Gezamenlijke opleiding UHasselt en KU Leuven

2020•2021
Faculteit Industriële Ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis
Hand Localization Using YOLO on Depth Data

PROMOTOR :
Prof. dr. ir. Eric DEMEESTER

COPROMOTOR :
Dhr. Stijn DEBRUYCKERE

BEGELEIDER :
Mevrouw Yanming WU

Maikel Both
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

2020•2021
Faculteit Industriële Ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Hand Localization Using YOLO on Depth Data

PROMOTOR :
Prof. dr. ir. Eric DEMEESTER

COPROMOTOR :
Dhr. Stijn DEBRUYCKERE

BEGELEIDER :
Mevrouw Yanming WU

Maikel Both

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Preface

This thesis contains the study and evaluation of different hand detection algorithms. It tries to evaluate these algorithms by comparing testing results. The tests include overall performance, accuracy ratings and their resistance to occlusion and different environments. This thesis is inspired by the Human Interface Mate (HIM), an augmented reality application developed by Arkite. This thesis has given me the opportunity to go in depth on hand detection and learn a tremendous amount of interesting information about it. I would like to thank Ir. Kim Rutten, Stijn Debruyckere, Prof. dr. ir. Eric Demeester and Yanming Wu for assisting me during the thesis and providing advice and ideas to guide me. I would also like to thank my family and friends for supporting me in my work.

Contents

List of Figures

List of Tables

Abstract

The Human Interface Mate (HIM) of Arkite is an augmented reality device that assists production operators in real time. It uses 3D sensors and a projector to inform and guide operators in their work area during the production process. It can be improved by implementing real-time hand detection to check when an operator enters a predetermined area with his or her hand before proceeding to the next step in the production process. It is important to be able to detect a person's hands with and without gloves equipped.

A literature study is first conducted, which leads to four promising solutions selected based on hardware requirements, performance and other criteria. Those possible solutions are YOLO, YOLO combined with OpenPose, Voxel-to-Voxel and Anchor-to-Joint. YOLO and OpenPose are pretrained for RGB images while Voxel-to-Voxel and Anchor-to-Joint require depth images. Work has been done to test each of the possible solutions after which it was decided to retrain YOLO to detect hands based only on depth images.

The retraining of YOLO is done on depth images to avoid any privacy concerns caused by RGB images. The ITOP and Arkite custom dataset is used for the training and testing. The results show a mean average precision (mAP) of 98.60% on the ITOP dataset and a good mAP of 13.59% on the Arkite dataset. Additional training is required to improve the mAP on the Arkite dataset.

Abstract in Dutch

De Human Interface Mate (HIM) van Arkite is een augmented reality apparaat dat productiemedewerkers in real time helpt. De HIM maakt gebruik van 3D-sensoren en een projector om medewerkers tijdens het productieproces in hun werkgebied te informeren en begeleiden. Dat kan worden verbeterd door realtime handdetectie te implementeren om te controleren wanneer een medewerker met zijn of haar hand een vooraf bepaald gebied betreedt, om door te kunnen gaan naar de volgende stap in het productieproces. Het is belangrijk om de handen te kunnen detecteren met en zonder handschoenen.

Een literatuurstudie is uitgevoerd die leidde tot vier veelbelovende oplossingen die gekozen zijn op basis van hardwarevereisten, performantie en andere criteria. Deze mogelijke oplossingen zijn YOLO, YOLO in combinatie met OpenPose, Voxel-to-Voxel en Anchor-to-Joint. YOLO en OpenPose zijn al voor getraind op RGB afbeeldingen terwijl Voxel-to-Voxel en Anchor-to-Joint dieptebeelden gebruiken. Er is werk verricht omtrent elk van de mogelijke oplossingen waarna er beslist is om YOLO te hertrainen om handen te detecteren op basis van dieptebeelden.

Het hertrainen van YOLO wordt gedaan op dieptebeelden om bezorgdheden omtrent privacy bij RGB beelden te voorkomen. De ITOP en Arkite's eigen dataset worden gebruikt voor het trainen en testen. De resultaten tonen een hoge gemiddelde precisie van 98.60% bij de ITOP dataset en een goede precisie van 13.59% bij Arkite's eigen dataset. Bijkomende training is vereist om de precisie bij de Arkite dataset te verhogen.

Chapter 1

Introduction

1.1 Situating

This master's thesis is conducted at Arkite NV. Arkite, with its head office at C-Mine Crib in Genk, develops and sells the Human Interface Mate (HIM). This is an augmented reality technology that assists production operators in real time. Arkite has put this technology into practice within several companies. One of these companies is Atlas Copco.

Atlas Copco was looking for a system to support the operator during the complex manual assembly of various devices. This provides a solution that eliminates incorrect assembly of components due to human error. Atlas Copco's goal was to reduce the number of defective products. The HIM addressed the issues mentioned and introduced a methodology to get new employees on board through its easy and efficient training capabilities [1].

The HIM technology uses a 3D sensor and a projector to inform and guide the operator in his work area during the production process. In addition to this guidance, the process steps that the operator carries out are also validated. Thanks to the HIM, an employee in a manufacturing company can perform complex operations with almost as much certainty as that of a robot. For the analysis of human actions, 3D sensor data are processed and combined with the various communication flows within the production equipment. The "virtual twin" of the workstation, which is created with the help of the 3D sensors and projector, consists of video images of 6.5 million pixels per second, on which about 500 meshes are defined. Optimal calculation processes therefore provide an almost unlimited number of virtual sensors to digitize these workstations [2].

The 3D sensors consist of an infrared camera and an RGB camera. The infrared camera sends out an infrared signal and picks it up again. The time interval between transmission and reception is measured and the distance to the camera is determined on the basis of this time-of-flight (TOF). The RGB camera monitors the working field and offers complementary footage. Human actions and the presence of objects are validated on the combined streams of the 2D RGB camera, the infrared camera and its 3D model of the working environment.

The graphical user interface (GUI) is built using the programming language C#. The image processing is written in C++ with the addition of various libraries such as OpenCV. C++ is a programming language that executes code very quickly because, unlike many other programming languages, the written code is converted directly into machine code.

1.2 Problem definition

The HIM currently detects when any volume enters a predefined area on the workbench to proceed to the next step in the process. However, it is not checked whether this is a person's hand, which indicates a certain intention, or something else, their elbow for example (accidentally). That could result in the HIM going to the next step in the process too quickly and skipping important steps. Detecting the location of the hand of the operator is a solution to this problem, ensuring that no steps in the process are skipped and to avoid accidental (false) triggers. The hand detection can be done on RGB images but this could possibly raise a privacy concern. That concern can be avoided when only using depth and/or IR images to detect the hands.

1.3 Goals

The overall goal is to implement hand detection based on IR and/or depth images (or as a backup solution on RGB images). The detection can be used to make sure the operator intentionally entered the expected area and to trigger the next step in the process. A first extension to this goal is to fully detect the hand and its orientation. While it is already very useful to know the location of the hand, it is even better if the orientation of the hand is known as well as the location of the wrist and fingertips. Another extension to this is to also be able to detect which pixel belongs to the hand and which does not, also known as hand segmentation. As a last extension, we can look at the exploitation of videos. Many solutions to detection problems are to look at each image separately. However, consecutive frames in a video can provide much more information which can result in a more robust hand tracking implementation.

