Technische Universität München

TUM School of Management

KEWAZO GmbH
www.kewazo.com

Android application development for processing the output data of computer vision and machine learning techniques for segmentation of geometrical primitives on RGB images

Interdisciplinary Project (IDP) SS18

Professur / Professor:	Prof. Dr. Nicola Breugst Professor for Entrepreneurial Behavior Arcisstr. 2, 80333 München
Eingereicht von / Submitted by:	Juan Du Jochbergweg 1, 85748 Garching bei München Matrikelnummer: 03680524 4th Semester
	Bilal Ahmed App 0302, Schröfelhofstraße 14, 81375 München Matrikelnummer: 03691688 3rd Semester
Betreuer / Advisor:	M.Sc Stefanie Federl
Praxisbetreuer / Advisor of Business Partner:	Eirini Psallida, Sebastian Weitzel
Anmeldedatum / Starting date:	02/04/2018
Abgabe am / Date of submission:	20/09/2018

Contents

1 Introduction

Construction automation has shown vast potential to increase construction productivity and boost profits. By focusing on the scaffolding process, a robotic system has been developed, providing an effective, cost-efficient and safe transportation of the building components during the scaffolding assembly process. The robot has been built by KEWAZO and is the 2nd iteration of the model. Currently, it is capable of moving up and down along pre-installed rails to the scaffold. In addition, the robot has integrated the function of delivering real-time data, which will allow data collection and control optimization.

In order to optimize further the processes in the construction industry it is important to know the number and type of scaffolding parts located in the storage space. Computer vision, as a substitute for human vision, provides a reliable way to replace the mundane counting work and to advance the uptake of automation of the existing logistics. In addition, the raw computing is now at the point where reliable vision algorithms can execute in real time on embedded computers, like an Android smartphone which most people already have. So it would actually make sense to develop an application, that assists the worker in figuring out the number of scaffolding parts quickly and accurately.

2 Objectives and goals

2.1 Business Requirements

The main objective of this project is to develop an application that is able to count objects of interest in images. The final product will be provided to users of KEWAZO robots as an assistant tool to reduce manual labor and increase work efficiency.
The application consists of two main components.

The Computer Vision Algorithm: The back-end part is responsible for processing and analyzing the input image and return a qualitative evaluation of number of the objects to the front user interface. The solution would comprehensively center on the computer vision algorithms and image theory.

The Android App: The app is there for the user to interact with, as well as select the Source and Template, communicating with algorithm and returning the results. Users should also be able to store and retrieve previous results. The front-end should be as user friendly and intuitive as possible.

Regarding the development plan, we chose the Iterative Model as our development cycle model. Iterative development starts with a simple implementation of a small set of the software requirements, which iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed. In our case, since the requirements for

this projects are initially fairly loosely defined, an iterative process would make sense as it should be able to adapt best to any late changes in requirements.

2.2 Technical Requirements of Computer Vision Algorithm

The counting of a specific type of an object in a given image can be considered as a detection problem [7], in which the fundamental task is to find and localize a given object where the image of the objects may vary in different viewpoints, sizes or even when they are translated or rotated. Many approaches to the task have been implemented over multiple decades.

To make sure the application can work efficiently and reliably, it is essential to design an optimal algorithm scheme for this specific problem. In our case, reliable algorithms must fulfill the following import requirements:

- Generality: Some algorithms or models are designed specially for one type of object. Although it usually shows high accuracy, it also means that these algorithms may underperform when applied to other types of objects. At the present stage, we expect our algorithm should handle six types of building components (see Fig. 1). Furthermore, the algorithm should have a proper level of generality considering the potential requirements of new types of objects from the users.
- Robustness: The algorithm should be robust with respect to noise and image variations, i.e. its performance should not degrade in case of slight variations of the object shape, poor image quality, changes in the lightning conditions etc.
- Speed: As the applications may be designed to work in real time, it implies certain demands on the speed of the image recognition process, which is expected to be completed within a few seconds.
- Accuracy: Accuracy is the core requirement of the algorithm. It would only make sense if the algorithm can return an accurate estimation of the number.
- High Automation: The process should be automated, i.e., it should work reliably without too much additional user interaction.

