Hochschule Reutlingen
Reutlingen University

INF
Informatik

Uwe Kloos, Natividad Martinez, Gabriela Tullius (Hrsg.)

19 Informatics Inside

Tagungsband

Hochschule Reutlingen

infoinside.reutlingen-university.de
infoinside@reutlingen-university.de

Hochschule Reutlingen
Reutlingen University

Uwe Kloos, Natividad Martínez, Gabriela Tullius (Hrsg.)

Informatics Inside
experience(IT);

Informatik-Konferenz an der Hochschule Reutlingen
8. Mai 2019

Inhaltsverzeichnis

Semi-automated image data labelling using AprilTags as a pre-processing step for machine learning*

Steven Cybinski
Reutlingen University
Steven.Cybinski@Student.Reutlingen-University.DE

Abstract

Data labelling is a pre-processing step to prepare data for machine learning. There are many ways to collect and prepare this data, but these are usually associated with a greater effort. This paper presents an approach to semi-automated image data labelling using AprilTags. The AprilTags attached to the object, which contain a unique ID, make it possible to link the object surfaces to a particular class. This approach will be implemented and used to label data of a stackable box. The data is evaluated by training a You Only Look Once (YOLO) net, with a subsequent evaluation of the detection results. These results show that the semi-automatically collected and labelled data can certainly be used for machine learning. However, if concise features of an object surface are covered by the AprilTag, there is a risk that the concerned class will not be recognized. It can be assumed that the labelled data can not only be used for YOLO, but also for other machine learning approaches.

Keywords

Semi-automatic Data Labelling, AprilTags, ArUco, Pre-Processing, Machine Learning, You Only Look Once (YOLO)

*

Betreuer Hochschule: Prof. Dr.-Ing. Cristóbal Curio
Hochschule Reutlingen
Cristobal.Curio@Reutlingen-University.de

Informatics Inside 2019
Wissenschaftliche Vertiefungskonferenz
08. Mai 2019, Hochschule Reutlingen

CR-Categories

D.2 [**ACM**]: SOFTWARE ENGINEERING;
I.2 [**ACM**]: ARTIFICIAL INTELLIGENCE;
I.4 [**ACM**]: IMAGE PROCESSING AND COMPUTER VISION

1 Introduction

Data labelling is a pre-processing step for the preparation of training data to train a neural network. The compilation of labelled image data is a time-consuming activity. The main reason for this is the need for a larger dataset consisting of many image data with corresponding labels. For each image, the required label information must be recorded, such as the position of the object(s). The formatting and storing of the label information depends on the later used framework for machine learning.

Many datasets can be found on the Internet. Some of them have been created automatically and therefore contain potentially incorrect information. The dataset may have to be formatted accordingly to the used framework understands it. But if it is a special object, the dataset itself must be created. In the context of a project at the university, for example, stackable boxes and the visible sides must be detected and classified. There is no suitable dataset for these stackable boxes on the Internet.

If a dataset needs to be created where the object has essentially no optical variation, this can be done with a single physically available object. In this case, the preparation of the object, for example with AprilTags, is not

a work-intensive task. The AprilTags, which will be explained later, can be detected using basic image processing techniques. If they are attached to an object, it is possible to link the object surfaces to a particular class. Thus objects that have such a marker can be localized and classified relatively easy. This approach enables the creation of datasets without the need for time-consuming manual input. With the help of camera calibration, additional information such as a 3D pose and the distance to the camera can be estimated.

1.1 Motivation and Objectives

As previously mentioned, AprilTags can form the basis to create a dataset. If it is well done, it will not matter if it is a stackable box or another object. The aim of this work is to develop a tool for creating a training dataset using AprilTags. So that the first step to machine learning can be completed as easy as possible. The tool should support the connection of Microsoft Kinect and Intel Realsense, but should also be easily expandable by further camera types. Through a dynamic configuration it is possible to set marker ID related classes as needed, so that the tool can be used as flexibly as possible. After configuration and preparation, the object can be recorded via a user interface and stored in a simple and framework-independent format.
The subsequent evaluation of the generated data will be performed for the previously mentioned university project scenario. The generated dataset trough the tool will be used for YOLO training. This will proof whether the dataset is usable for this task and whether YOLO itself can be trained. However, the prerequisite for this is that the data record is converted to the appropriate format.
The general aim is that the tool will support future projects in creating a machine learning solution.