1.4 Outline of the thesis

Chapter 2 is a literature study on various existing techniques that are related to hand detection. The chapter ends with a conclusion on the most promising methods. Chapter 3 explains the evaluation experiments of the most promising methods. Based on the evaluation result, YOLO is selected as the final implemented method. Chapter 4 explains how to train YOLO on depth images. Chapter 5 discusses the results achieved after training YOLO on the ITOP and Arkite dataset.

Chapter 2

Literature study

This chapter contains the literature study, which explores all the different techniques. They are all evaluated against each other after which the most promising ones are chosen.

2.1 Detection Methods

There are many existing hand detection and hand gesture detection methods. Within this literature study, eleven different methods are compared with each other based on the needs of Arkite.

The hand detection methods fall in three categories based on their output. The first category is hand localization, the methods within this category are able to localize the hand from a bigger picture and narrow it down to a bounding box. A second category is hand pose estimation, these methods usually start from the bounding box of a hand and use that to estimate the hand joints and fingers location. The last category is gesture recognition, the methods within this category are looking at a series of images to recognize certain gestures (such as waving or clapping) performed by a person. There are methods that belong in multiple categories but those are listed in the category that is best fitted for this thesis.

2.1.1 Hand Localization Methods

You Only Look Once (YOLO)

The YOLO algorithm [1] reframes object detection as a single regression problem. It only looks once at an RGB image and predicts the objects and their location within the image. A single convolutional network is used to evaluate the entire image and predict the bounding boxes and class probabilities for those boxes. Everything getting evaluated at once is one of the main reasons why YOLO has a good performance in terms of speed. The base algorithm used in the paper [3] works at a framerate of 45 FPS on a Titan X GPU. There is a faster version of YOLO, FastYOLO, that works at a framerate of 155 FPS on the same hardware. In comparison to other state of the art detection algorithms, YOLO makes more localization errors but less false positives. The algorithm mainly learns the general representation of the objects and does not look at details.

Further research into the YOLO algorithm has resulted in two updates that ensure it is faster and more accurate. YOLOv2 [4] and YOLOv3 [5] provide significant improvements on the framerate and precision rate.

YOLO can work at such a high speed that real-time detection is possible. Implementation of FastYOLO increases the framerate even more but also decreases the accuracy overall. A disadvantage of the method is that it only learns the general representation of objects. This means that the algorithm is only able to find the bounding box of the object and does not give any details about it, such as the orientation of detected hands.

Skeltrack

Skeltrack [6] is a library used for tracking human skeleton joints from depth images. It is mainly used to find a number of joints of a person from the image. The library tries to track the entire skeleton instead of only a particular part like the hands. It is able to track the hands but only so by finding the location of the hands and they have to be at least a predefined distance from the shoulders to be detected.

Viewpoint Invariant

The viewpoint invariant method proposed by Albert Haque et al estimates 3D human pose from a single depth image. The method is able to predict partial poses in the presence of noise and occlusion for different viewpoints. Unlike the V2V and A2J methods, the code for this approach is not publicly available [7].

2.1.2 Hand Pose Estimation Methods

OpenPose

OpenPose starts from an RGB image of which it calculates 2D confidence maps of body part locations and 2D vector fields that represent the degree of association between different body parts. A confidence map is a 2D representation of an area where a certain body part occurs in each pixel, a 2D vector field corresponds to a body part. Finally, the confidence maps and vector fields are aggregated in a greedy way to create 2D key points for every person in the image [8]. The calculated vector fields can be further used to connect different body parts of humans to form their skeleton. These vector fields provide information about the orientation of the body part.

One advantage of OpenPose is that it has already been trained for person detection, including body part orientation and hand detection with 2x21 key points. In addition, there is also functionality for tracking a single person. Finally, OpenPose has the option to use the built-in hand model with its key points on a bounding box that is provided. The disadvantage of OpenPose is that its speed depends on the number of people that are detected within the image, therefore it is slower than other algorithms such as YOLO. Another drawback is that OpenPose requires the person's shoulders or elbows to be visible to reliably detect the hands, however that can be solved by providing the bounding box of a hand to the algorithm.

Augmented Desk Interface

The paper by Yoichi Sato et al. [9] uses images from an IR sensor as input to detect hands and fingertips. The IR sensor measures the temperature of all the objects in front of it, which is then used to filter the pixels that have a temperature between 30 and 34 degrees Celsius. That threshold is around the body temperature, enabling the algorithm to find body parts very easily. Those pixels that lay within the threshold are further on used in a template matching strategy for finding fingertips.

Hand Shape Variation

This method uses a compact and efficient model of the surface deformation of human hands. Starting from noisy depth images, the hand shape is parameterized as a linear combination of a mean mesh in a neutral pose with a small number of offset vectors. The mesh is then articulated using standard linear blend skinning (LBS) to generate the control mesh of a subdivision surface. An energy is defined that encourages each depth pixel to be explained by the model, besides that the use of smooth subdivision surfaces allows optimization for all parameters jointly from a rough initialization [10].

The advantage of this method is that it is able to personalize the meshes of the hand. This allows the algorithm to adapt to a person's hand, improving the detection rate.

Hand Pose Estimation

The paper on Hand Pose Estimation from a Single Depth Image [11] uses a single noisy depth image to estimate the hand pose. That is done by applying three different steps to the input. Firstly, there is an initial estimation step that provides an estimation of the hand's in-plane orientation and location. The second step is the candidate generation step, producing a set of pose candidates from the Hough voting space by using the rotational invariant depth features. The last step is verification where the output is the result of an optimization problem.

Voxel-to-Voxel (V2V)

The paper on Voxel-to-Voxel (V2V) by Gyeongsik Moon et al makes use of a single depth map by casting the 3D hand and human pose estimation into a voxel-to-voxel prediction. The prediction uses a 3D voxelized grid where the per-voxel likelihood for each keypoint is estimated. This model is designed as a 3D CNN. The code of this approach is publicly available on Github [12].

Anchor-to-Joint (A2J)

The paper on Anchor-to-Joint (A2J) by Fu Xiong et al. uses an anchor-based regression network approach to estimate hand and body pose. Within A2J, anchor points able to capture global-local spatial context information are densely set on depth images as local regressors for the joints. An anchor proposal procedure is proposed to find informative anchor points towards certain joints. A 2D CNN is used as a backbone for A2J. The code of this approach is publicly available on Github [13].

2.1.3 Gesture Recognition Methods

Hand Gesture Recognition

The paper by Tomás Mantecón et al. [14] uses near-infrared images acquired by a Leap Motion sensor to recognize hand gestures. The system does not do any hand segmentation but computes Depth Spatiograms of Quantized Patterns, which is a global image descriptor. The descriptor is further on used to compress using a Compressive Sensing framework. The performed hand gesture is identified by analyzing the resulting descriptor by a set of Support Vector Machines.

Naive Bayesian Fusion

The research of this algorithm described in the paper [15] uses a Kinect camera to improve action recognition. Using a depth video sequence, the Depth Motion Maps (DMM) are computed from three projection views: front, side and top view. Following, the shape and texture features are extracted from the DMM. Those features are based on Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP) descriptors. Two fusion levels are used, the first one being a feature fusion level based on the concatenation of HOG and LBP descriptors. The second, a score fusion level, based on the naive-Bayes combination approach. The fusion levels are used to predict the action or gesture happening in the video sequence.

The method requires different projection views to perform as well as an entire video sequence to detect the gesture or action performed. That also means it is not detecting the hand itself in each frame.

