FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Implementation of smart part detection algorithms in OpenPNP open-source library
Student:	Bc. Nikola Karlíková
Supervisor:	Ing. Lukáš Brchl
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

The world of open-source manufacturing with DIY pick-and-place (PNP) machines is quite behind the latest trends of computer vision. In this field, the most popular tool is the OpenPNP library which is all-in-one software used to navigate PNP machines using cameras to accurately place parts on the PCBs. However, the current implementation of vision algorithms is insufficient for practical deployment. Therefore, the goal of the thesis is to implement innovative computer vision methods in this library to make it more robust for real-life scenarios.

1) Research existing methods of general parts detection in the manufacturing world and study the implementation in OpenPNP
2) Design a few computer vision algorithms that will be more robust than current stages in OpenPNP.
3) Implement the CV algorithms and accompanying methods for managing the CV pipelines/stages.
4) Create sample dataset for evaluation, compare your result to baseline algorithms and suggest further improvements.

Electronically approved by Ing. Karel Klouda, Ph.D. on 30 September 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Implementation of smart part detection algorithms in OpenPNP open-source library

Bc. Nikola Karlíková

Department of Artificial Inteligence
Supervisor: Ing. Lukáš Brchl

January 4, 2022

Acknowledgements

I would like to express my special thanks of gratitude to my supervisor Ing. Lukáš Brchl for providing guidance and support throughout this project even when things seemed uncertain. Thanks also to Ing. Tomáš Beneš for helping me with the data collection.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on January 4, 2022 . . . .

Citation of this thesis

Karlíková, Nikola. *Implementation of smart part detection algorithms in OpenPNP open-source library*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Počítačové vidění je důležitou součástí povrchové montáže plošných spojů, která zahrnuje i Pick and Place (P&P) operace, pro které je důležité mít spolehlivé počítačové vidění, které zvládne detekovat a zarovnat komponenty na správnou pozici, ve které jsou poté osazeny na desku plošného spoje. Na poli programů pro domácí osazování plošných spojů s open-source licencí je nejoblíbenějších programem OpenPNP. Jeho počítačové je implementováno jako sekvence OpenCV operací, která je aplikována na obrázek zachycující součástku. Toto řešení není spolehlivé a je pozadu s dnešními trendy detekce rotovaných objektů. Tato práce cílí na zlepšení procesu detekce součástek v programu OpenPNP pomocí implementace systému pro správu OpenCV sekvence operací a pomocí zavedení alternativní metody pro detekci, která by poskytla robustnější řešení vůči vnějším vlivům. Rešerše této práce byla soustředěna na dva aspekty; zkušenost s uživatelským prostředím pro detekci rotace OpenPNP programu a současné trendy v počítačovém vidění pro detekci objektů, který by mohly být použity pro P&P procesy. Na základě diskusí v rámci komunity uživatelů OpenPNP, jsem navrhla a implementovala řešení, které umožní uživatelům správu OpenCV operací, tím, že budou mít k dispozici znovu použitelné sekvence operací. V další části této práce jsem natrénovala čtyři detektory pro rotované objekty za použití vlastních dat. Nejlepších výsledků bylo docíleno za použití R^{3}Det detektoru. Výsledky detektoru jsem porovnala s výsledky ze současného řešení OpenPNP a dokázala jsem, že za použití strojového učení je možné dosáhnout řešení, které může sloužit jako alternativa pro současné OpenCV řešení. Toto řešení je navíc robustnější vůči vnějším vlivům a dokáže stabilně fungovat v reálném prostředí bez nutnosti časté uživatelské interakce.

Klíčová slova OpenPNP, Pick and place, plošné spoje, rotace součátky, detekce rotace, strojové účení

Abstract

Computer vision makes an important part in Surface Mount Technology processes including Pick and Place (P&P) operations. For P&P operation it is crucial to introduce strong computer vision solution to detect and align components to correct position before they are placed on the board. In the field of open-source software for DYI P&P machines the most popular tool is OpenPNP. Its computer vision takes form of a pipeline of OpenCV operations applied on the captured image of the component. This solution is not reliable and behind the latest trends of object detection. This work aims to improve the part detection process in OpenPNP tool by implementing better pipelines management system and introducing an alternative computer vision method that would be more robust to outer conditions. The prior research focused on two main aspects; user experience with OpenPNP object offset detection, and current trends in object detection that could be used for P&P detection purposes. Based on the OpenPNP community experience and discussions I designed and implemented a new solution that will allow users to manage OpenCV pipelines by making them reusable. In other part of the work, I trained four detectors for rotated objects and measure their performance upon custom dataset captured by P&P machine. The best detection results were achieved using R^{3}Det detector. I compared the detector results with current OpenCV solution and proved that solution using machine learning can be more robust alternative able to cope with real-time scenarios without the need of regular user input.

Keywords OpenPNP, Pick and Place, SMT, Component offset, Part offset, Part detection, Component detection, Rotation Detection, Machine learning,

Contents

Introduction **1**

Research background 1

Current problems 3

Thesis aims, objectives and significance 3

Known limitations 3

Structural outline 4

1 Parts detection in the manufacturing world **5**

1.1 Image pre-processing and feature extraction 8

1.2 Recognition (classification) methods 8

1.3 Object detection 9

1.4 OpenPNP 11

1.4.1 Using the OpenPNP software 12

1.4.2 Computer vision in OpenPNP 13

2 Analysis and design **15**

2.1 Part rotation detection 15

2.1.1 Rotation detectors available today 15

2.1.2 R^2CNN: rotational region CNN 17

2.1.3 RetinaNet 18

2.1.3.1 Focal loss for dense object detection 18

2.1.4 R^3Det: refined single-stage detector with feature refinement for rotating object [1] 20

2.1.5 Bounding box for rotated object detection via Kullback-Leibler divergence 22

2.1.6 SRCDet and SCRDet++ 22

2.1.7 Summary of the detections results 23

2.2 Impediments when using OpenPNP CVPipeline 23

2.3 OpenPNP improvements 27

2.3.1 Default CVPipeline improvement 27
2.3.2 Part detection management proposal 27
2.3.3 Part offset detection using machine learning 29

3 Implementation 33
3.1 Part detection management implementation 33
3.2 Offset detection using machine learning 36
3.3 OpenPNP migration 37
3.4 Data collection 38
3.5 Model setup 40

4 Model Evaluation 43
4.1 Dataset 43
4.2 Evaluation of the trained model 44
4.3 Detection results comparison 49
4.4 Discussion 51
4.4.1 Limitations and possible improvements 52

Conclusion 55
Future work 56
Summary 56

Bibliography 59

A Acronyms 63

B Contents of enclosed SD card 65

List of Figures

1.1 Example of a robotic arm (IRB 1400) [2] and Pick and Place machine for private use [3] . . . 5
1.2 Surface Mount Assembly line [4] . . . 6
1.3 Cognex's VisionPro software interface [5] . . . 7
1.4 The process of the PCB surface mounting showing solder paste printing and P&P operation [6] . . . 10
1.5 Definition of the printed solder paste offset [6] . . . 10
1.6 The process of upper pin group segmentation [7] . . . 11
1.7 OpenPNP User Interface . . . 12
1.8 OpenPNP definition of the offsets visualized . . . 13

2.1 The architecture of R^2CNN method . . . 17
2.2 Focal Loss function graph [8] . . . 19
2.3 The architecture of RetinaNet detection model . . . 20
2.4 The architecture of SCRDet detection model . . . 22
2.5 Results visualization on DOTa image. From the left there is result from RetinaNet-H, RetinaNet-R and R^3Det . . . 24
2.6 Example of a wrong detection using default CVPipeline in OpenPNP 25
2.7 Class diagram of the original OpenPNP computer vision solution . 26
2.8 Class diagram of the new proposed OpenPNP computer vision solution . . . 28
2.9 Visualization of the UI part of the proposed solution . . . 30
2.10 Structure of methods used in the RotationDetection project . . . 31

3.1 OpenPNP new UI Package panel . . . 34
3.2 OpenPNP UI Machine panel, Bottom Vision section . . . 35
3.3 OpenPNP coordination system visualisation . . . 39
3.4 GPU-Jupyter docker image Jupyterlab environment . . . 41

4.1 OpenPNP dataset objects representing categories . . . 44

4.2 Cross-entropy loss for all detectors, x-axis measures iteration, y-axis measures the loss value 45
4.3 RetinaNet-R loss graphs, x-axis measures iteration, y-axis measures the loss value 46
4.4 RetinaNet-R performance of validation data using thresholds 0.75, 0.8 and 0.95 46
4.5 R^3Det and R^3Det-KLD model total loss, x-axis measures iteration, y-axis measures the loss value 47
4.6 R^3Det and R^3Det-KLD performance of validation data using thresholds 0.75, 0.8 and 0.95 47
4.7 AP and F1 score measured for individual categories using all detection methods 49
4.8 Comparison of detection results from R^3Det detector and OpenPNP default pipeline on images with low lighting 50
4.9 Comparison of detection results from R^3Det detector and OpenPNP default pipeline on images with high lighting or visible nozzle . . . 51
4.10 Comparison of detection results from R^3Det detector and OpenPNP default pipeline on images with empty nozzle 52

List of Tables

4.1 Detection speed evaluation of trained detection methods on OpenPNP dataset. 47
4.2 Performance evaluation of trained detection methods on OpenPNP dataset. The shortnames for the categories are defined as: MC=Micro Crystal, Inf-Infineon, NS-Nordic Semiconductor, UB-U-blox, STM-STMicroelectronics 48

Introduction

Within the manufacturing world, the role of computer vision became essential. As physical dimensions of products, materials, and defects are decreasing, it is increasingly impossible for humans to fulfill high demands on the production precision and speed, and it makes the computer vision an important part in most of the manufacturing steps. One specific area of the manufacturing world is development of integrated circuits. Especially high demands are put on Surface Mount Technology (SMT) including Pick and Place processes (P&P), where object detection is the fundamental task requiring effective solution in terms of accuracy and speed. As the number of producers and commercial products grow, there is also increasing number of open-source P&P software. OpenPNP is one of the most popular open-source tools. Even though its popularity is huge, and the system is well designed, its computer vision still cannot compete with the commercial solutions present on the market.

This thesis aims to identify and evaluate existing solutions of object detection using computer vision that could be used in open-source OpenPNP tool to provide more precise and faster object offset detection. This chapter will provide and introduction to the thesis by providing the background and objectives of the research and proposed solution along with its benefits and limitations.

Research background

The focus of this thesis will be on the SMT P&P computer vision processes. SMT is a component assembly technology which mounts the electric components on the surface of a printed circuit board. The process of placing the component on the board consists of taking the part from a feeder using a suction nozzle and placing it on the board. To ensure the correct alignment of the mounted component several checks need to be performed including rotation of the nozzle with the picked part to the correct position using computer vision algorithm. To rotate the nozzle to the desired position the software first needs to identify the exact location of the component and calculate its offset represented by coordinates shift including a rotation angle. For that, the requirement is to have a computer vision algorithm that can detect the rotation of the object.

Current solution of OpenPNP software implements this rotation detection using OpenCV library. OpenCV is an open-source computer vision library that aims at real-time applications. OpenCV can provide strong object detection solution but its implementation in OpenPNP is not ideal. Based on the users' experience and demands the solution is not robust to outer conditions and requires regular adjustments. The current implementation of computer vision in OpenPNP using OpenCV library is therefore not sufficient and in many cases, it represents the weak spot of the software.

Computer vision has countless usages in the manufacturing processes from identifying defects and small parts to guiding the robot controllers in the environments. Even though every use case has its specific demands, there are practices that can be generalized. From the high-level overview, the computer vision tasks can be divided into two categories that are objects recognition and object detection. For P&P application the it is crucial to have a strong detection method. Most solutions are based on machine learning using Neural Networks (NN). Control systems using NN solution usually achieve high detection accuracy in real time object detection despite challenges like changing environment including occlusion, changing lighting, or camera pose.

Rotation detectors are currently a lot discussed field of study and many solutions were proposed in recent years. The pronounced challenges the studies are trying to solve are:

1. small objects,
2. densely arranged objects and
3. arbitrary object orientation.

Based on these challenges, detectors for small and rotated objects are a perfect fit for application in industry and thus for this thesis even though the manufacturing application is not the focus of the models' study.

Detection methods can be divided into two categories. First, two-staged detectors based on region features detection and classification based on fully connected layers. There exist state-of-art solutions based on two-stage detection model. They are usually based on R-CNN, Fast R-CNN, and Faster R-CNN and even though they are proved to perform well on challenging datasets they still have their speed limitations. On the other hand, recently developed single-stage detectors aiming to be fully convolutional proved to be fast in the object detection and performed competitive performance. Single-stage detectors were not yet able to achieve the same detection accuracy as state-of-art two-staged detectors, but they offer interesting trade-off between the detection performance and speed.

Current problems

Dynamic computer vision solution is critically important for the industry. Numerous studies have researched the field of computer vision in SMT and other applications with robotic elements. However, only a few studies focused on commercial or broadly used products and provided comparison of their results on real-life applications where the need for new practices is more demanding.

There are also only a few studies discussing last trends in computer vision and comparing their results in real-time industry environments. As a result, there is not compact comparison of recent hot topics in computer vision in manufacturing world providing a healthy competitiveness of open source and commercial products.

The implementation of computer vision in the OpenPNP software is behind the current trends. It brings many problems for practical usage and sets the software back from being compared to commercial tools.

Thesis aims, objectives and significance

In this thesis I will aim to research and evaluate the current field of part detection in the manufacturing world. I will also research current trends in rotation detection methods and evaluate possible benefits for this thesis. Based on the research results I will aim to resolve current OpenPNP impediments regarding computer vision by incorporating new computer vision solution for part offset detection. Another objective of the thesis will be to implement and evaluate the effectiveness of the new solution.

Questions to be answered in following chapters are what the current part detection methods in the industrial world are and what are the most recent trends in detecting rotated objects using machine learning. This thesis should also answer questions about current impediments in OpenPNP and propose and measure the improvements.

This work will contribute on the improvements of the widely used P&P open-source project by implementing new solution for component offset detection using last trends in machine learning for detection of rotated objects. This will help making the open-source SMT world more competitive to commercial tools. The results of the thesis should set a base for solving users demands and providing them with advanced software.

Known limitations

The focus of this thesis is narrowed only to the OpenPNP software and testing other implementations is not in the scope of this work. Therefore, the lack of comparison can be a big limitation. Another limitation that needs to be considered is the lack of previous studies trying to achieve similar results.

Except commercial software that are not publicly documented, there is no commonly known study trying to use recently researched machine learning solutions for rotation detection in SMT technologies and thus there are no practical usages that could be used as examples and to turn to.

Structural outline

In the first chapter, I will review the existing solutions regarding parts detection in the manufacturing world as well as describe the OpenPNP software and its computer vision for deriving the part location offset. In the second chapter, I will research rotation detectors available today. In another section of this chapter, I will analyze impediments when using current OpenPNP computer vision and provide design of a new solution in order to resolve users' problems. Following third chapter will describe implementation details of the solution proposed in the prior chapter. I will dedicate the fourth chapter for the new computer vision solution evaluation to measure its performance and state the results. In the last, fifth chapter, I will discuss the results of the new solution and propose future enhancements.

Chapter 1

Parts detection in the manufacturing world

Computer vision brings significant usage in automation of industrial processes. Those include for example burr measurement, surface inspection or grading and sorting products [9]. For Pick and Place automation operations in surface mount technology, the essential bottom vision techniques are detection and recognition of the objects to locate the workpieces. For such applications there are already off-the-shelf solutions available, but every machine has its specifications and limitations and not every solution can fit. Usually, it is also costly. Therefore, many manufacturers and developers of those systems are working on their own solutions.

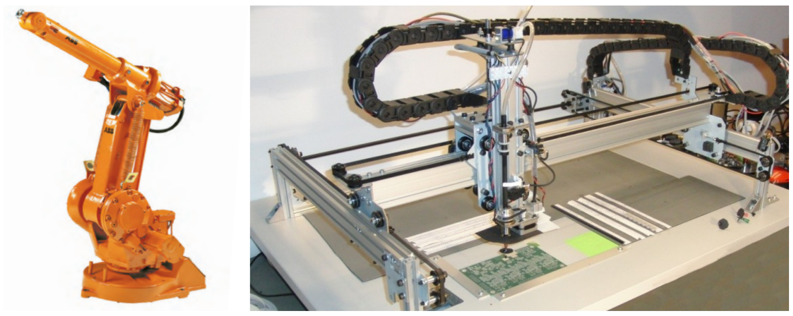


Figure 1.1: Example of a robotic arm (IRB 1400) [2] and Pick and Place machine for private use [3]

SMT is a method in which the components are mounted on the circuit board (PCB). Pick and Place operations (P&P) are one of the main processes the SMT comprises of. The electrical components are mounted directly on the solder paste that is placed on the PCB pads. During the P&P process it is usual that the components are not aligned and therefore there is a need of computer vision to automate the process of aligning the parts to their desired position [4]. The incorrect positioning can happen in multiple stages like picking, placing, or can be caused during board reflow. Figure 1.2 shows typical mount assembly process line.

There is not a lot of publicly available information about what computer vision techniques are used in real manufacturing world as the producers keep their processes hidden from the public. The only source for methods used in parts recognition are papers describing computer vision approaches used in pick and place machines or other application with robotic element. All applications I will mention here have been made to experimentally test the computer vision in real or simulated industrial environment. Even though there is a good number of papers and works available the majority is discussing solutions using pattern recognition and not the detection. Even fewer are discussing the rotation of the objects. Still, I will mention those techniques here as they are still relevant to the scope of this work.