1.2 Structure of the Paper

At the beginning of this paper the state of the art will be discussed. This includes technologies which were used for this project and one project which also dealing with semi-automated data labelling. Subsequently, the solution approach of labelling with AprilTags is explained. This includes the basic idea and process of labelling, on which a tool was implemented.
Then the result is validated in practice by training an object detection framework with the generated dataset. Finally, a conclusion is given based on the findings.

2 State of the art

This section explains the used technologies which already exists. Afterwards related papers with almost the same aim but another way to solve the labelling challenge are named.

2.1 Used technologies

In this subsection, the important technologies which will be used are briefly presented. This includes the aforementioned AprilTags with the library ArUco for detection of square fiducial markers. Then the real-time object detection system YOLO and the neural network framework Darknet is explained. YOLO and Darknet are only briefly mentioned here, as these technologies were only used for evaluation reasons and are not part of the actual project.

2.1.1 AprilTags

AprilTags are 2D barcodes developed and described in [1] by Edwin Olson for Robotics. At the first look they resemble the widely used QR codes. Unlike QR codes, they do not have more complex embedded information, such as web addresses, but only a single ID. Depending on the family, a certain number of unique IDs can be used. In [1] the authors recommended the family *36h11*, therefore this family was used for this project. In addition, the developed AprilTag detector is robust against illumination variations and occlusions and allows an exact localization of the markers [1].
There are also markers, such as the ARTags. But for this project the AprilTags were used because they delivered a good detection result. Moreover, a comparison between the dif-

ferent markers is not part of the work. In addition, the tests from [1] showed that April markers can be detected from a greater distance and at a greater angle to the camera. However, the configuration and use of ArUco allows many more markers to be used in this project, if they are supported by ArUco library.

2.2 *ArUco*

ArUco Marker and the detection library was developed by Rafael Muñoz and Sergio Garrido and described in [2]. One main point of the project described in the paper is the camera pose estimation under consideration of occlusion. The development was primarily focused on a solution for robotics and augmented and virtual reality applications.
The markers of ArUco look very similar to the AprilTags and also contain only a single ID. More interesting for this project is the library of ArUco, which can detect different square fiducial markers [2]. This includes the whole palette of AprilTags, ARTags, ArUco markers and many more. In [2] the authors show, that augmented reality work with AprilTags very well. If the marker and object size is known, it is possible to give the synthetic object a real size. For example, you can specify that a object should be exactly five centimetres high in the real world. To make this possible it is assumed that the camera has been calibrated beforehand.

2.2.1 You Only Look Once and Darknet

YOLO is a real-time object detection system developed by authors from [3]. YOLO, unlike other detection methods, looks at the image only once, this is where the name comes from. Due to this, YOLO is the fastest detection method, with up to 155 frames per second, and without much loss of quality [3]. This is possible because this method follows a different approach than the other methods. YOLO goes “straight from image pixels to bounding box coordinates and class probabilities” [3].
Darknet is a framework for neural networks which was developed by Joseph Redmon [3]. The author does not write a paper for Darknet, but you can see on his webpage[1] that it was developed especially for YOLO. In summary this framework makes it possible to use YOLO.

2.3 *Related work*

In [4], the authors Daniel Lopresti and George Nagy dealt with the topic of interactive labelling, classifier training, and automated labelling. There are several publications on this approach which were referenced in [4]. In their work, the authors have looked at how this approach can be improved by optimal data partitioning.
One system explicitly mentioned in the paper was the Computer Aided Visual Interactive Classification (CAVIAR) system. The idea is, that a classifier is trained interactively and iteratively with the help of a human operator. In [4] the procedure was explained as follows. A sequence of N patterns is manually labelled by an operator. After the sequence, a classifier is trained with this data and then applied to a new and slightly larger sequence of patterns. The operator now checks the result and makes corrections, such as removing false positives or adding new labels to false negatives. After this sequence, the classifier is retrained and the result is applied to a new sequence. This process is repeated until a certain abort criterion is met, e.g. a low error rate.
This approach allows a fast and comfortable training and could be extended by the functionality, that the labelled data can be stored separately and used for other models. However, it can be assumed that an untrained model initially requires a lot of work because its classification is insufficient. If an already sufficiently trained model is used, the quality can be improved relatively easy by this approach.

