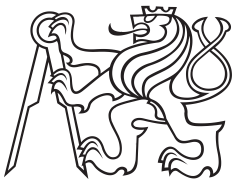
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Detection and Pose Determination of a Part for Bin Picking

Bc. Roman Sushkov

Supervisor: RNDr. Miroslav Kulich, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Roman S u s h k o v

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Detection and Pose Determination of a Part for Bin Picking

Guidelines:

1. Get acquainted with computer vision and image processing methods for object detection.
2. Design and develop a method for detection of a part (strut bracket) in a bin and determination of its position based on a camera image.
3. Design and realize a software framework for bin picking with a robotic arm.
4. Evaluate experimentally the developed framework in a real setup and discuss obtained results.
5. Discuss possible extension of the developed method for more complex parts.

Bibliography/Sources:

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems (pp. 1097-1105), 2012.
[2] Xie, Y., & Ji, Q.: A new efficient ellipse detection method. Pattern Recognition, 16th International Conference on (Vol. 2, pp. 957-960). IEEE, 2002.
[3] Dalal, Navneet, and Bill Triggs: Histograms of oriented gradients for human detection. Computer Vision and Pattern Recognition, IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.

Diploma Thesis Supervisor: RNDr. Miroslav Kulich, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 6, 2017

Acknowledgements

I would like to thank my supervisor for his guidance, my family for their support and my alma mater CTU for giving me interesting problems to solve.

Declaration

Author statement for diploma thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date .....

Signature .....

Abstract

This thesis discusses the visual bin picking task, which is the task of sequential unloading a bin one part at a time using a camera as a primary source of information. The semi-structured variant of the bin picking task is considered. In this thesis, a solution for this problem that is based on learning the appearance model of a part using convolutional neural networks is proposed. Thus, no hard-coded geometry of a part is required. The models in the developed system predict the poses of the parts and detect occlusions. The proposed system has been implemented and tested with a metallic strut bracket. The experiments have shown that the achieved estimated success rate of the system is 95 % of acquiring attempts.

Keywords: bin picking, convolutional neural networks, autonomous manipulation, pose estimation, occlusion recognition

Supervisor: RNDr. Miroslav Kulich, Ph.D.

Abstrakt

Tato diplomová práce se zabývá tématem vizuální úlohy vybírání, ve které se postupně vybírají součástky z bedny. Částečně strukturovná varianta této úlohy je uvažována. V této diplomové práci se navrhuje řešení této úlohy, které je založeno na konvolučních neuronových sítích. Díky tomu, programová specifikace geometrie součástky není nutná. Návrhovaný systém odhaduje pozici a orientaci součástky a detekuje překrytí součástek. Systém byl implementován a otestován s použitím kovové součástky. Odhadovaná kvalita systému je 95 % úspěšných pokusů vybrání.

Klíčová slova: úloha vybírání, konvoluční neuronové sítě, autonomní manipulace, odhad polohy, detekce překrytí

Contents

Chapter 1

Introduction

Presently, a large body of research is focused on automating various tasks in manufacturing in order to reduce the production cost. One of these tasks is bin picking, which is the task of autonomous sequential unloading a bin part by part using a robotic arm. Developing methods for bin picking is important for manufacturing and warehouses for various reasons: the task may be tedious for human workers; dangerous, if the parts are sharp; or the operation must be performed in an area where the access to humans is restricted. Various systems for bin picking have already been implemented before, and they typically use a combination of 3D and 2D vision systems and CAD models of parts for pose recognition. On the other hand, the research on using simple setups for bin picking—that do not use a 3D vision system or explicit CAD models of the parts, and employs only cheap components and materials—is sparse. By reducing the complexity of the setup, it becomes simpler to program, calibrate and maintain the system.

Although bin picking systems that use 2D vision exist, they generally deal with simple scenarios, where the parts are located on a planar surface, which simplifies the problem of pose estimation. This kind of a system is seldom used for complex scenarios where the parts may be arbitrarily oriented and partially occluded by each other. The goal of this thesis is not to create the most advanced system for bin picking, but rather to design a framework that uses cheap and simple components, such that the developed system would work in a realistic scenario, and that could be used for a wide range of parts.

Various implementations of bin picking systems use different hardware components. A 3D camera, a rangefinder or a digital camera is used for sensing. Part acquisition may be carried out using a vacuum or magnetic pads. The bin may contain objects of one or various types, and the degree of randomness of the objects is divided into three groups: structured, semi-structured, random. In structured bin picking, the objects are located in a structured array, and their positions are predictable. This is the easiest variant of this task since the part recognition is simplified. In random bin picking, the parts may have different orientations, may overlap, and their positions are generally unpredictable. This is the most complex variant since it is necessary to build a more complex model of the parts to cover different scenarios. In semi-structured bin picking, positions and orientations of the

parts may also be random, but with certain constraints, for example, the rotation of the part may be constrained to a certain range, or occlusions may be not present.

Various vision- and range finder-based bin picking frameworks have been developed and described in the literature. Early methods, such as [BH86], used a 3D range scanner for object localization. This system uses geometric features, such as edges and discontinuities, found in the scans of the parts, to generate the hypothesis about the pose of the part. A visual bin picking system with a stereo camera was introduced in [RK96]. Here, visual cues are used to recognize the pose of complex objects. Visual features, such as circles, are utilized for recognizing the position and orientation of the object. This approach has a disadvantage that it is necessary to define these visual clues and their 3D geometric representation.

In a more recent paper [LTV+12], an object detection and pose estimation method based on shape matching with a known 3D model using a multi-flash camera is proposed. Multi-flash camera (MFC) is a digital camera surrounded by multiple (8 in [LTV+12]) LEDs. This device is used to determine depth edges by capturing multiple images (one for each LED turned on). The depth edges are determined by comparing the difference in shadows in different images. The shape matching is done using Fast Directional Chamfer Matching, which is also proposed in [LTV+12].

Range sensors are divided into two groups: active, that use a separate light source, and passive, that do not emit additional light. Active sensors can roughly be divided into two groups: time of flight (ToF) and structured light sensors. In ToF sensors, the light is emitted, and the time between the emission and detection is measured, from which the distance is determined. In structured light sensors, a dot or lines pattern is emitted, and the shape is determined from the distortion of the pattern. Passive sensors usually use a stereo camera. In this case, the range image is formed by capturing two images and finding the correspondence between various parts of the images. The distances are then determined using triangulation.

Some of the active range sensors have a problem determining the distance in the outdoor environment because of the presence of the natural sunlight. The problem occurs when the intensity of the sunlight is higher than the intensity of the emitter, and it becomes difficult to distinguish the emitted light. Laser emitters are usually used in that case because they have a higher power. A similar problem arises when the measured surface is shiny, because the light reflected from other sources may have a higher intensity than the emitter. Multiple reflections is another issue that may cause difficulties. This is a problem which arises when the emitted light is reflected back to the detector after being reflected from multiple surfaces, and the device cannot determine which reflection is the correct one. The main problem of passive sensors, or stereo camera, is that it is difficult to determine correspondences between the parts of the scene. This is especially important for reflective surfaces, which may have a different appearance from different camera positions.

These problems raise the question whether these kinds of sensors may be

avoided altogether by using a simple camera and learning a model of how the part looks under different views. In this thesis, I propose a solution for this problem. The bin picking variant that is considered here is a semi-ordered bin picking with possible occlusions.

1.1 Computer vision and convolutional neural networks

Recently, the field of computer vision saw major breakthroughs in the classification and detection tasks due to the developments of deep learning and convolutional neural networks (CNN) [KSH12, HZRS16]. These advances led to increased employment of convolutional neural networks for solving the industrial [BDTD+16] and academical [SHM+16] problems. Despite the popularity of CNNs, they have not yet been widely applied for the bin picking task. Some of the applications have been described in [ZYS+16, WL15, LPKQ16].

In [ZYS+16], CNN is used to segment an object in an RGBD image (a color image with an additional depth channel), which is followed by shape matching of the segmented part and the known 3D model, which is carried out with iterative closest point [BM92] and 3DMatch [ZSN+16]. Another application is presented in [WL15], where the task of 3D pose estimation is divided into two parts: first, a descriptor of an image patch is generated; and second, the nearest neighbor search is used to determine the orientation and class of the object presented in the image. The descriptor is generated in such a way that similar image patches are described by similar vectors (whose absolute difference is a small value), and this allows creating a database of images of objects observed from different positions, and determine the orientation of an object in a test image as the nearest neighbor (by comparing the descriptor vectors) in the database. The task of creating a descriptor for an image patch (RBG or RGBD) is handled by a convolutional neural network. An advanced grasping system is described in [LPKQ16]. Here, learning of the grasping behavior is handled in an end-to-end manner, without an intermediate representation of the object pose. Learning the grasping model in this way is conceptually elegant, but is unfeasible for many researchers because of the cost of the equipment (multiple robotic manipulators are used for learning).

CNNs have been also used for pose estimation of a human body from ordinary color images. In [TS14], a framework for pose regression from raw image is proposed, where a deep neural network is presented with an image patch with a human, and it predicts the absolute image coordinates of the joints (the joints coordinates are predicted simultaneously). In order to increase the precision of the estimation, a cascade of networks is established, in which the precision is improved with each step. Another application has been described in [LC14], where the joints are not predicted in absolute image coordinates, but rather in 3D coordinates.

The main difference between estimating pose of human limbs and a rigid

object is that the human pose is described as an articulated structure containing multiple parts, whereas the pose of a rigid object is predicted as a whole. For that reason, in human pose estimation, it is usually not necessary to determine the orientations of the parts, in contrast to rigid body pose estimation.

1.2 Motivation

This thesis was motivated by a problem that Škoda Auto a.s. faced in one of its factories. The task of unloading a bin with metallic strut brackets at the rate of one part per minute is tedious for human workers, and is an inefficient use of human resources. This task has been posed to several research groups, one of which was our group of the Czech Technical University in Prague (CTU). The specification of this task stated that the minimum success rate of the system is autonomous acquiring of 70 % of the parts.

Solving this task required collaborative effort from several students and staff of CTU in order to create a functional bin picking system, which required building and connecting various hardware and software components. My commitment to this project was building the 'brain' of this system, which is the visual recognition system, and programming the behavior of the robotic manipulator. In this thesis, the developed components are described, implemented and tested.

1.3 Thesis structure

The thesis is organized as follows. A short introduction to the selected computer vision and machine learning methods that are used in this thesis is given in Chapter 2. The hardware components and the physical setup of the system is described in Chapter 3. The main part of this thesis—the visual recognition system—is introduced in Chapter 4. The control algorithm which defines the behavior of a robotic manipulator is presented in Chapter 5. The software implementation of the framework and the experimental results are described in Chapter 6. The achieved performance and potential improvements are discussed in Chapter 7.