2.2 Promising Methods

There are several different approaches to implement hand detection. These approaches range from using RGB images to depth video sequences as input and each have their own advantages and disadvantages. Furthermore, they each have different performance and benchmark results. These methods are all shown in table 2.1 in addition to the properties used to make a decision.

There are a multitude of limitations to keep in mind when selecting the most suitable solutions. The first limitation is the hardware limitation. Arkite uses an RGB camera and a near-IR / depth sensor where it looks at a person from only one viewpoint from above. Because of these constraints, the following methods are not suitable: Naive Bayesian Fusion [15], Augmented Desk Interface [9].

The second limitation is single frame detection. The algorithm needs to be able to detect the hand in each frame. Furthermore, the output needs to be the position of the hands visible in the input image and if possible multiple key points of the hands or hand segmentation. A dataset with annotated images of hands is used to train the method after which it needs to be able to detect hands in new images on its own. Because of that, the solution of Hand Gesture Recognition is not feasible as that method is only able to recognize pretrained hand gestures from a series of frames and is not able to detect a hand in every frame [14].

The next limitation is the performance. The solution is required to run at real-time with a framerate of at least 30 FPS on the hardware of Arkite. Because of this, methods showing a performance lower than 30 FPS in their paper are not suitable as the papers are often using

Name	Sensor	Output	Performance	Benchmark Results
YOLO [3, 4, 5]	Depth / RGB	Bounding box	YOLO: 45 FPS FastYOLO: 155 FPS on Titan X GPU	COCO mAP-50: 57.9 mAP
OpenPose [8]	RGB	2x21 hand keypoints Info on orientation	22 FPS on Nvidia GTX 1080 Ti	MPII dataset (wrist): 66.8 mAP
Voxel-to-Voxel (V2V) [12]	Depth	Coordinates of points of hand and fingers	Up to 35 FPS in multi-GPU environment	ITOP dataset (front-view, hands): 67.26 mAP ITOP dataset (top-view, hands): 62.44 mAP
Anchor-to-Joint (A2J) [13]	Depth	Hand: Lines representing fingers	Hand: 105.06 FPS on single Nvidia 1080Ti GPU	ITOP dataset (front-view, hands): 68.35 mAP ITOP dataset (top-view, hands): 59.35 mAP
Skeltrack [6]	Depth	Lines representing skeleton	60.4 FPS (Unknown Hardware)	
Hand Shape Variation [10]	Depth	Mesh of hand	Unknown	
Hand Pose Estimation [11]	Depth	Segmentation of each finger	12 FPS (i7 CPU 960 3.20 GHz, 24 GB memory) Not optimized, only 1 CPU used	
Viewpoint Invariant [7]	Depth	Location of different body parts	1.7 seconds per frame (0.588 FPS)	
Hand Gesture Recognition [14]	near-IR	Known hand movements		
Augmented Desk Interface [9]	IR	Segmentation of hand		
Naive Bayesian Fusion [15]	Depth 3 sensors	Action performed by hands		

Table 2.1: Every method with its properties

better hardware compared to Arkites HIM. Because of this limitation, the following methods are not suitable: Hand Pose Estimation with a performance of 12 FPS, Viewpoint Invariant with a performance of 0.588 FPS. In addition to that Hand Shape Variation does not have a performance reported in their paper and with no code publicly available it also gets excluded [10, 11, 7]. One exception to his limitation is OpenPose, reporting a performance of 22 FPS on the Nvidia GTX 1080 Ti. It does not get eliminated as it can be used to start from a bounding box and only detect the hand with its key points in the image. That can lead to a better performance [8].

Lastly, Skeltrack is not a promising method either. This method has the requirement that the hands need to be a minimal distance from the shoulders to be detected. The setup at Arkite uses a viewpoint from above resulting in the hands often being close to the shoulders or cases where the rest of the human skeleton is not visible. The lack of visibility of the human skeleton is resulting in a worse detection rate on hands [6].

Finally, the remaining methods are YOLO, OpenPose, Voxel-to-Voxel (V2V) and Anchor-to-Joint (A2J). Out of these methods, YOLO is the only method that can be used for hand localization. OpenPose, V2V and A2J can be used additionally for hand pose estimation.

YOLO and OpenPose are available pretrained on RGB images, however RGB images where the person is recognizable can cause privacy issues. Because of this, YOLO can be trained on depth images to find the bounding box of the hand. Following that, OpenPose can use that bounding box to get all the features of the hand. As OpenPose only uses the content of the bounding box, it is possible to use the RGB image for OpenPose only.

Voxel-to-Voxel and Anchor-to-Joint use depth images as input and are able to localize the fingers and other characteristics of the hand. Both methods have similar benchmark results scoring a mean average precision for top-view on hands of 62.44% (V2V) and 59.35% (A2J) on the ITOP dataset [13].

2.3 Datasets

Once a promising method is selected, it needs to be trained extensively. This is done by using a dataset that provides an extensive variation of annotated hands. The dataset should have varying environments and properties of hands, in particular when using RGB images the skin color or the presence of gloves can be important to provide a wide variety. Different levels of occlusion can help in the learning process but this heavily depends on the chosen method. Three datasets are listed below that may help the learning process, however it might be necessary to gather extra annotated images and use it in addition to the datasets to improve the results.

The University of Oxford has an annotated hand dataset of RGB images available consisting of 13,050 hand instances, of which 4170 are of high quality. The dataset contains many different people, including people with a different skin color. While collecting the data, there was no restriction imposed on the pose or visibility of people. The annotations only contain a bounding box, oriented with the wrist [16].

Indiana University's IU Computer Vision Lab has an RGB dataset that contains approximately 15,000 annotated hand instances. These annotations contain polygons with hand segmentations. Only four different people are shown throughout the dataset, resulting in a very low variety [17].

The ITOP dataset contains over 100,000 depth images from side and top views of a person in a scene. For each instance, the location of 15 human body parts including the hands are labeled with their location, relative to the sensor's position. 20 actors were used in gathering these instances [18].

2.4 Conclusion

Out of the eleven methods that are researched, there are four methods remaining as viable solutions. YOLO is the remaining solution for hand localization. OpenPose, Voxel-to-Voxel and Anchor-to-Joint are hand pose estimation methods and can be used once a bounding box is found by YOLO. The primary focus for testing is on YOLO with depth data, additionally the three hand pose estimation methods are tested as well. Figure 2.1 shows a summary of all the methods and the deciding factors.