2.3 Technical Requirements of the Android App

The Android App is required to bridge the gap between user interaction and the algorithm.

- Compatibility: The App should be able to run on a wide variety of mobile platform, both software wise (maximum Android OS penetration), and hardware wise(running on all types of cpu architectures armv7, armv7, x86, MIPS).
- Performance: The App should not take more than 0.5 seconds to startup and should not take more than 2 seconds for image detection to take place.

- Security: The features in the app should only be available to registered and verified users.
- Usability: The Application should be intuitive and user friendly for the user to be used, such that within 5 minutes, an untrained user should be fully able to grasp the usage of the application. Object recognition should also not take more than 5 clicks/taps.

3 Work packages and Milestones

3.1 Data collection and Analysis

The performance of an algorithm can heavily depend on the data that is applied to. Since this project is exclusively designed for users of KEWAZO robots, we find it necessary to create our own specialized dataset to develop a targeted and practical approach. We collected our dataset by taking pictures with smartphones on-site in a building warehouse. The resulting dataset is approximately in the size of 100 and consists of six categories, namely Aluboden, Bordbrett-I, Bordbrett-II, Riegel, Stahlboden and Standard. Some samples are presented in Fig. 1.

The next step is to explore our dataset and analyze its feature which can help us to make more informed decisions on the selection of algorithms. Several useful observations can be made as follows:

- The number of objects can vary significantly from a small size of less than 10 up to a larger number of more than 100. This can also happen in the same category.
- The shapes and textures of most building components are not complicated.
- Usually the objects are stacked up in an organized way. This is because most types of scaffolding material are used as a load-bearing base composed of batten or board decking components.
- The shape and appearance of the same object are profoundly viewpoint-dependent. Although we can suggest users capture pictures from the direct front to have a minimal distortion impact, sometimes it is hardly achievable because of the location of the items in real life scenario.

3.2 Research about computer vision algorithms

Our next step was to study algorithms that are suitable for the project. Object detection is a fundamental problem in computer vision and there has been massive work and research devoted to this area. Basically algorithms can be categorized into three methods: feature-based methods, template-matching methods, and learning-model methods. Combining the dataset feature and the user requirements, we decided to take Hough Transform [3] as a good starting point. This method uses a voting procedure from which object candidates are obtained as local maxima in a space constructed by the algorithm.

(a) Aluboden

(b) Bordbrett-I

(c) Bordbrett-II

(d) Riegel

(e) Stahlboden

(f) Standard

Fig. 1. Dataset Samples

(a) Quantity

(b) Viewpoint

(c) Shadow

Fig. 2. Potential Difficulties

Compared with other methods, Hough Transform has several key advantages: To begin with, it is conceptually simple and take less computation effort especially applied to objects with simple shapes, which means less running time and a better user experience. Furthermore, it is both powerful in dealing with noisy and occluded data and flexible enough to control the level of precision by setting a tolerance value that describes how big the maximum deviation from the requested geometric properties. Lastly, it is easily adapted to many types of forms which allows us to apply the same scheme to different group of objects with little effort.

3.3 Implementation of GHT Algorithm

The classical Hough Transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, etc. We first tested the classical algorithm on Standard, the simplest type in our dataset which can be basically represented as circles. The running time and accuracy both met our expectations, which naturally led us to the next step, the implementation of Generalized Hough transform [1]. The generalized version can be used to detect arbitrary shapes which do not have a simple analytic equation describing its boundary. The key of generalization is the use of directional information. Given any shape and a fixed reference point on it, instead of a parametric curve, the information provided by the boundary pixels is stored in the form of a look-up table in the transform stage. We applied our implementation to other types of building components and received good results under the condition that the parameters required by the algorithm were set to proper values.