Chapter 2

Selected methods of computer vision and machine learning

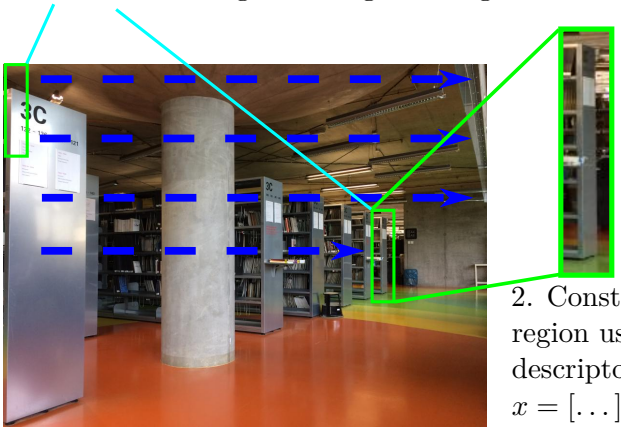
Computer vision is a part of computer science that deals with image analysis for obtaining high-level knowledge of the image content. Computer vision methods play an important role in the developed framework, as images are used to determine the current state, and this estimate is then used to guide the robot for bin picking.

Machine learning is widely used in computer vision. It is a technology that gives computers the ability to make decisions by learning an appropriate model of a given task based on data instead of explicitly programming the behavior of the system. Methods of machine learning are often used in the field of computer vision, because explicitly programmed models usually get quite complex. Machine learning is used to solve many tasks, but perhaps the most common ones are classification and regression problems. In both tasks, a specific attribute is predicted, and the difference between these problems is that in classification, this attribute is discrete, whereas in regression, it is continuous. A discrete value usually represents a class (hence the name classification), and the continuous value may represent a wide variety of concepts: a price, a position, a score. A sample, whose attribute is predicted, is usually represented by a vector of values called a feature vector. A feature vector contains multiple values that describe the sample. For example, in the famous iris dataset [Fis36] the feature vector contains the following measurements of iris flowers: sepal length, sepal width, petal length, petal width. In a more complex case of images, an image itself may be a feature vector.

In machine learning, training is the process in which the parameters of the model are learned with the goal of making accurate predictions. In classification and regression tasks, learning requires a collection of ground truth data, which is a collection of items (X, Y), where X is a feature vector of the sample and Y is the desired output. Training a neural network is described below in this section.

Object detection is a common problem in computer vision. This is the task of finding a region in an image where the target object is localized. The region is usually defined by a rectangular area, which is also commonly called a bounding box. A common approach is to use a sliding window method,

1. Extract local regions using a sliding window



2. Construct a feature vector of the region using a local feature descriptor
$x = [\dots]^T \in R^n$
3. Use a classifier to predict the class of the region
$\hat{y}(x) = C_{Shelf}$
4. Repeat for various window sizes for scale invariance

Figure 2.1: Object detection using the sliding window approach.

where a virtual window is moved across the various parts of the image. This approach is illustrated in Fig. 2.1. By examining individual local regions (also called image patches) of the given image, we can determine the locations where the object of interest is present. This is usually done by extracting a feature vector of this region and then using a classifier to determine whether this region is a part of a background or it contains the object of interest. A popular method for extracting a feature vector from an image patch is histogram of oriented gradients, which is described below. A more recent version of the sliding window approach is based on convolutional neural networks and it does not have a component for extracting a feature vector. Instead, a convolutional neural network may be trained to classify the image patch directly. However, this approach is not used in this thesis, because in practice, using convolutional neural networks may introduce additional difficulties in training and deploying.

2.1 Histogram of oriented gradients

Histogram of oriented gradients (HOG) [DT05] is a feature descriptor that is often used for detection in images. The idea of the descriptor is shown schematically in Fig. 2.2.

First, the gradient of the image patch is determined. In [DT05], a simple method that is based on filtering the image with kernels is used:

$$h_1 = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}. \tag{2.1}$$

The directions of the gradient (angle $\alpha \in [0°, 180°)$) are determined. The image is divided into small adjacent cells (e.g. of size 8x8), and for each of them, the histogram of gradient directions is determined. This histogram contains a compressed representation of a cell. The feature vector of the image patch is then formed as the concatenation of the individual cells histograms. In order to improve the accuracy, the blocks are introduced, which are larger regions of the image and that contain multiple cells. The histograms are then normalized with respect to the blocks. For a more detailed description of the algorithm, see [DT05].

In order to build a detector using the HOG feature descriptor, the sliding window approach is used. A window is moved across different regions of the image, and for each of the regions, the feature vector is determined. A classifier is used to determine whether the feature vector does or does not come from a region that contains the object. A linear SVM [CV95] is used for classification.

2.2 Artificial neural networks

Artificial neural networks are computational models that are widely used in artificial intelligence. They are composed of a collection of interconnected artificial neurons. A neuron (see Fig. 2.3) is a computational unit that is defined by multiple inputs, one output, activation function and a vector of parameters called weights. The output (also called the activation) of a neuron is determined using the following equation:

$$y = g(\mathbf{w}^\top \mathbf{x}), \tag{2.2}$$

where y is the output (which is a scalar value), $\mathbf{w}$ is the vector of weights, $\mathbf{x}$ is the input vector and g is the activation function. The purpose of the activation function is to introduce non-linearity into the model. Without a non-linear component, a neural network could be replaced by a simple matrix multiplication, which does not provide enough complexity for solving many problems.

Various activation functions have been described in the literature. Historically, one of the most common activation functions was a sigmoid, for example, the hyperbolic tangent function:

$$\sigma_{\tanh}(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.3}$$

Convolutional neural networks usually use another activation function: rectifier linear unit (ReLU), which is defined as

$$\sigma_{\mathrm{ReLU}}(x) = \max(0, x). \tag{2.4}$$

These activation functions are illustrated in Fig. 2.4.

Artificial neural networks are formed as a collection of artificial neurons. Although there are many architectures, here I will focus on a classical multilayer feed-forward network, which is shown in Fig. 2.3. In this type of neural

network, the neurons are organized in layers. There is one input layer, one output layer, and one or multiple hidden layers. In each layer i, the inputs of the neurons are connected to the outputs of the neurons in layer $i-1$ (except the input layer, which is the input). In this architecture, the neurons activations are

$$x_{ij} = g(\mathbf{w}_{ij}^{\top}\mathbf{x}_{i-1}), \tag{2.5}$$

where x_{ij} is the activation of the j-th neuron in the i-th layer, $\mathbf{w}_{ij}$ is the vector of weights of this neuron, $\mathbf{x}_{i-1}$ is the vector of activations of neurons in the layer $i-1$, and g is the activation function.

This model is used for forward stimulation of the network: by setting values of neurons in the input layer, the activations of other neurons in the network are computed one layer at a time, until the activations in the output layer are determined. The result of this computation is the prediction of the neural network, which is the activations of the neurons in the output layer. This result is determined only by the input layer and the parameters of the model (the weights). In order to make the prediction useful, it is necessary to learn these parameters, which is done by backpropagation.

Backpropagation [BDD63] (backward propagation of errors) is a method of learning the parameters of a model by using a forward stimulation. First, the input is presented to the network, and the output is computed, as has been described above. After that, the prediction is compared with the desired output and the error is computed using the loss function. Then, this error is propagated back from the output to the input. As the result of this, contribution to error is determined for each neuron, and this is used to correct the weights using an optimization technique (in the simplest form, gradient descent [BDD63], more recently, Adam optimizer [KB14]).

Two common types of tasks usually solved using artificial neural networks are classification and regression. From the perspective of learning the model, the difference between these two tasks is in the loss function. For the regression task, a squared error is used:

$$V(\hat{y}, y) = (\hat{y} - y)^2, \tag{2.6}$$

while the loss function that is usually used in the classification tasks is the cross entropy loss:

$$V(\hat{y}, y) = -y \ln(\hat{y}) - (1-y)\ln(1-\hat{y}). \tag{2.7}$$

In practice, neural networks are often trained in mini-batches, which means that N different inputs are presented to the networks, and the loss is computed as the average of individual errors:

$$\bar{V} = \frac{1}{N}\sum_{i=1}^{N}(V(\hat{y}_i, y_i)). \tag{2.8}$$

Although the standard feed-forward architecture is useful in situations where the number of input dimensions is low to medium, it was found that a

large input space leads to a complex model with a large number of parameters, and because of that, the training becomes more difficult. A large input space is commonly found in images, where each pixel is a part of the input vector. This problem has led to the development of convolutional neural networks.

A convolutional neural network is a type of a neural network that uses a special type of connectivity pattern between the neurons. In a regular feed-forward neural network, the neurons are organized in fully-connected layers, where each neuron in layer i is connected to each neuron in layer $i-1$. In convolutional neural networks, two new layer types are introduced: a convolutional layer and a pooling layer.

A convolutional layer is a 3D array whose size is specified by width, height, and depth. The input layer of CNN is usually an image, hence the width and the height of the input layer are defined by the size of the image, and the depth is either three (for a color image) or one (for a grayscale image). The later (hidden) layers are usually deeper, layers with the depth of 32 or 64 are not uncommon. This number corresponds to the complexity of the model: by selecting a larger number, more features are learned. A two-dimensional slice of a layer with constant depth is called a depth slice. For example, a color image has three depth slices, each for a color channel. Convolutional layers are no longer fully-connected, which means that each neuron in layer i is connected to a subset of neurons in the previous layer, which is called the receptive field (shown in Fig. 2.5). A receptive field is a rectangular region of neighboring neurons, whose position is equal to the position of the neuron. The functionality of a neuron in this kind of layer is the same as in regular neural networks: it computes the dot product of its weights and the outputs of the connected neurons in the previous layer. In convolutional layers, the parameters of the neurons are shared within a depth slice, which reduces the total number of parameters and makes it easier to train the network. The name convolutional layer comes from the fact that by sliding a neuron in layer i and computing the dot products of the kernel and the neurons in the receptive field, we essentially compute the convolution of layer $i-1$ and the kernel, which is defined by the weights of the neuron.

A pooling layer is a layer that performs the downsampling operation. Two types of pooling layers are usually used: max-pooling and average-pooling. This layer is usually placed after a convolutional layer to reduce its size: it essentially slides a window with a constant step size (called stride) and performs an aggregating operation on the contents of the window (finds the maximum or computes the average). For example, by using a window size 2x2 with stride 2, the layer is reduced 4 times.