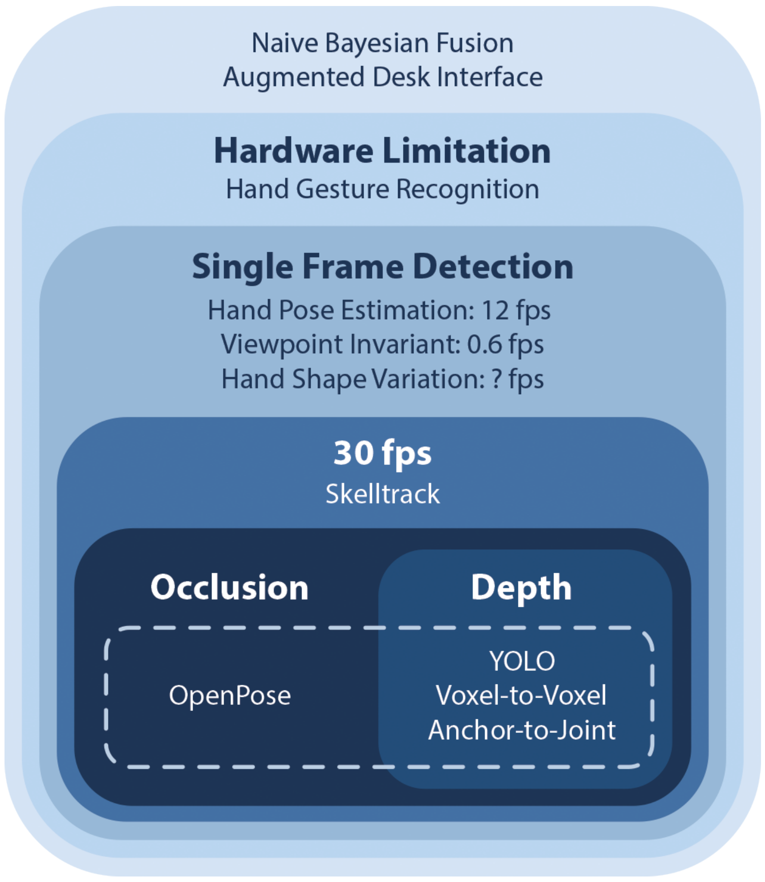

Figure 2.1: Methods and deciding factors

Using the right dataset is very important when training a method. The ITOP dataset is a very good dataset to train YOLO as it contains over 50,000 depth images of people from a top-view perspective, as in the Arkite use case. The dataset annotations include the location of the hand in the image and the segmentation of the hands. To improve the hand localization performance in the Arkite use, a custom dataset from Arkite can be annotated and added in the training dataset.

Chapter 3

Testing Methods

This chapter goes more in depth on the promising methods selected in Chapter 2, and explains the tests that are performed on those methods. The evaluation results are used to select the final method to be implemented.

3.1 YOLO

YOLO is the first of four promising methods to be tested. There are several versions of YOLO available of which YOLOv4 is chosen as this is the latest version written in C++. The tiny version of YOLO is preferred as the hardware in the HIM is not good enough to use the regular YOLO version in realtime. Because of that reason, tiny YOLO is used for testing, the accuracy is slightly lower than the regular version but the frame rate is significantly higher.

YOLO has a pretrained weights file available on their Github [19] which is used for this testing. The weights file is trained to detect 80 different classes but only the output for the person class is used. Running this version on a self-made video shows how it performs on new data similar to Arkite's. The model is not trained to detect people's hands specifically so a low accuracy is expected. This can be increased by retraining YOLO to detect hands specifically.

Figure 3.1 shows how YOLO is detecting the hands. The bounding boxes found by YOLO are not always good, the figure shows three different scenarios. The left image shows how both hands are detected but the bounding box is slightly small. The middle image shows how only one hand is partly detected even though there is nothing around the undetected hand that could possibly cause this. The right image shows how both hands are detected in one giant bounding box.

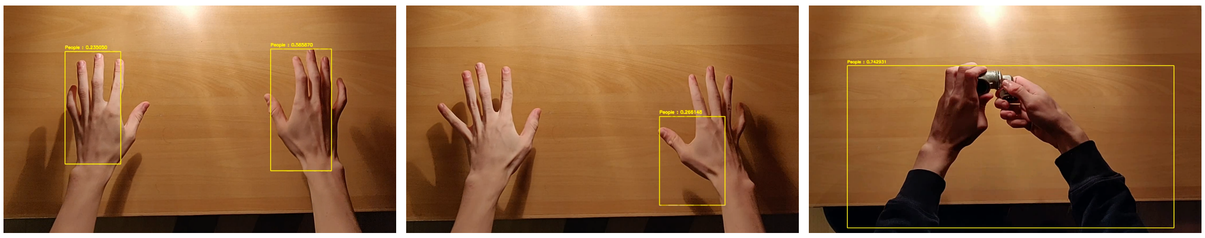

Figure 3.1: Results using the pretrained YOLO model.

3.2 YOLO and OpenPose

OpenPose is able to run on its own but to properly detect people's hands it needs to be able to see the shoulders clearly as well. Arkite has its camera positioned above the working place facing downwards. This results in a top view where the shoulders of the person are not always visible. That is the main reason why OpenPose is not a good option to run on its own.

OpenPose has the option to detect hands starting from a bounding box around the hand. The problem for this solution is that the bounding box is required first. YOLO can be used to easily get the bounding box of the hands and feed these to OpenPose. This allows OpenPose to annotate the hand further and find all the different keypoints.

Testing the combination of YOLO and OpenPose on new data shows that the image processing speed decreases significantly after using OpenPose. The frame rate decreases from 50 frames per second (fps) when only using YOLO, to 7 fps when adding OpenPose. Additionally the accuracy is very low due to OpenPose needing to know which hand is being detected in the bounding box. This information is not supplied by YOLO, causing a default value of the left hand to be given. When there are two bounding boxes detected, the left bounding box can be flagged as left hand and the right one as right hand, however this is not always accurate. Figure 3.2 shows on the left image what a good detection looks like and on the right an example where it was not able to detect anything from the given bounding box.

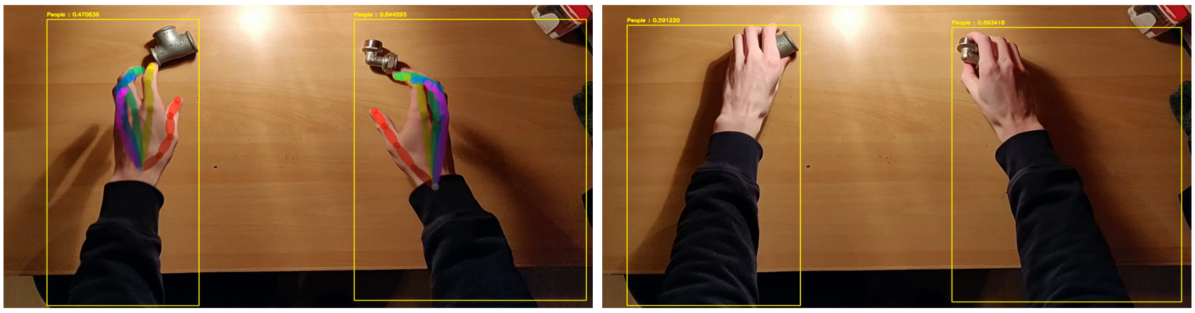

Figure 3.2: Results using OpenPose in combination with YOLO.

3.3 Voxel-to-Voxel

Voxel-to-Voxel (V2V) has its training code publicly available on Github [20], but there is no inference code available. A requirement for V2V is that the hand center estimation needs to be pre computed by DeepPrior++. Many hours and attempts have been put in V2V in an attempt to reproduce the results shown in the paper [12], but it kept on producing new errors which took too long to fix. Because of that, it has been decided to abandon this method and work on the other methods.

3.4 Anchor-to-Joint

Anchor-to-Joint (A2J) has its training code and inference code available on Github [21], but the inference code is heavily adjusted for each dataset as it starts off with the bounding box of the hand. The method requires a bounding box of a hand to base its detection on. Using its inference code and

In addition to that, attempts at reproducing the results in the paper have resulted in the detections shown on figure 3.3. When giving the bounding box of the hand to the algorithm, it returned the points that are seen as the grey dots on the hand visible in the image. This image is one of the images that was initially used by the writer of the research paper [13] to train the method.