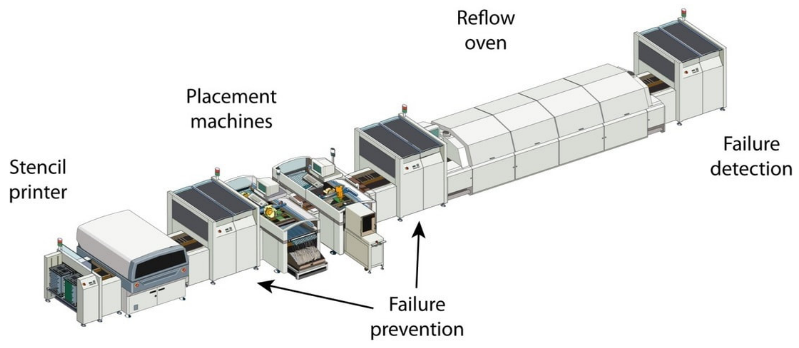


Figure 1.2: Surface Mount Assembly line [4]

The variety of object detection methods in the scope of manufacturing is huge and there is no common model that could be applied to multiple cases. Therefore, I aim to create an overview of different kinds of approaches used, and define what are the basics and widely used elements and methods.

When talking about the inspection techniques one of the simplest ones is CAD-based method providing inspections for a CAD models. However, this technique is not broadly used because it does not work for more complex parts especially when the model differs from the intended design but is still functionally acceptable. A widely used programming environment for the inspection is a LabView [10]. It is a platform that allows data acquisition, test automation, or embedded system design. It is usually combined with MATLAB to perform high level calculations upon the results from LabView to provide a classification.

From the commercially used vision software I can list Cognex's VisionPro [5] or MVTec Halcon [11]. Based on the specification by Cognex, the VisionPro does not require any image preprocessing. The system focuses on the critical acceptance relevant features ignoring any variations on the non-critical parts appearance. VisionPro software provides users with functions like object location, inspection, measurement, or alignment. Based on the specification the library operates on wide range of tools implementing methods like image filtering, OCR, patter matching and even smart and deep learning models. Halcon is a software for machine vision developed by Halcon. It supports multi-core platforms and GPU acceleration and serves various industries providing analysis, matching, measuring, object identification, 3D vision, and deep learning algorithms using the latest state-of-art computer vision technologies. Halcon is suited for embedded systems.

The typical workflow for the object recognition or detection can be broken down as follows. Firstly, image acquisition is performed to obtain an image of the detection object. The capture is then passed to preprocessing stage. Captured image is passed through various types of processes like filter banks or pooling operation to clean image data and provide the detector with normalized data to correctly generalize [12].

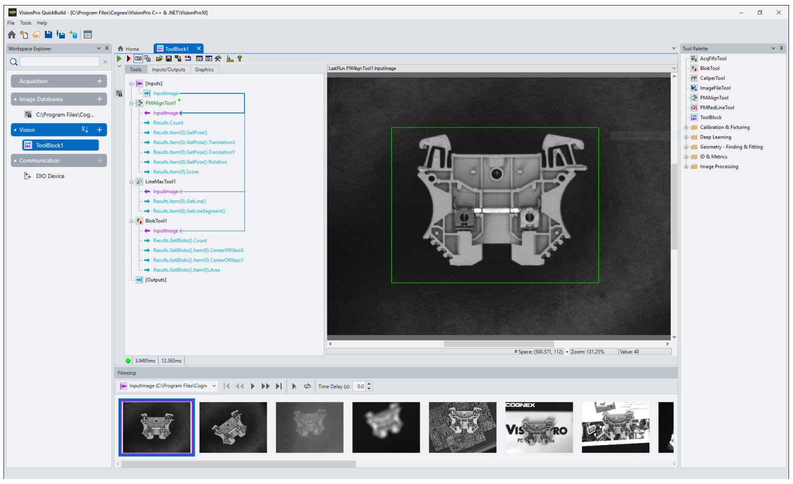


Figure 1.3: Cognex's VisionPro software interface [5]

Computer vision tasks require high quality input images to ensure the best results despite the challenges like occlusion, camera pose or unstable lightning condition. However, there are techniques to be addressed before applying image processing algorithms to improve the quality of the input image. Some engineers proposed a method using illumination system to increase the quality and illumination intensity of the capture acquired with a low-quality camera [13]. Bringing uniformity into illumination intensity distribution can reduce noise and shadows and thus can simplify image processing algorithms [14]. Almost every computer vision technique is at least adjusting the light system mainly to solve the uneven lightning around the vision system hardware as it makes the preprocessing easier. Another critical factor that influences the captured object is the color of the surface on which the object is captured. The color should be chosen as such the captured object is clearly distinguishable.

1.1 Image pre-processing and feature extraction

The next typical stage after the image acquisition is image preprocessing. The most common techniques use real-time computer vision functions as can be found in the OpenCV or MATLAB [13]. Commonly used techniques are masking, spatial filtering, color conversions or binary conversion [15].

Recently the most studied area as a step preceding the detection or classification is a feature extraction. It usually serves for removing unwanted data, providing new features to the training set for the classifier but can also be used for the detection itself. The feature extraction provides the features like centroid, shape, or color. The earlier feature extraction techniques focused on the visual features like edges or algebraic features. Nowadays, one reads mostly about features regarding the pattern withing the shape. The prior step is usually an image segmentation to separate the object in the image from the background to facilitate the process of feature extraction. The most robust detectors containing the feature extraction that are mostly discussed are Viola-Jones IP algorithm or Contour matching [16].

The feature extraction is usually based on simple thresholding algorithm. The usual methods are based on the edge detection as is a Canny method based on a Gaussian filter. Another step can be image dilatation and filling for achieving the object shape and drawing a bounding box around it. Thresholding can be also used for background removal performing the Otsu's thresholding algorithm [16] [15].

Feature extraction can be also understood as a basic image segmentation for which we can use a flood-fill algorithm [17]. Its purpose is to find areas of connected pixels of the same color. It is a simple iterative process, but its results cannot compete with the more sophisticated segmentation models based on CNNs.

1.2 Recognition (classification) methods

There are various approaches and models used in the manufacturing or industry-like environments. All the methods can be categorized as classification tasks and may serve not just for pure object recognition but also as an intermittent step in the object detection. Here, I am giving a general overview of the most popular methods currently available.

Most widely known and popular recognition techniques are statistical methods based on a distance measurement using Gaussian classifiers. They model many practical cases and are simple to implement. From this approach I can list a nearest-neighbor (K-NN) or minimum-distance rules [14]. K-NN method is used as a recognition tool in many different domains [18]. Given an input sample x and a corresponding training set the classifier assigns the sample to the class given by the closest k prototypes in the training set.

It is desirable to make the recognition system as autonomous as possible even for the statistical methods. For that, incremental learning algorithms were introduced. They can benefit from previously obtained data. It makes the recognition modules more robust and enables them to adapt to the dynamic environment. Incremental learning is based on improving the quality of training data eliminating mislabeled examples [19].

Neural networks give an alternative method for a statistical recognition in the context where the response time is as important as the recognition rate. A lot of solutions are talking about migration from statistical method to ANN [18]. From used models I can discuss solution done by TrainRp implemented in MATLAB.

From widely known image classification models, I will mention VGGNet lately outperformed by GoogleNet and ResNet using convolutional layers and max pooling technique. Even though single stage recognition systems do exist, most of the systems are employing two-stage process, Feature Extraction and Classification.

1.3 Object detection

During the object detection the machine vision solution needs to perform several steps including the classification. The high demand on the detection speed led to an introduction of a R-CNN and Faster R-CNN to replace a slower sliding window technique. Those techniques became a state-of-art solutions. Real time object detection became an area of an interest but there has always been a trade off between the detection accuracy and a speed, both crucial elements in robotics. Only Single Shot Multibox Detector (SSD) achieved a good balance between the speed and accuracy attaining 0.72 mAP at 58 fps and became another state-of-art detector [20] [21].

In [6] the authors are describing a use of computer vision in SMT for placement control when the object is placed on the wet solder paste. It can happen that the solder paste itself is printed with an offset as can be seen in the figure 1.5. It is a different perspective then the approach of this thesis, but the same operations can be applied, and it has the same importance. As the object is placed on the wet surface it can slump and its position might change before the board enters the heating stage. Authors designed an experiment considering variety of possible situations. The experiment used a Support Vector Regression machine learning model combined with other computer vision techniques. However, the component shifts were not addressed well.

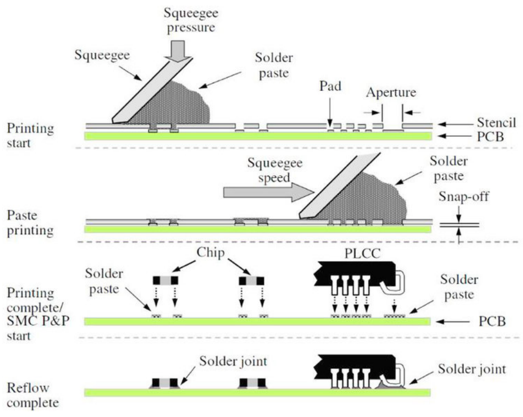


Figure 1.4: The process of the PCB surface mounting showing solder paste printing and P&P operation [6]

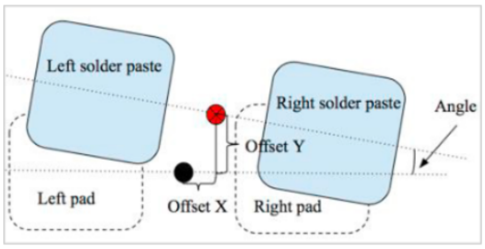


Figure 1.5: Definition of the printed solder paste offset [6]

In papers addressing the object rotation offset the vision processing is carried mostly by real-time computer vision best implemented in OpenCV. This approach gives developer a variety of different functions and integration with different operation systems and programming languages. Methods used are basically the same as described above in the image preprocessing section [22].

The [7] describes four types of chip detection methods: template matching, point set registration, pin line fitting and pin clustering. Template matching calculates the correlation between the target and the template [23]. Examples of such method are model-based matching or, for example, nearest neighbors. For identifying chips there are more suitable methods like line fitting and clustering. On the other hand, those methods are usually designed for specific chips and thus are not universal. In the paper, authors proposed a new method based on FAST model. FAST stands for Features from Accelerated Segment Test and it is a corner detection method. The newly proposed method uses an adaptive threshold that changes with the light. The rough position and rotation angle are established by minimum enclosing rectangle around all the FAST feature points. The executed experiments showed that the proposed method is promising even when compared with the commercial machine vision libraries. Figure 1.6 shows the process of component pin extraction using Otsu algorithm for preprocessing followed by segmentation as defined in the paper.

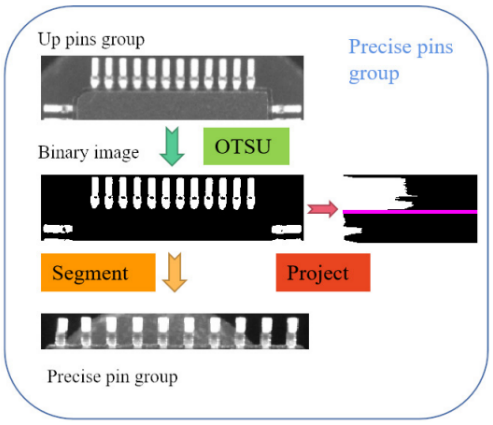


Figure 1.6: The process of upper pin group segmentation [7]

The neural network solution is recently being applied for most of the applications of the part detection and it is replacing the statistical methods [14]. The machine learning approach is almost in all cases preceded by object classification to determine whether the focused object is the object of interest. Few more recent papers are mentioning the use of single shot detectors (SSDs) that have recently brought a significant improvement in the performance and are currently used as an alternative to multistage detectors like Faster R-CNN.

1.4 OpenPNP

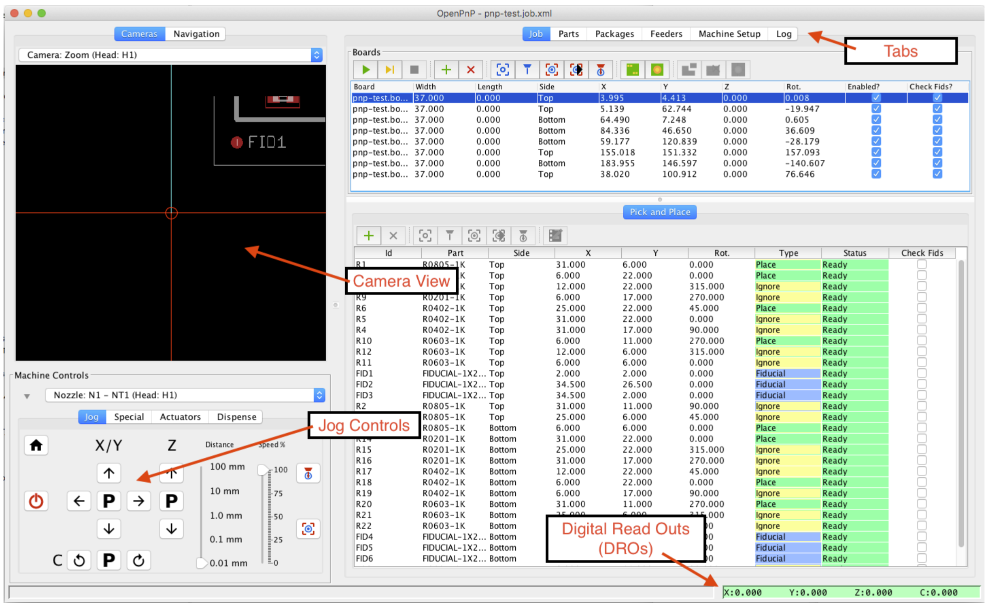
OpenPNP is an open-source software used for SMT pick and place machines. The official page provides ready to use software to run a pick and place machine of any kind whether it is a self-built one or a commercial one.

The repository with the code is provided on GitHub allowing users to build and modify the code freely. Contributors can make a fork and create pull requests whenever they want to contribute. Even though the code is under continuous development it is stable and heavily used. Over the ten years of its existence the repository has had over 50 contributors and has been forked for almost 400 times. Currently it is watched by nearly 150 users and has just about 1000 stars.

The project has built its own live community watching the GitHub project and having discussion groups, discord channel and twitter account. OpenPNP project does also provide users with help when building custom pick and place machines. [24]

1.4.1 Using the OpenPNP software

In the figure 1.7 one can see the user interface with the major components. The initial setup provides a demo configuration so user can try the application even without a machine connected. In this demo all machine parts needed for the interaction with the software as cameras, feeders, nozzles, and others are simulated.



Board	Width	Length	Side	X	Y	Z	Rot.	Enabled?	Check Fids?
pnp-test.bo...	37.000	0.000	Top	3.995	4.413	0.000	0.008	✓	✓
pnp-test.bo...	37.000	0.000	Top	5.139	62.744	0.000	-19.947	✓	✓
pnp-test.bo...	37.000	0.000	Bottom	64.490	7.248	0.000	0.605	✓	✓
pnp-test.bo...	37.000	0.000	Bottom	84.336	46.650	0.000	36.609	✓	✓
pnp-test.bo...	37.000	0.000	Bottom	59.177	120.839	0.000	-28.179	✓	✓
pnp-test.bo...	37.000	0.000	Top	155.018	151.332	0.000	157.093	✓	✓
pnp-test.bo...	37.000	0.000	Bottom	183.955	146.597	0.000	-140.607	✓	✓
pnp-test.bo...	37.000	0.000	Top	38.020	100.912	0.000	76.646	✓	✓

Id	Part	Side	X	Y	Rot.	Type	Status	Check Fids
R1	R0805-1K	Top	31.000	6.000	0.000	Place	Ready	☐
[illegible]	[illegible]	Top	6.000	22.000	0.000	Place	Ready	☐
[illegible]	[illegible]	Top	12.000	22.000	315.000	Ignore	Ready	☐
R9	R0201-1K	Top	6.000	17.000	270.000	Ignore	Ready	☐
R6	R0402-1K	Top	25.000	22.000	45.000	Place	Ready	☐
R5	R0402-1K	Top	31.000	22.000	0.000	Ignore	Ready	☐
R4	R0402-1K	Top	31.000	17.000	90.000	Ignore	Ready	☐
R10	R0603-1K	Top	6.000	11.000	270.000	Place	Ready	☐
R12	R0603-1K	Top	12.000	6.000	315.000	Ignore	Ready	☐
R11	R0603-1K	Top	6.000	6.000	0.000	Ignore	Ready	☐
FID1	FIDUCIAL-1X2...	Top	2.000	2.000	0.000	Fiducial	Ready	☐
FID2	FIDUCIAL-1X2...	Top	34.500	26.500	0.000	Fiducial	Ready	☐
FID3	FIDUCIAL-1X2...	Top	34.500	2.000	0.000	Fiducial	Ready	☐
R2	R0805-1K	Top	31.000	11.000	90.000	Ignore	Ready	☐
[illegible]	R0805-1K	Top	25.000	6.000	45.000	Ignore	Ready	☐
[illegible]	R0805-1K	Bottom	6.000	6.000	0.000	Place	Ready	☐
R14	R0201-1K	Bottom	31.000	22.000	0.000	Place	Ready	☐
R15	R0201-1K	Bottom	25.000	22.000	315.000	Ignore	Ready	☐
R16	R0201-1K	Bottom	31.000	17.000	270.000	Ignore	Ready	☐
R17	R0402-1K	Bottom	12.000	22.000	45.000	Ignore	Ready	☐
R18	R0402-1K	Bottom	6.000	22.000	0.000	Place	Ready	☐
R19	R0402-1K	Bottom	6.000	17.000	90.000	Ignore	Ready	☐
R20	R0603-1K	Bottom	31.000	11.000	270.000	Place	Ready	☐
R21	R0603-1K	Bottom	25.000	6.000	315.000	Ignore	Ready	☐
R22	R0603-1K	Bottom	[illegible]	[illegible]	[illegible]	Ignore	Ready	☐
FID4	FIDUCIAL-1X2...	Bottom	[illegible]	[illegible]	[illegible]	Fiducial	Ready	☐
FID5	FIDUCIAL-1X2...	Bottom	[illegible]	[illegible]	[illegible]	Fiducial	Ready	☐
FID6	FIDUCIAL-1X2...	Bottom	[illegible]	[illegible]	[illegible]	Fiducial	Ready	☐

X:0.000 Y:0.000 Z:0.000 C:0.000

Figure 1.7: OpenPNP User Interface

OpenPNP allows to setup the machine in Machine Setup tab. Here, user can configure all the aspects the machine consists of. This is the place where we want to start when using the software with our custom machine. The structure has a tree shape with root representing the machine itself. Every setup is mirrored in xml configurations that serves as an initial setup and place to persist the settings.