To illustrate the ideas presented above, a general architecture of a convolutional neural network is shown in Fig. 2.6.

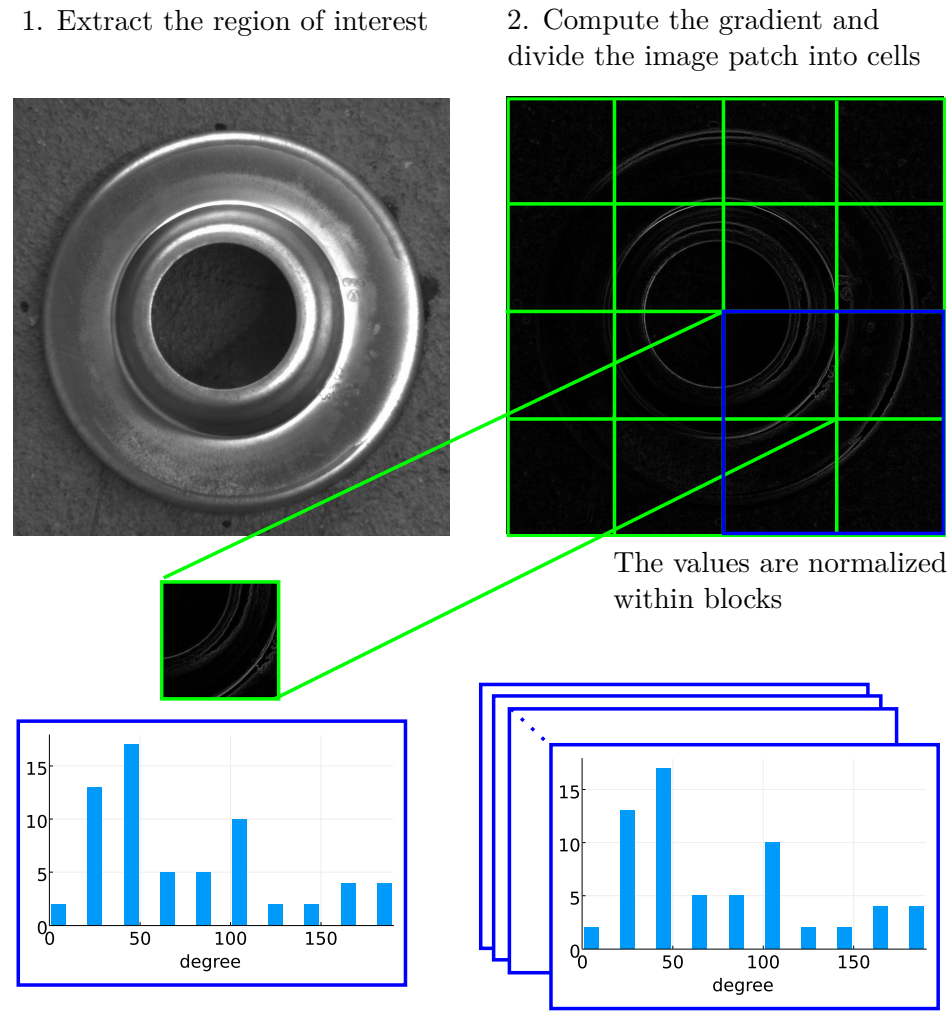


Figure 2.2: Histogram of oriented gradients.

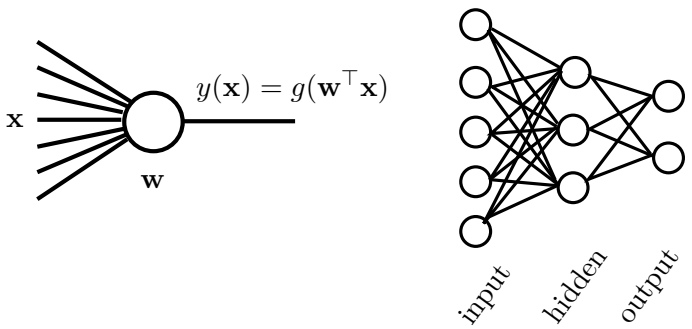


Figure 2.3: Neuron (left) and feed-forward artificial neural network (right).

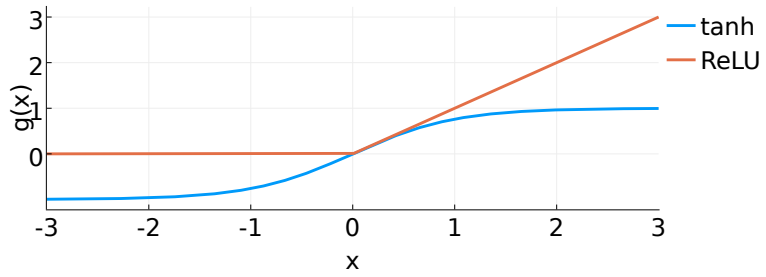


Figure 2.4: Activation functions that are used in artificial neural networks.

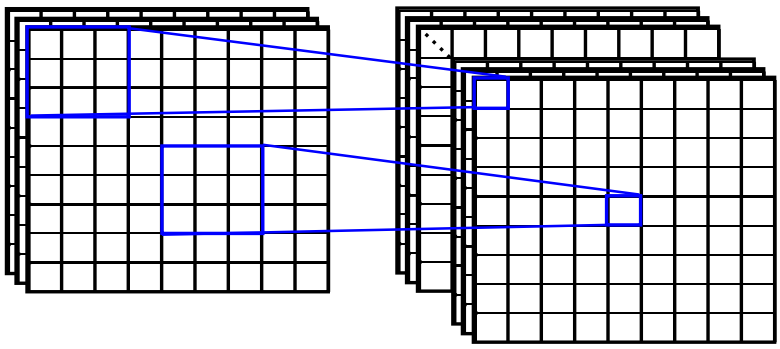


Figure 2.5: Receptive field of size 3x3 of two neurons in a convolutional layer.

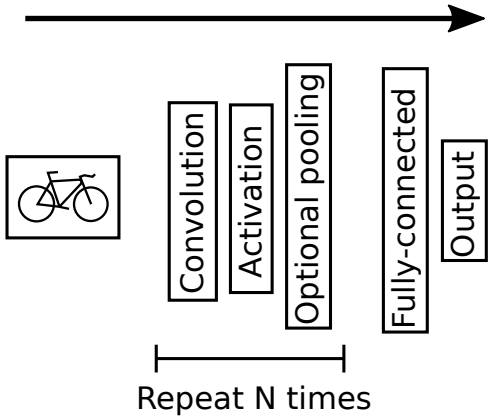


Figure 2.6: A general architecture of a convolutional neural network.

Chapter 3

System setup

The physical realization of the bin picking system is shown in Fig. 3.1. Here, a robotic manipulator has acquired a part from a pile in a bin and now it moves the part to the receiver. A gripper is mounted on the last link of the manipulator, and a closer view of the gripper is shown in Fig. 3.2. The gripper holds four vacuum pads that are used for acquiring the parts, a camera for visual detection and navigation, and a circuit board with light-emitting diodes (LEDs) for illumination. A pile of parts in the bin is shown in Fig. 3.4.

A visual recognition system is used in this framework to infer the state of the system, which is determined by the poses of the parts in the bin and the occlusions between them. Including the occlusions into the state description may seem redundant, because ideally, it is possible to determine which objects are occluded by knowing the poses. However, in practice, an estimation error is always present, and this redundancy helps to minimize the effect of the errors. Object pose is composed of the position and the orientation of the part relative to the camera or another coordinate system, as shown in Fig. 3.6b. Position p of the object origin is the vector from the camera to object origin

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.1}$$

and the part normal determines the orientation of the part.

$$n = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}, \ ||n|| = 1. \tag{3.2}$$

The part (shown in Fig. 3.5) that has been used in this framework is a rotationally symmetric metallic strut bracket with diameter 161 mm. The large flat disc surface on the top of the object is used as a contact surface for grasping by the vacuum pads. The part coordinate system is shown in Fig. 3.6a: the origin of the coordinate system coincides with the part origin, and the z axis is defined by the part normal. The orientation of the x axis can be selected arbitrarily, since the part is rotationally invariant. From this definition, we can see that this part has 5 degrees of freedom and that infinitely many coordinate systems may be assigned to a part. Despite this

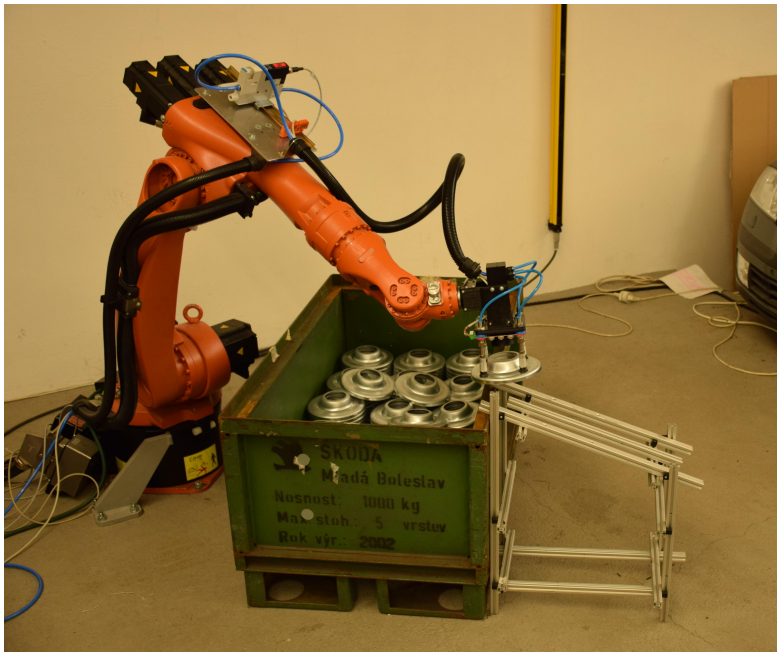


Figure 3.1: The bin picking system. The robot has acquired the part and moves it outside the bin.

ambiguity, this formulation is useful, and it is used in this framework for pose determination.

The robotic manipular that is used in the system is Kuka KR 5 arc, which is a 6-axis industrial manipulator intended for objects handling and welding. For the safety purposes, the manipulator speed has been set to 30% of the maximum speed during all operations with the robot.

Four vacuum pads Festo are used for acquiring parts. In order to acquire a part, the pads are placed on the flat surface on the top of the part, as shown in Fig. 3.3. Experiments have shown that the maximum allowed error of gripper position is ± 6 mm in the xy plane of the part and the maximum inclination of the normal is $\pm 7°$. When this limit is exceeded, the grasping becomes unreliable, and because of that the precision requirements of the pose estimation component are relatively high.