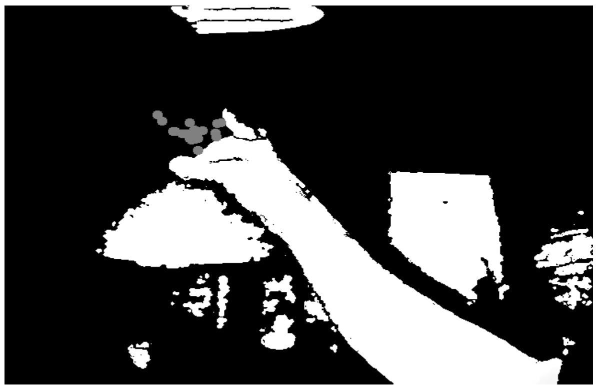

Figure 3.3: Results using Anchor-to-Joint

3.5 Decision

Work performed on V2V and A2J has resulted in either no results or bad results. Those methods are not looked into further at this stage due to the bad preliminary results and limited time constraints. This leaves YOLO with the possible addition of OpenPose as the remaining option. OpenPose has not shown great results, which partly might be because of the bounding box of the hand not always correctly annotated. Since hand localization is the primary goal, the priority is now to retrain YOLO on depth data. The retraining has to happen from scratch as it is based on depth data to prevent any privacy concerns.

Chapter 4

Training YOLO

This chapter explains the different steps on training the YOLO model. First, the configuration changes compared to the default model are explained. Second, the ITOP dataset and how it is used to train YOLO is explained. Following that, Arkite's data are explained and how they get annotated using LabelImg. The last part goes more in depth on the training process of YOLO, how it is initiated and how the training can be continued on another dataset. This also explains how the automatic mAP calculation from YOLO can be used on the datasets.

4.1 Configuration

The configuration is saved in the `yolov4-tiny.cfg` file. This file includes all the variables that can be adjusted. The default configuration for tiny YOLO is used, which gets further adjusted to the project needs. Instead of the 80 different objects the regular model can detect, only one object needs to be detected here. This is adjusted by changing the classes variable in each of the YOLO layers within the configuration. Additionally, the last convolutional layer before each YOLO layer has the filters variable that gets adjusted to `(classes + 5) * 3`, which is 18 in this case.

Depth images are used instead of RGB images and because of that change, additional variables get changed within the configuration file. The number of channels gets changed from three to one. The variables for angle, saturation, exposure and hue get commented out as these are not needed for one channel images.

4.2 ITOP Dataset

The ITOP Dataset is used to initially train the new configuration of the YOLO network. The dataset contains depth images of a person from a top-view perspective. Along the images, the dataset provides the coordinates of each body part detected as well as the segmentation of each body part. The images shown in figure 4.1 show the coordinates of left and right hand as well as their segmentation. The coordinates are visualized by a circle where the middle point is the given coordinate. The right hand is shown in red and the left hand is shown in green in the images.

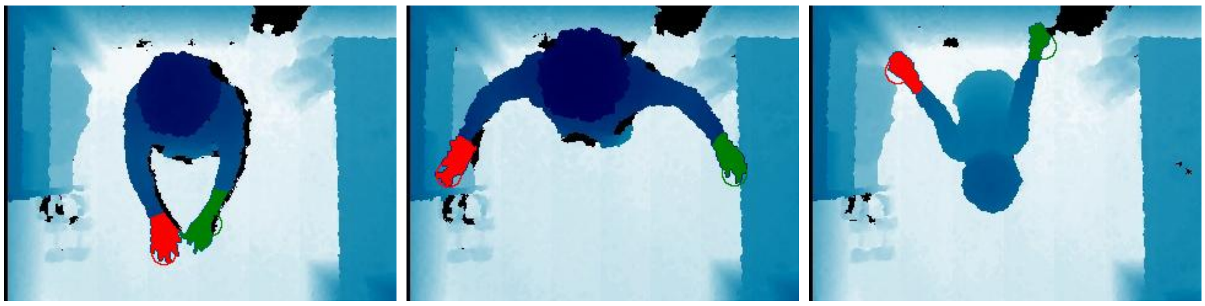

Figure 4.1: Three examples of ITOP Dataset with good segmentation.

4.2.1 Filtering Dataset

Figure 4.1 visualizes a good segmentation, however there are several cases where there is either something wrong with the segmentation or the coordinates of the hands. The first irregularity is where segmentation is correct but the coordinates of the hand do not match with the segmentation of the hands. Three instances of this are shown in figure 4.2

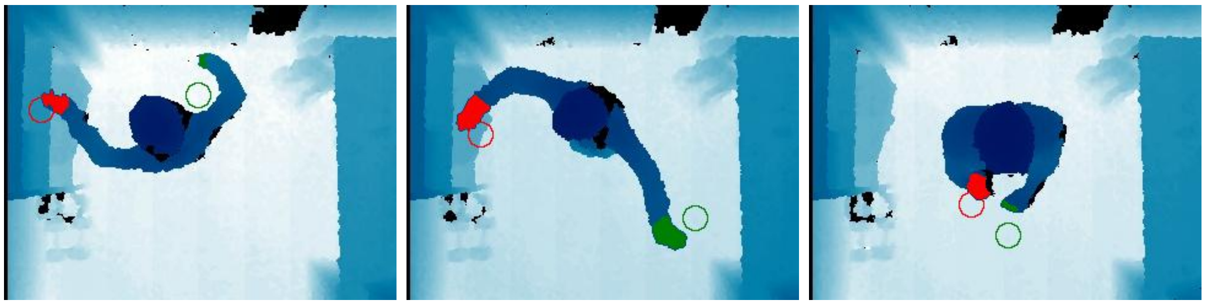

Figure 4.2: Three examples where coordinate of hand and segmentation do not match up.

A second irregularity is where both the segmentation and the coordinates of the hand are incorrect. The images shown in figure 4.3 show how either a hand is being detected as both hands, both hands are detected as one hand or the segmentation and coordinates are completely incorrect.

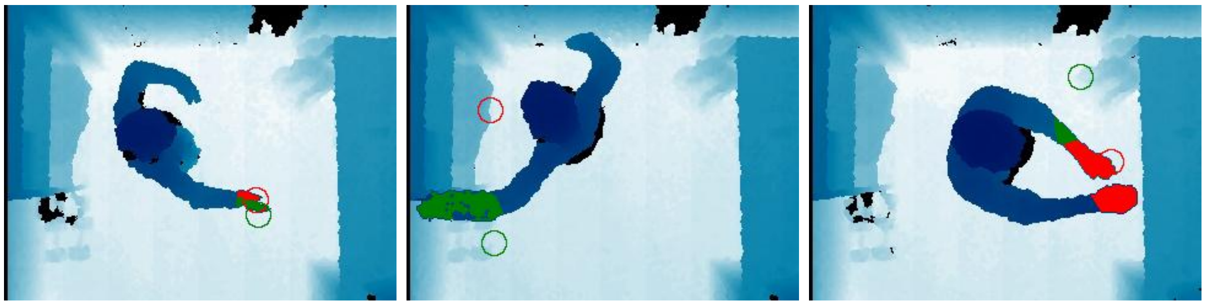

Figure 4.3: Three examples where hand coordinates and/or segmentation are completely wrong.

The dataset is filtered down to only contain the examples where the segmentation and coordinates of both hands match up with each other. This gets rid of the images that are partly wrongly labeled where either the hands are segmented incorrectly or the coordinates of the hand are wrong. This filter removes any images where only one or no hands are visible. There are more sophisticated ways possible for filtering, but not used in this thesis due to time limitation and scope of the thesis. This filtering reduced the amount of images from 39,795 items down to 2,257 items. Within these 2,257 images there are still some bad annotations but the majority of these are correct.