3 Labelling with AprilTags

This chapter will explain the semi-automatic labelling approach of this paper. As already

[1] See `https://pjreddie.com/darknet/`

mentioned in chapter 2.2, a 3D pose can be estimated from the detected markers, like in augmented reality. If there is also a definition of the ID and class relation as well as the size proportions, then the required information for semi-automatic labelling is given.
The labelling process basically consists of four steps. Each frame reads in the first step passes through the entire process as long as the tool is running. In summary, image resources are retrieved and the markers are detected. If markers have been found and the marker ID is known, the marker information and the configuration will be used to create labels. In order to not record too much similar data, the operator must indicate if he wants to record this frame and label. This assumes that at least one object was detected and labelled.
The following subchapters first explain the camera calibration and configuration. Subsequently, the four previously mentioned steps are explained in more detail.

3.1 Camera calibration and dynamic configuration

In this subchapter, two important points are explained which must be given as a precondition for the labelling. This includes camera calibration and configuration.

3.1.1 Camera calibration

The camera calibration is an essential step for this project. The labelling can only run properly with a good calibration. An projection of three-dimensional coordinates onto a planar image can be carried out by the so-called pinhole camera model, which requires a camera calibration [5]. In this project the calibration was done with the AprilTags. There are also alternative methods, such as calibration using a chessboard.
The calibration is performed by recording a chessboard or a board with AprilTags from different perspectives. A batch is created from the sum of the detected features, which is later used to calculate two matrices using the ArUco library. One resulting matrix is the camera matrix or the pinhole camera model, which is responsible for mapping 3D points on an image plane. “However, using the ideal pinhole model in isolation will poorly capture the dynamics of most real world lenses, especially those with a wide field of view” [5]. Therefore, the model is extended by another distortion matrix that is used to correct the lens distortion [5].

3.1.2 Dynamic configuration

The configuration of the tool defines important basic information for the application and labelling. In order to not only provide a solution for the stackable boxes scenario, it was defined as a requirement for the tool that the labelling should be as dynamically configurable as possible.
First, the marker family to be detected is determined. Below is a list with all objects that should be labelled by the tool. The objects contain a list of classes, such as front, back, left and right. Each class gets unique marker ID as well as the marker size and the size of the white marker border. In addition, the objects dimensions will be defined from the respective point of view. Since the markers cannot always be placed in the centre of the object, it is possible to set offsets to adjust the distance from the center. To keep the same objects apart, they should be defined with different marker IDs as often as needed. This allows the tool to determine which marker belongs to which object, even if it is of the same type, e.g. box.

3.2 Read frame from interface

In the developed tool, an interface was used that allows relatively simple integration of additional cameras. The interface provides functions for fetching RGB and depth frames. The initialization and connection of the camera source is defined in the constructor. Another predefined function is the one that specifies how to disconnect from the camera source. Therefore, with a few definitions, almost any camera can be supported by the tool, regardless of the resolution or frame rate of the camera.

3.3 Detect and interpret markers

Marker detection is the basic part of labelling, as it is used to derive further information for labelling. As mentioned before, the ArUco library was used for marker detection.