The camera that is used in this framework is Smartek GC2591MP with sensor CMOS Aptina MT9P031 1/2.5 and resolution 2592x1944 pixels. The used lens is Computar M0814MP2 with focal length 8 mm and the angle of view 67°.

The reason for using dedicated LEDs is to suppresses both ambient and specular components from external light sources. Ambient light may potentially complicate the configuration of the system, since the external conditions, such as daylight illumination, may change, and it would be necessary to address this change. The specular component is also harmful, since it produces a greater variation of the part appearance, and because of that, a larger training set would be required.

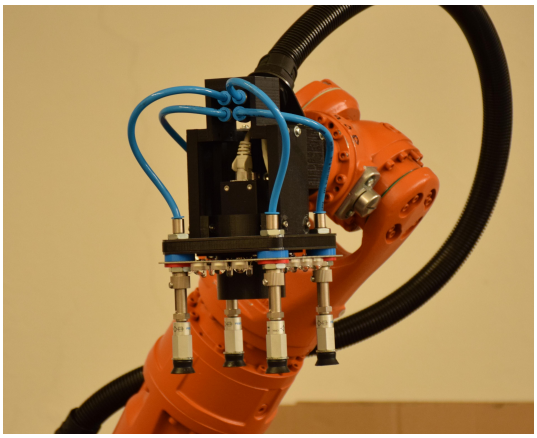


Figure 3.2: A close look on the gripper.

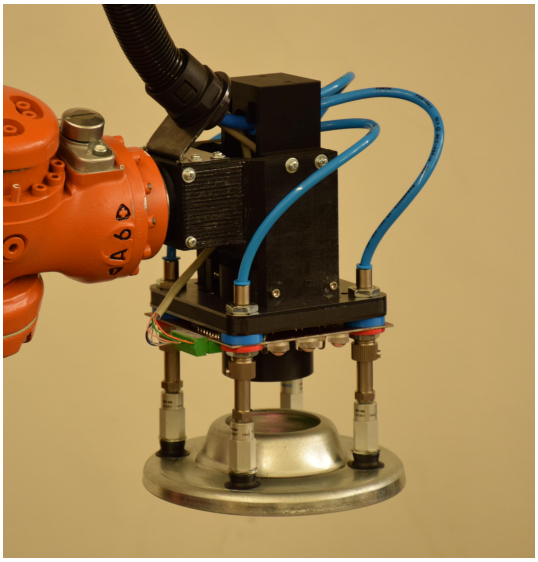
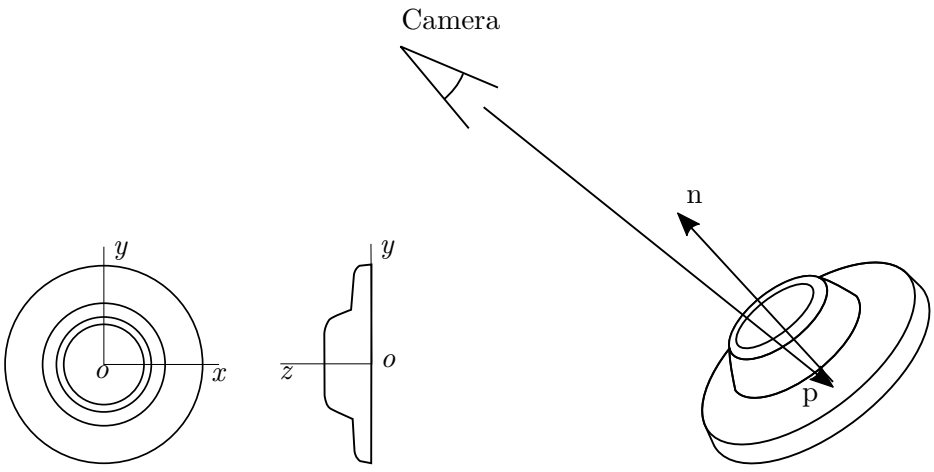


Figure 3.3: In order to acquire the part, the gripper is placed on the flat surface of the part.

Figure 3.4: A pile of parts in the bin.

Figure 3.5: The part that is used in this system (a metal strut bracket).



(a) : Object coordinate system **(b) :** Position and orientation of a part

Figure 3.6: The part coordinate system, position and orientation of a part

Chapter 4

Visual recognition system

The visual component forms a major part of this work, because the information about the state of the system is inferred from images. The developed image analysis framework is shown in Fig. 4.1. The framework contains several components. First, an image is captured and parts that are present in the scene are detected. The detections determine the areas of the image (bounding boxes), where the objects have been recognized. Second, for each detection, an image patch is extracted, which is used to determine the pose of the detected object. Third, the occlusion recognition component determines which objects are occluded, and thus cannot be grasped. This information is used to control the position of the robot, and, after multiple iterations of adjusting the position of the camera, grasp and remove an object from the bin.

All the components used in this framework are trainable, which is in many cases a desirable property, because this allows easier generalization to a new problem. The only geometrical information assumed in this framework is that this object is rotationally symmetric; the particular shape of the object is learned, rather than programmed. No other geometrical features such as ellipses, edges or curves are used for analysis, hence, reusing the implementation for a different object should theoretically only require adding new data and retraining the components.

Application of methods of machine learning, instead of hard-coded geometric features, is also motivated by the fact that the surfaces of the objects considered in this thesis are metallic, and it is difficult to model the visual features that appear on the surface under different lighting conditions. An example of this effect is shown in Fig. 4.2, where the appearance of one part viewed from different positions is evidently different: the object on the left is brighter.

Trainable components learn these features and do not require additional effort from the programmer to cover the varying conditions. The lack of control from the programmer may be a downside of learning in the situations when the performance of the models is insufficient or it is difficult to gather data. It may also require additional effort to debug the learnable components, compared to hard-coded models.

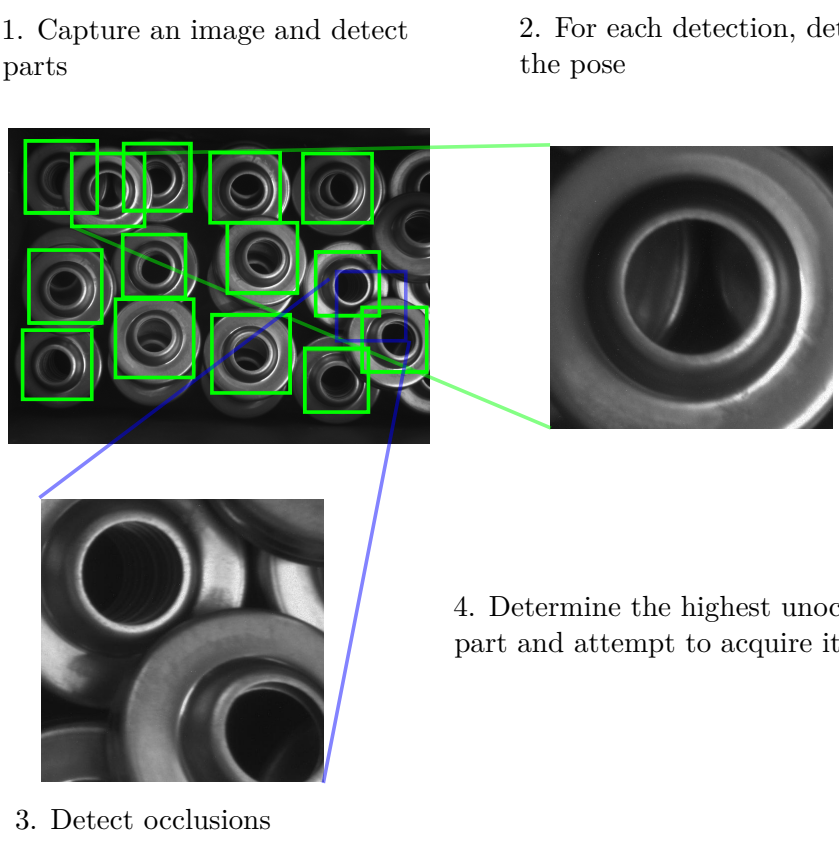


Figure 4.1: The developed framework consists of a part detector, a pose estimator and an occlusion detector.

4.1 Object detection

Object detection component is the first part of the framework. The goal of the detector is to examine a raw image captured by the camera and to find the locations where the objects are present. These locations are represented by rectangular regions of the image, also called bounding boxes. The detector has been realized using histogram of oriented gradients (HOG). This choice has been governed by the fact that this is a model that is easy to train and that it often performs well for a range of different problems. Indeed, the training set contains only 1058 training images (53 positive and 1005 negative). The parameters of the HOG detector are: block size 16x16, cell size 8x8, image patch size 64x64.

An example of detections is shown in Fig. 4.3, where the bounding boxes are drawn around the detected objects. The detections do not determine the poses of the objects in the image, but rather they determine in which parts of the image the objects are located. Image patches extracted from these locations are used in the next stages of the framework to estimate the poses

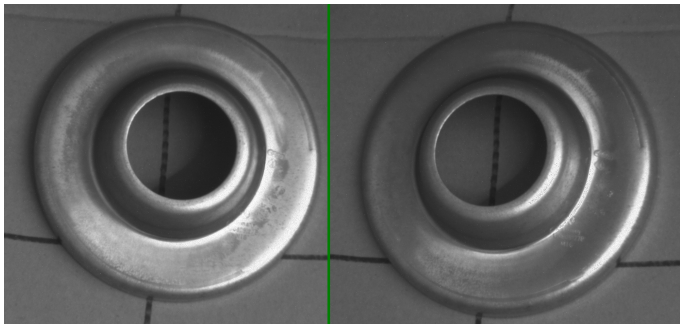


Figure 4.2: An example of how the appearance of the object changes by changing the view angle.

