without occlusion (size:1344). Each data set includes three types of images, the number of which roughly the same. It is important because unbalance distribution of training data would add bias to our learning model, so it would be prone to the one with the largest number in order to get higher accuracy.

We do image processing and play with different data augmentation methods. Firstly, all the images are resized to 256×256. To keep the color information each image has 3 channels. Then some images are chosen randomly and been added with some changes, like rotating, horizontal flipping, scaling, adding Gaussian noise and gamma correction, whose purpose is to ensure the trained model robust to these uninterested changes on images and find the true distinguishing features successfully.

4.2 Normalization and PCA

Each input image is changed to a 1D vector with length of $256 \times 256 \times 3 = 196608$. It means we face with a very high dimensional feature space and would be time-consuming to train with. Put it another way, many features are not that useful to distinguish our objects, like colors. So Principle Component Analysis (PCA) would be very suitable for our problem to reduce the dimensions.

Before doing PCA, our data must be scaled, otherwise the PCA will select to project as much as possible in the direction of data with big scalar to get greater variance. Another reason is that most machine learning algorithms are not scale invariant, so it is important to scale data before further operation. Scale each feature on the input vector X to [0,1](Minmax Scalar), or standardize it to have mean 0 and variance 1 (Standard Scalar). Here we use Minmax Scalar.

After doing normalization and PCA, you can see the variance explained in figure 6.

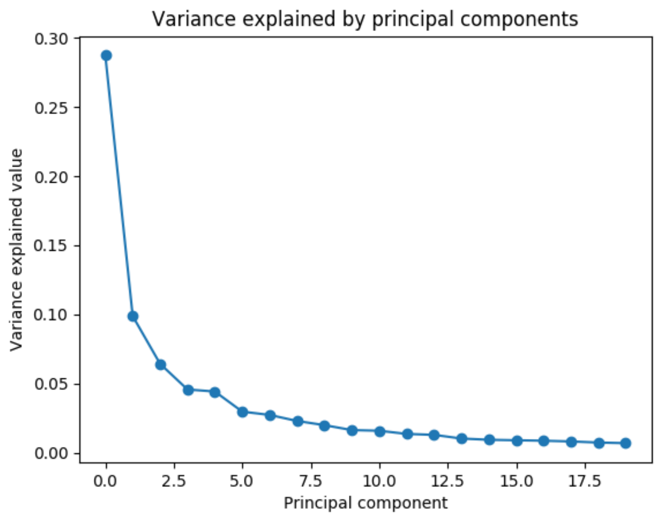

Figure 6: Variance explained by PCs

From the figure 6, we found that very few principal components can explained most variance and the components more than 20 have only little contribution. So we just choose top 20 components and project all the training samples onto these 20 components. The result of projection onto PC1 and PC2 is shown in figure 7.

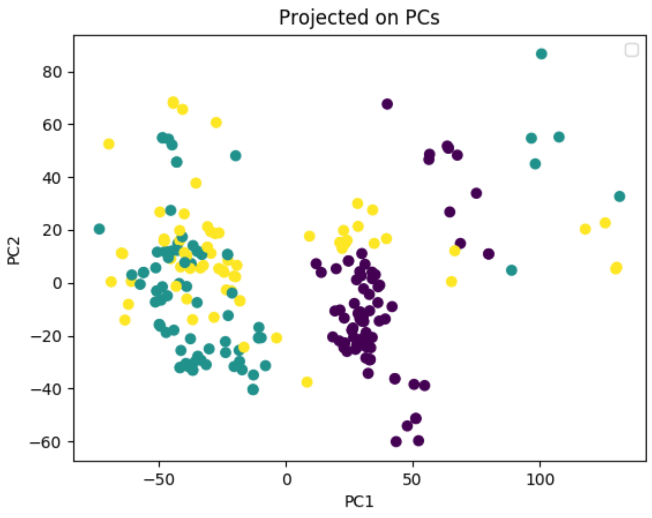

Figure 7: projections on PC1 and PC2

4.3 Classifier

We use two kinds of classifier, one is SVC and the other is CNN.