The goal of this dataset is first to test the configuration and to verify that YOLO works correctly using depth images. To verify this, the small batch of 2,257 images is used to train and see if YOLO can process the images. The bounding boxes used to train in this test are calculated by using the segmentation of the hands. They are calculated in such a way that all pixels of the segmentation are included in the box. Visualization of the bounding box is shown in figure 4.4.

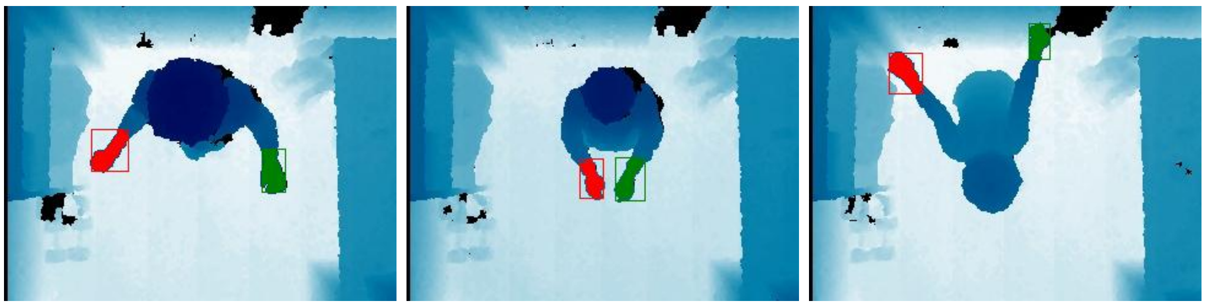
Figure 4.4: Three examples visualizing the bounding box around the segmented hands.

4.2.2 Test results

The accuracy of YOLO after training on those 2,257 images is not important as this is purely a test to see if YOLO can work correctly with depth images. After training YOLO, several new images are used to test if YOLO is able to correctly process them. The result shown in figure 4.5 proves that YOLO is able to work correctly on depth data and in this case is already able to detect one of the hands.

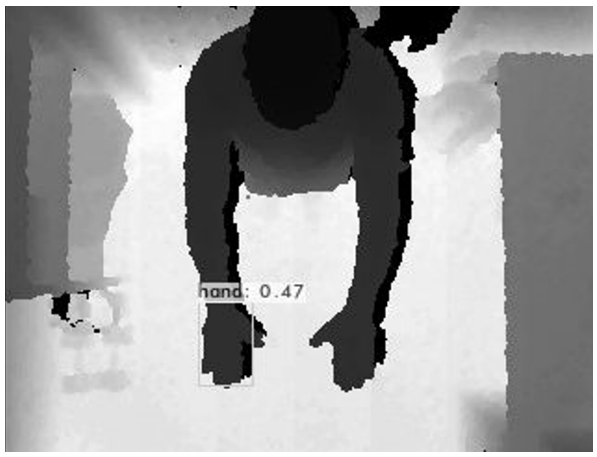

Figure 4.5: YOLO detection on new image after initial training.

4.2.3 Annotation Solution

Using the entire dataset is very beneficial for the end result of the detections. Because of that, a solution is needed to update the bad annotations. The required edits to the annotations is often a minor move of the bounding box or resizing it. To update these bounding boxes, the LabelImg tool is used. This tool is explained more in depth in the next section.

4.3 Arkite Data

The data provided by Arkite are recordings saved in a custom file extension that can be read by using their reader written in C++. The training of YOLO is done in Python, which means that the data needs to be converted to a file that Python can read. As YOLO requires images to train its model, the data is converted from video into images of each frame that can be saved as PNG files.

Arkite saves their depth data in 16 bit signed integer format. The data is signed integer instead of unsigned integer because it uses the value of -11 when no depth value can be calculated by the sensor. The first step of converting the data is to change all these negative values to zero, this makes it easier as the data can now be seen as 16 bit unsigned integers.

The data gets loaded in C++ using the OpenCV library. OpenCV has the ability to save images but only in 8 bit unsigned integer format. Converting the Arkite data from 16 bit down to 8 bit would mean a great loss in quality. To avoid that, the OpenCV library is used to save each frame in yaml files. These files store the images by writing the value of each pixel in a matrix. The yaml file is then read using a Python script and converted to a 16 bit image using the PIL Python library. This image can now be used for annotation and training YOLO.

The last step to be able to use the images for YOLO, is annotating the hands in each image. The graphical image annotation tool LabelImg is used to do this task [22]. This tool allows the user to annotate using bounding boxes, which then automatically gets saved in the YOLO format in a separate text file for each image. The tool is shown in figure 4.6.

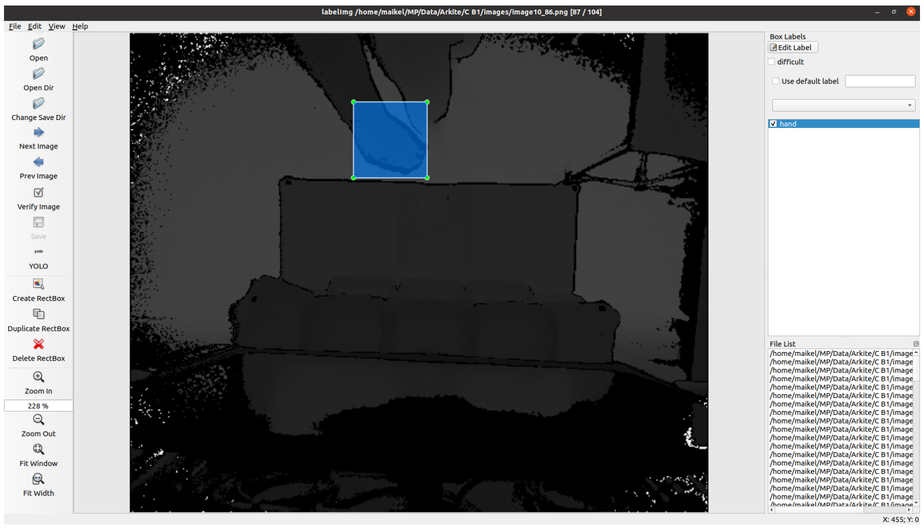

Figure 4.6: Annotating images using the LabelImg tool.

4.4 Training Process

4.4.1 Preprocessing

The ITOP and Arkite dataset are both used to train the model. Preprocessing is required to ensure that these two datasets work well together. Arkite stores its data by saving the distance of each pixel to the sensor in millimeters. ITOP on the other hand saves the distance in meters. The Arkite data is converted into meters by dividing each value by 1,000.

As the data is originally saved in float numbers, it is converted over to integer values ranging from 0 to 65,535 (16 bits). This is done by dividing every value by five and multiplying with the highest possible number, 65,535. The value of five is chosen so every object detected within five meters of the sensor is still shown. Anything beyond this distance ends up having a value of over the maximum value and will default to the maximum value.

4.4.2 Requirements

Training YOLO requires three different files, a data file named `obj.data`, a configuration file named `yolov4-tiny.cfg` and an initial weights file named `yolov4-tiny.conv.29`. The data file contains the amount of classes, paths to the training and validation text files, the path to the object names file named `obj.names` and the path to the backup folder. The training and validation text files consist of the path to each image used for training or validation. The object names file is a text file that has the name of each detectable object on separate lines. The line number of the object names file, starting at zero, is the id that gets assigned to each detectable object. An example of the obj.data and obj.names file is shown below. The names file is only one line in this case because YOLO is training to only detect the hands.