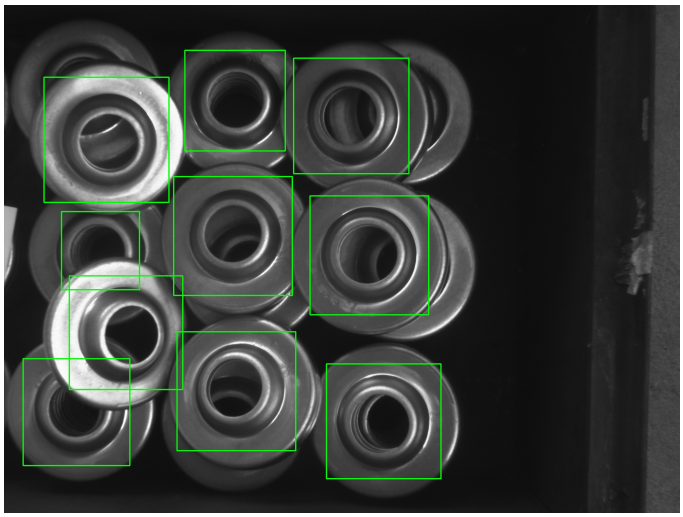


Figure 4.3: Bounding boxes around detected parts.

of the objects. As shown in Fig. 4.3, some objects overlap, and some of the overlapped objects are detected in this stage. These objects are still mostly detected by this components, however, because they cannot be acquired, they are recognized in the occlusion recognition component.

4.2 Pose estimation

After identifying the regions of the captured image that contain an object, image patches are extracted for pose estimation. An example of such a patch is shown in Fig. 4.4. The pose, which includes the orientation and the position of the object expressed in the camera coordinate system, is determined using two different models, which are described in Sections 4.2.1, 4.2.2.

The position of the object is a vector of the coordinates of its base:

$$p_o = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix}. \tag{4.1}$$

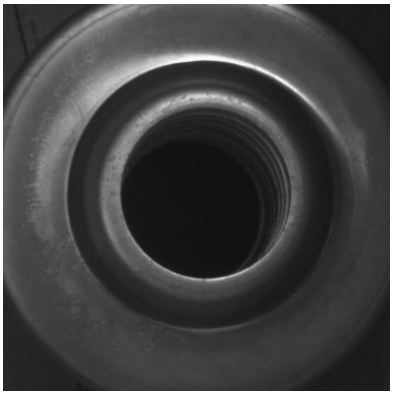


Figure 4.4: An example of an image patch.

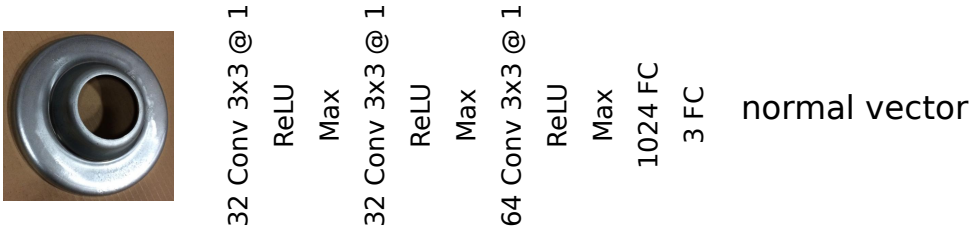


Figure 4.5: CNN architecture for object normal vector estimation.

Since the object is rotationally symmetric, its orientation is fully specified by its rotation axis defined by the vector

$$n_o = \begin{bmatrix} n_{xo} \\ n_{yo} \\ n_{zo} \end{bmatrix}. \tag{4.2}$$

4.2.1 Estimation of the object normal vector

Precise estimation of the orientation of the object is necessary for the picking task in an unstructured environment. Incorrect alignment with the object during contact with the gripper may lead to an unsuccessful grip, or even damaging the gripper or other robot parts which may be difficult or expensive to repair.

In this thesis, I propose a regression model based on CNN for the task of part normal estimation. The structure of this network is shown in Fig. 4.5. The input is an image patch containing an object resized to 64x64 pixels. It is followed by three convolutional layers with the ReLU activation function, followed by max pool layers. So many max pool layers are used here to minimize the number of parameters of the model, which is needed because of the sparsity of data. The last layer of the network is a fully-connected layer with 3 neurons, whose output is the predicted value of the object normal vector n_o.

The size of the input (64x64) has been selected as a trade-off between the level of detail of the image and the number of parameters of the network.

Having a larger input would capture the object with more detail, but it also would increase the total number of weights of the network, which would require more data and increase the number of iterations for training.

Training

The pose data is generated by capturing an image of a part, whose pose is known, from a known position of the camera (both poses are in the base coordinate system). Since both poses are known, the system is fully determined. However, in order to train this model, it is necessary to express the part pose in the camera coordinate system. This transformation is described in this section.

In order to train the model, it is necessary to derive the coordinates of the part normal with respect to the camera coordinate system using the known values: part pose and camera pose. The homogeneous coordinates of the object normal expressed in the object coordinate system are

$$\bar{p}_{zo} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \tag{4.3}$$

and the coordinates of the object origin are

$$\bar{p}_{0o} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.4}$$

These vectors can be expressed in the gripper coordinate system:

$$\begin{aligned} \bar{p}_{zg} &= T_{go}\bar{p}_{zo}, \\ \bar{p}_{0g} &= T_{go}\bar{p}_{0o}, \end{aligned} \tag{4.5}$$

where T_{go} is the transformation matrix from gripper to object. Object normal expressed in the gripper coordinate system is the difference between these two vectors in non-homogeneous coordinates:

$$p_{ng} = p_{zg} - p_{0g}, \tag{4.6}$$

where p_{zg} and p_{0g} are non-homogeneous versions of $\bar{p}_{zg}$ and $\bar{p}_{0g}$. Eq. 4.6 is used to determine the value of the normal vector from the pose of the camera and the pose of the object.

A loss function used for optimization is the mean squared error function:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{4.7}$$

where $\hat{y}$ is the predicted value, y is the ground truth and n is the batch size. Adam optimizer with learning rate 0.001 was used, 5000000 training iterations were required and dropout rate 0.5 was used for regularization.

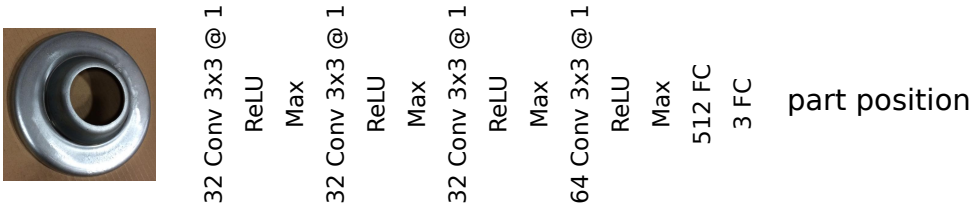


Figure 4.6: CNN architecture for object position estimation.

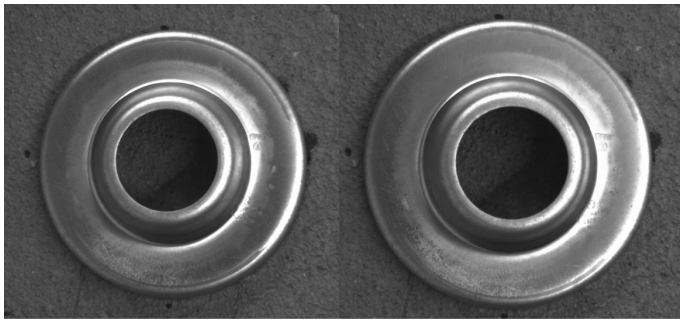


Figure 4.7: Two image patches of the same part look similarly from different distances.

Convolutional neural networks typically require a large dataset for training, so artificially augmenting the dataset is beneficial. In order to increase the size of the dataset, the image patches are rotated by 90, 180 and 270 degrees and flipped (upside-down), yielding 8 times more data samples.

4.2.2 Estimation of the object position

In the previous section, the model for object normal prediction has been described. The model is built in such a way that it predicts the normal vector directly: the output value of the neural network *is* the value of the normal vector. However, using this approach for prediction of the part position causes several difficulties. For example, consider distance prediction. The first problem is that it is difficult to see the difference in the distance between two image patches. For example, in Fig. 4.7, two image patches are shown. The object on the left-hand side is approximately twice as far as the object on the right-hand side, but it is difficult to distinguish which one is more distant after the images have been resized. A possible way around this problem is to use the information about the size of the image patch for estimation. However, this would complicate the structure of the neural network, since it would require two entry points: the first for the image patch and the second for the image patch size (the second entry point is necessary, since the image patch is normally resized before being fed into a convolutional neural network). The second problem is a potential difficulty with gathering the data: it would be necessary to gather training data for the whole range of distances. My solution to overcome this problem is to avoid predicting the distance directly, and instead of that use the image patch width for the first-order estimate

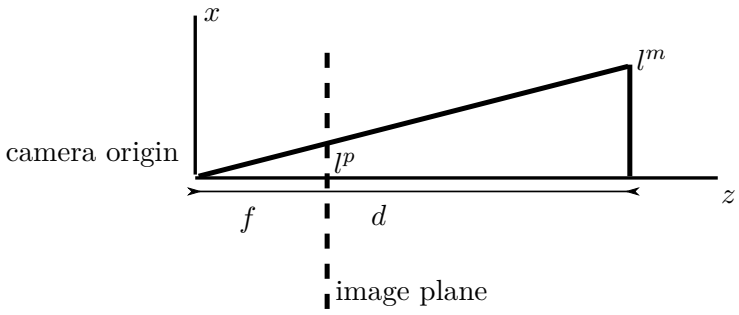


Figure 4.8: Pinhole camera model.

of the distance, and train the neural network to predict a coefficient that improves the initial accuracy.

Similarly to the model for predicting the normal vector, the model for estimation of the object position is also based on CNN. Here, the input is also an image patch with an object, and a similar architecture is used (see Fig. 4.6). The main difference between these models is that in the position network, the position is not predicted directly, but the *position coefficients* are predicted instead:

$$\hat{y} = \begin{bmatrix} k_d \\ x_{bo}^p \\ y_{bo}^p \end{bmatrix}, \tag{4.8}$$

which are used to determine the position of the object relative to the camera coordinate system. The predicted coefficients k_d, x_{bo}^p and y_{bo}^p are used to determine the distance from the camera to the object and the translation of the camera in the direction of x and y axis. Meaning of these coefficients is explained further in this section.

In order to describe the functionality of this model, first, let's discuss the projection on the image plane. Consider Fig. 4.8. Here, an object with world coordinates (l^m, d) is present. This object is projected onto the image plane at coordinate (l^p). The focal length f determines the distance from the origin of the camera coordinate system to the image plane. Assuming that the focal length is known, the projection can be determined as

$$l^p = \frac{l^m f}{d}. \tag{4.9}$$

This equation is useful for our purposes, because now we can predict the image coordinates of the object and transform them into world coordinates. For example, if the diameter of the object in image coordinates is known, the distance to the object can be determined.