In its base the software is a Computer Numerical Control (CNC) machine controller. The main section in the program is the Jobs tabs. There, user can define a set of jobs the machine is supposed to execute, and the software sends the corresponding commands to the hardware. When defining a job user needs to configure the location of all the aspects needed for the execution which means defining location of boards, parts, packages, feeders, and corresponding final placements.

For this thesis, section dedicated for parts is the most relevant. From OpenPNP perspective, part is a record referring to its origin and final placement, also containing an information about its physical properties to help to place the part accurately to a desired place. Every part belongs under a specific package. Package closer specifies the part's physical properties. It provides OpenPNP with information about the appearance including shape, count of the pads or pins, and the size.

1.4.2 Computer vision in OpenPNP

Computer Vision in OpenPNP is managed in module called Bottom Vision [25]. It helps to place the components more accurately by identifying center point offset or rotation error and correcting the position of the part on the nozzle. It can also determine if the pick was successful. Main elements of the Bottom Vision procedure are up-looking camera along with CVPipeline. The whole operation consists of:

1. picking the part from a feeder,
2. centering the nozzle over the up-looking camera and take an image and
3. applying CVPipeline to determine the errors.

CVPipeline stands for Computer Vision pipeline and consists of stages represented by *CVStage* class. Each stage represents an OpenCV operation or another procedure to help with the image processing and debugging such as image capturing and saving. OpenPNP implements editor for user to be able to specify concrete stages and their order of the execution along with individual stage parameters. CVPipeline, when executed, provides image capture and part detection, and computes the errors. During the execution, original image is updated after each stage is applied providing user option to debug the whole CVPipeline.

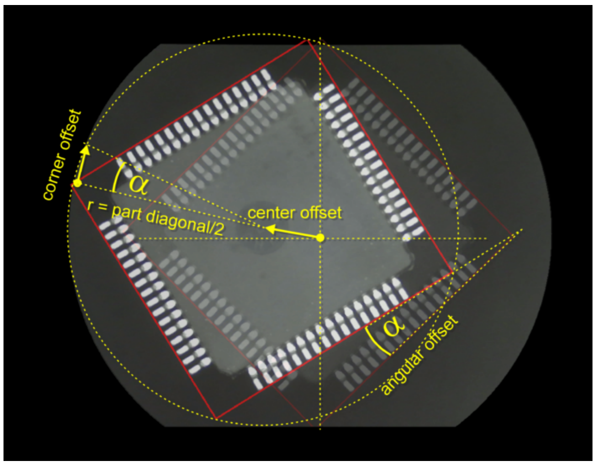


Figure 1.8: OpenPNP definition of the offsets visualized

The output of the CVPipeline is rotated rectangle that represents a minimal bounding box of the picked up and captured component. The rectangle is represented by width, height, center coordinates and rotation. Using this we calculate the error in the offset and in the rotation.

Bottom Vision configuration offers to apply rotation action prior to the vision. That means that the nozzle will pre-rotate to its final placement position. Enabling this pre-rotation allows user to apply also multipass vision. During the multipass, part is centered based on the calculations derived from the error multiple times. The aim is to cancel any potential errors caused by wrong camera caption. These errors have various causes. Part on the nozzle can be seen from a slight angle, a section of the part can be out of focus or there may be reflections due to angled surfaces.

OpenPNP Bottom Vision also provides user with part size check. Based on the selected method and defined tolerance, the part alignment can fail if the component size differs. CVPipeline can check for body size or for whole footprint including pads.

Chapter 2

Analysis and design

In this chapter I will present current trends in machine learning techniques for detecting rotated objects. In other section I will analyze impediments when using current computer vision solution in OpenPNP tool. Based on previous research and analysis I will propose a design for new functionality for OpenPNP tool to solve current problems and to introduce new updated solution.

2.1 Part rotation detection

Main applications of the rotating image detectors are scene texts and retail scenes. Scene text is text that appears in the pictures from outdoor scene. It is difficult to locate a multiangled object and distinguish it from the background. Existing detection methods work with objects as they would be located along the horizontal line. Those are known detectors like Fast R-CNN, Faster R-CNN, SSD or YOLO. However, the problem of arbitrary rotated objects was not addressed by the above detectors. Due to the value the detectors for rotated objects would bring, there are new detection methods currently developed. To achieve better accuracy, many of the detectors are introducing refinements directly to the existing solutions.

2.1.1 Rotation detectors available today

Rotation detection is a challenging task and only few solutions exist and are well documented. Next to the large aspect ratio and densely arranged objects, arbitrary orientation of objects is one of the challenges pronounced today and therefore new rotation detectors have been proposed. Methods using Deep Neural Networks can be generally grouped into two categories [26].

The first category are two-staged detectors. They usually calculate the sparse set of feature proposals that should contain all the objects in the first stage, and then apply the classifiers or regressors in the second stage. The region proposal methods are usually models like R-CNN, SPPnet, Fast or Faster R-CNN and R-FCN. The outputs are then post-processed by methods such as non-maximum suppression (NMS) to get the detection results. Due to their robustness and ability to classify objects in dense scenes, two-staged detectors are still dominant. However, when we look at the performance and accuracy of the detection of rotated objects there is still a space for improvement. ICN, SCRDet or Gliding Vertex are examples of state-of-art-detectors. The main drawback of the two-staged detectors is the complexity of their structure that is causing the speed bottleneck. Recently the studies and works showed that the two-staged detectors can overcome this bottleneck by reducing input image resolution and by number of other proposals. Above mentioned rotation detectors are based on horizontal detectors such as Faster R-CNN, Region-based Fully Convolutional Networks (R-FCN) or Feature Pyramid Network (FPN). Those advanced techniques were introduced to improve the speed of sliding window approach. These two-staged frameworks are consistently achieving top results on the COCO benchmark [27] considered as one the most challenging and used for comparing the performance of state-of-art algorithms. Neural Networks have shown that they can achieve high detection accuracies in real time detection, up to 0.73 mAP but there has always been a tradeoff between accuracy and speed. To achieve better accuracy, many detectors are using cascaded model like The Cascade RCNN, HTC or FSCascade performing multiple concatenated classifications and regressions in the second stage.

Second method, on the other hand, does not rely on region proposals. Detectors using this method are single shot detectors based on convolutional neural network which makes them much more attractive due to their effectiveness. This family of detectors directly estimate object candidate same as SSD or YOLO. This method saves detection speed without losing to much accuracy. Single-stage detectors are based on and introduce refinements to models like RetinaNet or RefineDet. RetinaNet is especially popular due to its ability to detect small scale objects in a dense scene that are typical for aerial and satellite images. The performance of single-shot detectors is mainly limited by features misalignment. Maintaining fully convolutional structure is the important requirement so the feature misalignment cannot be fixed by adding refinements from two-staged detectors as the speed advantage would be lost.

Even though two-stage detectors are still achieving better results, single-stage detectors maintain faster detection speed. From existing and most documented detectors for rotated objects, I can list R^{2}CNN, RetinaNet-H and RetinaNet-R, FCOS or R^{3}Det. In the sections bellow I will give a brief description of a few detectors including state-of-art method.

2.1.2 R^{2}CNN: rotational region CNN

This model was first proposed in 2017 as a novel method for arbitrary-oriented text detection. While traditional text detection methods are based on sliding window or Connected Components (CC), R^{2}CNN proposes deep learning base method as it has been recently widely studied.

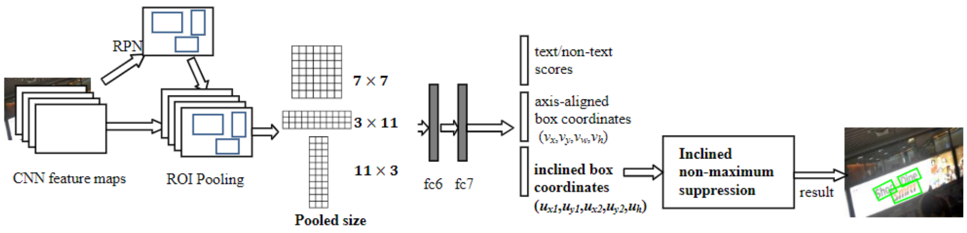

RPN
CNN feature maps
ROI Pooling
7 × 7
3 × 11
11 × 3
Pooled size
fc6
fc7
text/non-text scores
axis-aligned box coordinates (v_x, v_y, v_w, v_h)
inclined box coordinates $(u_{x1}, u_{y1}, u_{x2}, u_{y2}, u_h)$
Inclined non-maximum suppression
result

Figure 2.1: The architecture of R^{2}CNN method

Rotational Regional CNN is based on Faster R-CNN architecture. The procedure first uses Region Proposal Network (RPN) to propose axis-aligned bounding boxes around the texts. The next stage is classification and refinement of the boxes to predict the minimum area boxes using the Fast R-CNN. Last process is a post-process using non-maximum suppression to get results [28]. Figure 2.1 shows the architecture.

The approximation of the detected arbitrary-oriented scene text is done by inclined minimum area rectangle. Instead of using angle to represent the rotation, the model uses coordinates of the first two points and the height of the rectangle (x_1, x_2, y_1, y_2, h). The first and second points always mean the left-top and the right-top corners. To increase the performance the method adds additional axis aligned bounding boxes to each predicted inclined box.

The detection strategy is based on two stages. In the first stage, the RPN is first used to generate region proposals. Proposals are axis-aligned bounding boxes enclosing the arbitrary oriented texts. It is also called cascade stage and reduces the set of all possible objects from almost infinite to thousands by eliminating the easy negatives. The ROIPooling is then performed upon the proposals to generate pool features for further classification and regression. The Faster R-CNN uses ROIPooling with pooled size 7x7 pixels. R^{2}CNN adds additional two pool sizes 11x3 pixels to catch more horizontal features and 3x11 pixels to catch more vertical features. The pooled features and fully connected layers are then used to generate scores, axis-aligned boxes, and inclined minimum area boxes.

Non-maximum suppression (NMS) is used to post-process the detection candidates. NMS uses Intersection-over-Union (IoU) and modifies it to calculate the IoU between two inclined bounding boxes. The second NMS is applied to axis-aligned bounding boxes respectively. The loss function is the summation of the text/non-text classification loss and the box regression loss.

The regression loss consists of the loss of axis-aligned boxes and the loss of inclined minimum area boxes.

The model was trained on images from ICDAR 2015 dataset and custom images containing incidental scene texts with arbitrary orientation. Based on the results from the experiment the R^2CNN reached recall of 79.68%, precision of 85% and F-measure of 82.54%. Compared to Faster R-CNN the proposed method outperformed the detector by 6% which was mainly caused by Faster R-CNN ignoring the rotation of the objects. In overall, the evaluation shows that the model reached competitive results on the selected dataset. The method is also general and could be applied to other general detection frameworks like YOLO or SSD.

2.1.3 RetinaNet

RetinaNet is one of the best single-shot detectors. It consists of the backbone network, and classification and bounding box regression subnetworks [29]. The backbone network is an off-the-self convolutional network responsible for extracting a feature map from the entire input image. As its base settings it uses tuple representing an arbitrary oriented rectangle [30]. It consists of five parameters (x, y, w, h, θ) representing center of the object, its width and height and rotation angle ranging in $[-90, 0)$ degree range. Theta denotes the acute angle between the box and the x-axis. This definition is consistent with OpenCV structures.

2.1.3.1 Focal loss for dense object detection

When we speak about RetinaNet we cannot but mention the Focal Loss as the base that makes RetinaNet such a strong detector. The imbalance between foreground and background class is one of the main reasons why single-stage detectors do not reach the accuracy of two-staged methods. Focal Loss (FL) can, as a novel loss function, remedy this issue and make the single-shot detectors more powerful. When the RetinaNet was firstly introduced, it outperformed every other model on the market mainly thanks to the Focal Loss [31].

In the case of class imbalance there is a surplus of easy negatives that tend to overwhelm the foreground class which can lead to generalization of the model. The class imbalance also leads to a training inefficiency as the high number of easy negatives leads to a lot of useless signals. The general solution to overcome this impediment is to perform hard negatives mining. The FL stands as a contrast to these methods showing that it can naturally handle the class imbalance for one-stage detectors.

The function is described in the figure 2.2. It is a dynamically scaled cross-entropy loss. When the confidence in the correct class increases the scaling factor gamma decays to zero. This scaling factor can make the model focus

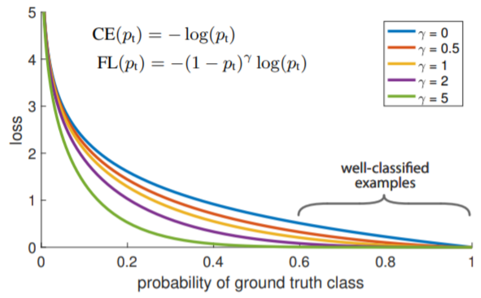


Figure 2.2: Focal Loss function graph [8]

on the hard examples by downweighing the easy examples contribution and is commonly used to address the class imbalances [8]. For the notation we classify:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise.} \end{cases} \tag{2.1}$$

It can be observed that even for well-classified examples ($p_t \gg 0.5$) the loss in non-trivial.

The gamma ($\gamma \;\geq 0$) plays role in the so-called modulation factor $1 - p_t)^\gamma$ that is added to a cross-entropy loss. The Focal Loss is then defined as followed:

$$FL(p_t) = -(1 - p_t)^\gamma log(p_t). \tag{2.2}$$

When the p_t is small, the modulation factor is close to one and the loss is unchanged. In the opposite case when the p_t goes to 1 the factor goes to zero and the loss is downweighted. It means the loss contribution of easy examples Is reduced. The gamma parameter has the power to adjust the rate at which the easy examples are reduced. Clearly, the case of $\gamma = 0$ corresponds to a cross-entropy. From the graph in the figure 2.2, it can be observed that the Focal Loss function also extends the range where examples receive a low loss and thus put more importance on the misclassified examples.

In practice, there is another α-balanced variant:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma log(p_t). \tag{2.3}$$

This form comes out of the balanced cross-entropy loss. α is a weighting factor that takes value of $[0, 1]$ for class 1 and $1 - \alpha$ for class -1. It may be derived either with inverted class frequency or as a cross-validation hyperparameter. The precise form is not crucial is the authors of the paper proved during the experiments.

To improve the training stability in case of heavy disbalanced classes, authors introduced the concept of prior pi. The value p for the rare cases is at the beginning estimated to π to overcome the instability in the phase of early training.

The Focal Loss is used as the loss on the output of the classification subnet. Experiments showed that in practice, $\gamma = 2$ works well and the model is robust when the γ is in $[0.5, 5]$. In general, the α should be decreased slightly when γ increases and for the $\gamma = 2$ the best value of α was observed to be 0.25.

2.1.4 R^{3}Det: refined single-stage detector with feature refinement for rotating object [1]

The detector aims to solve current problems with existing single-stage detectors and provide accurate and fast solution for rotated objects detection. Namely the problems analyzed are large aspect ratio, densely arranged and arbitrary oriented objects. The main design improvements proposed are progressive regression using combination of rotated and horizontal anchors and feature refinement module to solve feature misalignment problem. The base of the detector is a Refined Rotation RetinaNet (R^{3}Det) as can be seen in the figure 2.3. At the beginning there is a bottom-up and top-down pathway to build the feature pyramid network (FPN). The refinement is added to the classification and regression subnetwork in the form of refinement stage for bounding box and feature refinement module (FRM) to reconstruct the feature map. It can be repeated multiple times.