3.4 Research and implementation of Native code in an Android enviroment

Once it was decided that the General Hough Transform Alogrithim was to be written in C++ (native), we started researching on how to implement native code. Android apps are written in Java so we had to find a way to integrate our native code without rewriting it. Our reason for not rewriting the CV algorithim in Java was that Image recognition is a very computationally demanding process. Smartphones are not as powerful as full fledged desktop computers so we decided to inconvenience ourselves a little and use C++ code as it is (as it is the most speedy and efficient programming language for graphics work), if we wanted to reach our target performance requirements.

We found out that Google provides a set of libraries NDK (Native Development Kit) and support for a build tool CMake to work alongside the Android build tool (Gradle). We also to write a JNI (Java Native Interface) which acted as sort of a 'bridge' between the Native and Java code. Apart from setting up and configuring the NDK, we also had to integrate a set of Computer Vision libraries called OpenCV for Android that contained the functions used in our GHT algorithm.

We started work on our Android app by selecting a mimumum SDK of 19 which meant that our app would work on approximately 90% of smartphones [4]. We managed to successfully implement the algorithim taking a source and template image provided by the user.

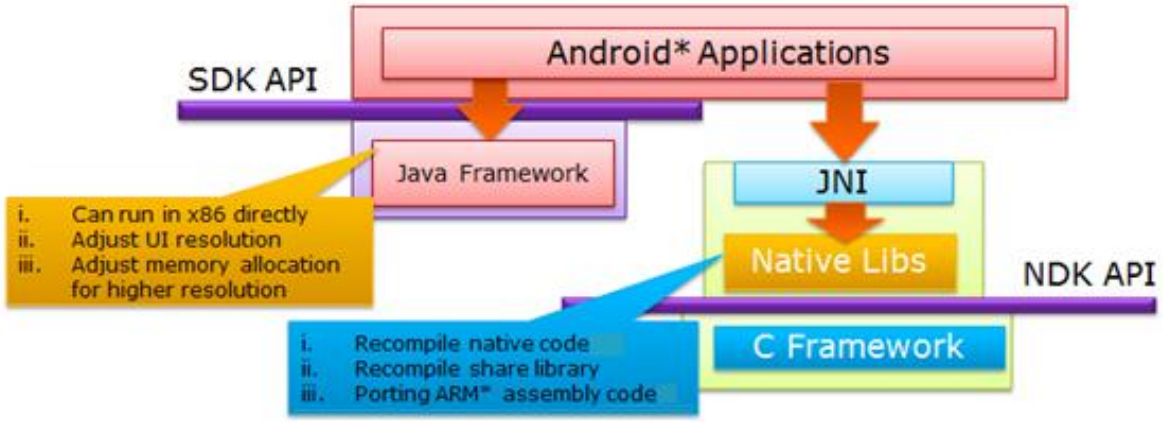


Fig. 3. JNI Libraries [6]

3.5 Trade-off between automation and user control

Following the iterative process, we presented the experiment results to our supervisor and discussed which requirements were well satisfied and which still need extra improvement. During the discussion, we identified two main problems. The Hough transform method only performs well under the condition that the shape of the objective item is a prior knowledge. Because the number of types is limited to seven, in our first version of implementation the solution was storing all the shape templates in different scales and viewpoints in advance and then selecting the best match when executing on some specific image. In this way, the degree of automation was maximized as the algorithm itself took full responsibility and required no user input. However, the drawbacks came along as well. Not only more computation was consumed in this selection phase, it also yielded a lower accuracy rate of counting due to the variation between the detected object and the predefined template.

Another issue was about the dependency on manual adjustment of some parameters to reach good results. The adjustment is necessary because different types of objects have different computation complexity and therefore different thresholding tolerance. The basic version of implementation relied on manual adjustment by proposing them an interactive interface button. Unfortunately, this requires a specific knowledge which most of our target users lack and therefore is better to be accomplished by the algorithm side.