The detection is explained by [2] in four essential steps, which are explained for illustration in relation to figure 1. In *(a)* you can see the original image for which a marker detection is performed. The first step is the image segmentation. Here the most important contours from the greyscale image are segmented by means of local adaptive thresholding *(b)* [2]. In the second step the contours are extracted and filtered. The Suzuki and Abe algorithms will be used to extract the contours from the segmented image, resulting in a set of image contours *(c)* [2]. By applying the Douglas-Peucker algorithm, a polygonal approximation is performed, causing the contours that are not attached to four vertex-polygons to be discarded [2]. After simplifying the contours by keeping only the outer contours, the result will look like in *(d)* [2]. The third step deals with marker code extraction. A homography matrix is calculated and the respective tag is normalized *(e)* [2]. The optimal threshold value is then determined using the Otsu's method and applied to the image [2]. This results in a binary image that is divided into a grid where each cell is assigned to the value 0 or 1, depending on the majority of pixels *(e,f)* [2]. If not all border regions have the value 0, an additional analysis is aborted and the marker candidate is discarded [2]. In the fourth step the marker identification and error correction are performed. The aim is to check which marker candidates belong to the dictionary and which are just part of the world [2]. The dictionary is converted into a binary tree and searched for the concatenated bits of the marker [2]. A marker candidate is valid if it was found in the dictionary.

The interpretation of the marker IDs is performed using the configuration from chapter 3.1.2. A list of visible objects is created from the object definitions of the configuration and the detected markers. All further steps are performed on the basis of this list. If the multiple IDs of the same object are visible, such as front and left, an object with four pages is created in this step. If several IDs of the same object are visible, such as front and left, this is registered in the particular object. Therefore, even if the objects are of the same type, they should have unique IDs, as mentioned in chapter 3.1.2. Up to this point, the labelling provides information about which objects are visible with which classes and where exactly their reference points (markers) are.

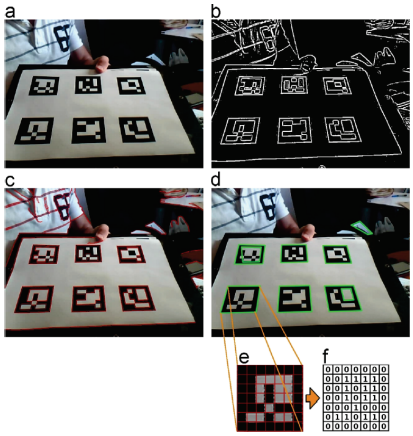

Figure 1: "(a) Original image. (b) Result of applying local thresholding. (c) Contour detection. (d) Polygonal approximation and removal of irrelevant contours. (e) Example of marker after perspective transformation. (f) Bit assignment for each cell." [2]

3.4 Generate label information

Through the previous step a list of visible objects was created from the detected markers. This chapter explains how further labelling information is generated from these objects. The generation of label information is performed with four additional steps, which will be explained in the following subsections.

3.4.1 Define and inpainting

Since the markers should not be included in the target dataset, inpainting is used to remove the markers. Inpainting enables the reconstruction of an image area using the surrounding image environment. Inpainting was implemented in the tool with the Open Source Computer Vision Library (OpenCV). For all detected markers, the corner points were first expanded by the white border defined in the configuration. This was specified larger, so that the newly defined area is slightly larger than the original marker, see 2 *(b)*. These areas were then drawn on a mask *(c)*, that specifies where a reconstruction should be made. When choosing a range of equal size, the white border is taken into account by the algorithm. In this case this region looks like a bright spot. OpenCV provides a few inpainting algorithms, from which the Telea method was chosen because of its performance and quality. The Telea method was developed by Alexandru Telea and described in [6]. It is based on a Fast Marching Method and tries to reconstruct relatively fast with consideration of gradient direction and photometry [6].

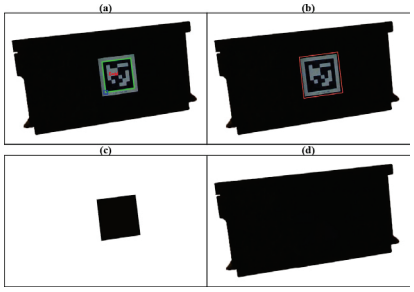

Figure 2: Illustration of the inpainting process. (a) Detection of the marker. (b) Resizing the detection area. (c) Inpainting mask. (d) Result of the inpainting.

Once the inpainting has been applied, the result should look like figure 2 *(d)*. In this step no relevant labelling information was created, but preparation of the image material was made, so it can be used for machine learning.

Now the markers do not falsify the result of the detector to be trained. In order to not waste too much computing power, the inpainting is only used if the corresponding preview option has been activated via the GUI or a recording is performed.
Unfortunately, this step has a weakness. If only a small part of a marker is visible and therefore it is not detected, no inpainting will be performed for this region. As a result, individual recordings may occur that falsify the dataset. Further information on this can be found in chapter 4.