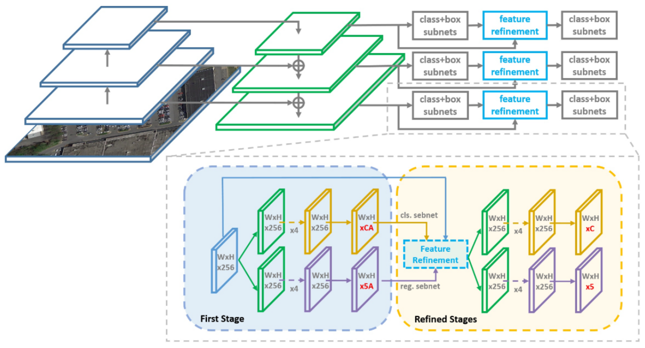


Figure 2.3: The architecture of RetinaNet detection model

As a loss function the SkewIoU is used as it can differentiate box sets with different aspect ratio. Commonly used smooth L1 loss value does not consider aspect ratio in its calculations, only the angle and size of the objects. IoU in general is an effective regression loss function and is widely used but it is not derivable which makes it unusable for direct usage. In the refinement for rotating objects the derivable approximate SkewIoU loss function is used and defined as follows:

$$L = \frac{\lambda_1}{N} \sum_{n=1}^{N} obj_n \frac{L_{reg}(v'_n, v_n)}{|L_{reg}(v'_n, v_n)|} |f(SkewIoU)| + \frac{\lambda_2}{N} \sum_{n=1}^{N} L_{cls}(p_n, t_n). \tag{2.4}$$

$$L_{reg}(v', v) = L_{smooth-l1}(v'_\theta, v_\theta) - IoU(v'_{\{x,y,w,h\}}, v_{\{x,y,w,h\}}). \tag{2.5}$$

'N indicates the number of anchors, obj_n is a binary value. v_0 represents the predicted offset vectors, v denotes the targets vector of ground-truth. While t_n indicates the label of object, p_n is the probability distribution of various classes calculated by sigmoid function. SkewIoU denotes the overlap of the prediction box and ground-truth box. The hyper-parameters λ_1, λ_2 control the trade-off and are set to 1 by default. The classification loss $L_c ls$ is implemented by focal loss.' [1]

In addition, SkewIoU loss is sensitive to slight changes in rotation therefore authors introduced the refinement of the prediction boxes using various IoU thresholds to improve the recall rate. Multiple refined stages are joined together using different combination of foreground and background thresholds. The overall loss is then defined as a sum of the refinement stages loss values.

Because the detector was designed to identify the rotated objects in moving images the FRM came in place to pick up any misalignments caused by changing location of the bounding boxes. This improvement does not consider me as I will be using the picture from a static camera. However, it is still worth short mention as it improves the accuracy of the whole model. The FRM serves as an addition to refined bounding box re-encoding its position to corresponding feature point causing reconstruction of the whole feature map to achieve the correct alignment. ROIAlign is the equivalent solution for solving feature misalignment in two-stages detectors as mentioned in the previous section. Compared to ROIAlign, FRM has about 25% less sampling points which gives it a speed advantage. In addition, FRM maintains a full convolution structure that compared to fully connected structure in ROIAlign leads to a higher efficiency and fewer parameters.

During the experiments there were various datasets and protocols used to measure the results and choose the combination with the best performance. As one of the datasets, authors used public DOTa dataset comprising of over 2000 aerial annotated images. As next datasets used there was HRSC2016, ICDAR2015 and UCAS-AOD datasets.

Three baseline methods were used for training. RetinaNet-H using horizontal anchor boxes made the model less accurate. RetinaNet-R baseline with rotated anchor boxes on the other hand performed better in dense scenes but was less efficient because it adds angle parameters and thus multiplying number of anchors. R^{3}Det refined detector without the FRM combines the best of two previous baselines. First, it uses horizontal anchors and then applies the rotating refined anchors to overcome the problem that dense scenes cause. Performance of the R^{3}Det baseline was measured to 63.52%. Adding two refined stages, the FRM and approximate SkewIoU loss improved the performance and shot the percentage up to 73.79%. Results showed that the new method achieved the best performance among all the detectors. In overall, the experiments and studies showed that the results achieved state-of-art models‘ accuracy and efficiency.

2.1.5 Bounding box for rotated object detection via Kullback-Leibler divergence

a regression framework using Kullback-Leibler Divergence (KLD) was proposed in [32] to overcome limitation of regression loss on dense scenes and objects with large aspect ratio. The paper is based on the idea that different parameters are important for different types of objects like angle parameter and center point and thus the regression loss should be self-modulated, and the learning process could use more dynamic optimization. First the rotated bounding boxes are converted into Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ and as a standard distance metric the KLD is used to calculate the distance between the predicted and ground-truth box.

The KLD method is compared to L1 Smooth Loss and Gaussian Wasserstein Distance (GWS). The study and experiments found that KLD has more complex parameter optimization and the gradient of the parameters can be dynamically adjusted based on the object of interest. Additionally, it is scale invariant and can be degenerated into the l_n-norm loss.

2.1.6 SRCDet and SCRDet++

Same as the R^{3}Det model described above this solution is trying to offer a model that would perform well on scene images with small, cluttered, and arbitrarily oriented objects. It tries to minimize the interference on both background and foreground objects by introducing a denoising module.

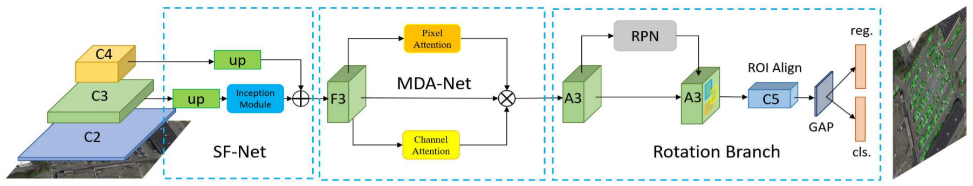


Figure 2.4: The architecture of SCRDet detection model

The proposed architecture is a two-stage model based on RPN network. In the first stage authors added SF-Net and MDA-Net to reduce noise and increase feature information amount in the feature map. This stage is still using regression for horizontal boxes. The second stage on the other hand uses improved regression with five parameters and rotation non-maximum suppression operation for each proposal. The whole method is described in the figure 2.4.

Because using the pooling layers can lead to a loss of small object feature information, the model introduces a feature fusion such as Top-Down Modulation (TDM) o FPN. Based on the deeper analysis of the feature fusion and finer sampling method authors designed the finer sampling and feature fusion network (SF-Net) [33].

Next the Multi-Dimensional Attention Network (MDA-Net) was designed to enhance the object's features and weaken the non-object information. Compared to other existing attention methods this one is supervised and thus able to learn specific purposes.

The model uses the same parameters to represent ground-truth boxes as OpenCV and thus the previous above-described detector (R^{3}Det). Also, the loss function is based on the same method, namely SkewIoU [34]. Based on SkewIoU computation there is also post-processing operation using rotation non-maximum suppression (R-NMS).

2.1.7 Summary of the detections results

All above mentioned detectors and corresponding papers show good results on various datasets. The most interesting results are achieved by R^{3}Det detector on DOTa dataset. This dataset contains aerial images and thus providing samples with small, rotated, and cluttered objects. The paper provides results comparison between R^{3}Det detector and SCRDet and RetinaNet detectors. Among the single-stage detectors, R^{3}Det achieved the best detection results around 89%. RetinaNet-R outperformed R^{3}Det detector only one on HRSC2016 dataset containing aerial images of ships. Even though two-stage detectors are still dominant, the R^{3}Det detector achieved the best and promising performance on DOTa dataset. Figure 2.5 shows the detection compared on image from DOTa dataset.

The results indicate that above mentioned detectors could be a good fit for purposes of this work and for SMT processes in general.

2.2 Impediments when using OpenPNP CVPipeline

OpenPNP default configuration offers a default OpenCV pipeline. The default pipeline described below gives a nice overview of possible pipeline configuration. The OpenPNP implementation contains more stages than described here. There are over 60 stages implemented using around 26 OpenCV opera-

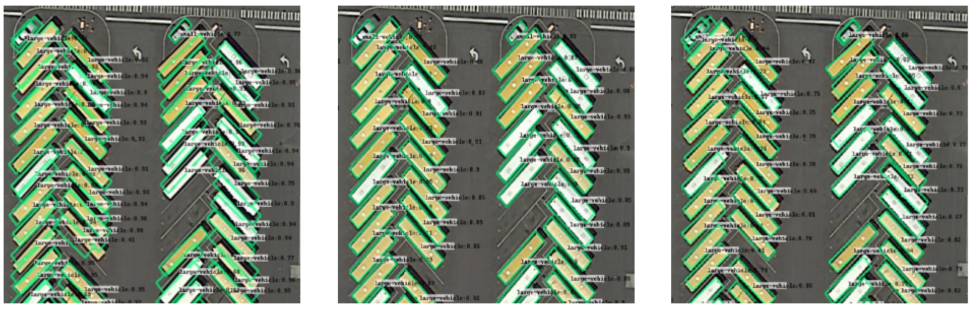


Figure 2.5: Results visualization on DOTa image. From the left there is result from RetinaNet-H, RetinaNet-R and R^{3}Det

tions. Considering how big library the OpenCV is, the number of operations implemented is quite small.

The default pipeline consists of stages listed below.

1. Capture image.
2. Gaussian blur: Performs minor blurring to reduce noise in the image.
3. Circle mask: Blacks out everything outside of a circle of a given diameter.
4. Convert RGB color to HSV.
5. HSV Mask: Searches the image for any pixels that match a certain hue and turns them black to remove green pixels from the image. Green is the color of the nozzle holder.
6. Convert color back from HSV to RGB.
7. Convert color from RGB to grayscale.
8. Threshold: Turns the image into a binary image.
9. Find connected contours in the image.
10. Remove any contours from the previous stage that are smaller than a specified value.
11. Draw all the of the remaining contours in white on the black background.
12. Use Minimal Area Rectangle to create a Rotated Rectangle that fits around any non-black pixels in the image.
13. Draw the Rotated Rectangle in red overtop the original image.
14. Write the resulting image out to a file.

The default CVPipeline is not a reliable solution. It introduces a lot of mistakes during the part segmentation and users of the program usually need to adjust the pipeline for separate parts or packages. To support this statement there exist several proposals and discussions upon the usage of the CVPipeline and users are repeatedly confirming that part detection using CVPipeline is at least unintuitive, and it takes time to become acquainted to setup the pipeline correctly considering also other setup like lightning or lenses. Next, a segmentation error can occur when the part is not picked up correctly, and part of the nozzle remains visible in the capture. The nozzle is not distinguished from the part and the rotated rectangle then covers bigger area then it should. This mistake can be seen in the figure 2.6.

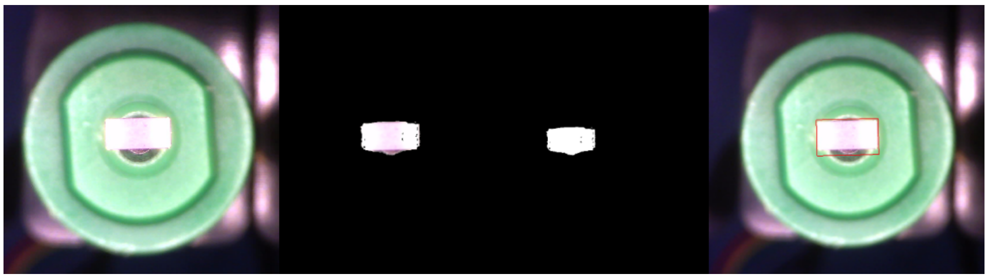


Figure 2.6: Example of a wrong detection using default CVPipeline in OpenPNP

For nice captures that can be processed correctly it is important to set up the light and camera exposure correctly. When the outside condition changes during different parts of the day it requires making lightning changes quite often. Slight change in the aperture of the image can mean errors during the rotation detection. It means that the CVPipeline is not a robust against the changes in the lightning.

Another obstacle can be the pipeline management. User can define different pipelines for different parts and he or she can also copy and paste the pipeline as a whole. But there is no option how to save the defined pipeline under unique and distinguishable name for quick access and assignment to individual part. In the background there is a container mapping the part to an individual pipeline, however, it is not visible to the user and it is not maintainable. This container can be seen on class diagram in the figure 2.7. The structure holding the pipelines is represented as a map of *PartSettings* objects living only in a *ReferenceBottomVision* class. This class represents Machine Bottom Vision settings.

Many parts that belong to the same package usually have very similar footprint and thus it would be convenient to have the option to assign or at least define pipeline for the whole package. This option is completely missing, and package stands almost completely apart from the logic described in the diagram. It must be said that there exists a global object called Configuration holding all the instances of model classes, but it is not reachable from the UI.

There have been some proposals and CVPipeline improvements from both the authors and users. One of them is a CVStage for detecting Circular Symmetry [35]. This stage can be used to detect objects like fiducials, nozzle tips and others that take form of a concentric circles. It could be used for detections of an empty nozzle and thus the failure to pick up a part. Unfortunately, current CVPipeline management do not allow any conditional behavior so there is currently no possibility to first run automatic empty nozzle detection and then, in case of failure, detection of the part. This stage is currently used purely for navigating the machine head using fiducial markers.

Another step was made by introducing stage for detecting rectangles. It was based on the idea that most of the parts are rectangular. The stage used Hough Transform with some further processing. However, this stage was not introduced to the default pipeline nor brought any significant improvement.

Unfortunately, none of the existing nor proposed solution is in form of out-of-box technique that would fit many cases. The CVPipeline still requires thorough pre and post processing. Generally, world of image detection advanced and nowadays there exist faster and more accurate solutions. OpenPNP is a powerful solution for home pick and place machines and having more advanced and powerful computer vision algorithm could make it even bigger player on the field on SMT machines.

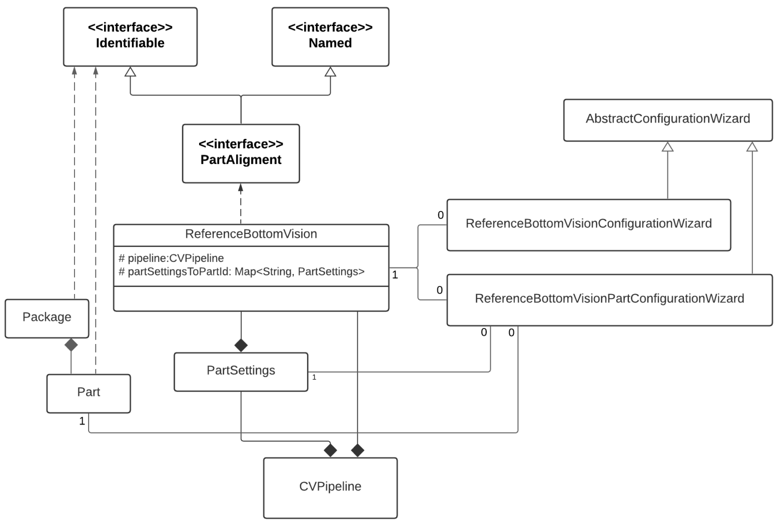

<<interface>> Identifiable
<<interface>> Named
<<interface>> PartAligment
AbstractConfigurationWizard
ReferenceBottomVision
# pipeline:CVPipeline
# partSettingsToPartId: Map<String, PartSettings>
ReferenceBottomVisionConfigurationWizard
ReferenceBottomVisionPartConfigurationWizard
Package
Part
PartSettings
CVPipeline
0
0
1
0
0
1
1

Figure 2.7: Class diagram of the original OpenPNP computer vision solution

2.3 OpenPNP improvements

As already mentioned, the current part detection solution is not strong and robust. In this section, I will propose and design a new solution, by which I will try to improve the process of part offset detection. The proposal does not contain only improvements in the accuracy of the part detection but also in the manageability and maintainability of the whole process. OpenPNP software is implemented using Java Swing that is used to create a window-based application, so for the analysis purposes it was easy to navigate through the code.

2.3.1 Default CVPipeline improvement

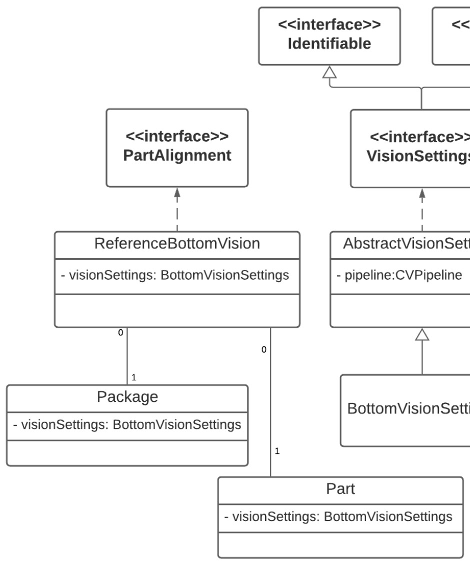
As it was already said and shown in the previous section, the default pipeline doesn't provide users with functional solution and the CVPipeline usage in general can be a tedious process. During the analysis I considered an option to propose and build functional pipelines for separate packages that could be used as a reliable out-of-box solution. During the research of today's object detection options, I decided to withdraw this solution as it is outdated and I would not be able to ensure a high accuracy and ease usage. Even though there are solutions to make the process using OpenCV operations more robust and dynamic by automation, the OpenCV solution is still not that robust to handle wide range of outer conditions and provide universal solution.

2.3.2 Part detection management proposal

To ease the usage of CVPipelines I will propose a solution, how to make them more maintainable and manageable by introducing a logic for reusing them. From the beginning, I intended to get a new patch with those changes to the repository because it would solve a lot of users' complaints and, I believe, would be a beneficial contribution.