After discussing and evaluating the user ability and preference, we decided to take a balance point in terms of automation. On one hand, it is not difficult for a human to identify a proper template in the detected image, which can be carried out by simple interactive operations through the front-end interface. Conversely, the adjustment of parameters should be automated by optimizing the algorithm scheme.

3.6 Optimised Implementation and final framework

The optimized version was integrated with a pre-processing procedure and a post-filtering procedure. The pre-processing phase consists of several key image enhancement techniques [8] including contrast stretching, color conversion, and Otsu thresholding. Experiment results revealed that these operations could make a significant difference to our final results by converting the image into a form which is better suited for machine interpretation. Another improvement came from the post-processing procedure. We used our previous version to generate region proposals, which were a set of bounding boxes potentially contributing to the count number. Then we performed a series of dedicated operations which include non-maximum suppression, score evaluation, and double threshold to filter out the false items.

3.7 Updated Android App and additonal features

Upon completing the optimzed algorithim, it was integrated it into our app again. This time we were getting much more accurate results. Once we had completed the main feature of the app, our attention went towards developing some extra features like an authenthication login page (to prevent unauthorized use of our app), cosmetic changes, and connection to the KEWAZO web server where results from the algorithm along with some additional data (like date taken, location, part category) were also stored.

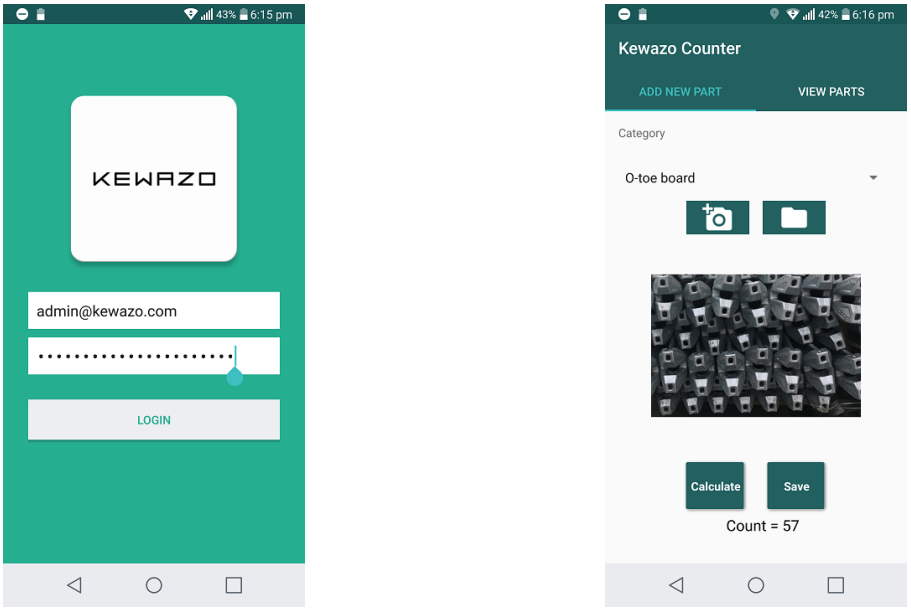


Fig. 4. Login screen (left) and Main screen (right)

4 Team coordination

Our development team composed of two main members responsible for the Android App and the Computer Algorithm respectively. To boost the efficiency of the whole

project, we have followed some principles to build a good communication and coordination throughout the project.

Clear priorities and timelines To accelerate the development process, the core functionality was first developed as a C++ project on a Unix machine. This allowed a proof of concept to be created early on in the development stages and gave a more controlled atmosphere for testing. Once the basic implementation had been completed, the migration from Unix to Android was reasonably straightforward. And the front-end design and the back-end optimization were carried out simultaneously.

Meetings on regular basis Regular meetings were held between our team and KEWAZO supervisors to discuss the progress and further tasks. We made sure that both members attend each others iteration ending demo of our work product in order to know in a broad sense what the other is working on.