Listing 4.1: Example of obj.data

```
classes = 1
train = path/to/train.txt
valid = path/to/valid.txt
names = path/to/obj.names
backup = path/to/backup/
```

Listing 4.2: Example of obj.names

```
hand
```

4.4.3 Initial Training

The training of YOLO starts off with the entire ITOP dataset. It is initiated by going to the folder where the YOLO repository is cloned and running the command in the command prompt shown below.

```
./darknet detector train path/to/obj.data path/to/yolov4-tiny.cfg
path/to/yolov4-tiny.conv.29 -map
```

The graph shown on figure 4.7 pops up when the training is initiated and shows different statistics of the training process. The current average loss, the current iteration and approximate time left are shown at the bottom of the graph. The graph itself displays the average loss changing throughout all the iterations. A weights file gets saved every 1,000 iterations and also at the end of the training. Once the training is finished, the final weights file can be used later on for detection. The red graph indicates the mean average precision on the testing set throughout the training process.

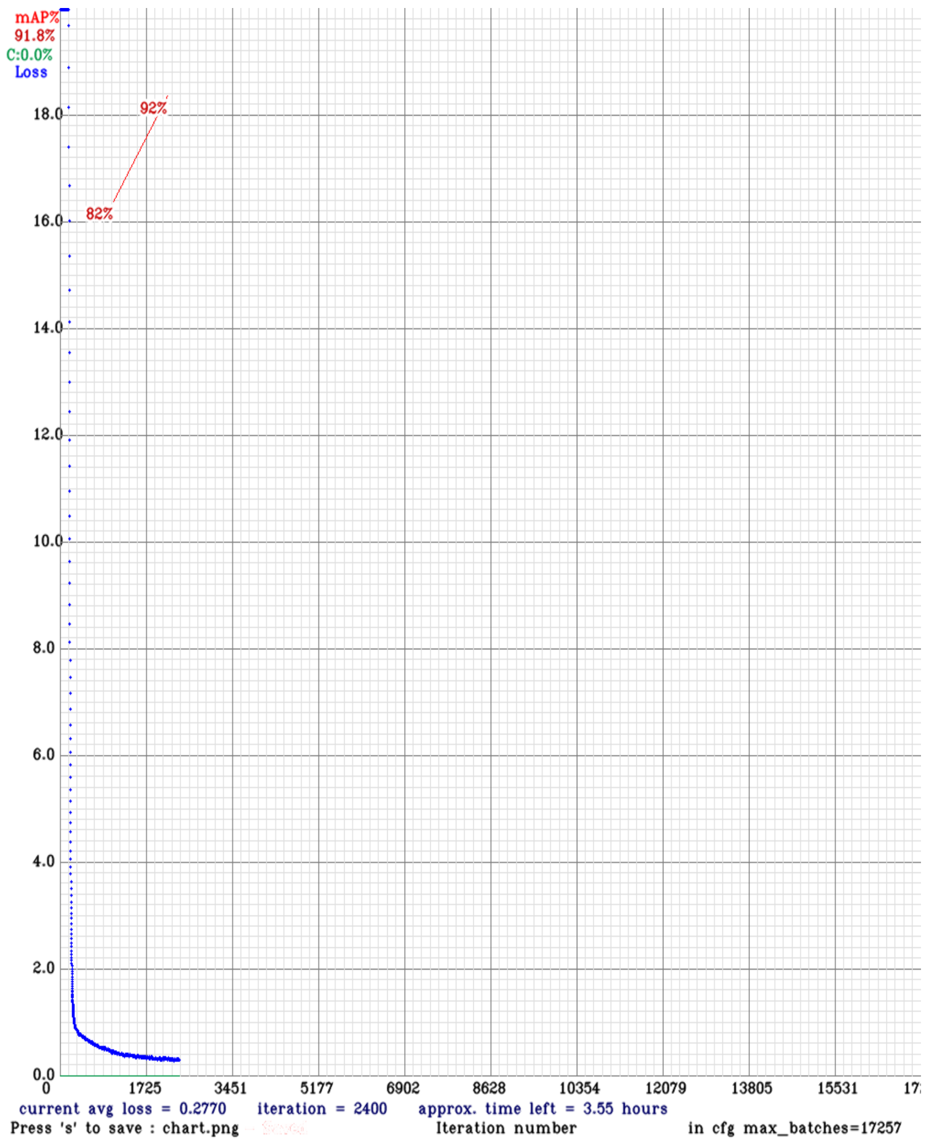

Figure 4.7: YOLO training graph.

The training process can be paused at any time by stopping the process. To resume training the latest weights file saved in the backup folder is used. Resuming training is initiated by running the following command in the command prompt:

```
./darknet detector train path/to/obj.data path/to/yolov4-tiny.cfg
path/to/latest.weights
```

4.4.4 Transfer Learning On New Data

After the initial training of YOLO, the latest weights file is used to initialize the training while training for new images. To add new images, the path to each image is added to the `train.txt` file and the values of max batches and steps are adjusted in the `yolov4-tiny.cfg` file. Lastly, the training process is started up again using the following command in the command prompt:

```
./darknet detector train path/to/obj.data path/to/yolov4-tiny.cfg
path/to/latest.weights
```

Chapter 5

Results and Discussion

5.1 Training on ITOP Dataset

YOLO is first trained on the entire ITOP Dataset. Figure 5.1 shows the learning curve throughout the training process. The blue and red graph, that display the current average loss and mean average precision (mAP), stabilizes towards the end of the graph and they are not changing values anymore after 14,000 iterations. This indicates that additional training is not needed on the ITOP dataset and that the remaining 26,000 training images are not required anymore.

The mAP value shown in the top left corner of the figure is the precision value at the end of the training. This shows that this trained version of YOLO has an average precision of 98.67% on the test set.

Using the trained model on the Arkite test set shows a very low mAP value of 7.35%. A reason for this significant drop is that the ITOP dataset consists of people moving around with a clear background and no tables nearby. Most of Arkite's data consists of people moving their hands above a table which immediately gives a different background.

5.2 Transfer Learning on the Arkite Dataset

The next part of the learning process is Transfer Learning on the Arkite dataset. The weights file generated by training on the ITOP dataset is now used as the starting point for training on the Arkite dataset. The Arkite dataset available is rather small in size compared to the ITOP dataset so different learning rates will be used.

The Transfer Learning process is done three times, each time with a different learning rate. The three learning rates used are 0.00261, 0.00087 and 0.000261. The first value is the same learning rate used on the ITOP dataset, the second and third rates are one third and one tenth of the first value. The different learning rates change how much the weight file adapts to the new Arkite data. The learning rate is reduced to not lose the weight values obtained with the ITOP dataset too fast.