The standard way how to contribute to the OpenPNP repository is described directly in wiki section of the Github repository. First, a discussion must be created to get other users' support on the vision solution and their insights. It is also the best way how to connect with the project's maintainers and get their approval. After a successful discussion, a Pull Request in the Github repository with the proposed changes can be created. Other users are then given a space to comment, review, approve or reject the code. The code needs to be approved mainly by maintainers of the repository. After that it goes to testing branch so users can thoroughly test the solution on real machines.

There already was a recent discussion talking about the part of the code related to computer vision on up-looking camera and the way how to make



<<interface>> Identifiable
<<interface>> Named
<<interface>> PartAlignment
<<interface>> VisionSettings
ReferenceBottomVision
- visionSettings: BottomVisionSettings
AbstractVisionSettings
- pipeline:CVPipeline
0
1
0
1
Package
- visionSettings: BottomVisionSettings
BottomVisionSettings
Part
- visionSettings: BottomVisionSettings

Figure 2.8: Class diagram of the new proposed OpenPNP computer vision solution

it manageable[1]. The thread was based on the idea that the whole vision module including the pipeline should be reusable. It was a slightly different approach than mine but still based on the same foundations, making the process manageable. The older discussion has created at the beginning of the year 2021 so I created a new thread backing this one[2]. With one of the maintainers of the repository, we agreed and cooperatively designed a solution making the vision module containing CVPipeline reusable[3].

First thing that needed to be taken into consideration was how to represent the new Vision object. In the original code the vision consisted of different UI elements that were brought together in the object mapped to the part it belonged too. The object was given no id so it could not be referenced, and it lived only in the global machine setup. The proposed solution aims to create an identifiable object representing the vision settings that would stand on the same level as part and package.

Same as parts and packages have their tab in the UI, the vision objects would also have dedicated tab to provide users with the possibility to manage the settings. User should be given an option to assign the vision object to every part and newly also to the package. The proposed UI design is presented in the figure 2.9.

[1]The older discussion is available at https://groups.google.com/g/openpnp/c/7DeSdX4cFUE/m/VYDG6x6-AAAJ

[2]The newly created discussion is available at https://groups.google.com/g/openpnp/c/BehcYrn-qhs/m/oldc9a07AAAJ

[3]The discussion is available at https://github.com/openpnp/openpnp/pull/1318

As the object scope will be extended, it is important to extent the reset functionality and inheritance. Currently there is an option to reset any pipeline to the default one saved in the machine setup, but the default pipeline can also change so it is hard to keep track of what pattern is the specific pipeline following. Original code is also not providing an option to reset other elements in the vision settings like tolerance or part size check method. The code also doesn't count with the package to play part in the vision setup.

The new solution would introduce a dynamic inheritance between parts, packages, and machine. The first link in the chain of inheritance is the part. If the part doesn't have any vision assigned, the program will automatically assign the package's vision to it. Next, if the package doesn't have any vision assigned, it will be provided with the machine's default vision settings.

The base of the solution is modeled in the class diagram in the figure 2.8. It shows only a portion of the classes that needs to be refactored but those are the core ones. In the middle there stands the new class representing the new bottom vision object replacing the *PartSettings* class. The new object is called *BottomVisionSettings*. It is visible that from the vision settings point of view the *Package* now stands on the same level as the *Part* or the Machine *Bottom Vision*. I am not showing any Wizard classes in this diagram. There should be one for each respective class, so I am omitting them from the diagram to maintain the simplicity.

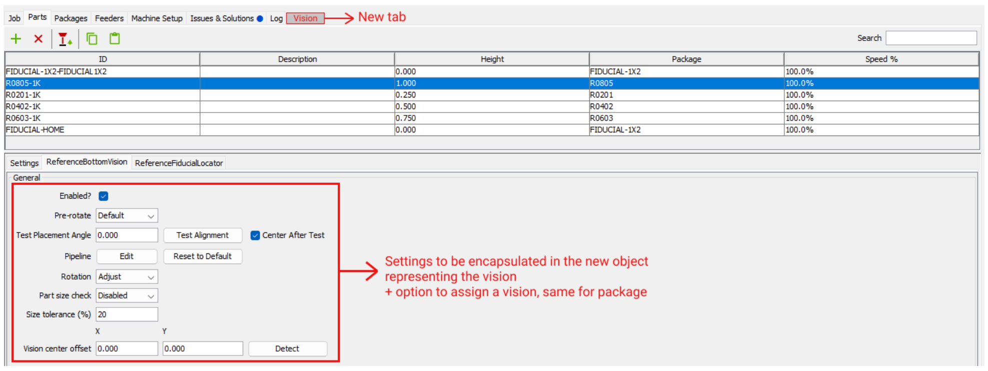
New code would be backed with a slightly different xml configuration, so it is crucial to introduce a migration solution. During the migration, the old *PartSettings* mapping solution should be removed and replaced by a new xml element representing the vision and its settings and every part and package should be provided with an ID of the migrated vision to serve as a foreign key.

Current proposal supplies a solution only for an up-looking camera vision but there is also a vision belonging to the down-facing camera for machine head navigation. It is important to implement the new patch the way that would be in the future easily extended to different kinds of *Vision*. Current design counts on it and therefore the *BottomVisionSettings* class is designed as an extension to an abstract vision settings class. When new kind of vision is needed, it can simply extend this abstract class.

2.3.3 Part offset detection using machine learning

After researching existing solutions and methods used in manufacturing, I decided to propose a new part offset detection using neural networks as an alternative way to the OpenCV pipeline. The software allows to run a script so adding this method is simple and straightforward.

Papers describing the observed detectors showed very promising results. Based on the character of the data collected by OpenPNP software, I can assume that I could achieve very good results and help to introduce a robust solution that could replace current OpenCV pipeline.



ID	Description	Height	Package	Speed %
FIDUCIAL-1X2-FIDUCIAL 1X2		0.000	FIDUCIAL-1X2	100.0%
R0805-1K		1.000	R0805	100.0%
R0201-1K		0.250	R0201	100.0%
R0402-1K		0.500	R0402	100.0%
R0603-1K		0.750	R0603	100.0%
FIDUCIAL-HOME		0.000	FIDUCIAL-1X2	100.0%

Enabled? ✓
Pre-rotate: Default
Test Placement Angle: 0.000
✓ Center After Test
Rotation: Adjust
Part size check: Disabled
Size tolerance (%): 20
Vision center offset: X 0.000, Y 0.000

Settings to be encapsulated in the new object representing the vision
+ option to assign a vision, same for package

Figure 2.9: Visualization of the UI part of the proposed solution

To cope with the existing code and make the whole process as simple as possible, running the model would be the same as running a single stage in the CVPipeline. Like that, there could be a new Vision created containing a pipeline that would be running the model. The whole pipeline would at its base have a stage to capture the image from the camera and run the model upon the capture. If needed, prior stages can be added to preprocess the image as required. At the end of the pipeline, the software would derive the offset based on the rotated bounding box the same way as now.

As already described in the previous chapter, there exists some solutions of detectors for arbitrarily rotated objects. I decided to choose the R^{3}det single stage detector. From the results published in the paper it seems like a promising solution. There is a complete code available as well as a working benchmark called RotationDetection. The benchmark does not contain only the implementation of R^{3}Det detector but also other detectors like two-stage SCRDet, Refined RetinaNet or other variations using like Gaussian Wasserstein Distance Loss function or Kullback-Leibler divergence. It will allow me to train multiple detectors and compare the results to choose the best performing one. The project is well documented and includes the guide to run the detectors upon custom data. The project is written in Python and the benchmark I am going to use is built on TensorFlow 1.x. The structure of the project is displayed on the figure 2.10.

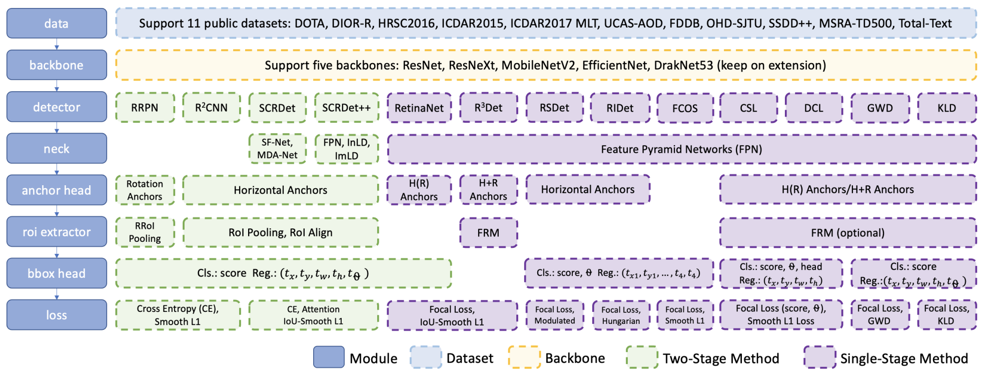

data
Support 11 public datasets: DOTA, DIOR-R, HRSC2016, ICDAR2015, ICDAR2017 MLT, UCAS-AOD, FDDB, OHD-SJTU, SSDD++, MSRA-TD500, Total-Text
backbone
Support five backbones: ResNet, ResNeXt, MobileNetV2, EfficientNet, DrakNet53 (keep on extension)
detector
RRPN
R^{2}CNN
SCRDet
SCRDet++
RetinaNet
R^{3}Det
RSDet
RIDet
FCOS
CSL
DCL
GWD
KLD
neck
SF-Net, MDA-Net
FPN, InLD, ImLD
Feature Pyramid Networks (FPN)
anchor head
Rotation Anchors
Horizontal Anchors
H(R) Anchors
H+R Anchors
Horizontal Anchors
H(R) Anchors/H+R Anchors
roi extractor
RRoI Pooling
RoI Pooling, RoI Align
FRM
FRM (optional)
bbox head
Cls.: score Reg.: $(t_x, t_y, t_w, t_h, t_\theta)$
Cls.: score, θ Reg.: $(t_{x1}, t_{y1}, \ldots, t_4, t_4)$
Cls.: score, θ, head Reg.: (t_x, t_y, t_w, t_h)
Cls.: score Reg.: $(t_x, t_y, t_w, t_h, t_\theta)$
loss
Cross Entropy (CE), Smooth L1
CE, Attention IoU-Smooth L1
Focal Loss, IoU-Smooth L1
Focal Loss, Modulated
Focal Loss, Hungarian
Focal Loss, Smooth L1
Focal Loss (score, θ), Smooth L1 Loss
Focal Loss, GWD
Focal Loss, KLD
Module
Dataset
Backbone
Two-Stage Method
Single-Stage Method

Figure 2.10: Structure of methods used in the RotationDetection project

Chapter 3

Implementation

In this chapter I will describe my approach and used methods when implementing the new feature for OpenPNP software. Following text is divided into sections together describing the implementation of the part detection management and an introduction of machine learning model for the offset detection. As the last, I will provide details about data collecting for future training purposes.

Testing of the new solution was done using unit tests and manual tests. After the pull request is approved and merged to the repository the code goes first to the testing branch so users can try it out and test it before it will go to the main branch.

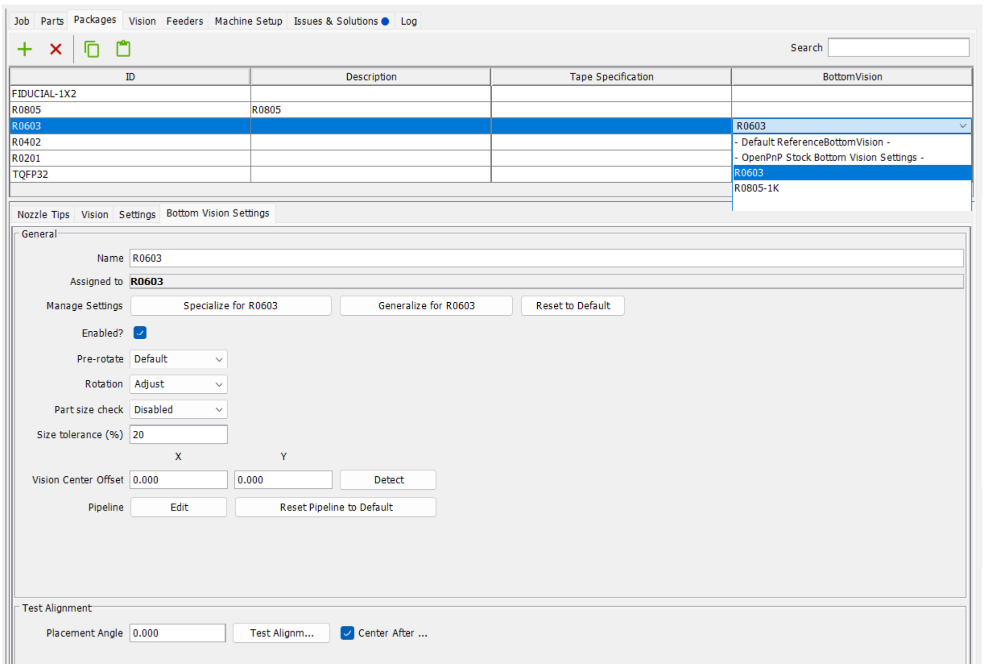
The implementation is based on the analysis stated in the previous chapter. The proposed design was done by me but as the new feature for OpenPNP software was developed in a cooperation with the repository maintainer the final design and functionality was extended. The extension does not make impact on the scope of this work. I will briefly mention the extended functionality, but I will not present the details and it is important to state that it is not clearly my implementation.

3.1 Part detection management implementation

To start working on the code I forked the repository to create my custom copy to work on, following the standard way of contributing to a GitHub repository. I implemented all the code changes upon a test branch and at the end I created a pull request so my code could be reviewed by the repository maintainers.

In the middle of everything there stands *BottomVisionSettings* (BVS) class. This class represents the up-looking camera settings model that was created to replace the previous *PartSettings* class. Above the attributes the BVS took from *PartSettings* it now contains a name and an id. This is by default given by inheritance from *Named* and *Identifiable* interface. Between the *VisionSettings* interface and the BVS class there stands abstract class *Ab-*

stractVisionSettings. Its purpose is to provide extendable model in case there will be another *VisionSettings* needed. It holds the common attributes every vision should have including the CVPipeline.



ID	Description	Tape Specification	BottomVision
FIDUCIAL-1X2			
R0805	R0805		
R0603			R0603
R0402			
R0201			
TQFP32			

- Default ReferenceBottomVision -
- OpenPnP Stock Bottom Vision Settings -
R0603
R0805-1K

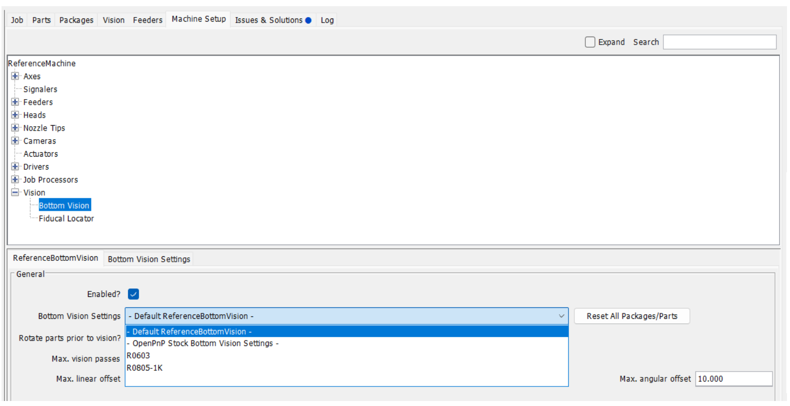
Name: R0603
Assigned to: **R0603**
Manage Settings: Specialize for R0603 | Generalize for R0603 | Reset to Default
Enabled? ✓
Pre-rotate: Default
Rotation: Adjust
Part size check: Disabled
Size tolerance (%): 20
Vision Center Offset: X 0.000, Y 0.000 | Detect
Pipeline: Edit | Reset Pipeline to Default
Placement Angle: 0.000 | Test Alignm... | ✓ Center After ...

Figure 3.1: OpenPNP new UI Package panel

The BVS inheritance is done in the same manner as described previously in the analysis chapter. If there is no BVS assigned to a part, it inherits the settings from its package. Same goes with the package which inherits from the Machine default Bottom Vision. In addition, user can specify, which BVS will be used as the default one. He or she can choose from the list of all defined BVS, default BVS migrated from the previous configuration and stock BVS. Stock BVS is a setting that comes from the original configuration. It represents built-in configuration that is immutable and is identified by the unique id.

Multiple reset scenarios needed to be handled. The set of reset operations differs slightly based on the placed where the BVS is handled. The BVS can be reset to the default or user can reset only the pipeline. The default pipeline is the pipeline saved in the default BVS. In the package's *Bottom Vision Settings* panel there is an extra button for generalization. This function resets the BVS for all the parts within the package. It simply unassign any special BVS from the parts so they will inherit the package's BVS by default. The same logic is implemented for the Machine, allowing user to reset all the packages and parts.

Another model classes like Part and Package are now having an extra attribute representing the assigned BVS. In the UI, every Part and Package is organized in the table. The table has been extended by one more column with a combo box, so user is able to choose and assign the BVS manually to individual part or package. When part or package is selected it is possible to update the vision attributes in the *Bottom Vision Settings* panel. The layout of the panel is almost the same as in the *eferenceBottomVision* panel original version except few extra elements. Now, there are options to change the name of the assigned or inherited BVS, specialize the BVS for the selected part or package, or reset the BVS to the default. Specialization is another way, how to assign the inherited BVS.



Enabled? ✓

Bottom Vision Settings: - Default ReferenceBottomVision -

- Default ReferenceBottomVision -
- OpenPnP Stock Bottom Vision Settings -
R0603
R0805-1K

Rotate parts prior to vision?

Max. vision passes

Max. linear offset

Max. angular offset: 10.000

Figure 3.2: OpenPNP UI Machine panel, Bottom Vision section

There is a new panel dedicated for the BVS objects displayed in the table following the same pattern as with panel for *Part*, *Package* or *Feeder* objects. The table contains two columns with name and list of part or packages the BVS is assigned to. When a BVS is selected, the UI provides user with the settings panel same as with the part or package.

When loading and saving the configuration, the BVS class represents the model class same as *Part* or *Package* so it is loaded from the xml configuration file and follows the same persisting logic. The link between BVS and *Part* and *Package* is represented by xml attribute bottom-vision-id. The attribute represents the foreign key relationship.

Extra functionality for fiducial vision was added by the repository maintainer on the top of my implementation. To navigate the machine head the program uses down-looking camera and computer vision related to it. This fiducial computer vision uses the same principle of OpenCV pipeline as the *BottomVision*. It looks for fiducial markers on the board and process the captured images by CVPipeline. The implementation is using the same base implementation by creating new *FiducialVisionSettings* class extending the *AbstractVisionSettings* class.

As already mentioned in the analysis chapter, the functionality was designed and implemented in the cooperation with one of the OpenPNP repository maintainers[4]. The created Pull Request was successfully merged to the repository to testing branch. There was an announcement and a video made in the discussion channel, informing about the changes and showing the use case of the new feature [9].

3.2 Offset detection using machine learning

OpenPNP has a built-in scripting engine. It allowed me to add a machine learning solution for the offset detection without the need to change the source code. To modify the code as little as possible I introduced a new stage that represents the trained model. The new stage is represented by *ModelRun* class extending *CVStage* abstract class. To add the stage to the list of all stages so it could be listed and selected I needed to register it in the *CVPipelineEditor*. The name of the stage needs to be *"results* so the *DrawRotatedRects* stage can retrieve it and draw the rectangle in the image.

```java
public class ModelRun extends CvStage {

    public RotatedRect rectangle;

    @Override
    public Result process(CvPipeline pipeline) throws Exception {

        File file = Configuration.get()
                        .createResourceFile(getClass(), "Test_", ".png");
        Imgcodecs.imwrite(file.getAbsolutePath(),
                          pipeline.getWorkingImage());

        Map<String, Object> globals = new HashMap<>();
        globals.put("imageFile", file);
        globals.put("stage", this);

        Configuration.get().getScripting().on("Model.Run", globals);

        return new Result(null, rectangle);
    }
}
```

Listing 3.1: Code of a *process()* method in the *ModelRun.java* stage class

The code of the stage can be seen in the listing 3.1. From the pipeline, the code simply loads the image that was captured in the previous stage. Then it defines variables that will be sent to the script and runs the script using the scripting engine. The return value is a *Result* object containing the rotated rectangle. The return value is saved in the pipeline mapped on the stage so it can be retrieved any time during the pipeline execution.

[4]The complete Pull Request with discussion and further implementation details can be seen on
https://github.com/openpnp/openpnp/pull/1318

```
import os
import tensorflow as tf
import org.opencv.imgcodecs as imgc
import org.opencv.imgproc as imgp
from org.opencv.core import RotatedRect

img_data = imgc.imread(imageFile)
img_plac = tf.compat.v1.placeholder(dtype=tf.uint8, shape=[None, None, 3])
img_batch = tf.cast(img_plac, tf.float32)

with tf.compat.v1.Session() as sess:
    saver = tf.train.import_meta_graph(checkpoint_path + '.meta')
    saver.restore(sess, tf.train.latest_checkpoint(checkpoint_path))

    img_h, img_w = img_data.shape[0], img_data.shape[1]
    new_h, new_w = min(int(short_size * float(img_h) / img_w), max_len),
        short_size
    img_resize = imgp.resize(img_data, (new_w, new_h))

    resized_img, detected_boxes, detected_scores, detected_categories =
        sess.run([img_batch], feed_dict={img_plac: img_resize[:, :, ::-1]})

    if detected_boxes.shape[0] != 0:
        resized_h, resized_w = resized_img.shape[1], resized_img.shape[2]
        detected_boxes[:, 0::2] *= (img_w / resized_w)
        detected_boxes[:, 1::2] *= (img_h / resized_h)

    x_c, y_c, w, h, theta = detected_boxes[:, 0][0], detected_boxes[:, 1][0],
        detected_boxes[:, 2][0], detected_boxes[:, 3][0], detected_boxes[:,
            4][0]

    stage.rectangle = RotatedRect([x_c, y_c, w, h, theta])
```

Listing 3.2: Code of the script running the trained model

The script takes the file name and stage class as arguments so it can write to the stage's variables and provide the result in the for in rotated rectangle. First, it loads the image and creates a TensorFlow session. The session is used to restore the trained model from the checkpoint saved in the configuration. After the session is restored and run upon the captured image, it will return the detected box. The box is in the form of four points array, so it needs to be converted to format with the angle. I omitted this part from the code listing.

The implementation of the machine learning solution was discussed among the OpenPNP community but the Pull Request was not yet created. Due to the complexity of the problem the code is still in progress and requires more testing. After the trained model is fully improved and all the scenarios are tested, the code patch can be published for a review. Based on the experience with pipeline management implementation and approving process the code could be successfully merge within next few months.

3.3 OpenPNP migration

Because I changed the xml configuration structure, it was necessary to introduce a migration solution that transforms the original configuration to the new one. The migration code is run after the configuration is fully loaded. Therefore, the original structures need to be kept so the xml files are loaded and mapped to the object flawlessly.

After everything is loaded successfully the migration takes the map of *PartSettings* and transforms them one by one to the new BVS objects. Before that the stock settings and default settings are migrated to avoid duplications. This process is shown in the listing 3.3. To make the code more compact I omitted some parts that are not important for the logic, like null or invalid data checking, logging or attributes setting. To avoid duplicates, every newly created BVS is stored in a hash map. Every BVS is assigned to the respective part and saved in the configuration. At the end of the migration process the *partSettingsByPartId* map and original default pipeline are set to null so they will not be present in the configuration anymore.

```
protected void migratePartSettings(Configuration configuration) {
    HashMap<String, BottomVisionSettings> bottomVisionSettingsHashMap = new
        HashMap<>();

    // Create stock settings, add it to the configuration and the map above
    // Migrate the default settings
    // reset vision settings in all parts and packages

    partSettingsByPartId.forEach((partId, partSettings) -> {
        Part part = configuration.getPart(partId);

        String serializedHash = createSettingsFingerprint(partSettings);
        BottomVisionSettings bottomVisionSettings =
            bottomVisionSettingsHashMap.get(serializedHash);
        if (bottomVisionSettings == null) {
            bottomVisionSettings = new BottomVisionSettings(partSettings);
            bottomVisionSettingsHashMap.put(serializedHash,
                bottomVisionSettings);

            configuration.addVisionSettings(bottomVisionSettings);
        }

        part.setVisionSettings((bottomVisionSettings !=
            defaultBottomVisionSettings) ?bottomVisionSettings : null);
    });

    partSettingsByPartId = null;
}
```

Listing 3.3: Code of the xml configuration migration

3.4 Data collection

To train the selected detector and get accurate results in the form of bounding boxes with correct offset and rotation it was necessary to collect enough data directly from the pick and place machine. For the training dataset to be accurate it was essential to collect the images from the bottom vision camera and save corresponding annotations. From the annotations it was possible to describe the images correctly and specify a ground-truth boxes for the training process.

The OpenPNP uses right-hand coordination system as can be seen in the figure 3.3. This coordination system assumes that we are standing above the machine looking down at it. Nozzle then moves on the X axis from right to left where right is positive, on the Y axis forward and back, forward being negative and on the Z axis up and down. The last direction C specifies the rotation. Counter-clockwise rotation has positive values. The default unit for the movement on X, Y and Z axis is a millimeter. Rotation is given in degrees.

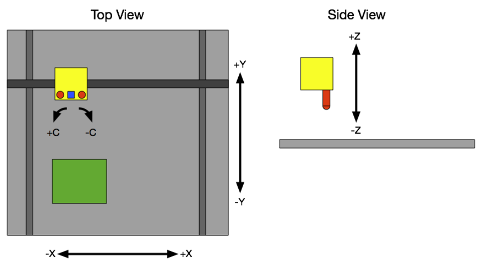


Figure 3.3: OpenPNP coordination system visualisation

The easiest option how to collect the data and be able to interactively see and control the process in the OpenPNP application was to implement it directly in the software. In the OpenPNP every CVStage class contains process() method. That method runs the corresponding stage's operation producing appropriate results. I used this functionality to create my own custom stage which results in the set of images and annotations. The code structure of the custom process() method can be seen in the listing 3.4 bellow.

```
public Result process(CvPipeline pipeline){
    // initialize the starting parameters

    for (int exp = expMinValue; exp < expMaxValue; exp++) {
        camera.getExposure().setValue(exp);

        for (int light = 3; light <= lightCount; light++)

            for (int cap = 0; cap < captureCount; cap++) {
                float curXDeviation =
                    (new Random().nextFloat() - 0.5)*2 * maxXDev;
                float curYDeviation =
                    (new Random().nextFloat() - 0.5)*2 * maxYDev;
                float curRDeviation =
                    (new Random().nextFloat() - 0.5)*2 * maxRDev;

                Location offsetLocation = new Location(curXDeviation,
                    curYDeviation, 0, curRDeviation);
                nozzle.moveTo(baseLocation.addWithRotation(offsetLocation));

                float curLight = (1 << light) -1;
                actuator.actuate(curLight);

                BufferedImage bufferedImage = camera.settleAndCapture();
                Mat image = OpenCvUtils.toMat(bufferedImage);
                Imgcodecs.imwrite(outFile, image);

                // assemble and save json data containing part parameters
                // and all the deviations
            }
        }
}
```

Listing 3.4: Code of a process() method in the TestCapture.java stage class

The code sets different values of camera exposure and light to simulate changing and unstable conditions. For every capture and the combination of the exposure and light, the part on the nozzle was moved to a random location. The location was given by random deviations. The deviations were applied on the x, y location and on the rotation. For every part the capture process was run twice. First time the process captured images of the part picked by the nozzle the standard was. Second time the part was deflected from its position on the nozzle manually to simulate the wrong pick up when the part of the nozzle is visible in the capture.

After the dataset was collected it was necessary to go through the pictures manually and delete the outliers that could confuse the training process. As an outlier I considered captures with too high or too low exposure, so it was not clear, what part is in the picture.

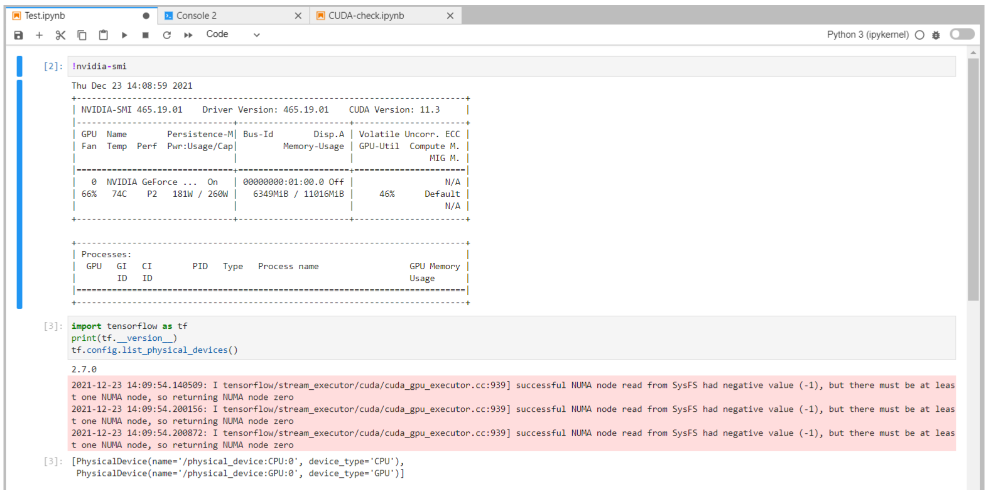
3.5 Model setup

As described in the analysis chapter I chose to use TensorFlow-based detection benchmark. In order to start training I had to fork the repository and made some adjustments. All the steps required are put in the documentation. The model was originally trained and tested upon several dataset but the mainly used one was DOTa dataset.

First step to do was to download the pretrained weights. The model I was using was pretrained Resnet model. After that I needed to compile the project by running the predefined scripts and the code for ready to train the DOTa dataset. To train my own dataset I needed to modify some variables like the name of the dataset to be loaded, number of classes and the version. The training script is loading the configuration from *cfgs.py* file that differs slightly based on used model or data.

The training process loads the data from the tfrecord file and as the main format of the image annotation it uses the xml format. The repository already contained the script for transforming the data into desired format. All I needed to do was to introduce logic to parse my json data annotations into the xml format and migrate the offsets into the bounding box format defined as $[x1, y1, x2, y2, x3, y3, x4, y4]$.

I trained my models using Fraktal, a powerful computer placed in Impro-Lab equipped with GeForce RTX 2080 graphic card. The computer disposes of CUDa driver 11.3 version and GPU memory of 11016 MiB. In the image, I installed CUDa compilation tool in version 11.2. I was connecting to it using ssh connection. On the computer I was using a docker image from GPU -Jupyter project. This project uses NVIDIa CUDa image and install the Python, R and Julia toolstack on top of it. It enables GPU calculations in Jupyter notebooks [36]. The environment is show-cased in the figure 3.4.



```
[2]: !nvidia-smi

Thu Dec 23 14:08:59 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 465.19.01    CUDA Version: 11.3     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0 Off |                  N/A |
| 66%   74C    P2   181W / 260W |   6349MiB / 11016MiB |     46%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+

[3]: import tensorflow as tf
     print(tf.__version__)
     tf.config.list_physical_devices()

2.7.0
2021-12-23 14:09:54.140509: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at leas
t one NUMA node, so returning NUMA node zero
2021-12-23 14:09:54.200156: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at leas
t one NUMA node, so returning NUMA node zero
2021-12-23 14:09:54.200872: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at leas
t one NUMA node, so returning NUMA node zero

[3]: [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
      PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Figure 3.4: GPU-Jupyter docker image Jupyterlab environment

The TensorFlow version 1.x which the benchmark is built on is not compatible with CUDa driver version 11.0 and higher. It was necessary to migrate the code to TensorFlow 2.x. For that I run the official migration script and replaced imports of tensorflow.contrib.slim library with tf_slim.

Chapter 4

Model Evaluation

In this chapter I will present the training and testing process of the detection models. Prior to that, I will enclose a brief description of the dataset upon which the model was trained. For training I chose 4 different methods. I selected three single-stage methods including RetinaNet for rotated objects using rotated anchors (RetinaNet-R), R^{3}Det detector and R^{3}Det combined with Kullback-Leibler divergence (R^{3}Det-KLD). As the last detector I chose a state-of-art two-stage detector SCRDet. All four detectors were introduced in the Analysis chapter. At last, I added a section for discussion where I will discuss the results and propose possible future enhancements and improvements.

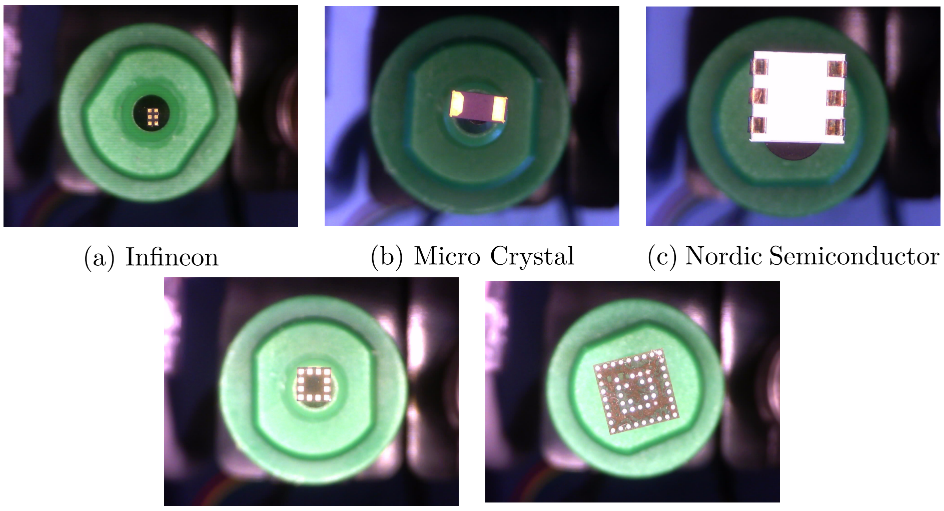
4.1 Dataset

For the training I was using custom dataset collected from up-looking camera on P&P machine using OpenPNP tool. The dataset contains 1074 images of 5 categories defined as Infineon, Micro Crystal, Nordic Semiconductor, STMicroelectronics and U-blox. Each category represents a component picked up on the nozzle. The image size is 1944x2592 pixels and contains exactly one object with an arbitrary center point offset and rotation offset.

Dataset was collected so it represents various situations. It contains images with different lighting conditions as well as images simulating wrongly picked component where part of the nozzle is visible. Every image in the dataset is annotated by xml file containing the category of the object and its location represented as $[x1, y1, x2, y2, x3, y3, x4, y4]$ bounding box. The smallest object is Infineon amplifier with size of 0.7x1.1 millimeter which makes something around 49x77 pixels in the image and the biggest is Nordic Semiconductor of size 4.5x4.5 millimeters and something around 314x314 pixels. Figure 4.1 shows all parts in dataset labeled with the respective category. Images in the figure are cropped so the objects of interest are clearly visible.

The dataset is split into training, validation and testing sets using 80:10:10 ratio. To avoid model overfitting, I doubled the size of dataset by duplicating each image and applying an image augmentation to each image before it enters the training, validation or testing process. Augmented data for validation and testing is not the usual practice, but it allows more accurate measurements. The augmentation contains a sequence of random rotation and vertical or horizontal flip. Each image is also normalized by simple pixel mean subtraction.

Originally, the dataset also contained images of empty nozzles but the there was an error in collected annotations and I was not able to use them for training. Therefore, I omitted those images from datasets as it would bring an error to the results.



(a) Infineon (b) Micro Crystal (c) Nordic Semiconductor

(d) STMicroelectronics (e) U-blox

Figure 4.1: OpenPNP dataset objects representing categories

4.2 Evaluation of the trained model

For consistency and fairness in the model comparison all experiments were performed with the same parameter settings. Maximum number of epochs was set to 13. Checkpoints with respective weights were saved every 20 000 to 30 000 iterations. In total, every model generated around 15 checkpoints. All the models used pretrained ResNet50 model as it was recommended. All the loss graphs where generated using TensorBoard tool.

When observing the cross-entropy loss graph in the figure 4.2, all the detectors performed well. They reached almost zero loss in the first 20 000 iterations and thus it does not make sense to observe the cross-entropy loss further. Same goes for the learning rate. All detectors' learning rate changed in the first 15 000 iterations.

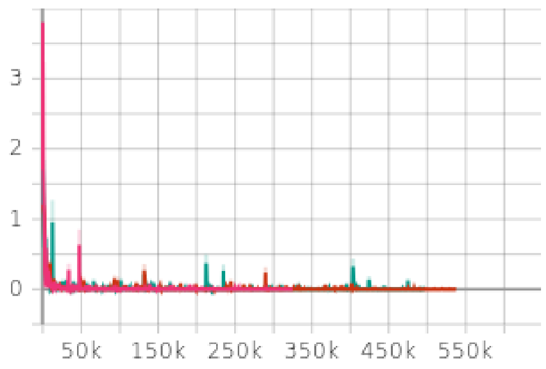


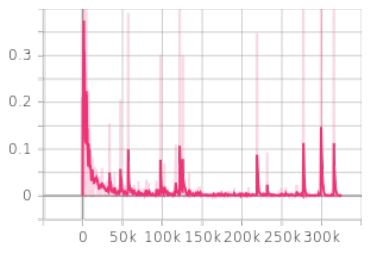
Figure 4.2: Cross-entropy loss for all detectors, x-axis measures iteration, y-axis measures the loss value

For measuring the performance, I am using mainly mean average precision (mAP) and detection speed. During the validation/testing number of true and false positives were collected by calculating intersection over union (IoU) between detected bounding boxes and ground-truth boxes using different values of threshold. From collected numbers precision, recall and F1 scores, average precision (AP) and mAP were calculated. AP is generally the area under precision-recall curve and is calculated for each category. The mAP is then calculated by averaging all the AP values. For the model evaluation I was using three values of thresholds: 0.75, 0.8 and 0.95.

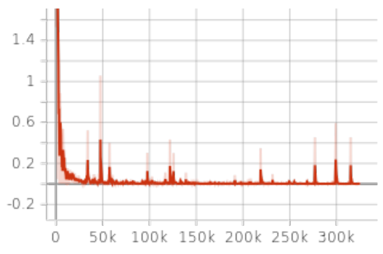
For purpose of this work, the final detector should have the best trade-off between the performance and detection speed even when high threshold is used. The tolerance of component misalignment is always specific to the component [37][38], but it was empirically measured, and we can assume that for the part placement threshold of 0.85 is sufficient. Therefore, for comparing testing results I will be using this threshold.

As first I trained the RetinaNet-R model as the single-stage model that serves as a base for various refinements. The model was trained for 300 000 iterations. The regression loss and total loss graphs are in the figure 4.3. Validation data performance in the figure 4.4 shows that the best results were achieved by using threshold 0.75 as expected but using threshold 0.8 the performance is also good. On the other hand, the performance when using higher threshold of value 0.95 is low and the model is useless.

Next two trained models are R^{3}Det single-stage detector and mixed method R^{3}Det-KLD. Compared to the RetinaNet-R loss graphs the R^{3}Det loss line in the figure 4.5 has more outliers making the line less smooth. The R^{3}Det-KLD loss did not converge at all and it was constantly oscillating between values 1 and 4. I was not able to achieve significantly better results not even after trying to modify the hyperparameters of the model. However, both models achieved quite good performance on the validation dataset. With threshold



(a) RetinaNet-R regression loss



(b) RetinaNet-R total loss

Figure 4.3: RetinaNet-R loss graphs, x-axis measures iteration, y-axis measures the loss value

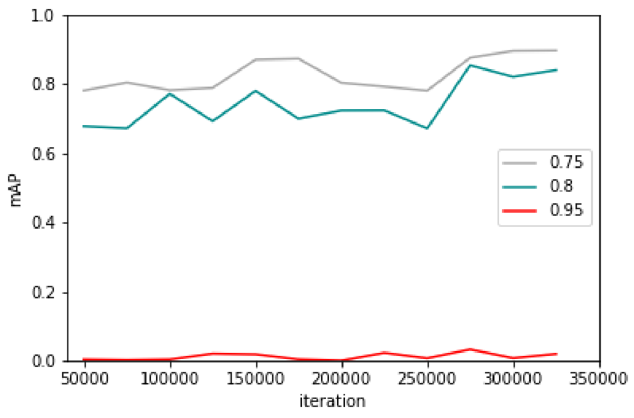


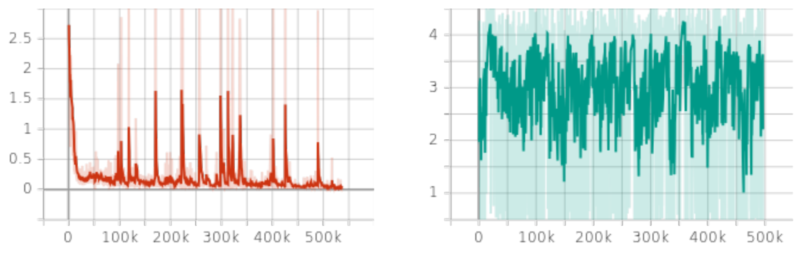
Figure 4.4: RetinaNet-R performance of validation data using thresholds 0.75, 0.8 and 0.95

value 0.75 the R^{3}Det outperformed RetinaNet-R by 8% in the best result. Using threshold 0.8 and 0.95 the result of RetinaNet-R and R^{3}Det are similar. R^{3}Det-KLD result were lower but still reaching maximum value of 0.8 mAP when using 0.8 threshold.

As last model to train, I chose state-of-art two-stage detector SCRDet based on Faster R-CNN. When observing the loss function graph and validation performance, the results are close to RetinaNet-R results, but it seems that RetinaNet even outperformed the SCRDet model when using higher threshold.

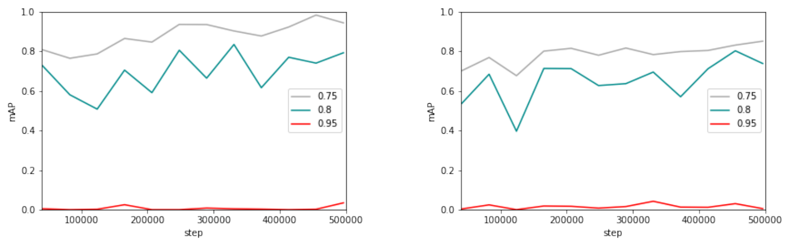
Table 4.1 lists the comparison between detectors detection speed. The lowest detection speed was measured for SCRDet model as expected as it is the only two-staged detector. The best detection speed was reached by RetinaNet-R. The model speed was 23 fps outperforming R^{3}Det by 6 fps.

I took two best performing detectors R^{3}Det and RetinaNet-R and tried to train them for additional 4 epoch. When evaluating the trained models



(a) R^{3}Det total loss (b) R^{3}Det-KLD total loss

Figure 4.5: R^{3}Det and R^{3}Det-KLD model total loss, x-axis measures iteration, y-axis measures the loss value



(a) R^{3}Det performance (b) R^{3}Det-KLD performance

Figure 4.6: R^{3}Det and R^{3}Det-KLD performance of validation data using thresholds 0.75, 0.8 and 0.95

Detector	Speed (fps)
R^{3}Det	17
R^{3}Det_KLD	18
Retina	23
SCRDet	14

Table 4.1: Detection speed evaluation of trained detection methods on OpenPNP dataset.

during the training I did not observed any improvements in the results.

Based on the performance on the evaluation data and loss function I selected the best performing checkpoint for each detector and measured the testing performance adding one more threshold value of 0.85. The results can be seen in the table 4.2. The first two best results are highlighted considering not just value but also the threshold. Most of the best (green) values are in the 0.8 threshold section and the rest is in the 0.75 threshold part.

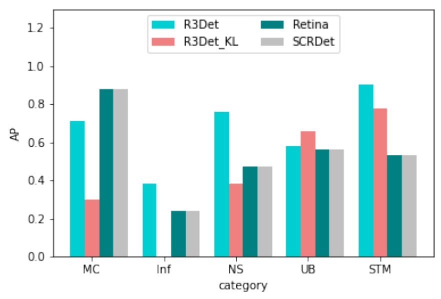
When observing the performance upon individual categories and detectors, the R^{3}Det and SCRDet performed well on all categories. R^{3}Det scored the highest mAP for 0.85 threshold. It was also the only method that scored over 80% mAP in higher threshold for Infineon category that seems like to

Method	MC	Inf	NS	UB	STM	mAP
threshold = 0.75						
R3DET	1.00	**0.88**	**1.00**	**1.00**	1.00	**0.98**
R3DET_KLD	**0.89**	0.15	**1.00**	**1.00**	1.00	0.81
Retina	1.00	0.37	**1.00**	**1.00**	1.00	0.87
SCRDet	1.00	0.84	**1.00**	**1.00**	0.90	0.95
threshold = 0.8						
R3DET	**1.00**	**0.86**	**1.00**	**1.00**	0.91	**0.95**
R3DET_KLD	0.69	0.00	0.86	**1.00**	**1.00**	0.71
Retina	**1.00**	0.18	**1.00**	**1.00**	**1.00**	0.84
SCRDet	**1.00**	0.54	**1.00**	**1.00**	0.88	0.88
threshold = 0.85						
R3DET	0.71	0.38	0.76	0.58	0.90	0.67
R3DET_KLD	0.30	0.00	0.38	0.66	0.78	0.42
Retina	0.75	0.08	0.61	0.70	0.79	0.59
SCRDet	0.88	0.24	0.47	0.56	0.53	0.54
threshold = 0.95						
R3DET	0.01	0.00	0.14	0.19	0.04	0.07
R3DET_KLD	0.00	0.00	0.00	0.01	0.06	0.01
Retina	0.05	0.00	0.11	0.20	0.00	0.07
SCRDet	0.04	0.00	0.15	0.01	0.03	0.04

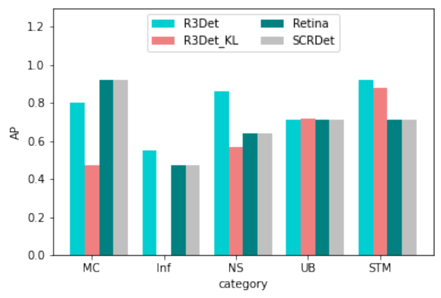
Table 4.2: Performance evaluation of trained detection methods on OpenPNP dataset. The shortnames for the categories are defined as: MC=Micro Crystal, Inf-Infineon, NS-Nordic Semiconductor, UB-U-blox, STM-STMicroelectronics

most difficult one. R^3Det_KLD was the detector that scored the worst values. Its mAP for 0.8 threshold was 71% where other detectors scored more than 80%. RetinaNet-R also achieved good results competing with other mentioned detectors. Figure 4.7 shows the same data for categories along the F1 score visualized using the most relevant and performing threshold of value 0.85.

When run upon testing data and using the threshold of value 0.85, the R^3Det detector achieved 0.67 mAP outperforming the SCRDet detector by 13%. The detector was able to detect 166 of 220 images correctly reaching 75% of sensitivity. The best results of 0.9 AP it achieved for the STMicroelectronics category. We can consider the Infineon category as the most difficult one. Generally, the results upon this category where lower due to its size and thus it is more sensitive to bounding box deviation. The R^3Det detector reached 0.38 AP that is by 14% better than the state-of-art SCRDet detector.



(a) AP of threshold = 0.85



(b) F1 of threshold = 0.85

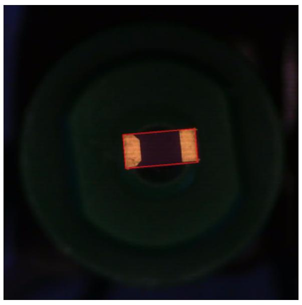
Figure 4.7: AP and F1 score measured for individual categories using all detection methods

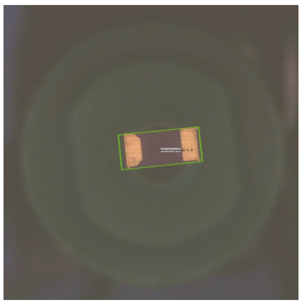
4.3 Detection results comparison

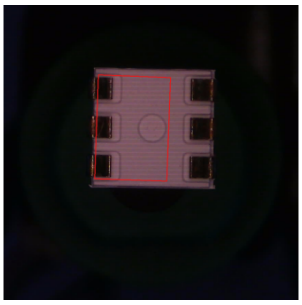
The aim of this work is to provide OpenPNP software with computer vision solution that is robust and dynamically adapts to outer conditions. Therefore, I did not make any image adjustments before testing to simulate the real-time situation. To make the OpenCV default pipeline perform better it is possible to adjust the parameters of individual stages however, it would be necessary to adapt the pipeline differently for each testing capture. Therefore, when testing the current OpenPNP solution, I left the default OpenCV pipeline without any changes. Figures 4.8 and 4.9 show examples of final detections compared to results from OpenPNP default OpenCV pipeline. I chose different kinds of pictures representing the difficult detection cases.

Figure 4.8 shows captures with low camera exposure settings. It can be seen, that in one case (4.8a and 4.8b) the default pipeline performed even better than detector, however, figures 4.8c and 4.8d show better result for the detector.

Figure 4.9 shows captures with high camera exposure settings and capture representing wrong pick where big part of the nozzle is visible. OpenPNP detection in 4.9a is wrong due to the bright color of the nozzle caused by high


(a) Micro Crystal OpenPNP detection


(b) Micro Crystal R^{3}Det detection


(c) Nordic Semiconductor OpenPNP detection

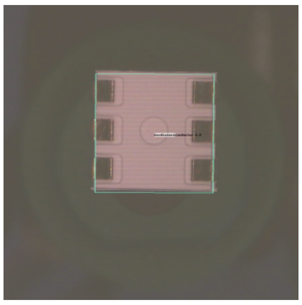
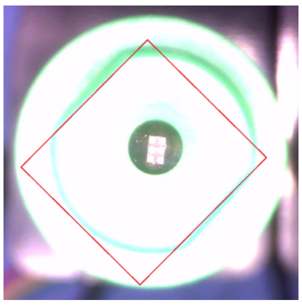
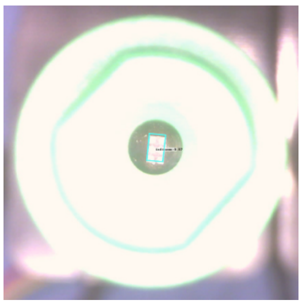

(d) Nordic Semiconductor R^{3}Det detection

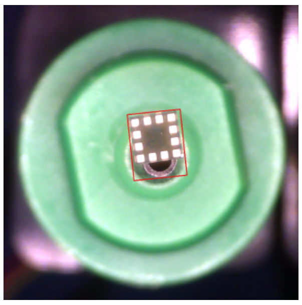
Figure 4.8: Comparison of detection results from R^{3}Det detector and OpenPNP default pipeline on images with low lighting

exposure. When capture is converted to gray scale and image threshold is performed, big part of the nozzle is selected and outlined by the bounding box. In case of visible nozzle (4.9c) the same mistake with image threshold happened and the nozzle was selected and outlined by the OpenPNP pipeline. The detector in both cases performed well.

I tried to run the model upon images of empty nozzles. For most of the nozzle types the detector successfully did not detect any object. The model was not trained upon the nozzle category, so it is not possible to measure the detection accuracy but even so, the model was able to detect no object on 78% of nozzle images. Most of the detection mistakes were done on one type of the nozzle shown in the figure 4.10. The Figure shows the result in detecting empty nozzles when using R^{3}Det detector and OpenPNP computer vision. OpenPNP is not able to detect empty nozzle. Its pipeline is based only on image thresholding and because the color of the nozzle is similar to color of any component, it is not able to distinguish between those two objects. It is also visible that the nozzle where the detector did mistake of detecting an object is not clearly circular and its inner shape is rectangular like. Because of its size, it was detected as Infineon category.


(a) Infineon OpenPNP detection


(b) Infineon R^3Det detection


(c) STMicroelectronics OpenPNP detection

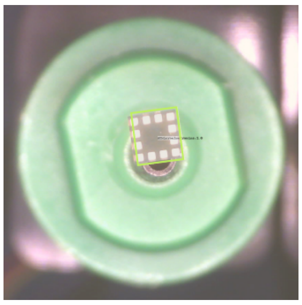

(d) STMicroelectronics R^3Det detection

Figure 4.9: Comparison of detection results from R^3Det detector and OpenPNP default pipeline on images with high lighting or visible nozzle

4.4 Discussion

Above stated results show that recently developed detectors for rotated objects perform well on the custom dataset collected from the P&P machine. The best results were achieved by R^3Det single-stage detector. It also achieved better results than the state-of-art SCRDet detector. The best results were attained for categories like Micro Crystal, Nordic Semiconductor and STMicroelectronics. For the Infineon category the detector performed the worst. The performance upon this category was anticipated to be lower due to its size but it was expected to be higher that the achieved results. Overall, the detector's true positive rate was 75% and considering the high threshold of 0.85, the results fulfilled the expectations.

R^3Det detector also performed the detection task in the same speed as OpenPNP pipeline. We did not achieve any improvement in this direction but the speed is already sufficient so it can be considered as success that the NN solution did reached the same detection speed as current method.

Comparing the results to the results stated in the research papers the performance upon OpenPNP dataset is better. Compared to the DOTa dataset that the detectors were evaluated for, the OpenPNP dataset is not that diffi-

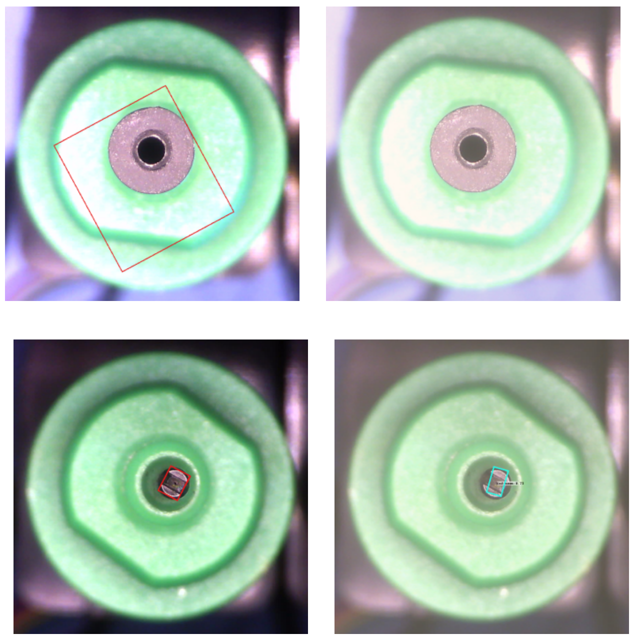


Figure 4.10: Comparison of detection results from R^{3}Det detector and OpenPNP default pipeline on images with empty nozzle

cult. However, in the OpenPNP dataset, it is important for the objects to be defined precisely and due to the application, the accuracy is more crucial.

In cases when the captured image has ideal lightning, the component of the interest is in focus, and the nozzle is fully hidden behind the component, the OpenCV pipeline can still give more precise results. Even so, the detectors for arbitrarily rotated and small objects proved to be effective solution for computer vision of P&P part alignment process. Due to the character of the application, it can be assumed that the detectors would be a good fit to other manufacturing processes where component detection is needed. Single-stage detectors can provide accurate and fast detection and thus improve the industry procedures.

4.4.1 Limitations and possible improvements

Even though the model performed well there are still some limitations and space for improvements. Without the context, the results would be considered good but for purposes of this work better results should be expected for higher thresholds. The models were trained to be used in the P&P software that requires high speed of the recognition and high accuracy in the offset detections. The SMT processes usually manipulate with small size objects and therefore, the trained model should be able to provide high accuracy results for smaller components and higher thresholds like 0.95 on which, in this case, the models performed poorly.

Improving the training data could be one way how to achieve better results. The initial dataset of 1074 samples is small and after splitting the dataset, the train set contains something around 800 samples. Even with this small number of train images I was able to achieve good results, mainly thanks to the augmentation, but I believe increasing the dataset size would bring even better results. The dataset also contained only 5 categories. By gathering more component captures with various environment and machine settings, the final dataset could be more diverse.

Another way how to improve the training dataset would be to improve the offset annotations and ensure that all the ground-truth boxes are defined correctly. Current training data were collected directly from the P&P software using the default OpenCV pipeline for offset detection. It is possible that some offsets were not detected precisely or there was an error introduced when setting the environment and part placement manually. By introducing another check of the offset annotations, the training process could be improved. Different offset check would also help the testing and it would have provided better data when comparing the results between detector and OpenPNP. It would help to generate valid and unbiased performance data for the OpenPNP pipeline and detector itself and it would allow me to measure the improvement of the offset detection.

Another aspect is the testing itself. Currently the results can be compared only to original OpenCV method. By having access to wider range of applications the results could be compared to other methods from manufacturing and it would allow me to see other ways of improvement.

As last way how to improve the training process I will mention settings of hyperparameters of the models. In the papers describing the detection methods, authors were mentioning setting different anchor scales for different objects. For smaller objects it was necessary to setup finer anchors scale then for objects that are bigger and in the foreground. It is possible that this setup could cause improvement not just in the performance but also in the detection speed.

The training in this chapter was performed upon real components used in P&P operations but it is also important to detect empty nozzle and alert error in picking the component. Due to an error in dataset images with empty nozzle I was not able to train the model for those cases. By providing dataset with correct annotations for empty nozzles, the detection could be further improved.

Conclusion

In this section, I will conclude the thesis by summarizing the key points of the work and relate them to the work aims and questions and discuss the contribution thereof. I will also discuss the limitations and opportunities for future research and enhancements.

In this thesis I aimed to apply a new computer vision solution for part offset detection in OpenPNP project. The solution should have been based on research of last trends in machine learning for rotation detection. I was aiming to provide the OpenPNP tool with new implementation making it more competitive in the field of SMT software.

In this work I successfully researched and organized methods used in the industry for part detection and created an overview of publicly available approaches confirming the important part of machine learning in parts detection. I made an overview of recently developed detectors solving challenge of detecting small and rotated objects and proposed a solution, how those detectors could solve the prior stated problems and impediments in OpenPNP computer vision component. The results of the research of detection methods used in manufacturing indicated flaws in current OpenPNP implementation. OpenPNP computer vision is outdated not unable to provide users with robust detection.

Further investigation found recent rotation detectors with high performance that seemed like a perfect fit for the purpose of this work. Taking the researched detectors, I was able to train four of them on custom dataset collected from a P&P machine. I achieved a decent performance and detection speed that could be compared to current solution. Based on the design, I implemented the solution incorporating the trained model into the OpenPNP software. The solution is not replacing the current computer vision completely but serves as an addition to current solution.

I tested the trained detectors and compared the results to current solution. The detectors achieved high performance when using standard thresholds but when testing the data with threshold higher that 0.85 the performance decreased rapidly. The main reason for that could be insufficient dataset size and variability and insufficient model setup. Despite that the model performed good and proved to be sufficient and robust solution that has the potential to replace current OpenCV based computer vision. As part of fixing the limitations of the OpenPNP computer vision module I designed and implemented a solution for managing current solution based on OpenCV operation pipeline. This improvement was implemented as a contribution to existing repository and was successfully approved and merged to the testing branch of the repository. The implementation solves problems discussed among the community and it will help incorporating the machine learning solution more smoothly.

Future work

In the last chapter I detected limitation of the solution and proposed some methods for future work and enhancements.

First limitation of this work was the data quality. The quality of the data could be increased by introducing extra method for generating the data annotations that would serve also for the testing and more accurate results comparison between the detector and OpenPNP solution. Along with increasing the data quantity the overall training process could be improved.

In the evaluation chapter I also mentioned the limitation regarding detection of an empty nozzle images. Current OpenPNP solution is provided with an option to detect circular symmetry. This method could in theory detect empty nozzle, but current solution does not provide users with optional pipeline execution. If this optional execution would be implemented, it would allow users to combine empty nozzle detection with running the train model to detect the part rotation. Unfortunately, I was not able to introduce such solution and therefore I am mentioning it here as a proposal for future work.

Summary

In this thesis I researched computer vision methods used in manufacturing world mainly in Surface Mount Technology for part detection. Next, I researched last trends in object detection for rotated objects. Based on the research I designed and implemented computer vision solution for OpenPNP tool that could serve as an alternative for current OpenCV method providing user with more robust detection. As part of the OpenPNP computer vision improvements I successfully implemented a new solution for OpenCV pipeline management. This implementation resulted in the contribution to the OpenPNP code repository and after approval it was merged to the testing branch.

At the end of this work, I analyzed the results and evaluated the limitations. Based on the limitations I also proposed options for future work to further improve the new solutions.

Bibliography

[1] Yang, X.; Yan, J.; et al. R3Det: Refined Single-Stage Detector with Feature Refinement for Rotating Object. 2020.

[2] IRB 1400 - industrial Robot. Accessed: 2021-12-02. Available from: https://library.e.abb.com/public/99bb3fb8ff6495cfc1257b130056d120/IRB1400_R3-US%2002_05.pdf

[3] LitePlacer – The Prototyping Pick and Place Machine for Your Lab. Accessed: 2021-12-02. Available from: https://liteplacer.com/

[4] Introduction to Surface Mount Technology. Accessed: 2021-12-02. Available from: https://www.surfacemountprocess.com/articles.html

[5] QUICK AND EASY: PICK-AND-PLACE OF MICROPARTS WITH VISIONPRO. Accessed: 2021-12-02. Available from: https://www.cognex.com/en-cz/applications/customer-stories/medical-devices/quick-and-easy-pick-and-place-of-microparts-with-visionpro

[6] Cao, S.; Parviziomran, I.; et al. Prediction of Component Shifts in Pick and Place Process of Surface Mount Technology Using Support Vector Regression. *Procedia Manufacturing*, 2019: pp. 2–6.

[7] Liu, W.; Yang, X.; et al. A novel industrial chip parameters identification method based on cascaded region segmentation for surface mount equipment. *IEEE Transactions on Industrial Electronics*, 2021: pp. 2–5.

[8] Lin, T.-Y.; Goyal, P.; et al. Focal Loss for Dense Object Detection. 2018.

[9] Sharan, R. V.; Onwubolu, G. C. Measurement of end-milling burr using image processing techniques. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, Mar. 2011: pp. 448–452.

[10] Alnowaini, G.; Alttal, A.; et al. Design and simulation robotic arm with computer vision for inspection process. In *2021 International Conference of Technology, Science and Administration (ICTSA)*, 2021, pp. 2–4.

[11] HALCON – THE POWER OF MACHINE VISION. Accessed: 2021-12-02. Available from: `https://www.mvtec.com/products/halcon`

[12] Riordan, A. D. O.; Toal, D.; et al. Object recognition within smart manufacturing. 2019: pp. 2–4.

[13] Cabré, T. P.; Cairol, M. T.; et al. Project-Based Learning Example: Controlling an Educational Robotic Arm With Computer Vision. 2013: pp. 1–2.

[14] Zhang, Z.; Yang, X.; et al. Weighted Smallest Deformation Similarity for NN-Based Template Matching. *IEEE Transactions on Industrial Informatics*, 2020: pp. 6787–6795.

[15] Sharan, R. V.; Onwubolu, G. C. Automating the Process of Work-Piece Recognition and Location for a Pick-and-Place Robot in a SFMS. *International Journal of Image, Graphics and Signal Processing*, Mar. 2014: pp. 3–6.

[16] Kumar, R.; Lal, S.; et al. Object detection and recognition for a pick and place Robot. In *Asia-Pacific World Congress on Computer Science and Engineering*, 2014, pp. 1–4.

[17] Schindler, K.; Suter, D. Object detection by global contour shape. *Pattern Recognition*, Dec. 2008.

[18] Sanz, P.; Marin, R.; et al. Including efficient object recognition capabilities in online robots: from a statistical to a Neural-network classifier. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2005: pp. 2–6.

[19] Marin, R.; Sanchez, J.; et al. Object recognition and incremental learning algorithms for a web-based telerobotic system. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No .02CH37292)*, 2002, pp. 2719–2724 vol.3.

[20] Kipkosgei, P.; Njiri, J. G.; et al. Real Time Object Detection using Single Shot Multibox Detector Network for Autonomous Robotic Arm. *JOURNAL OF SUSTAINABLE RESEARCH IN ENGINEERING*, Sep. 2020.

[21] Liu, W.; Anguelov, D.; et al. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, 2016: p. 1–4.

[22] Andhare, P.; Rawat, S. Pick and place industrial robot controller with computer vision. In *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 2016, pp. 1–4.

[23] Shih, H.-C.; Yu, K.-C. A New Model-Based Rotation and Scaling-Invariant Projection Algorithm for Industrial Automation Application. *IEEE Transactions on Industrial Electronics*, 2016: pp. 4452–4460.

[24] OpenPNP. Accessed: 2021-12-02. Available from: `https://openpnp.org/`

[25] OpenPNP. Bottom Vision. Accessed: 2021-12-02. Available from: `https://github.com/openpnp/openpnp/wiki/Bottom-Vision`

[26] J., H.; J., S. Detecting Rotated Objects Using the NVIDIA Object Detection Toolkit. 2020, accessed: 2021-12-02. Available from: `https://developer.nvidia.com/blog/detecting-rotated-objects-using-the-odtk/`

[27] Veit, A.; Matera, T.; et al. COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images. 2016.

[28] Jiang, Y.; Zhu, X.; et al. R2CNN: Rotational Region CNN for Orientation Robust Scene Text Detection. 2017.

[29] Ahmad, M.; Abdullah, M.; et al. Small Object Detection in Aerial Imagery using RetinaNet with Anchor Optimization. In *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, 2020.

[30] How RetinaNet works? Accessed: 2021-12-02. Available from: `https://developers.arcgis.com/python/guide/how-retinanet-works/`

[31] RetinaNet: The beauty of Focal Loss. Accessed: 2021-12-02. Available from: `https://towardsdatascience.com/retinanet-the-beauty-of-focal-loss-e9ab132f2981`

[32] Yang, X.; Yang, X.; et al. Learning High-Precision Bounding Box for Rotated Object Detection via Kullback-LeibleDivergence. 2021.

[33] Yang, X.; Yang, J.; et al. SCRDet: Towards More Robust Detection for Small, Cluttered and Rotated Objects. 2019.

[34] Yang, X.; Yan, J.; et al. SCRDet++: Detecting Small, Cluttered and Rotated Objects via Instance-Level Feature Denoising and Rotation Loss Smoothing. 2020.

[35] Openpnp. DetectCircularSymmetry. Accessed: 2021-12-02. Available from: `https://github.com/openpnp/openpnp/wiki/DetectCircularSymmetry`

[36] GPU-Jupyter. Jul 2021, accessed: 2021-12-02. Available from: `https://hub.docker.com/r/cschranz/gpu-jupyter`

[37] Infineon Technologies AG, 81726 Munich, Germany. *Recommendations for Printed Circuit Board Assembly of Infineon QFN Packages*. 5 2012.

[38] STMicroelectronics. *Mounting instructions for SMD (surface mounting device) packages*. 5 2019.

Appendix A

Acronyms

P&P Pick and Place

SMT Surface-mount technology

NN Neural Network

CNN Convolutional Neural Network

R-CNN Region-based Convolutional Neural Network

PCB Printed Circuit Board

CAD Computer-Aided Design

OCR Optical Character Recognition

K-NN K-Nearest Neighbor

FAST Features from Accelerated Segment Test

SSD Single Shot Detector

CNC Computer Numerical Control

CC Connected Components

RPN Region Proposal Network

NMS Non-maximum Suppression

IoU Intersection-over-Union

FPN Feature Pyramid Network

FRM Feature Refinement Module

FL Focal Loss

KLD	Kullback-Leibler Divergence
GWS	Gaussian Wasserstein Distance
AP	Average Precision
mAP	mean Average Precision

Appendix B

Contents of enclosed SD card

- `readme.txt` .... file with CD contents description
- `RotationDetection` .... directory with rotation detector source code
  - `dataloader` .... directory with scripts for data preprocessing
  - `libs` .... directory with helper functions and training scripts
    - `models` .... directory with scripts for building the models
  - `r3det` .... directory with training and testing script for r3det detector
  - `r3det_kl` .... directory with training and testing script for r3det_kl detector
  - `retina` .... directory with training and testing script for RetinaNet detector
  - `scrdet` directory with training and testing script for SCRDet detector
- `OpenPNP` .... directory with the source codes for OpenPNP
  - `gui` .... directory with gui implementation
  - `machine` ..... directory with machine interface implementation classes
  - `model` .... the directory with data classes
  - `stages` .... directory with CVStages
  - `config` .... directory with xml configurations
- `text` .... thesis text and LaTeX source directory
  - `thesis.pdf` .... thesis text in PDF format
  - `thesis` .... directory of LaTeX source codes of the thesis