Batch Reading Densely Arranged QR Codes

Binyao Jiang, Yisheng Ji, Xiaohua Tian, Xinbing Wang
School of Electronic Information and Electrical Engineering,
Shanghai Jiao Tong University, China.
Email: {emberspirit,acetanil,xtian,xwang8}@sjtu.edu.cn.

Abstract—**This paper presents BatchQR, a mobile APP that can batch read the densely arranged QR codes attached to caps of the tubes and vials in clinical and biological labs. The basic idea of BatchQR is to detect each code in the image and then decode in an one-by-one manner. However, the unique characteristics of the QR code and the application scenario bring technical challenges: First, off-the-shelf lightweight object detection mechanisms are unable to distinguish those densely arranged codes that are highly similar to each other; second, the focus area of the camera is limited, which blurs or distorts parts of the image. To this end, we propose a lightweight code detection mechanism, which can adaptively adjust operating parameters to identify densely arranged QR codes in practice. We also propose a simple but effective image refocus mechanism, which takes an auto-focused image and multiple refocused ones, and then replaces the blurred or distorted code parts with the high-quality counterparts in the refocused images. Comprehensive experimental results show that BatchQR can read 160-180 Version 1-L QR codes in batch with 90%-95% accuracy in 10-14s, which is only 4% of the time consumed by the regular QR code reader in the same situation.**

I. Introduction

Tubes and vials are the most frequently used equipment in clinical and biological labs for containing chemical or biological samples. The routine lab experiments need a large number of such containers, which must be labeled with the information such as the sample's source, type and infectiousness. In order to efficiently identify and track chemicals and specimens, barcode labels are usually attached to the tube. In particular, barcode labels are attached to the body of the tube or vial, for which standards are developed to regulate how the code should be designed [1], [2]. In order to prevent loss and improve management efficiency, labeled containers are then centrally stored in the holder rack; therefore, the barcode of the container in the inner layer is blocked by those containers in the outer layer, which makes it troublesome to search or select specific ones.

To deal with the issue, the barcode for the body of the tube is normally associated with a small QR (Quick Response) code that is attached to the cap of the tube [3], [4], which makes it possible to identify certain tubes without removing them out of the holder. However, the QR codes attached to those containers' caps are densely arranged as shown in Fig. 1, which are snapshots of real situations in the clinical lab of a local hospital. Although the introduction of the on-cap QR code can facilitate management of those containers, using regular QR code readers is still inconvenient in this particular application scenario. Existing code reading mechanism is unable to read the densely arranged QR codes in batch, instead, the user still needs to scan the code one after another when searching for a specific tube or logging the information.

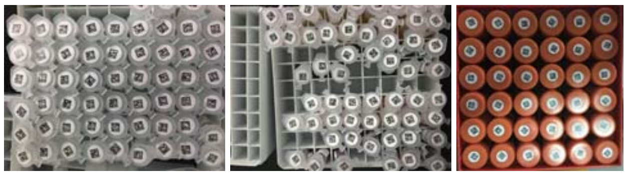


Fig. 1: Densely arranged tubes, a snapshot of real situations in the clinical lab of a local hospital.

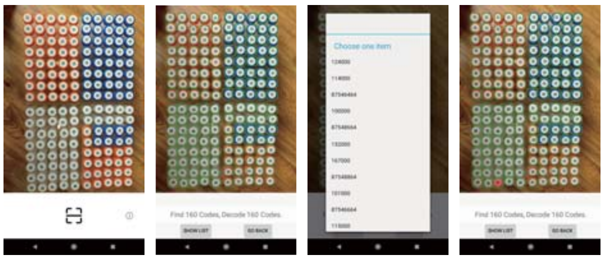


Fig. 2: Screen capture of BatchQR APP, where QR code tags within the bounding box can be decoded successfully. It can also show contents of those codes, and locate the chosen one, which are filled with red color.

In contrast to batch reading, generating QR codes in bulk is much easier [5], [6], where the data can be input in certain format and the corresponding QR codes can be generated in bulk automatically. The ZXing team creates an open-source, multi-format 1D/2D barcode image processing library implemented in Java [7]; the library is termed as "Zebra Crossing", which however is only able to process regular QR codes images in an offline manner in the batch reading scenario, while the field operator dealing with the situation as shown in Fig. 1 requires to read in those QR codes on the spot in an online manner. RFID techniques are also applied to container management [8], [9]: each tube or vial is embedded with a RFID chip or attached to a passive RFID tag; the intelligent holder rack can automatically register the position of the inserted container and the information is transmitted to the cloud database. Nevertheless, RFID embedded tubes are much more expensive than the regular ones, and the entire system is even more costly.

Online handling the situation as shown in Fig. 1, we could extract the codes in the image, and then decode the extracted codes one by one in the background. It is seemingly that the code extraction can be easily realized by applying off-the-shelf object detection mechanisms; however, the unique characteristics of the object and the application scenario factually bring technical challenges:

- Learning-based object detection mechanisms require high-performance computational platform to achieve high accuracy and low latency [10], [11], which can not be satisfied by the mobile device favored by the application scenario; non-learning-based ones detect objects by the shape and color features [12], [13], which however are almost the same for all those QR codes as shown in Fig. 1, thus the approaches usually mistake multiple codes to a single object.
- The focus area of the camera is limited, some subregions in the image containing QR codes therefore can be blurred or distorted if there are many codes in the batch. It is still possible to detect those code areas even in the blurred or distorted part of the image; however, the errors incurred by the low image quality usually go beyond the error correction capability of the standard QR code reader itself, thus the code can not be correctly decoded.

In this paper, we present "BatchQR", an APP that can read densely arranged QR codes as shown in Fig. 1. The screen shots of the APP are as shown in Fig. 2. Our technical contributions in realizing BatchQR are as following:

- We present a QR code subregion detection mechanism, which is dedicated to identify each code in the batch QR codes image (**Section III**) . The mechanism can recognize codes in different sizes by dynamically adjusting design parameters, which accommodates the practical scenario that there are various numbers of codes in the batch. The mechanism is not based on learning but leverages a light-weighted learning based classifier (**Section IV**) , which requires tractable computational resources and presents high performance compared with pure learning-based and non-learning-based object detection schemes.
- We propose a simple but effective image refocus mechanism to deal with the issue of blurred and distorted image in batch QR code reading (**Section V**). We make the camera take another 4 refocused images besides the auto-focused one when reading the batch codes, where the focus of the 4 images are on the 4 corners of the batch codes image. Since blur and distortion usually occur in corners and edges of the batch codes, we could use the high-quality code image in the refocused images to replace the blurred or distorted one in the auto-focused image.
- We implement the BatchQR APP in a COTS smartphone (**Section VI**), and conduct comprehensive experiments to examine the performance (**Section VII**). Results show that BatchQR can read 160-180 Version 1-L QR codes in batch with 90%-95% accuracy in 10-14s. In contrast to the regular QR code reader dealing with the same situation taking about 6-8 minutes, our APP consumes only less than 4% of the time.

II. Background and Related Work

The QR code is a kind of 2D matrix barcode standardized in [14], [15], which has been widely used in the clinical and biological labs for identifying or tracking tubes or vials containing chemical samples or specimens [3], [4]. Another option for the scenario is the RFID based lab management system, which however has been proved expensive [8], [9]. In particular, the price for a typical passive RFID tag is 7-15 cents [16], [17], [18], but a very frequently used 0.5ml microcentrifuge tube as shown in the left part of Fig. 1 costs less than 3 cents [19], where the label is even more expensive than the tube itself. Consequently, majority of the labs are still using 1D/2D barcode in the routine clinical and biological experiments.

To generate a QR code, one needs to first analyze the data to be encoded and make several important design decisions, including the code mode, error correction level and version (size). The character capacity table could facilitate the process [15], which tabulates the amount of characters can be encoded under different possible combinations of those design factors. Then the data could be translated into the codeword, which is a string of bits including the error correction codes generated through Reed-Solomon (RS) coding procedure [14], [15]. Regular QR codes normally contain some special patterns, e.g., the black and white squares in the corners of the code area can facilitate the reader to locate the coding area in a code image. The codeword bits will be placed in the code area after the special patterns are in the positions following certain rules in [14], where black and white indicate bit 1 and 0, respectively. Decoding is basically the inverse process.

Efforts have been devoted to streamlining design of regular QR codes [20], [21]. In contrast, our work in this paper preserves the standard encoding and decoding procedure for the single QR code in order to accommodate the standard QR code printers that have been widely used; the advancement made is to locate and separate each code in a lot of densely arranged codes, which makes the user feel reading those codes as reading a regular one.

To realize batch reading, we propose an object detection mechanism to find the subregions in the codes image, which contain partial QR code. Object detection can be realized using the deep learning based approach such as Mask R-CNN [10] and Faster R-CNN [11], which however normally runs on high-performance servers equipped with GPU, and requires a lot of training to fine-tune the CNN. Moreover, as for non-learning based approaches such as Selective Search[13] and Edge Boxes[12], our preliminary experiments indicate that such approaches are inappropriate for detecting densely arranged objects with similar color and shape. Our proposed object detection method resolves the issue mentioned above, which particularly suits the mobile APP.

In order to identify the falsely-positively detected subregions of the codes image, we utilize the MobileNetV2 [22] to perform classification of all detected subregions. In comparison with the machine learning based approaches such

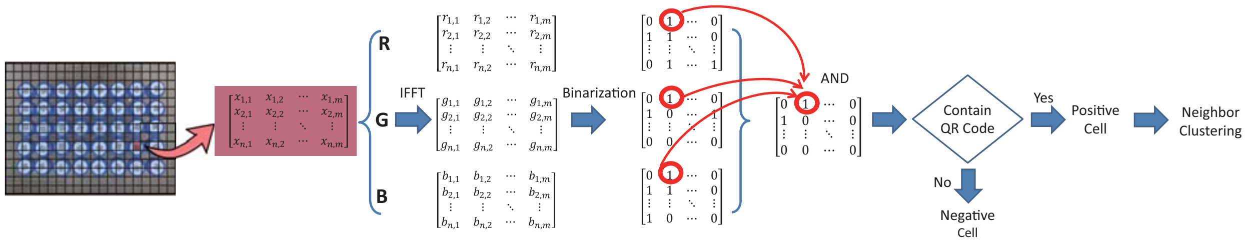


Fig. 3: Processing flow for code detection algorithm.

as SVM, Random Forest, and XGBoost [23], MobileNetV2 can achieve higher accuracy; In comparison with the deep learning based approaches such as CNN, VGG [24] and Inception-v4 [25], which requires high-performance processor and high power consumption, MobileNetV2 is power efficient and fast. MobileNetV2 has a novel layer module taking a low-dimensional compressed representation as an input, which is first expanded to high dimension and then projected back to a linear convolution representation [22].

Our work in this paper propose new techniques and streamline existing mechanisms to present a systematical design to deal with the issue of batch reading densely arranged QR codes.

III. Code Detection

The first step towards batch reading QR codes is to identify the subregion in the image which contains the QR code. We here describe how to frame the QR code area into the bounding box as shown in Fig. 2.

A. Basic Procedure

The basic procedure is shown in Fig. 3. We divide the image into $M \times N$ cells, with the length of each cell set to be d pixels. For each of the cell, we need to determine if it contains a complete or partial QR code. If positive, the cell will be labeled and then clustered with neighboring labeled cells, so that we could roughly frame those codes in the image.

In order to determine if a cell should be labeled, we first decompose the image piece in the cell through *RGB* channels, which yields three matrices. For the matrix obtained through channel *R*, each element $r_{i,j}$ denotes the value of the pixel's red color in the original image. Similarly, $g_{i,j}$ and $b_{i,j}$ denote the values of the green and blue color of the pixel, respectively. It is straightforward that subregions containing QR code fragments present intensive luminance change, as the QR code is consisted of small black and white squares. To make such feature more salient for later processing, we perform *Inverse Fast Fourier Transformation (IFFT)* to the matrix from each channel. It is worth mentioning that we could also choose to perform IFFT to the grey-scale image derived from the original codes image; however, this could result in high false positive ratio, because the intensive luminance change in those *GRB* matrices are basically caused by the black-and-white pattern of the QR code, while that in the grey-scale image matrix could result from many practical environmental factors such as the lighting condition and the color of the tube's cap.

Observe the resulted matrices after IFFT, ideally, great values of the pixels in a given cell indicate the intensive color change, thus it is likely that the cell contains the QR code fragment. However, it may occur that values of certain elements are abnormally high due to unpredictable lighting conditions and camera angles. This will make the value of the corresponding cell abnormally high, which however is not due to the presence of the QR code. To this end, we perform binarization to the three yielded matrices after IFFT. In particular, we set a threshold Th_b to binarize each elements in the matrix.

We now have three binarized matrices representing the *RGB* channels respectively as shown in Fig. 3, and next we will combine the three matrices into one that represents the image of the given cell. In particular, we AND elements that are located in the same position of those matrices, with the result set to be the value of the element in the same position of the combined matrix. If the ratio of 1-elements in the combined matrix is greater than another threshold Th_f, we label the corresponding cell in the original image, indicating that the cell contains at least a QR code fragment. For a given labeled cell, if the four neighboring cells of the cell are also labeled, we cluster those cells and put them into a bounding box. In this way, we could roughly identify and frame the subregion in the image which contains the complete or partial QR code.

B. Self-Adaptive Parameters Configuration

Issues in practical scenarios. If we realize the basic procedure as mentioned above and apply it to the practical scenario, the phenomena as shown in Fig. 4 may appear: In Fig. 4 (a), only a couple of cells are correctly labeled; in Fig. 4 (b), some cells containing the QR code fragments can not be identified; in Fig. 4 (c), some cells do not contain code fragments are incorrectly labeled. The root cause of the phenomena is that the QR code areas in those images are different in size, while the edge length of the grid d applied in those images remains the same.

In particular, the configuration of the two parameters d and Th_f in the basic code detection procedure is the main reason why this can occur. Recall that Th_f is used to determine if the cell represented by the combined binary matrix contains code fragments. The value of d and value of Th_f should be appropriately set so that we can accurately capture and separate the cell containing the QR code's black-and-white color feature. In Fig. 4 (a), (b), the value of d is much smaller than the actual size of a QR code, and each cell could contain

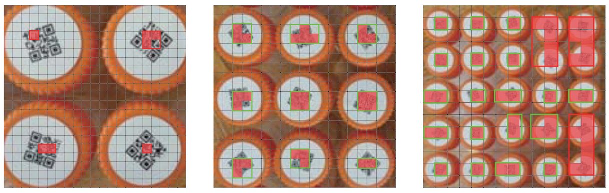

(a) Tiny-cell Issue (b) Small-cell Issue (c) Multi-tag Issue

Fig. 4: Three potential issues in practice when d and Th_f are fixed.

only a very small part of the QR code. In this case, the color feature of the QR code is not that salient and we should set a lower threshold Th_f to make it easier to identify the cell containing the code fragment. In Fig. 4 (c), the value of d is comparable to the actual size of a QR code, but we can see that some bounding boxes contain codes from multiple tags because Th_f is over low with the given d, because the area between two tags is falsely recognized as a part of QR code. We should make Th_f higher to these particular bounding boxes, so that the standard to label a cell is more strict.

Consequently, the values of the tuple $< d, Th_f >$ should be dependent on the size of the code area in the image, which however is dependent on the application scenario thus can not be predicted. In order to deal with the issue, we propose a self-adaptive parameters configuration mechanism to automatically adjust values of $< d, Th_f >$, which can accommodate the practical scenario.

Algorithm architecture. Algorithm 1 shows the main process of parameter adjustment, where I refers to the input image with height, width and the *RGB* channel index being H, W and C respectively. We first try to set $d = d^s$ and $Th_f = Th_f^s$, where d^s is the length of 13 pixels and $Th_f^s = 0.60$. Those values are obtained by conducting 500+ times of experiments in different environments for reading about 160 QR codes. With such initial values, we invoke the operation *DetectOnce*, which performs automatic parameter adjustment starting from $< d^s, Th_f^s >$. The details about *DetectOnce* will be described later; the set of bounding boxes yielded from the operation is denoted by $\boldsymbol{\Lambda^s}$. We let those bounded subregions of the image in $\boldsymbol{\Lambda^s}$ go through a classifier. The details of the classifier will be described in Section IV, and at this point, it only needs to understand that the output of the classifier is the tuple for each of the bounding boxes, which indicates the probabilities that the given bound box are correctly and incorrectly labeled, respectively. We average the correct labeling probabilities for the bounding boxes in $\boldsymbol{\Lambda^s}$, which is denoted by $Score^s$.

We then use $< d^l, Th_f^l >$ to process the image in the similar way, where d^l is the length of 75 pixels and $Th_f^l = 0.02$. Such values normally work good when reading 1-20 codes, which is verified by our 500+ experimental results. The set of bounding boxes yielded with the configuration is denoted by $\boldsymbol{\Lambda^l}$; they go through the classifier and the corresponding average of the correct labeling probabilities is denoted by $Score^l$. We then compare the two scores and choose the set with higher score, and classify them again to be the final bounding boxes set. Thus the problem occurred in Fig. 4 (a) can be solved.

```
Algorithm 1: SubregionDetect
  Input: $< d^s, Th_f^s >$, $< d^l, Th_f^l >$, $I_{H \times W \times C}$
  Output: TAG bounding box set $\Lambda$
1 $\Lambda^s \leftarrow DetectOnce(d^s, Th_f^s, I)$ ;
2 $Score^s \leftarrow$ Averaged positive probabilities of $\Lambda^s$ ;
3 $\Lambda^l \leftarrow DetectOnce(d^l, Th_f^l, I)$;
4 $Score^l \leftarrow$ Averaged positive probabilities of $\Lambda^l$ ;
5 if $Score^s > Score^l$ then
6     $\Lambda \leftarrow Classify(\Lambda^s)$
7 else
8     $\Lambda \leftarrow Classify(\Lambda^l)$
```

It is worth noting that we will not apply both the results of Λ^s and Λ^l in our later processing since it may heavily increase the processing time, especially when image refocus mechanism to be mentioned in Section V is triggered multiple times (single triggering can add about 2s). We note that it is also possible to evaluate the resulted bounding boxes by training a classifier to measure the value of intersection of union (IoU) between the yielded boxes and its potential tightly enclosure [10], [11]. In particular, we have to manually draw the bounding box that tightly enclosures the QR code in the image as the ground truth, which is to serve as the training dataset for the classifier. However, manually generating the ground truth dataset is all consuming. In contrast, the classifier used in our scheme only needs labeling the cell that contains the code fragments, which brings much convenience in ground truth data collection. More details will be discussed in Section IV.

Parameters configuration adjustment. The detailed algorithm is summarized in Algorithm 2, and we here describe the basic idea of adaptive parameters configuration adjustment. With the given initial values of the parameters, we run the basic procedure and then check if

$$\eta_{mdn} \geq Th_l \times d^2, \tag{1}$$

where η_{mdn} denotes the median size of those obtained bounding boxes and $Th_l = 11$ from experimental results. The inequality above indicates that the cell is too small as shown in Fig. 4 (b), thus we need to increase d and reduce Th_f since they are highly relevant. In particular, we let $d = d \times \lambda_{inc}$, $Th_f = Th_f / \mu_{dec}$, and run the basic procedure again. This operation will be repeated until the inequality (1) no longer holds. This is to tentatively increase the size of the cell.

We then check if

$$\eta_i \geq Th_m \times \eta_{mdn}, \tag{2}$$

where η_i is the size of any yielded bounding box and Th_m is set to 3.5 according to our empirical study. This indicates that the bounding box may contain falsely labeled cells as shown in Fig. 4 (c), thus we need to grid this bounding box with finer granularity. In particular, we let $d = d/\lambda_{dec}$, $Th_f = Th_f \times \mu_{inc}$, and run the basic procedure to this particular bounding box again. We will repeat the operation until the condition no longer holds.

Algorithm 2: DetectOnce

```
   Input: $d$, $Th_f$, $I_{H \times W \times C}$
   Output: TAG bounding box set $\Lambda$
1  Initialize $Label_{\frac{H}{d} \times \frac{W}{d}}$ = matrix filled with 0;
2  Grid $I$ by $d$ into $CellSet_{\frac{H}{d} \times \frac{W}{d} \times C}$;
3  foreach $x \in [1, \frac{H}{d}]$ do
4      foreach $y \in [1, \frac{W}{d}]$ do
5          Initialize $idftg$ = matrix filled with 1;
6          foreach $c \in [1, C]$ do
7              grid $g \leftarrow CellSet(x, y, c)$;
8              $idftg_c \leftarrow$ idft($g$);
9              Binarize each element in $idftg_c$ by $Th_b$;
10             $idftg \leftarrow idftg$ AND $idftg_c$;
11         if ratio of 1s in $idftg > Th_f$ then
12             $Label(x, y) \leftarrow 1$;
13 $\Lambda \leftarrow NeighborCluster(Label)$;
14 while Either Eqn. 1 or Eqn. 2 satisfy do
15     foreach $Z^i \in \Lambda$ satisfy Eqn. 2 do
16         $d \leftarrow \frac{d}{\lambda_{dec}}$;
17         $Th_f \leftarrow Th_f \times \mu_{inc}$;
18         Remove $Z^i$ from $\Lambda$;
19         Add $ObjectProposal(d, Th_f, Z^i)$ to $\Lambda$;
20     if $\eta_{mdn} > Th_l \times d^2$ then
21         $d \leftarrow d \times \lambda_{inc}$;
22         $Th_f \leftarrow \frac{Th_f}{\mu_{dec}}$;
23         $\Lambda \leftarrow ObjectProposal(d, Th_f, I)$;
```

IV. False Positive Classification

A. Purposes of Classification

We implement a classifier to distinguish if the bounding box obtained indeed contains a partial QR code. The result of classification can help determining the parameters configuration as described in Section III; moreover, it can help saving the batch reading time.

In particular, the code subregion detection process is unable to identify those QR codes in 100% accuracy, because the algorithm design is based on detecting the black-and-white color feature of QR codes. It is supposed that the subregion containing code fragments has intensive luminance change, which however may also happen in certain subregions containing no code fragments. The consequence of the false positive judgment is that the reader can not recognize the code. In the regular code reading process, we could just leave the unrecognizable code area behind.

Nevertheless, in the batch reading scenario, it may happen that certain part of the codes image is blurred or distorted because of the factors including the camera's focus area, resolution and the angle of reading. Those also result in the unrecognizable code area, which can not be left behind (How to deal with this issue will be discussed in Section V). The problem is, if the unrecoganizable code area emerges, we have no idea if this is because the subregion indeed contains no code or the region is blurred/distorted. If it is the former case, the reader will still invoke the process to be described Section V to deal with the issue. This will take 2-8s and make the user uncomfortable. We need to eliminate the false-positively detected bounding boxes as many as possible, in order to avoid such unnecessary processing.

B. Classifier Implementation

To identify which subregions indeed contain QR codes and which are false-positively detected is a typical classification problem. A number of classifier models could be used to resolve the problem, such as SVM, Random Forest and XGBoost [23]. We first conduct experiments to test if such off-the-shelf models can be utilized directly. We collect 52000 batch QR codes images in different environments with different kinds of tubes for training. We then use another 4900 images to test this model. The accuracies of the three frequently used classifier models are 92.6%, 92.7%, 92.2%, respectively. The performance looks fairly good in percentage, but not good enough for the specific application scenario. For example, one of our experiments is to read 160 codes in batch, then the classifier models above still have about 13 bounding boxes mistakenly judged, successful decoding rate will decrease 9% and its corresponding processing time may increase 2-8s.

During the experiments, we find that collecting training data in the application scenario is not too hard. We can collect some images containing multiple QR codes and run basic code detection procedure with simple parameter adjustment but without operations involving the classifier. We then are able to obtain up to 100-200 positive and negative samples from a single image. It takes only about 8 hours to collect 52000 training images. This motivates us to try deep learning based classifiers. Deep learning models such as traditional CNN and its variants [24], [25] require high-performance computing platform, which can not be satisfied in the scenario; therefore, we choose to implement our classifier with MobileNetV2[22], a lightweight and accurate architecture suitable for mobile equipments. First, it can greatly reduces the memory overhead during inferencing. Second, it can greatly enhance the inference speed about 8 to 9 times [26] by using a bottleneck block as the building block of the neural network.

We refer our readers to look at MobileNetV2 paper [22] for the detailed network structure. It is worth noting that we use width multiplier α [22] to slim our network in our implementation, which means that the number of channels in the same layer will be α times compared to before. Then the computational loss will be about α^2 times. Thus, if we can reasonably select this parameter, we can process a lot faster while maintaining a good performance. The detailed configuration of our neural network will be shown in Section VI. Finally, we train this model with the dataset mentioned earlier, and the testing accuracy can be 98.7%. That is, the

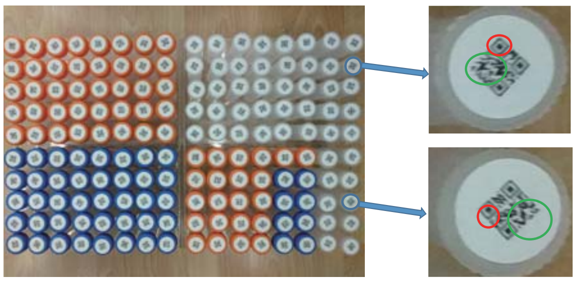


Fig. 5: Distortion and obscurity brought by the limited resolution of camera where green circles represent distortion and red circles represent obscurity