Distance prediction

The model for predicting the distance from the camera to the object is based on the following equation:

$$d = \frac{w f}{w^p}, \tag{4.10}$$

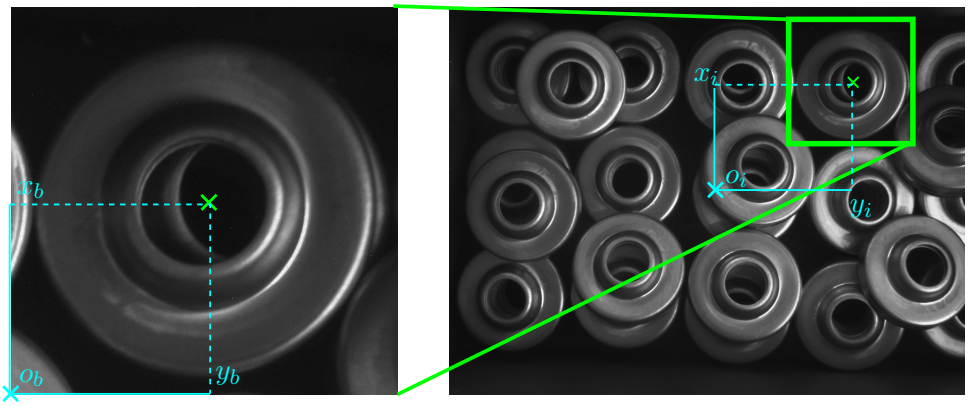


Figure 4.9: CNN architecture for prediction the translation in xy plane.

where d is the distance, w is the diameter of the object, f is the focal length of the camera and w^p is the diameter of the object measured in pixels. The general idea of this approach is to predict the size of the object in pixels, and use this equation to determine the camera-object distance. However, in order to use this approach, it is necessary to know the values of the focal length and the size of the object. Measuring these values is not complicated, however, it is more convenient to derive a more general formula and to avoid using concrete values when possible. Let me simplify Eq. 4.10. First, w^p is expressed as

$$w^p = \alpha w_b^p, \tag{4.11}$$

where w_b^p is the width of the image patch and α is a coefficient whose value is different for every image patch, but it is usually close to 1 because the width of the image patch is approximately equal to the size of the object. This yields

$$d = \frac{wf}{\alpha w_b^p}. \tag{4.12}$$

Now, substitute

$$k_d = \frac{wf}{\alpha} \tag{4.13}$$

from which we get the following equation:

$$d = \frac{k_d}{w_b^p}. \tag{4.14}$$

The difference between Eq. 4.10 and Eq. 4.14 is that in Eq. 4.14 we do not need to know the specific parameters of the object and camera w and f. By training the model to predict k_d (which is described below in this section), we can then use it to determine the distance to the object.

Prediction of translation in xy plane

The idea of this model is shown in Fig. 4.9. First, the pixel coordinates of the origin relative to the image patch are determined. Second, the coordinates of

this pixel are transformed to the image coordinate system. And finally, the translation in the camera coordinate system is determined.

Since multiple coordinate systems are used in this section, it is necessary to establish a naming system for the coordinates. The superscript determines whether the coordinate is measured in the image plane or in space: x^p or x^m; the first subscript determines the coordinate system: x_i is the camera (image) coordinate system, x_b is the bounding box coordinate system; the second subscript is used to select the item whose coordinates are measured (whether coordinates of the object x_{io} or the bounding box x_{ib}).

The coordinates of the object are determined using the following equation:

$$\begin{aligned} x_{io}^m &= \frac{x_{io}^p d}{f}, \\ y_{io}^m &= \frac{y_{io}^p d}{f}, \end{aligned} \tag{4.15}$$

where d is the distance from the camera to the object and f is the focal length of the camera.

In order to determine the pixel coordinates x^p, y^p, two coordinate systems are established. The *image* coordinate system is the system with the origin in the center of the image and axes as shown in the right-hand side of Fig. 4.9. The *detection* (or *bounding box*) coordinate system is the system with the origin in the bottom-left corner of the bounding box surrounding the object (see left-hand side of Fig. 4.9). The transformation between these coordinate systems is performed using the following equation:

$$\begin{aligned} x_{io}^p &= x_{ib}^p + x_{bo}^p, \\ y_{io}^p &= y_{ib}^p + y_{bo}^p. \end{aligned} \tag{4.16}$$

Now, when the correspondence between the world coordinate system and the image projection is established and the transformation between the detection and image coordinate systems is known, we have come to the main point of this section: the coordinates of the object in the camera coordinate system are determined by (a) estimating the pixel coordinates in the detection coordinate system and then (b) applying Eqs. 4.16 and 4.10 to obtain the (x, y) coordinates in the camera coordinate system. The coordinates of the objects can be the expressed as

$$\begin{aligned} x_{io}^m &= \frac{x_{io}^p d}{f} = \frac{x_{ib}^p + x_{bo}^p}{f} d, \\ y_{io}^m &= \frac{y_{io}^p d}{f} = \frac{y_{ib}^p + y_{bo}^p}{f} d. \end{aligned} \tag{4.17}$$

Training

As has been stated in the previous section, in this model, the distance coefficients are predicted: k_d, x_{bo}^p, y_{bo}^p. Their meaning has been explained,

however, I did not yet describe how to train the model. In this section, a method for training the position model is described. The model is a CNN which is trained in the supervised-learning fashion. The training data consists of a set of pairs (X, Y), where X is the input of the network (an image patch of a detected object) and Y is the output (a vector of position coefficients that determine the object position)

$$Y = \begin{bmatrix} k_d \\ x^p_{bo} \\ y^p_{bo} \end{bmatrix}. \tag{4.18}$$

In order to generate the training data, the position coefficients need to be determined. Assuming that the camera pose and the object pose are known, the position coefficients are obtained as follows. The first coefficient k_d is determined using Eq. 4.14:

$$k_d = dw^p_b. \tag{4.19}$$

The second and the third position coefficients are determined using Eq. 4.17:

$$\begin{aligned} x^p_{bo} &= \frac{x^m_{io} f}{d} - x^p_{ib} \\ y^p_{bo} &= \frac{y^m_{io} f}{d} - y^p_{ib}. \end{aligned} \tag{4.20}$$

After obtaining the training data, the parameters of the position network are learned in the supervised-learning fashion using the Mean Squared Error, defined in Eq. 2.6. Adam optimizer with learning rate 0.001 was used to find the optimal weights. The training required 5000000 iterations, dropout rate 0.5 was used.

4.3 Occlusion detection

If the objects are randomly placed inside the bin, some objects may be occluded by the others, and be temporarily unavailable for picking. The occluded objects need to be removed from the list of candidates for picking for two reasons: (a) in order to maximize the success rate of the controller and (b) to avoid damaging the parts of the robot or the gripper.

An image with four objects (among which only one is not occluded) is shown in Fig. 4.10. All of the objects in the image have been detected, but since some of them overlap, it is necessary to find the ones that lie on top. The occlusion detector used in this framework examines an image patch between two close objects and determines whether one of these objects is occluded by the other. Two detections a, b are considered close if the following condition is satisfied:

$$d(a, b) < 1.4 \frac{w(a) + w(b)}{2}, \tag{4.21}$$

where $d(a, b)$ is the distance between the centers of two bounding boxes and $w(x)$ is the width of the bounding box (measured in pixels).

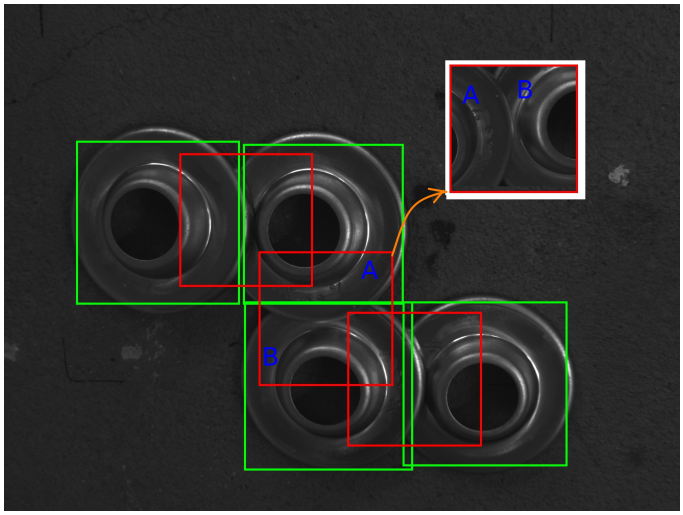


Figure 4.10: Model for occlusion recognition. Regions between close detections (red squares) are tested.

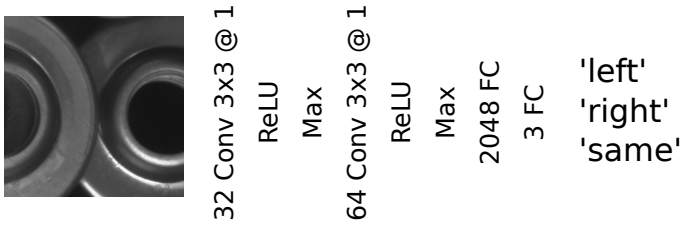


Figure 4.11: CNN architecture for occlusion recognition.

The possible outputs of this model are 'left', 'right' or 'same'. Other variants (like 'up', 'up-left') are excluded because the considered image patch is rotated and aligned with respect to the tested objects as shown in Fig. 4.10, where the image patch is rotated by 90 degrees such that the object A is on the left-hand side.

The reason why this model examines a part of the boundary of an object, and not the object itself using the image patch from the detection is because the former has less 'degrees of freedom': there are always two objects presented in the test image patch, and they normally are located in roughly the same place. On the other hand, if the whole object is considered, then it is necessary to address many different cases: the number of objects at the boundary and their positions may vary. These variants must be present in the training dataset to adequately learn the model, which requires additional work.

One of the downsides of this approach is that it is necessary to see the whole context of the object. If the neighbors are not fully in the image, and they are not detected, then the boundary is not examined, and a possible occlusion cannot be recognized.

The occlusion classifier uses a convolutional neural network whose architecture is shown in Fig. 4.11. It consists of two hidden convolutional layers that are followed by the ReLU activation function and max pool layers. The last

fully-connected layer contains three neurons

$$l_5 = \begin{bmatrix} l \\ r \\ s \end{bmatrix} \tag{4.22}$$

whose outputs correspond to one of the possible classifications: 'left', 'right', 'same', and the neuron with the highest value is the final prediction of the model. Unlike the previously described models, the size of the input layer is 32x32 in this network, which reduces the training time when compared to size 64x64.

Training data for this model is labeled manually. Dataset is augmented 4-fold by (a) rotating the image 180 degrees, (b) flipping the image upside-down and (c) combining steps *a* and *b*.