3.4.2 Estimate 3D pose

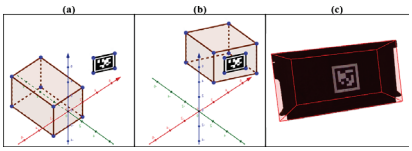

Figure 3: 3D pose estimation by projection. (a) Definition of the 3D object and the position of the detected marker. (b) Transformation of the 3D object to the marker. (c) Result based on a real recording.

Based on the previously generated list of objects and detected markers, the 3D pose of the object is determined. The object is first defined in the origin using the configured size, as shown in figure 3 *(a)*. In this case, the origin corresponds to the centre of the marker. Therefore, the points of the front side seen from the marker are defined with the Z coordinate 0 and the points of the back side with a corresponding negative Z coordinate, since the object is located behind the marker. The centre of the front must be the origin itself, if it is assumed that the marker is centred on the object. If an offset is defined in the configuration, the entire object is transformed accordingly. Then, as shown in *b*, the defined 3D object must be transformed to the position of the marker and then assigned to the image plane. For the transformation and the mapping OpenCV offers a function called *projectPoints*. As input parame-

ters the function needs the points of the 3D object, the output rotation vector (rvec) and output translation vector (tvec) of the respective marker as well as the camera and distortion matrix. Where *rvec* and *tvec* are used for the object transformation and the camera and distortion matrix for the projection. The resulting image points can now be drawn to the image plane, as *(c)* illustrates.
After this step we know exactly where the detected objects are and which pose they have. The obtained image points are also stored in the respective object label and used in the next step.

3.4.3 Extract detection windows

In this step, a detection window is extracted from the previously generated image points. To define the detection window the first step is to iterate through the eight image points to find the minimum and maximum values of the coordinates. Then the four corner points of the detection window *DW* are defined as follows.

$$DW = \{[x_{min}, y_{min}], [x_{max}, y_{min}], [x_{max}, y_{max}], [x_{min}, y_{max}]\} \quad (1)$$

Figure 4 shows the recognition window determined in this way, whereby the projected 3D pose is essentially framed.
This step is uncomplicated, because the positions of the objects is already known. The definition of a detection window was nevertheless done, because many training datasets only aim at a pure object detection and no 3D pose detection.

3.4.4 Estimate distance

The distance between the camera and the respective object is determined by the Euclidean distance. The camera position, which in this case is the origin *o = [0, 0, 0]* itself, and the *tvec* of the marker are required for this calculation. Than the Euclidean distance is calculated as follows.

$$distance = \sqrt{(tvec_0 - o_0)^2 + (tvec_1 - o_1)^2 + (tvec_2 - o_2)^2} \quad (2)$$

If the markers and object size are specified in millimetres in the configuration, the result must also be interpreted in millimetres. The distance to the objects is also stored in the label information of the respective object.

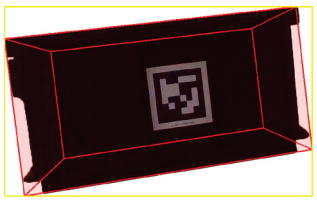

Figure 4: Determination of a detection window using a known 3D pose.

3.5 Create a record

Records are triggered by the operator. The creation of a record is done by generating a JavaScript Object Notation (JSON) object from the label information and applying inpainting on the RGB image. Recording is only possible if at least one object has been recognized, so that the label contains at least one object. Each object has a unique ID for this recording, as well as the previously generated label information. Among other things, each object contains at least one visible side that contains a class name and a distance. If an object is positioned so that two markers are recognized, both visible sides could be listed here.
Subsequently, the JSON object, depth and RGB image is stored with the same name, but with the corresponding prefix, in the configured path. The three data types are organized in three separate subfolders, so that they are not mixed. A framework specific preparation as well as composition of the data was intentionally left out, so that the datasets can be used in a wide range.

4 Project result evaluation

This chapter evaluates the previously presented project. First a look is taken at the generated label information and then a practical test is performed.