4.3.1 SVC

We have only three classes so the Support Vector Classifier (SVC) is enough to solve it. The figure7 shows that this problem is not linearly separable. So we changed the kernel to Radial Basis Function(RBF), the kernel function is :

$$exp(-\gamma \|x - x^{'}\|^{2}) \tag{11}$$

For SVC classification, we are interested in a risk minimization for the equation:

$$C\sum_{i=1}^{n} L(f(x_i), y_i) + \Omega(\omega) \tag{12}$$

where L is a loss function and Ω is a penalty function.

So there are two important parameters to be considered : regularization parameters C and influence parameter γ. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. γ defines how much influence a single training example has. The larger γ is, the closer other examples must be to be affected.[2]

The fact show that the our model is very sensitive to the two parameters. So We try different values by using GridSearchCV tool in sklearn, which can optimize the parameters by cross-validated grid-search. The best parameters we found are 'C': 10.0, 'γ': 0.0001 with a score of 0.97. The training result is listed below.

$$\begin{aligned} \text{training score: } & 1.0 \\ \text{validating score: } & 0.97 \\ \text{confusion matrix: } & \begin{bmatrix} 24 & 1 & 0 \\ 1 & 20 & 0 \\ 0 & 0 & 14 \end{bmatrix} \text{ (from left to right are book, box and cup)} \end{aligned}$$

4.3.2 CNN

Apart from the SVM method, we also build a Convolutional Neural Network (CNN) on PyTorch. To keep the video real-time, the classifier works when and only when a new target appears.
Considering the more training parameters the network has, the longer it takes the classifier to work, even though we prove ResNet18 performs well in this project, we designed a shallow but neat network to do the work, whose architecture is shown in figure 8. Two batch normalization layers are added to enhance the network generalization ability. To fascinate the classification speed, input images are converted to gray maps and resized to 128×128 $pixel^2$. PCA is not used here whereas images are standardized to have average equal to 0 and variance of 1.

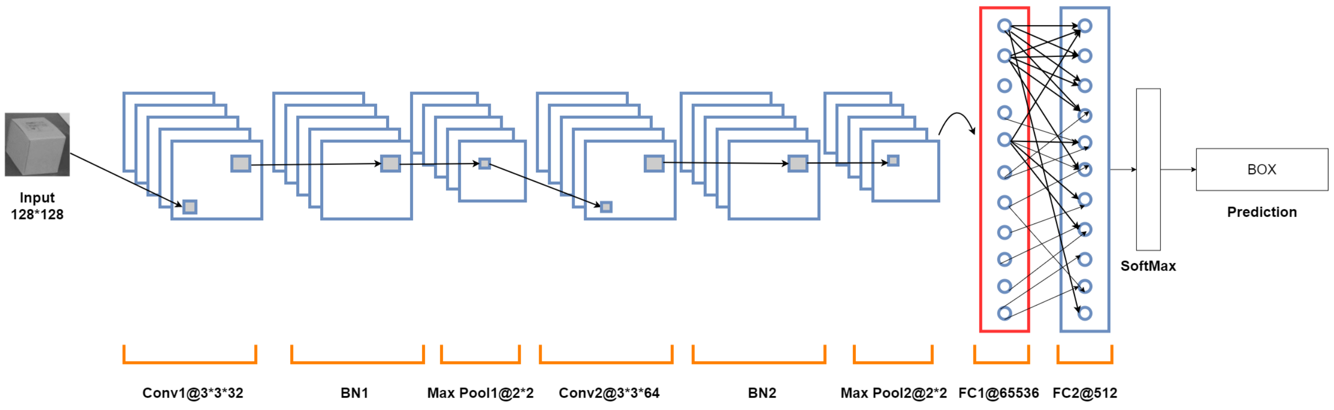

Figure 8: CNN architecture

The network is trained on Tesla K80 with batch size of 30 and ADAM optimization for 30 epochs. The learning rate is set 10^{-4} to get a high accuracy.
The trained network has an accuracy of 100% on validation set and performs well when applied on videos. Since the network working speed is faster than the tracking algorithm, there is no obvious stutter in video during classification.