Timely communication At the beginning of our project, we set up a messaging channel in the Slack, which allows for immediate communication and simple file sharing during the whole project.

Feedback from project supervisors and clients The two supervisors from the KEWAZO company played a vital role in deciding how to reach a balanced point between the automation (the back-end) and manual adjustment (the front-end).

API Documentation The back-end developer kept an up-to-date, accurate documentation for the API. This meant that content could be driven into the front-end application without any back-end framework in place.

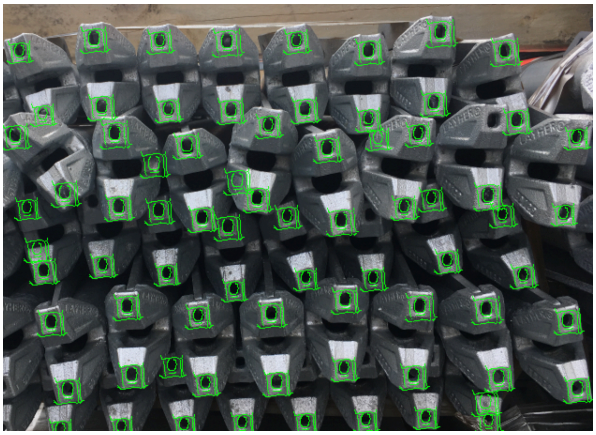
Regular code commits Both the Android App and the CV algorithm had their own code repositories which all parties could access at any time to see the progress on the work. Also in case of faulty changes or data loss, the project could be reverted to a previous working state.

Publishing app versions after each feature After each major feature change, we would publish the application on a distribution platform called HockeyApp which allowed KEWAZO to test the app without the effort of building and compiling the code. This allowed for immediate feeback and bug reporting.

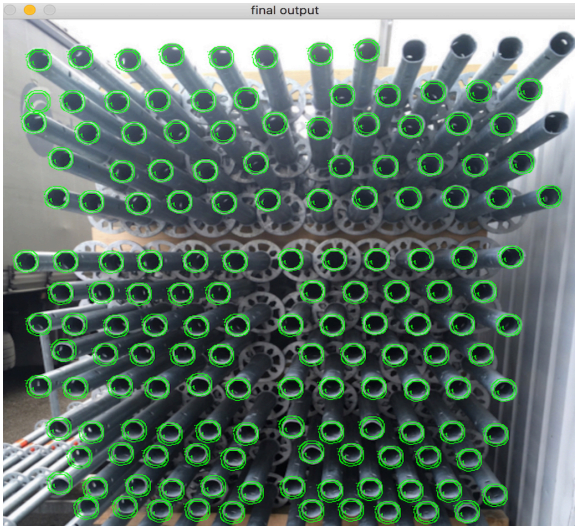
5 Decision about the Technologies

Prior to the actual development work, we held a team meeting to discuss the requirements and thoughts from both the user interface developer and the algorithm developer.