4.1 Label information

In this subchapter the 3D pose estimation and the derived detection window quality are considered. Subsequently, a small experiment will show how accurate the distance estimation is.

4.1.1 3D pose estimation and detection window

The quality of the 3D pose estimation and the detection window cannot be measured directly, because there is no reference data for comparison. Therefore, the position is taken in relation to the visual impression. Table 1 shows a ranking that was performed in the context of this work. Essentially it was looked at how the camera resolution and marker size affect the quality of the 3D pose estimation. It is assumed that the configuration of the markers and object size as well as the offsets were set correctly and accurately during this evaluation. It is also assumed that the calibration of the camera was done properly. Moreover, not all variations were examined, since the individual estimations already allow a good estimation of the results. The ratings of the table have the meaning *dash* for bad results, *circle* for acceptable results and *plus* for good results.
As shown in the table, the resolution of the camera has the greatest effect on the result. At a low resolution, an acceptable result can be achieved with a large marker, while at a higher resolution, slightly smaller markers allow a good result. The best result, with an optically very well-fitting 3D pose estimation, was achieved with the resolution *1280px x 720px* and a marker size of *50mm x 50mm*. Therefore, this constellation was used for the dataset evaluation in chapter 4.2. Since the detection window is derived from the projected 3D pose and is therefore in a qualitative relation, no separate evaluation was performed for this.

4.1.2 Distance estimation

This chapter examines the accuracy of the previously mentioned distance estimation is. The proposed setup from chapter 4.1.1 was used for this purpose. For the experimental setup, the camera was set to the height of the marker, so that the distance is not distorted by a vertical shift. To avoid distortion of the results by a horizontal shift, the marker was always aligned so that the center of the marker corresponds to the center of the camera image. The marker was then measured in 100 mm increments, as shown in table 2 *Real distance (mm)*. It is assumed that a proper camera calibration was performed.
During the experiment, it was observed that the distance estimation did not show a fixed value. Rather, this value varied from frame to frame, so that the maximum and minimum estimated distance were recorded. Even the minimum and maximum values show that the estimate can be distorted by about one centimetre.
In summary, the distance estimation is quite good, even if the label information is not strictly suitable for critical applications that need to accurately determine the distance. If necessary, an exact determination of the distance should be done by a more robust solution anyway. For simpler scenarios such an estimate is quite enough.

Table 1: 3D position estimation in the context of camera resolution and marker size

Cam. resolution	Marker size	Ranking
640x480 px	15x15 mm	-
640x480 px	25x25 mm	-
640x480 px	50x50 mm	o
1280x720 px	15x15 mm	-
1280x720 px	25x25 mm	o
1280x720 px	50x50 mm	+

4.2 Evaluation of the generated dataset

In this chapter, a dataset generated by the tool is tested in practice. This includes the preparation of the dataset, the training of a YOLO detector and the discussion of the results.
The stackable box problem was chosen as the experimental scenario. A total of 449 la-

Table 2: Evaluation of distance estimation

Real distance (mm)	200	300	400	500	600	700	800	900	1000
Min. estimated distance (mm)	197	298	399	498	597	698	789	890	991
Max. estimated distance (mm)	203	302	401	501	600	703	810	908	1008
Average difference (mm)	0	0	0	0,5	1,5	0,5	0,5	1	0,5

belled images were recorded for this experiment using the setup suggested in chapter 4.1.1. This preparatory work took a little over an hour. This step includes the mounting of AprilTags, configuration of the tool and the recording itself. This dataset contains the four classes mentioned in chapter 3.1.2.

4.2.1 Prepare dataset

As mentioned before, the dataset is stored in a way that is independent of the framework. Therefore, in this step the dataset must be transferred to the structure that YOLO can understand. The basic definition of the training is a fundamental task and is not relevant for this paper. More interesting is the transformation of the dataset itself, YOLO needs the data in a folder with image and label pairs each. The image can be easily adopted, but would have to be resized depending on the configuration. However, YOLO cannot handle the label information from the developed tool, since this framework only needs the detection windows with the respective classes.
The required information is available in the generated dataset, so it only needs to be parsed. As part of this experiment, a parser was built that needed about three to five seconds to parse the 449 labelled images. The preparation of the dataset was therefore quick and easy, since the JSON format is simpler to process and the expected YOLO format does not require any special things.