4.4 Testing Result

After getting a pre-trained model, we will applied it on the provided video to test if it can generalize to more unseen images. Unfortunately, although the training and testing result is quite well on our self-selected data set, the result on the provided video is not that good.

The box and book are confused sometimes, it can be understood because they have very similar features. But it is weird that cup is often be recognized as book or box. While in training data set, cups are always 100% right classified. We deduct that it is because the backgrounds of objects in training dataset are always pure color while the objects in video are cropped out with some part of the conveyor. The conveyor has some features that are similar with book and box, like straight lines.

We also try to use online pictures with diverse backgrounds as training dataset, but it didn't turn out well. At last, we train the classifier with provided video without occlusion and applied it on the videos with occlusion, the accuracy of result up to 100%.

5 Result Presentation and Conclusion

5.1 Results

Concluding our project we have summarized two solutions.

5.1.1 Solution 1: GC+CMM+CNN

In the videos, targets are bounded with RED box while other candidates with GREEN ones. YELLOW points represent the predicted position of the target centroid. Front and top views are used to present the 3D motion of targets.
Under occlusion: https://www.youtube.com/watch?v=GWvI1aQguXo
Without shield: https://www.youtube.com/watch?v=-YFX-JQ75JU
From the videos we can see that our software is effectively capable of tracking visible objects through the use of a Background Subtractor in conjunction with the GC processes respectively defined in 3.1.2 allowing for a very clear and noise free recognition of objects. Whilst these are not entirely independent algorithms, their application and tuning improved the results significantly. Stereo matching is implemented here to fetch the target depth. For tracking objects occluded by the cardboard shield we experimented with 1-order Classical Motion Modeling which provided adequate results. It can be seen in the video that the motion prediction is not perfectly accurate but provides an estimate in position. Lastly, when it comes to object classification our algorithm based on a Convolutional Neural Network with excellent accurracy.

5.1.2 Solution 2: FLC+KF+SVC

The 3D coordinates of targets are shown in the left upper corner of the video.
Both with and without occlusion: https://www.youtube.com/watch?v=IoGjnXz-$_XI$
In second solution we tried a totally different way to solve the problem. FLC is introduced to detect and track objects in 2D. Triangulation acts as an important role in 3D reconstruction. Two independent KF emulate the target motion when it is under occlusion. For classification, SVC is used and trained with our self-taken pictures so the performance is not perfect.

5.2 Conclusion

Due to time constraints, many of our ideas have not been realized. Although our scenario is relatively simple, BS based on KNN is not as robust to the change of light and noises as we estimated, resulting in errors during tracking. BS related with Gaussian Mixture Model or Bayes can be tried in the future. When we put the methods for motion prediction into practice, KF works well in box path prediction while CMM performs better when estimating the motion of cups. Therefore, we deem an ensemble algorithm a direction to develop. As feature point extraction bothers a lot in SOF and triangulation, more related algorithms are needed to be discovered.For classification, we still lack a effective training data set.

To conclude, we have accomplished all main objectives set out through the effective use of a variety of technologies and methods, whilst greatly furthering our understanding of technologies that were not core components of the course. Whilst we recognize our solution is not perfect and applies strategies that would not be suitable for an industrial environment we deem it adequate for this project.

References

[1] Gunnar Farnebäck. “Two-Frame Motion Estimation Based on Polynomial Expansion”. In: *Image Analysis*. Ed. by Josef Bigun and Tomas Gustavsson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370. ISBN: 978-3-540-45103-7.

[2] *RBF SVM parameters*. URL: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py.

[3] *The return value of Farneback Algorithm*. URL: https://stackoverflow.com/questions/38131822/what-is-output-from-opencvs-dense-optical-flow-farneback-function-how-can-th.