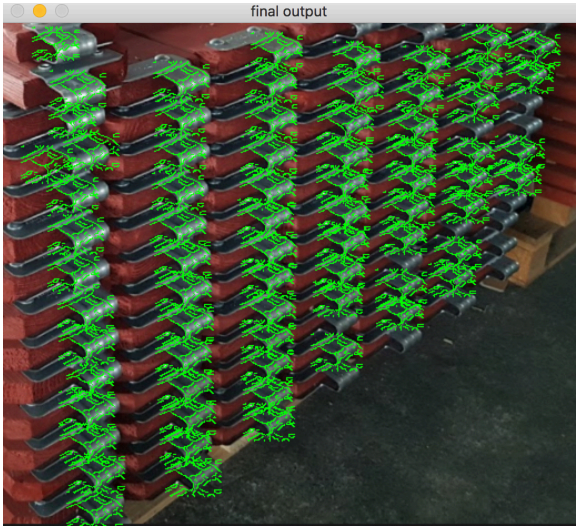
(a) Aluboden



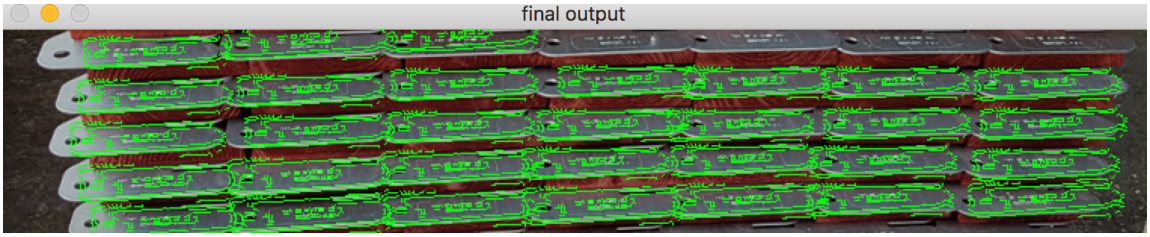
(b) Riegel



(c) Standard



(d) Bordbrett-II



(e) Bordbrett-I

(f) Stahlboden

Objects	result / true number
Aluboden	24 / 24
Riegel	34 / 32
Standard	152 / 156
Bordbrett-II	81 / 86
Bordbrett-I	31 / 35
Stahlboden	38 / 38

(g) Accuracy

Fig. 5. Result Samples

Combined with our clients need, we successfully agreed on some decisions on the tools and technologies we were going to exploit in our later development.

Opencv library [2] OpenCV is an open source computer vision library, free for both academic and commercial use. It supports multiple platforms, including Android, and has a Java interface. OpenCV provides a substantial set of computer vision algorithms, based on an easy-to-use framework.

Programming Language Regarding the programming language, the algorithm was implemented in C++ in view of two reasons. The main limitation came from the OpenCV library, the newest version of which has all packages and interfaces written in C++. Moreover, C++ is usually more performant than the dynamically written languages because the code is type-checked before it is executed.

Android platform As for which mobile platform to develop for, we chose to develop for Android rather than iOS because according to our initial survey, majority of our clients were more likely to use Android devices, thus we opted to build for Android platforms. We also used various third party libraries for features such as image cropping and REST commands [9] [5].

6 Conclusion and Future Work

Tests have proved that our current algorithm performs well on six types of building components in terms of speed, accuracy, and generality. It is also possible to be extended to support more types of objects with little effort and adjustment. A future enhancement to the algorithm might be to figure out some technique to eliminate the step of letting the user select a template in order to make the application work completely automatically, or real-time image recognition without taking a source.

References

[1] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[2] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobbs journal of software tools*, 3, 2000.

[3] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[4] Google. Distribution dashboard. Retrieved 15 September 2018, from https://developer.android.com/about/dashboards/.

[5] Google. google/volley. (2018). github. Retrieved 15 September 2018, from https://github.com/google/volley.

[6] Intel. Intel software android* application development and optimization on the intel atom platform. Retrieved 15 September 2018, from https://software.intel.com/en-us/articles/android-application-development-and-optimization-on-the-intel-atom-platform.

[7] PS Khude and SS Pawar. Object detection, tracking and counting using enhanced bma on static background videos. In *Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on*, pages 1–4. IEEE, 2013.

[8] Raman Maini and Himanshu Aggarwal. A comprehensive review of image enhancement techniques. *arXiv preprint arXiv:1003.4053*, 2010.

[9] Arthur Teplitzki. Arthurhub/android-image-cropper. github. Retrieved 15 September 2018, from https://github.com/ArthurHub/Android-Image-Cropper.

Declaration of Authorship

I hereby declare that the IDP report submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the report in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted Here.

This paper was not previously presented to another examination board and has not been published.

Munich, 20.09.2018

Juan Du Bilal Hadid