Training with the same learning rate as the ITOP dataset results in a mAP of 42.79% on the ITOP dataset and 22.05% on the Arkite dataset. Using one third of the learning rate results in a

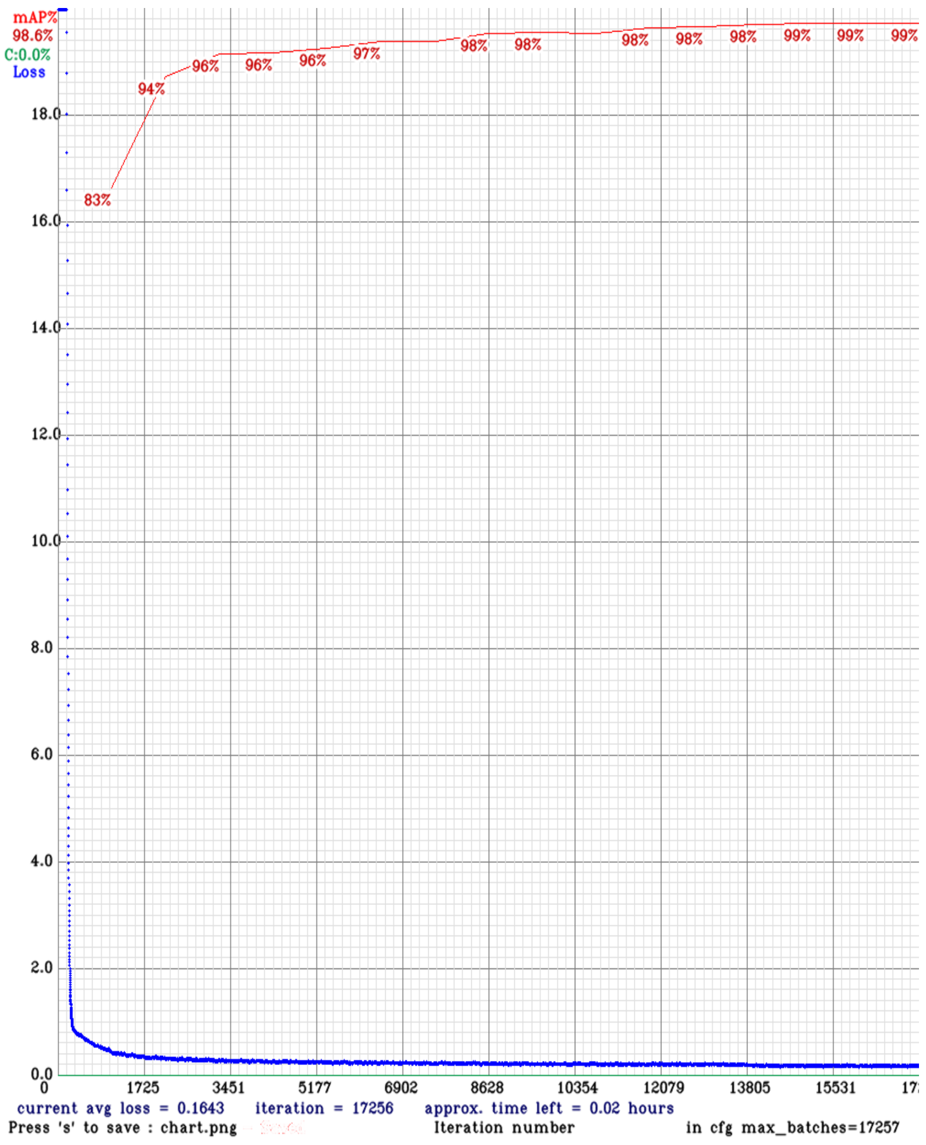

Figure 5.1: YOLO training graph on ITOP dataset.

mAP of 98.60% on the ITOP dataset and 13.59% on the Arkite dataset. Using one tenth of the learning rate results in a mAP of 98.66% on the ITOP dataset and 9.72% on the Arkite dataset.

Those numbers show that when using the same learning rate that accurate detection on the ITOP dataset gets much harder quite quickly. The learning rate that is one tenth of the ITOP learning rate shows that when the rate is too low, it results in a very small improvement on the Arkite mAP and almost no loss on the ITOP mAP. The learning rate that is one third shows the best results. The ITOP mAP decreases a small amount but the Arkite mAP is also increasing a decent amount.

Chapter 6

Conclusion

The purpose of this thesis was to research different hand detection methods for Arkite's HIM device, and to choose the best one to test and implement. The literature study has shown that YOLO was the best method of all the researched ones based on the requirements of Arkite. YOLO has been trained to work on depth data to detect hands by using the ITOP dataset and Arkite's custom dataset.

YOLO was initially trained on the (filtered) ITOP dataset that resulted in a mean average precision (mAP) of 98.67%. Using the method of transfer learning has shown that the learning rate should be a third of the rate used for the ITOP dataset when training on the Arkite data. It improved the mAP on the Arkite dataset by 6% to reach 13.59% while decreasing the ITOP mAP by only 0.07%.

The YOLO model can be improved by adding new Arkite data to improve the average precision rating. Additional work can be put into Voxel-to-Voxel and Anchor-to-Joint to get those methods properly working and be able to compare them to OpenPose. With that, the output of the YOLO model can be used further by the hand pose estimation methods to find the location of the fingers and wrist.

Bibliography

[1] Arkite, "Meet our customers." Accessed on: August 4, 2021. [Online]. Available: https://arkite.com/references/.

[2] Arkite, "Proposal master's thesis."

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection,"

[4] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger,"

[5] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement,"

[6] J. Rocha, "Skeltrack - reference manual." Accessed on: August 4, 2021. [Online]. Available: https://people.igalia.com/jrocha/skeltrack/doc/latest/.

[7] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, "Towards viewpoint invariant 3d human pose estimation,"

[8] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: Realtime multi-person 2d pose estimation using part affinity fields,"

[9] Y. Sato, Y. Kobayashi, and H. Koike, "Fast tracking of hands and fingertips in infrared images for augmented desk interface," in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, pp. 462–467.

[10] S. Khamis, J. Taylor, J. Shotton, C. Keskin, S. Izadi, and A. Fitzgibbon, "Learning an efficient model of hand shape variation from depth images," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2540–2548. ISSN: 1063-6919.

[11] C. Xu and L. Cheng, "Efficient hand pose estimation from a single depth image," in *2013 IEEE International Conference on Computer Vision*, pp. 3456–3462. ISSN: 2380-7504.

[12] G. Moon, J. Chang, and K. M. Lee, "V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] F. Xiong, B. Zhang, Y. Xiao, Z. Cao, T. Yu, J. Zhou Tianyi, and J. Yuan, "A2j: Anchor-to-joint regression network for 3d articulated pose estimation from a single depth image," in *Proceedings of the IEEE Conference on International Conference on Computer Vision (ICCV)*, 2019.

[14] T. Mantecón, C. R. del Blanco, F. Jaureguizar, and N. García, "Hand gesture recognition using infrared imagery provided by leap motion controller," in *Advanced Concepts*

for Intelligent Vision Systems (J. Blanc-Talon, C. Distante, W. Philips, D. Popescu, and P. Scheunders, eds.), Lecture Notes in Computer Science, pp. 47–57, Springer International Publishing.

[15] A. Ben Mahjoub, M. Ibn Khedher, M. Atri, and M. El Yacoubi, *Naive Bayesian Fusion for Action Recognition from Kinect.*

[16] A. Mittal, A. Zisserman, and P. Torr, "Hand detection using multiple proposals," in *Proceedings of the British Machine Vision Conference 2011*, pp. 75.1–75.11, British Machine Vision Association.

[17] "EgoHands: A dataset for hands in complex egocentric interactions | IU computer vision lab."

[18] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, "ITOP dataset." type: dataset.

[19] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020. Git code https://github.com/AlexeyAB/darknet.

[20] G. Moon, "V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map." Git code https://github.com/mks0601/V2V-PoseNet_RELEASE.

[21] B. Zhang, "A2j: Anchor-to-joint regression network for 3d articulated pose estimation from a single depth image." Git code https://github.com/zhangboshen/A2J.

[22] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg.