Warehouse Management Using Real-Time QR-Code and Text Detection

Debjoy Saha*, Ganesh Shiridi Balaji Udayagiri†, Parakh Agarwal, Biswajit Ghosh, Somesh Kumar‡
IIT Kharagpur, West Bengal, India

ABSTRACT

The objective of this project is developing the computer vision tools for efficient inventory management of packages of a warehouse. Packages are identified by unique QR-codes(Quick Response codes) and are required to be matched with the alphanumeric codes of the shelves on which they are kept. Dimensions of the shelves are pre-known. Figure 1 shows the setup available to us at the competition. QR codes are pasted on the body of the packages and alphanumeric codes are put on the shelves. QR-code identification relies on OpenCV(Open-source computer vision) library ZBar which provides quite reliable and robust output. Corresponding alphanumeric code identification is done using deep learning text detection library Tesseract. All modules were integrated into a ROS(Robot Operating System) architecture and the output preserved in a CSV file. In addition to it, we developed an algorithm for robust and precise warehouse management. The novelty of our approach lies in the detection of text in Tesseract computationally inexpensively using pre-known information. In general, this paper can be used as a working guide for text detection using Tesseract under similar conditions. All libraries used are explained in detail.

1 INTRODUCTION

Real-time text detection from visual input is a useful aspect for the development of autonomous robots. Most of the information that we receive from our surroundings is in the form of images. And a major part of those images is text. Text identification and decoding is thus essential for the development of fully autonomous drones. This project concerns with text identification part only. It also concerns with the decoding of another kind of visual input, QR codes which are comparatively in more machine recognizable format, given the standardized representation of these codes. As already explained, the problem which we are concerned with involves a warehouse containing boxes on shelves. We need to make an inventory, listing QR-codes pasted on the objects and matching it with the alphanumeric codes pasted on their respective shelves. The last part of this paper deals with the algorithm for warehouse management. The warehouse management is done using a quadcopter autonomously with an attached gimbal for image stabilization, implementation similar to [1]. We produce all the steps that we took to improve the performance of our algorithm, the test inputs and their results. A basic flow chart of the developed algorithm is shown in Figure 2.

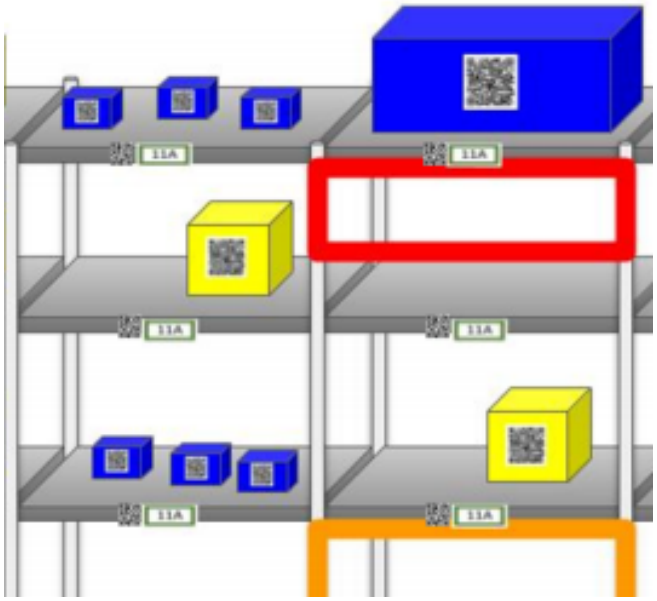


Figure 1: Sample shelf (Source: www.imav2019.org)

2 PRIMARY PRE-PROCESSING FOR QR-CODE DETECTION

Images obtained from the video feed is from a fish-eye camera. A fish-eye camera is used owing to the wide-angle that it captures. The images obtained from this camera are distorted. To utilize the data and properly decode QR-codes and text in the image we have to undistort the images. We use camera_calibration package of ROS [2] and OpenCV [3] to calculate the camera matrix, distortion coefficients and other parameters using multiple checkerboard images at different positions, scale and skew angles.
Having calculated all the intrinsic and extrinsic components,

*Email address(es): sahadebjoy10@gmail.com
†Email address(es): balajiatvizag@gmail.com
‡Supervising professor

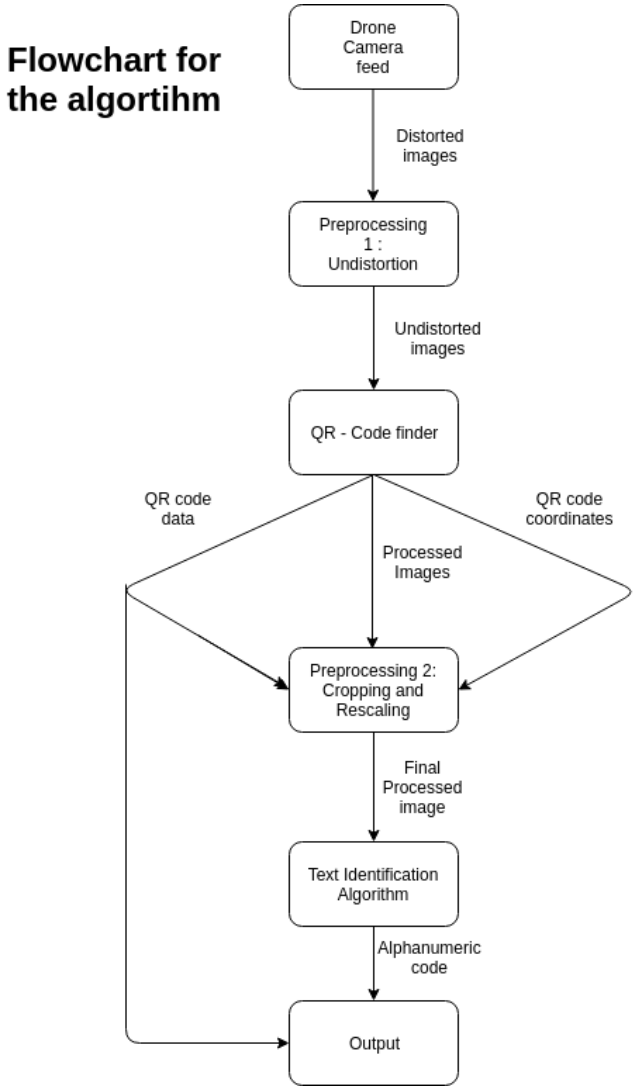


Figure 2: Flowchart for the algorithm

(a) Input

(b) After undistortion

Figure 3: Image before and after undistortion

we undistort the images and publish them for further usage. An example of the undistortion process is given in Figure 3. Some more pre-processing steps are used on the undistorted images using information extracted from QR codes, which are explained in section 4. For now, we skip the remaining pre-processing steps, since they are computationally expensive and not necessary for QR code detection. So, unless QR codes are observed from camera feed, we do not perform those steps.

3 QR-CODE DETECTION

We use the OpenCV ZBar library [4] to detect QR-codes. We pass the images obtained from the undistorted image feed(as referred to in section 2), to the ZBar class object created [5]. It scans the two-dimensional image to produce a stream of intensity samples. The decoder searches a stream of intensity values obtained by processing the images from the video feed for recognizable patterns and produces a stream of completely decoded symbol data from which the QR code text are decrypted. We store the obtained QR-code data and its coordinates for further usage and move on to Alphanumeric code detection.

4 ALPHANUMERIC CODE DETECTION

The most challenging hurdle to the task was detecting the alphanumeric codes accurately. For this task we used Google's open-source library Tesseract[6]. Tesseract is an OCR(Optical character recognition) engine with support for Unicode and the ability to recognize more than 100 languages. It can be trained to recognize other languages. Tesseract is efficient only in the case of extremely well defined text in a page (like we obtain for a .pdf (portable document format) file). It shows quite poor performance for detection of text from noisy images, such as those obtained from a drone camera. Parameters need to be properly tuned and images properly pre-processed to produce optimal output. To obtain this, we follow a step-by-step approach. First, using information extracted from the QR-code (center and dimensions), we attempt to further process the input image and create more identifiable images. Then we apply Tesseract text

recognition on it. We list the various parameters that we modify to obtain perfect solution. Finally, we propose a method to further improve the result obtained using Tesseract.

4.1 Secondary Image pre-processing for alpha-numeric code detection

After the initial pre-processing step (section 2), further processing needs to be done on input feed for better detection of alpha-numeric characters. The methods mentioned below helped improve performance.

- Rescaling: Tesseract gives useful results with specific text size in the image. To improve the result obtained by Tesseract, we need to resize the image appropriately.
- Cropping desired area: Cropping reduces the ROI(Region of interest), thus improves the text detection by increasing confidence in the prediction.
- Blurring: The noise in the image gets reduced and results thus get better.
- Thresholding: It can be used to binarize the image to improve the performance and eliminate shadows. [7].
- Rotation / Deskewing: The image could be rotated to align the text which helps gain better results. [8].

Out of the listed methods, only rescaling and cropping improved output quality, so we developed algorithms that used those methods appropriately.

4.1.1 Cropping the image

For the text recognition to work well we observed that the results are better when the domain on which search for the text has to be done is smaller. The sides of the image is cropped to reduce the search area. We intend to crop our image with respect to the QR-code. We do not crop the image from top or bottom as this may lead to loss of information of the text and QR-code. Our cropping assumes the following:-
1. Both the QR-code and the alpha-numeric code occur in the same frame.
2. The QR-code is positioned along the left or right edge of the shelf and the alphanumeric text is positioned at the center of the shelf.
3. Only one shelf was detected per image.

Assumptions 2 and 3 were taken as limiting cases of the alphanumeric detection problem. Our objective is to ensure that the qr code and text both appear in the final cropped image irrespective of their relative separation. Since we know the alphanumeric text is placed at the center of the shelf, the position of maximum separation is if the QR-code is near the edge. Also, the maximum separation in pixels can only be obtained if we consider one shelf per image.
Consider the length of the label of the alpha-numeric code be a and the length of the shelf be L. Position of the center of the QR-code label be x. And the length of the QR-code is l_q. The part of the image in between the position x. (refer to Figure:4)

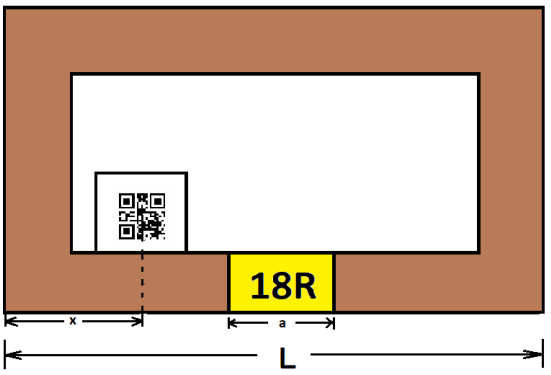


Figure 4: Showing the parameter of the shelf

Distance between farther ends of QR-code and alphanumeric code -

$$d = ((L + a)/2) - x \tag{1}$$

When $x = 0$ (The extreme case),

$$d\_max = (L + a)/2 \tag{2}$$

This implies that in a d_max neighbourhood of the QR-code, the alpha-numeric code exists. Now, since the image is undistorted, we can directly compare the ratio of distances to obtain the part of the image we need to retain. The ratio we obtain is

$$k = (L + a)/(2 * L) \tag{3}$$

In our current problem, we find this ratio to be 0.56 which we approximately 60%. So we crop of 2/5th of the distance of both ends of the image, measured from the center of the QR-code. So, in the final cropped image, we retain 60 % of the frame on either side of the QR-code. Using this we can be sure that the alphanumeric text is detected and at the same time reduce the region of interest considerably.

4.1.2 Re-scaling the image

The image obtained after cropping does not always yield good results. This reason is attributed to the size of the the text in the image, which has to be close to a particular value for best results. For proper detection, therefore, we need to estimate the text size before-hand to scale the image accordingly. There are many machine learning frameworks for detection of text regions but usually, they are computationally very expensive. We observed that the text recognition occurs optimally at a particular text size in the image (174 X 74 pixels using the test drone). This text size in the image occurs at a particular distance of the camera (here the drone) and the

object that contains text (here the shelf). When the camera is at some other distance, the text size in the image is different from the optimal size and the output is not desirable. We propose a method for text size approximation in the image that resizes the image using the information from the camera matrix [9] (obtained through camera calibration) [10, 11] and the distance of the shelf from the drone. The drone is maintained at a constant distance from the shelves using a depth estimate from a stereo camera mounted on parrot bebop drone. We make use of known text frame size (on the shelf) and the depth at which it occurs. Our objective in this method is to find the scaling factor by which we would scale our image.

Camera matrix =

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where fx and fy are focal-length times a scaling factor.

$$fx = f * mx \tag{4}$$

$$fy = f * my \tag{5}$$

(where mx and my are the scaling factors along corresponding axes). In the camera we used, the scaling factors along both axes are the same hence we take the focal length f as the average of fx and fy.

$$y' = y * f/z \tag{6}$$

$$y'' = y' * m \tag{7}$$

where,
y' = object_size_in_image_sensor
y = object real size
f = focal length of camera
z = distance from object containing text
y'' = object size in pixel
m = pixels per millimeter

Final step is re-scaling by a factor such that the final output image used for detection has optimal text pixel size. Scaling by:

$$k = a/y'' \tag{8}$$

a = optimal text size in image(in pixels) = 74 px
k = scaling factor

To re-scale, we use OpenCV's [3] built-in re-scaling function and the obtained ratio k. The representation of the algorithm is provided in Figure 5.

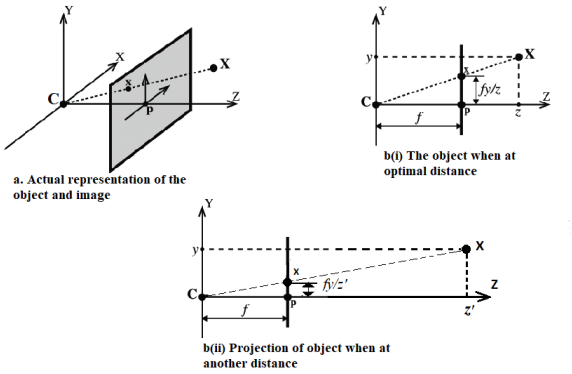


Figure 5: Re-scaling : The image (a) shows the actual representation of the object in 3-dimensions. The image b(i) shows the projection of object on the image. And the character recognition at this distance of the object is optimal. (ii) shows the projection of object on image at another distance that is not the optimal distance for character recognition. Our objective is to process this image so that it gives optimal output for character recognition, i.e. achieve situation(i).

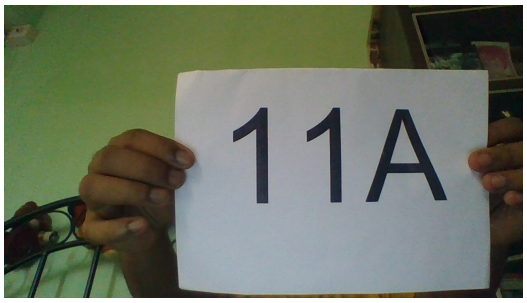


Figure 6: Sample testing image

4.2 Improving Tesseract performance

Furthermore, Tesseract contains many intrinsic parameters whose values can modify output results as well as its frequency. In the following section, we list those parameters and their observations one by one. All provided results are sampled from a larger output of the algorithms, as tested on sample Figure 6, therefore they are only indicative of the actual results and the effects observed on modifying those parameters. The final results are provided in section 6.

4.2.1 Setting appropriate Page segmentation method

Page segmentation mode(psm) is the first of those parameters. By default, Tesseract expects a page of text when it segments an image. How Tesseract will read the page will depend upon how it segments the image. And we can specify that using psm. Tesseract provides 13 supported page segmentation modes for different scenarios. PSM_SINGLE_WORD

(i.e Treating the image as a single word) was considered due to better performance at detection over other methods, PSM_SINGLE_LINE (i.e treating the image as a single text line) and PSM_DEFAULT.

S.no.	PSM_SINGLE_WORD	PSM_AUTO
1	11A	
2	11A	11
3	11A	114
4	11A	41h
5	11A	11A

Table 1: Results for different page segmentation modes

Increased frequency and accuracy of prediction, given predefined knowledge as to the type of text present (Refer to Table 1).

4.2.2 Disabling Tesseract built-in dictionaries and including custom word-lists and patterns

Modifying the load_system and load_freq parameters allows us to enable/disable Tesseract dictionaries, which are responsible for producing output close to the general dictionary words. Disabling the dictionaries, Tesseract should increase recognition since the text we need to detect isn't dictionary words but alphanumeric codes having a specific pattern.
Common character patterns were added (/d/d/c for digit, digit, character) to further trim down image output to more accurate results. It should be noted that this method only increases the probability of correct predictions. We need to perform a further trim(section 4.2.5) to ensure it. The following table summarizes the improvements.

S.no	Using Custom word-patterns	Default
1	11A	141A
2	J1A	J1A
3	11A	111A
4	11A	A1A
5	11A	1A

Table 2: Custom word pattern results

Notice the improvement in detection of both digits instead of just one (row 5) and of correct text-pattern (row 4) (refer to Table 2).

4.2.3 Using White-list of characters

White list is another parameter which can be assigned the value of a string containing all characters which we want to be used for recognition purposes. Since we are sure to encounter only single alphanumeric codes for detection, we can safely omit special characters and blank spaces from the list of characters. Thus we can add alphabets and digits under Tesseract white-list, only they will be used for prediction of text.

S.no.	With White-list	Without white-list
1	11A	
2	11A	11
3	114	114
4	11A	11&
5	11A	!1A

Table 3: Results for character white-list

Using white-list has suppressed occurrence of special character '!' (row 5) and '&'(row 4) and has improved confidence in prediction (Refer to Table 3).

4.2.4 Using an iterator object to examine sub-strings

Owing to background noise, a lot of stray text is also detected, which is not a part of the text in the input image. To suppress those occurrences, we use an iterator object to scan each string detected separately and select the text sorted on basis of confidence in prediction and word-pattern. We modify the code to involve an iterator object to achieve this result.

4.2.5 Trimming the output further

The next step is parsing the output for the desired result and selecting the most probable alphanumeric code corresponding to a specific QR-code. We run a loop over the obtained text to find for data of type (int)(int)(character) having the maximum number of observations.

Algorithm 1 explains the full algorithm.

Algorithm 1 Overall Algorithm per frame

Ensure: QR code & Text Matching
Input: Camera feed image, Depth
Undistorted_image←Preprocess(INPUT)
(QR_dims, QR_data) ← QR_detection(Undistorted_image)
Cropped_image←Cropping(Undistorted_image, QR_dims)
Resized_image ← Rescaling(Cropped_image, depth)
Text ← Text_detection(Resized_image)
Alphanumeric_Code ← Trim(Text, QR_data)
Output: (QR_data, Alphanumeric_Code)

5 WAREHOUSE MANAGEMENT ALGORITHM

Using the above pre-processing steps guarantees a good recognition using very minimal computation but during use, to reduce any possible errors, we improvised on the motion

of the quadcopter to improve efficiency and quality of results. The improvisations are listed below:

- The current warehouse consists of various shelves, and each of the shelf is used to store various packages. The shelves are further divided into various rows and columns to store the packages. We observed that detection when traversing along columns was considerably slower than when traversing along rows. So a row-wise traversal of the shelves was used finally.
- Also, since Tesseract gives optimal result at only a definite text size, we implement an algorithm, in addition to section 4.1.2, to maneuver forward and backward from the mean position. This to and fro swaying motion perpendicular to the shelf changes the text size by a small amount thus correcting any drift errors in the position of the drone and thus assuring an appropriate output even at an erroneous distance from the shelves. It also reduces the scope of error in re-scaling of the image. An oscillation of 10 cm was observed to produce maximum results.
- Having observed that Tesseract gives better results when the text is on the bottom side of the page, we align it more that way to get a better result using the pre-known position of the QR code from section 3.
- Once a QR-code is detected, we initiate a hover time so that sufficient detection results are obtained thus minimizing error during final processing.
- Also, the presence of gimbal ensures that the QR-code is viewed normally at all times, reducing perspective distortion in text.

6 TESTING

Figure 7: Drone used for testing (Source: www.parrot.com & www.amazon.in)

We used a Parrot Bebop 2 drone (Figure 7) for testing purposes which has a built-in gimbal camera, to eliminate all hardware limitations. The result obtained was quite satisfactory. QR code and corresponding alphanumeric text could be correctly recognized from a wide range of distances from the shelf, depending upon the quality of the camera feed of the drone. Experiments on Bebop drone gave impressive results for distances up-to 3m, after which the camera feed deprecates due to re-sizing (section 4.1.2). The Average computation time observed was low, thus the algorithm was capable of real-time detection. Output frequency and accuracy was sufficiently high, thus the hover time required before each package was low (Results listed in detail in Table 4).

Parameter	Freq(min^{-1})	Acc(%)	Frame-Rate(fps)
None	95	96.8	7.10
Config-File	68	95.4	5.96
White-List	97	42.2	7.31
PSM	11	51	7.21

Table 4: Result: Excluding different parameters

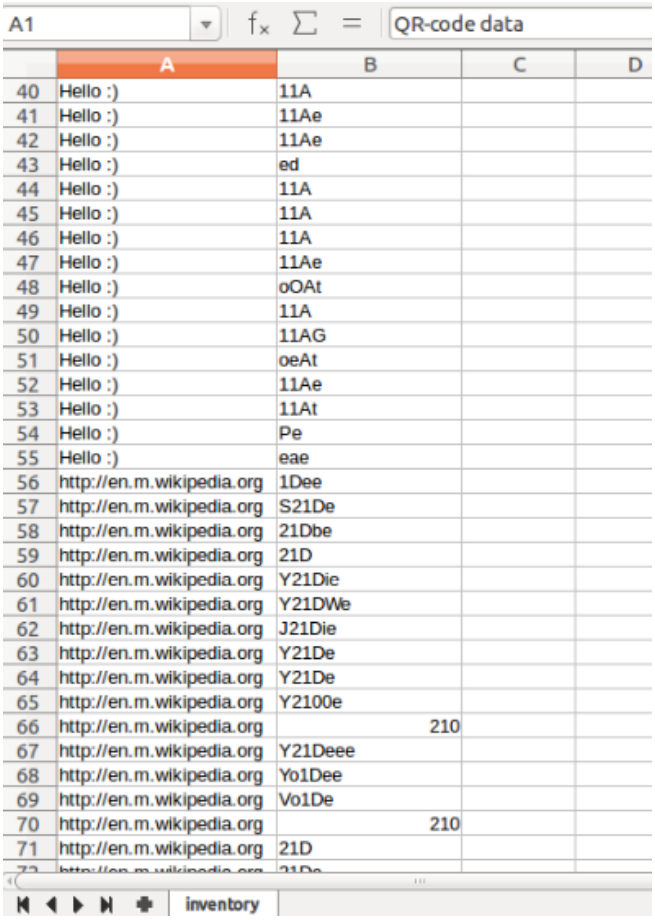
Table 4 lists in detail the effect of the absence of any of the tuned parameters tested in section 4.2. The first column is the parameter excluded. The second column represents the number of correct observation observed per minute. The third column is the percentage of correct observations in prediction. The fourth lists the frame-rate observed during detection. These results are obtained during testing on a desktop PC.

Major takeaways from the results table are -

- An impressive 95 observations/minute were observed on the final algorithm and an accuracy of 96.8 % without the final trimming step (section 4.2.5).
- Removing config-files (specifying custom user-patterns and dictionaries, section 4.2.2) decreases the frame rate as well as the observation frequency. However, on the limited observations, the accuracy observed is quite high (95.4%).
- Removing white-list (section 4.2.3) or changing page-segmentation-mode (psm) (section 4.2.1) has quite drastic results on detection results.

7 CONCLUSION

This article documents all steps and corresponding results of the warehouse management sub-module of the IMAV competition 2019. In this article we have mentioned the ways to optimize the two major modules of warehouse management i.e QR-code detection and alpha-numeric detection.The QR-code detection module is simpler and provides accurate results even with no modifications. However, the text detection module has higher complexity and has to be dealt with



A1 | QR-code data

	A	B	C	D
40	Hello :)	11A		
41	Hello :)	11Ae		
42	Hello :)	11Ae		
43	Hello :)	ed		
44	Hello :)	11A		
45	Hello :)	11A		
46	Hello :)	11A		
47	Hello :)	11Ae		
48	Hello :)	oOAt		
49	Hello :)	11A		
50	Hello :)	11AG		
51	Hello :)	oeAt		
52	Hello :)	11Ae		
53	Hello :)	11At		
54	Hello :)	Pe		
55	Hello :)	eae		
56	http://en.m.wikipedia.org	1Dee		
57	http://en.m.wikipedia.org	S21De		
58	http://en.m.wikipedia.org	21Dbe		
59	http://en.m.wikipedia.org	21D		
60	http://en.m.wikipedia.org	Y21Die		
61	http://en.m.wikipedia.org	Y21DWe		
62	http://en.m.wikipedia.org	J21Die		
63	http://en.m.wikipedia.org	Y21De		
64	http://en.m.wikipedia.org	Y21De		
65	http://en.m.wikipedia.org	Y2100e		
66	http://en.m.wikipedia.org	210		
67	http://en.m.wikipedia.org	Y21Deee		
68	http://en.m.wikipedia.org	Yo1Dee		
69	http://en.m.wikipedia.org	Vo1De		
70	http://en.m.wikipedia.org	210		
71	http://en.m.wikipedia.org	21D		

inventory

Figure 8: Initial output

great precision. We have described various methods of using Optical character recognition. To improve the results, we used undistortion, re-scaling of the input image, cropping of the input image. We obtained a frequency of 6-7 results per second and the correct alphanumeric code was embedded in about 90-95% of identified strings. Refer to Figure 8: Initial Output, without the final trim(steps up to section 4.2.3) and Figure 9: Final Output. The results were quite satisfactory. Given a hover time of 4-5 seconds (modifiable during final testing of the algorithm at the venue), we obtain enough test data to predict the correct QR-code with >90% accuracy on test input. We look forward to reproducing this performance at the competition too.

ACKNOWLEDGEMENTS

We would like to thank our team members and supervisors at ARK (Aerial Robotics Kharagpur) to provide us with valuable insights on this project and all necessary equipments. We would also like to acknowledge Parrot for the Bebop-2 drone that we have used for testing. We would also like to thank the authorities at Indian Institute of Technology, Kharagpur to provide us the funds and support for the presented work.

	A	B	C
1	QR-code data	AlphaNumeric Code	Shelf Code
2	Hello :)	11A	
3	http://en.m.wikipedia.org	21D	
4			
5			
6			
7			

Figure 9: Final output

REFERENCES

[1] A. Anand, S. Agrawal, S. Agrawal, A. Chandra, and K. Deshmukh, "Grid-based localization stack for inspection drones towards automation of large scale warehouse systems," 2019.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Wikipedia contributors, "Zbar — Wikipedia, the free encyclopedia," 2019. [Online; accessed 5-June-2019].

[5] I. Szentandrási, A. Herout, and M. Dubská, "Fast detection and recognition of qr codes in high-resolution images," in *Proceedings of the 28th spring conference on computer graphics*, pp. 129–136, ACM, 2013.

[6] R. Smith, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, pp. 629–633, IEEE, 2007.

[7] Z.-K. Huang and K.-W. Chau, "A new image thresholding method based on gaussian mixture model," *Applied Mathematics and Computation*, vol. 205, no. 2, pp. 899–907, 2008.

[8] Y. Zhang, J. Zhong, H. Yu, and L. Kong, "Research on deskew algorithm of scanned image," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 397–402, Aug 2018.

[9] O. Semeniuta, "Analysis of camera calibration with respect to measurement accuracy," *Procedia CIRP*, vol. 41, pp. 765–770, 12 2016.

[10] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000.

[11] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, pp. 965–980, 1992.