For training, Adam optimizer with learning rage 0.0001 was used with 2000000 training iterations. The dropout rate was set to 0.5. The loss function used for training was cross entropy loss:

$$V(\hat{y}, y) = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y}). \tag{4.23}$$

Chapter 5

Control algorithm

In the previous sections, the vision components have been described. In this section, a control algorithm for bin picking that employs these components is described. The main objective of bin picking is to recognize and acquire a part. In this framework, this procedure is done in multiple iterations, during which the part pose is determined more precisely, which is schematically shown in Fig. 5.1. The control algorithm for the picking task establishes a hierarchy of three behaviors: selection, picking, and aligning, each performing increasingly more low-level control.

```
Function SelectAndPick
    while True do
        object := FindObject()
        if object != None then
            PrepareForPick(object)
            TryToPick(object)
        else
            return
        end
    end
end
```

Algorithm 1: Main picking loop.

```
Function FindObject
    objects := DetectObjects()
    objects := FilterUnoccluded(objects)
    selected := SelectHighest(objects)
    return selected
end
```

Algorithm 2: Find and select an object in a pile.

Acquiring of a part begins with the selection phase (shown in Algs. 1, 2, 3). In the beginning of this phase, the robot scans the interior of the box by capturing two images from two different locations. The reason why two images are captured is not to provide the stereo information, but to see the

```
Function PrepareForPick(object)
    pose := EstimatePose(object)
    x, y, z := GetPosition(pose)
    zAbove := z + 0.6 // meters
    MoveGripper(x, y, zAbove)
    return
end
```

Algorithm 3: Prepare the robot for aligning with the selected object.

```
Function TryToPick
    // Picking loop
    for i in 1..3 do
        // Aligning loop
        for j in 1..10 do
            pos := EstimatePos()
            AlignGripper(pos)
            if GripperSufficientlyAligned then
                TryPick()
                if PickSuccess then
                    return
                end
            end
        end
        // Cannot align
        return
    end
end
```

Algorithm 4: Picking loop.

```
Function AlignGripper
    detection = FindObject()
    objectPose = GetPose(detection)
    newGripperPose = objectPose + δ× NormalVector(objectPose)
    MoveGripper(newGripperPose)
end
```

Algorithm 5: Align gripper with a part.

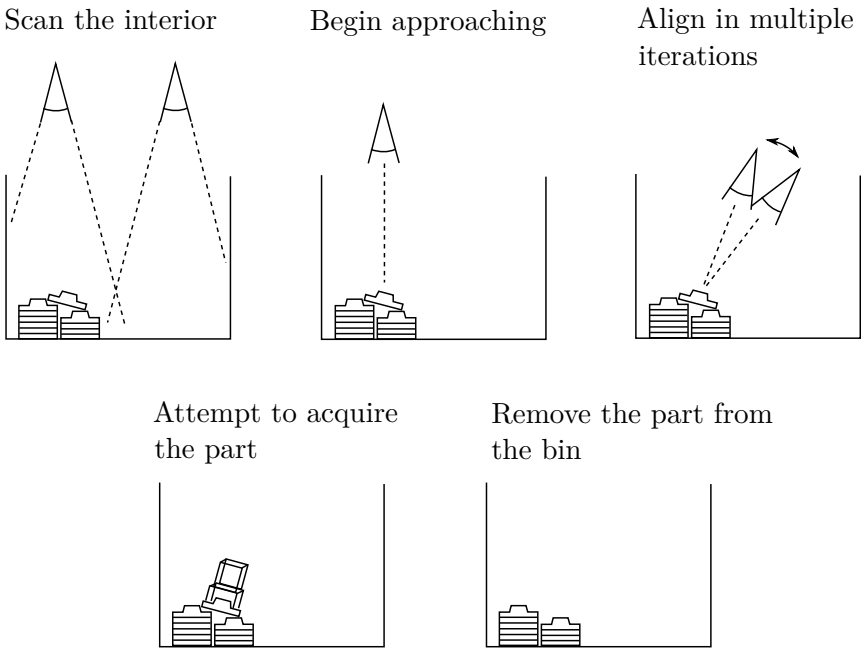


Figure 5.1: Control algorithm.

whole interior of the bin, since it is not fully visible in one shot. By using a smaller bin or a different camera lens, it would be possible to omit capturing additional images. After that, the parts are detected, and the occlusions are recognized. Occluded objects are removed from the list of candidates and positions of the unoccluded parts are determined. The part that lies on top (whose predicted position is the highest) is selected for acquiring. Finally, the robot moves directly above the selected part and captures the image of the part once again, and the approach phase begins.

In the beginning of the approach phase (shown in Alg. 4), the gripper is located above the previously selected part. The goal of this phase is to find the position of the gripper in which it is aligned with the object. After aligning the gripper and the object, an attempt to acquire the object is made: the gripper is moved to the position of the part, and the vacuum pads are enabled. After that, it is verified whether the part has been acquired by measuring the pressure in the vacuum pads. It is possible to distinguish the unsuccessful attempts, because if the vacuum pads are not in contact with a flat surface, the pressure inside them is lower. If the attempt is unsuccessful, then the approach phase is repeated two more times. Eventually, in case if the attempt to pick has been successful, the object is removed from the bin and new acquiring iteration begins.

During the alignment phase (shown in Algs. 4, 5), the gripper is iteratively aligned with the normal vector of the part. In each iteration, the pose is estimated, and the camera is moved to the position where its z-axis matches the estimated normal vector at constant distance δ from the estimated part origin. The reason why this is done in iterations and not in one step is to minimize the absolute error. Typically, 5 iterations are required to complete

this phase.

The reason why the controller architecture has been separated into a hierarchy with three levels is that this structure offers a trade-off between the complexity of the implementation and complexity of behavior: if fewer levels were used, it would be more difficult to achieve efficient and intelligent behavior; and if more levels were used, it would be more difficult to implement the algorithm and maintain the implementation.

In order to avoid collisions, a simple strategy is used. The bin is placed at a known predefined position, and the software limits are set to prevent the collision with the bin walls. Collisions with the objects in the pile are prevented by always acquiring the objects that lie on top.

5.1 Configuration assumptions

There are multiple situations that cannot be resolved using the developed system. The objects that are flipped upside-down are not considered for simplicity, and they should be removed from the bin manually. The geometry of the gripper does not allow to reach the objects that are inclined towards the wall. When this situation occurs, the robot attempts to align with the part, but after it reaches the iteration limit, it signalizes that the object is unreachable.

Chapter 6

Implementation

6.1 Data

Since the components that are used in this framework are trainable, obtaining data is a necessary part of the implementation. During data gathering, a camera is placed in a known position, an object is placed in front of the camera, and an image is captured. The coordinates of the camera and the object are saved, as well as the image. The camera position is selected randomly at various distances to the object. After capturing the image, a new random position is selected. This process ensures that the object is captured at various angles (with the rotation angle up to 45 degrees). The total number of samples that were used for training the model is 1726 images.

The knowledge of the pose of the object is required for obtaining the data, because otherwise, it would not be possible to train the pose model within the developed framework. The easiest way to determine the object pose without using additional measuring equipment is by using the measurements from the robot. The gripper is manually positioned to a place where the vacuum pads are in contact with the object. Since the gripper pose is known, it is straightforward to determine the object pose by subtracting the known offset.

Manual labeling of the images is virtually impossible, because it is difficult to determine the position and orientation of the object just by looking at the image. For the same reason gathering pose data requires additional attention (compared to typical classification tasks), it is not easy to visually confirm that a training data sample is correct. If, for example, the implementation of the framework contains bugs, they can be difficult to spot, and a large portion of the automatically generated data may be corrupted.

One problem that occurs while acquiring the data is that the camera that was used in this system produces corrupted images (some portions of the image might be missing or come from a different image, see Fig. 6.1). Approximately 2 % of the images are corrupted with varying levels of damage. However, whether this has a negative effect on the quality of the system is unclear, since during the experiments, no malfunctioning due to this kind of error was observed.

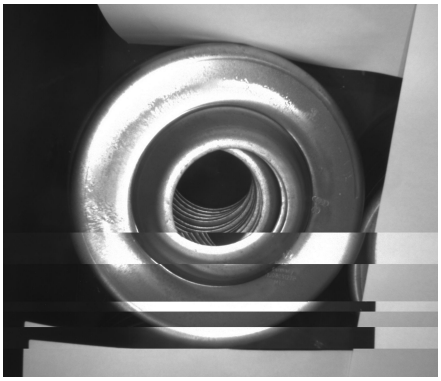


Figure 6.1: Some images are damaged during capturing. Here, a portion of the image comes from another shot.

6.2 Software Implementation

Building a computer vision application is simplified by a wide availability of open-source image processing and machine learning libraries. OpenCV [Its15] is a widely used library for digital image processing and computer vision. It contains many modules that provide efficient implementations of popular computer vision algorithms. The library provides APIs for multiple languages, is open-source and cross-platform. Tensorflow [AAB+15] is an open source library for machine learning developed by Google. Tensorflow provides efficient components and a high-level interface for building, training and running neural networks and other machine learning models. It supports different hardware for training and running the models and is cross-platform. Building distributed applications that are written in multiple languages requires using a method for inter-process communication. ZeroMQ is a library that provides this functionality via TCP sockets, allowing to run concurrent applications.

The proposed framework has been implemented as a software package written in C++ and Python. The controller logic was written in C++. OpenCV has been used to handle basic operations with images and to implement the detector, since OpenCV implementation of HOG has been used.

The components that are based on convolutional neural networks (the pose estimation and occlusion recognition) have been implemented using Tensorflow. Unfortunately, Tensorflow v0.11 does not provide mature API in C++, so Python API has been used. The communication between the components and the application core has been implemented using ZeroMQ.

6.3 Experiments

In order to test the functionality of the proposed system, the experiments have been conducted. For the computations that are required for using the developed models (for the detector, pose estimation and occlusion recognition)

a regular laptop (called lab PC later) with 4-core Intel i5 @ 1.6 GHz CPU has been used. No graphics card has been employed and the same computer has been used for learning the models.

Running the models that are based on the neural networks does not require extensive computational resources and the following computational time was required for one feed-forward run on the lab PC (averaged over 500 executions):

- Orientation: 53 ms
- Position: 55 ms
- Occlusion: 43 ms

Memory that is used for loading the models is

- Orientation: 166 MB
- Position: 87 MB
- Occlusion: 172 MB

Since the models in this systems are comparatively small, it is feasible to perform the training without using a GPU. Training one model requires approximately 14 hours on the lab PC.