4.2.2 Training with the dataset

After the training configuration and the dataset preparation, the training could be done with Darknet. With the *YOLOv2*[2] model and the 449 labelled data, the training has reached an average loss of *0,28* after about six hours of training with an *Intel Core i7-6700K* and *Nvidia GeForce GTX 1080 Ti*, which is a pretty good result. Darknet itself aborts the training as soon as the average loss dropped below *0,3*.

[2]The YOLO models are presented in `https://pjreddie.com/darknet/yolo/`.

4.2.3 Training results

After the training was completed, YOLO was able to detect the four classes, which are shown in figure 5. In addition, an attempt was made to illustrate how well *YOLOv2* was trained. Of course, the quality also depends on the framework itself. Since the developer's paper has already shown that the detection of YOLO can provide pretty good results, it is interesting to find out how YOLO performs with the self-generated dataset.

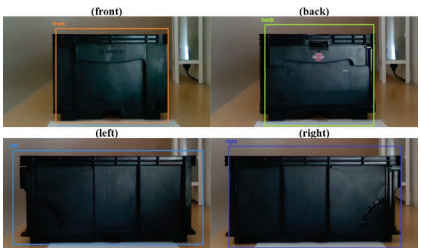

Figure 5: Label definition in YOLO.

A total of 2000 frames, with 500 frames per class, were processed with YOLO for the experiment. The object was rotated, placed further or closer to the camera and exposed to changing light reflections during the experiment. Table 3 shows the result of the experiment. The detection of the front and back sides has achieved a pretty good detection rate. However, as soon as the light conditions became too poor, causing the object to be parti-

Table 3: Detection results of YOLOv2

Frames with	Correct	Wrong	Not
Front class	92,4%	2%	5,6%
Back class	93,8%	0%	6,2%
Left class	63,4%	13,88%	22,72%
Right class	50,4%	31%	18,6%

ally overexposed or underexposed, the object could not be detected. Whereby light conditions are meant, under which the human eye already gets problems to determine the object. The false detections occurred as soon as the essential features of the class, the raised font on the front side and the red logo on the back side, were not recognizable. The detection of the left and right side did not perform this well, as these two classes are more susceptible to light conditions. The main reasons for this are the weakly recognizable features and indentation on the left and right side, see figure 5 *(left)* and *(right)*. As soon as the object was slightly over- or underexposed, this feature could not be detected, so that the object was partly not or incorrectly detected.

5 Conclusion

In this paper an approach for a semi-automated labelling tool was presented, which was implemented in the context of this work. The tool achieved a very accurate labelling result with a camera resolution of *1280px x 720px* and a marker size of *50mm x 50mm*.
The practical test using a dataset of 449 labelled images of stackable boxes to train a YOLO network resulted in a fairly good result. The evaluation revealed that the marker had to be attached in such a way that it did not cover any important features of the class or object. The whole procedure for creating a detector, without an existing dataset, took about seven to eight hours along with the training. For this purpose, a small parser was developed, which converted the independent dataset structure into a structure understandable by YOLO.
In the future the tool could be extended by the approach of [4]. A detector could be trained in parallel, which also performs detections during labelling. The markers could be used to identify false defects and simultaneously create a dataset with negative samples.
In summary, it can be said that the use of AprilTags is suitable for semi-automated labelling. Scenarios such as the identification of living creatures or other objects that cannot be prepared with markers are not suitable for this approach. For scenarios such as the detection of visible sides of stackable boxes, this approach is very well suited.

Literatur

[1] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011.

[2] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. volume 47, pages 2280–2292. Elsevier BV, 2014.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[4] D. Lopresti and G. Nagy. Optimal data partition for semi-automated labeling. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 286–289, 2012.

[5] A. Richardson, J. Strom, and E. Olson. Aprilcal: Assisted and repeatable camera calibration. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013.

[6] A. Telea. An image inpainting technique based on the fast marching method. volume 9, pages 23–34. Informa UK Limited, 2004.