Executing the HOG detector takes 809 ms (averaged over 100 executions). Image size was set to 778x583, since the input image is downscaled by the factor of 0.3 for performance.

Two variants of the system have been developed that differ in how the models have been trained. In the first version, the LEDs have been disabled and the parts were positioned horizontally during training. In the second version, the LEDs were used and the parts were oriented randomly in the training data. In the first variant a portion of the training data is corrupted: a wrong camera position has been logged during data gathering. Because of that, the training dataset contains items with wrong orientation and position, which has a harmful effect on the quality of the models. It is known that these items are present in the dataset, but it is impossible to determine which ones are damaged. In the second variant of the models, the data is correct. During the testing, two configurations have been considered: the horizontally oriented parts and randomly oriented parts with maximum inclination 29°.

The first variant of the models has been tested for two scenarios: for a horizontally oriented parts and for randomly oriented parts. Horizontally oriented objects are grasped with success rate 98 % (results based on 96 trials). Randomly oriented parts are acquired with lower probability 72 % (based on 80 trials). This was determined by repeatedly removing different parts from the bin in different configurations. The parts are removed one by one without operator assistance, and the bin is periodically refilled (after removing approximately 20 objects). Such a big difference in quality between two configurations is caused by multiple factors. First, the corrupted items in the dataset that were mentioned above prevent from training the correct

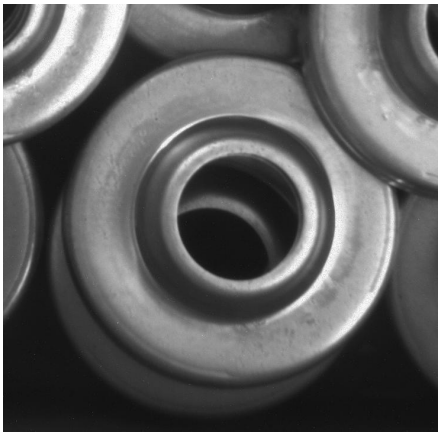


Figure 6.2: An example of an occluded object where the occlusion cannot be recognized in the proposed system, since the occluding object is not seen.

model. Second, the objects look differently from different positions of the observer due to the external lighting and the specular surface. Because of these problems, the model apparently does not generalize well to new conditions. These issues lead to building the second model, where LEDs are used and and where the dataset is created correctly.

The tests of the performance of the second variant have shown that the success rate of the grasping algorithm is 93 % (based on 150 trials). During the testing, the parts were randomly oriented. The most common reason of failure is a wrong distance estimation, when the predicted distance is lower than the actual distance. In that situation, the part is not grasped by every vacuum pad (usually, it is grasped by two out of for pads), and the pressure sensor incorrectly identifies this configuration as 'grasped'. In other cases, the part slips under the gripper, which also may result in unsuccessful grasp. In this situation, the next attempt is usually successful, because the object is located in a more stable position.

During the testing, I faced a problem with the occlusion recognition module. In some cases, in the beginning of the acquisition cycle, it may happen that the occlusion is not recognized. This happens due to the way how the recognition system is built: in order to detect the overlap, both of the objects have to be present in the image. An example of an image where the overlap is not detected is shown in Fig. 6.2. Even though the occlusion module is reliable when both of the objects are present in the image, it may fail in some cases, and it causes approximately 6 % of the failures in the beginning of the acquisition cycle (based on 100 trials).

Chapter 7

Discussion

It follows from the experiments that the success rate of the grasping procedure is 98 % for horizontal parts and 93 % for random parts, and the probability of success of the of the selection phase is 94 %. Since grasping follows the selection phase, and their success rates are independent, the performance of the whole system can be determined. Assuming a typical scenario in which a bin with parts is filled with 70 % horizontal unoccluded parts and 30 % random (and possibly occluded) parts with maximum inclination 29°, the expected performance of the proposed system is

$$\begin{aligned} R &= (P_{ph}P_{sh} + P_{so}P_{pr}P_{sr}) \times 100\% \\ &= (0.7 \times 0.98 + 0.94 \times 0.3 \times 0.93) \times 100\% = 94.8\%, \end{aligned} \tag{7.1}$$

where P_{ph} is the probability that the grasped part is horizontal, P_{sh} is the success rate for a horizontal part, P_{so} is the success rate of the selection phase (for the horizontal unoccluded parts it is assumed that the success rate is 1, since they are not occluded), P_{pr} is the probability that the grasped part is random, P_{sr} is the success rate for a random part. This rate exceeds the required performance 70 % by a margin of 25 %.

The part that has been considered in this thesis has 5 degrees of freedom (3 for the position and 2 for orientation). Using this framework for a rotationally nonsymmetric part would require modifying the orientation model by adding the rotation around the normal to the prediction. Since the rotation $\phi + 2\pi$ and ϕ is the same angle, the angle should be probably encoded using its sine and cosine.

Since the proposed system does not rely on hard-coded properties of the part, using this framework with a part whose geometry is more complex should not bring additional difficulties, as the models will learn the necessary visual features.

Two main problems that were identified in Section 6.3 were occasional faults of the distance prediction and the occlusion detection models. In order to improve the success rate, these problems have to be addressed. One solution for improving the performance of the system is to add contact sensors on the vacuum pads. This modification would reduce the requirements for the level of quality of the distance measurement, since the possible estimation error would be corrected by sensor feedback. Another way to improve the system

is to improve the occlusion recognition component by adding an additional model for verification.

The functionality of the system requires training the models of a part. Because of this, preparation of models for new parts is time-consuming, and it may take several days to finish. Improving this process and making it more time-efficient would largely benefit the framework, as it would allow building more precise models.

Another improvement that would largely improve the system is applying methods of unsupervised learning for this task. Since acquiring labeled data for the models requires a lot of time and effort, there is a need in algorithms that are capable of building a model from unlabeled sensor data, which is usually much cheaper.

It is also worth investigating alternative CNN architectures for pose estimation. Reducing the size of the models (the number of parameters) would shorten the training time and also may improve the performance.

Chapter 8

Conclusion

In this thesis, a recognition system for visual bin picking system was presented. The developed framework includes models for estimation of position and orientation and for occlusion detection.

The part that has been used for testing the system was a metallic strut bracket. However, the proposed system does not require hard-coded knowledge of the part, and uses trained models to determine key attributes of the object, and the application of the proposed system is not limited to the tested part. These models are based on convolutional neural networks; their implementation and testing procedure were described.

A controller for a robotic manipulator has been implemented for testing of the framework. Two variants of the framework have been implemented and described. It has been shown that the system works reliably in the tested conditions, but requires improvements for full unsupervised autonomy.

The main contributions of this thesis are introduction of a useful representation of part position and orientation that is suitable for training a neural network, and introduction of a method of using this representation for bin picking.

The achieved estimated performance of the system which is the probability of successfully acquiring a part is 94.8%, which greatly exceeds the requirements that were initially posed (70%).

To my best knowledge, this is one of the first works described in the literature that uses CNN-based pose estimation for the bin picking task. Hopefully, methods based on deep learning will be more widely used in industry in the future.

Possible modifications of the software and hardware components of the system for improving the performance and simplifying the usage of the framework have been discussed.

CD contents

The controller application and the scripts for training the models are located in the directory `bin_picking`. A copy of this thesis and a video demonstration are located in the root.

```
CD
├─ thesis_sushkov.pdf
├─ video1.mp4
└─ bin_picking # the application
   ├─ # controller modules
   ├─ detector
   ├─ server
   ├─ server_primitives
   ├─ camera
   ├─ support
   ├─ svm_light
   ├─ exe
   ├─ dbg
   ├─ cmake
   ├─ # training scripts and models
   ├─ occlusion_middle_detection
   ├─ pose2_regression
   ├─ xyd_regression
   └─ runtime_data
```

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, Software available from tensorflow.org.

[BDD63] Arthur E Bryson, Walter F Denham, and Stewart E Dreyfus, *Optimal programming problems with inequality constraints*, AIAA journal **1** (1963), no. 11, 2544–2550.

[BDTD+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al., *End to end learning for self-driving cars*, arXiv preprint arXiv:1604.07316 (2016).

[BH86] Robert C Bolles and Patrice Horaud, *3dpo: A three-dimensional part orientation system*, The International Journal of Robotics Research **5** (1986), no. 3, 3–26.

[BM92] Paul J Besl and Neil D McKay, *Method for registration of 3-d shapes*, Robotics-DL tentative, International Society for Optics and Photonics, 1992, pp. 586–606.

[CV95] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Machine learning **20** (1995), no. 3, 273–297.

[DT05] Navneet Dalal and Bill Triggs, *Histograms of oriented gradients for human detection*, Computer Vision and Pattern Recognition,

2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, IEEE, 2005, pp. 886–893.

[Fis36] Ronald A Fisher, *The use of multiple measurements in taxonomic problems*, Annals of eugenics **7** (1936), no. 2, 179–188.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[Its15] Itseez, *Open source computer vision library*, `https://github.com/itseez/opencv`, 2015.

[KB14] Diederik Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, 2012, pp. 1097–1105.

[LC14] Sijin Li and Antoni B Chan, *3d human pose estimation from monocular images with deep convolutional neural network*, Asian Conference on Computer Vision, Springer, 2014, pp. 332–347.

[LPKQ16] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen, *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection*, arXiv preprint arXiv:1603.02199 (2016).

[LTV+12] Ming-Yu Liu, Oncel Tuzel, Ashok Veeraraghavan, Yuichi Taguchi, Tim K Marks, and Rama Chellappa, *Fast object localization and pose estimation in heavy clutter for robotic bin picking*, The International Journal of Robotics Research **31** (2012), no. 8, 951–973.

[RK96] Krisnawan Rahardja and Akio Kosaka, *Vision-based bin-picking: Recognition and localization of multiple complex objects using simple visual cues*, Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on, vol. 3, IEEE, 1996, pp. 1448–1457.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., *Mastering the game of go with deep neural networks and tree search*, Nature **529** (2016), no. 7587, 484–489.

[TS14] Alexander Toshev and Christian Szegedy, *Deeppose: Human pose estimation via deep neural networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1653–1660.

[WL15] Paul Wohlhart and Vincent Lepetit, *Learning descriptors for object recognition and 3d pose estimation*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3109–3118.

[ZSN+16] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, and Jianxiong Xiao, *3dmatch: Learning the matching of local 3d geometry in range scans*, arXiv preprint arXiv:1603.08182 (2016).

[ZYS+16] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker Jr, Alberto Rodriguez, and Jianxiong Xiao, *Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge*, arXiv preprint arXiv:1609.09475 (2016).