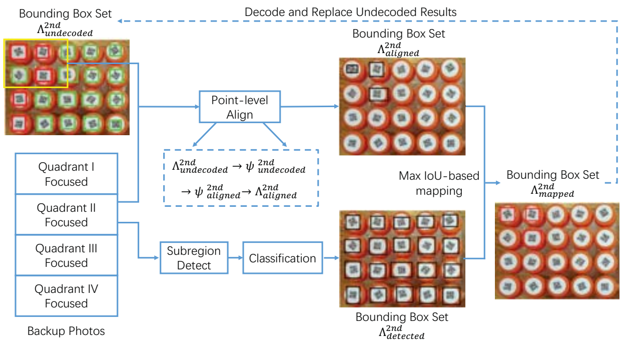
number of mistakenly judged bounding boxes is 4 in 160 batch reading scenery, which means only 1/3 error rate compared with SVM, Random Forest and XGBoost [23]. We integrate the architecture into our system and the inference speed of each input is only about 1ms, which is highly satisfactory.

V. Image Refocus

We need to decode the QR code captured in the bounding box after the process described above. Figure 5 shows the bounding boxes obtained from our experiments. We can see that some of the code subregions in the image are blurred or distorted. The two unfavored phenomena could result from a number of factors such as the lighting conditions and the angle between the camera and the tubes; however, according to our experiments, the main reason is because the camera of the mobile device has limited focus area, and some of the codes are outside the area when the user wants to read a lot of codes. Since the focus area of the camera is normally in the center of the image, the blur and distortion usually happen in the corners or edges of the image.

The decoding process of the standard QR code requires the quality of the code image to be reasonably good. Our experimental results show that some seriously blurred or distorted QR codes can not be decoded. A straightforward way to resolve the issue is to utilize deblurring algorithms [27], [28] to reverse the blurring phenomena, which however are based on the assumption that the blur phenomenon follows certain patterns. Such assumptions can hardly hold in the practical batch codes reading scenarios.

We propose a simple but effective image refocus approach to deal with the issue. Algorithm 3 shows the main process and its corresponding procedure is shown in Fig. 6. In particular, we let the camera take 5 images of those tubes in the background when batch reading the codes. As shown in Fig. 6, the first image is the auto-focused one. After that, we take the 4 images of those codes as backups, with the focus on the top left, top right, bottom left and bottom right part of those codes respectively, because the blurred or distorted usually occur in the corners and edges of the image. If the auto-focused image contains some blurred or distorted codes that can not be decoded, we first figure out the phenomena occur in which



Bounding Box Set $\Lambda^{2nd}_{undecoded}$
Decode and Replace Undecoded Results
Bounding Box Set $\Lambda^{2nd}_{aligned}$
Point-level Align
$\Lambda^{2nd}_{undecoded} \rightarrow \psi^{2nd}_{undecoded} \rightarrow \psi^{2nd}_{aligned} \rightarrow \Lambda^{2nd}_{aligned}$
Quadrant I Focused
Quadrant II Focused
Quadrant III Focused
Quadrant IV Focused
Backup Photos
Subregion Detect
Classification
Max IoU-based mapping
Bounding Box Set Λ^{2nd}_{mapped}
Bounding Box Set $\Lambda^{2nd}_{detected}$

Fig. 6: Pipeline of image refocus algorithm where red rectangles in initial phone means this tag cannot be decoded, green rectangles means the opposite.

Algorithm 3: ImageRefocus

```
   Input: $\boldsymbol{\Lambda_{undecoded}}, I^{auto}, I^{1st}, I^{2nd}, I^{3rd}, I^{4th}$
1  $quadrantSet \leftarrow \{1st, 2nd, 3rd, 4th\}$;
2  Divide $\boldsymbol{\Lambda_{undecoded}}$ into
     $\{\boldsymbol{\Lambda^{1st}_{undecoded}}, \boldsymbol{\Lambda^{2nd}_{undecoded}}, \boldsymbol{\Lambda^{3rd}_{undecoded}}, \boldsymbol{\Lambda^{4th}_{undecoded}}\}$;
3  foreach $qd \in quadrantSet$ do
4      if No undecoded bounding box in quadrant $qd$ then
5          skip the current quadrant $qd$;
6      Initialize Point Set $\Psi^{qd}_{undecoded}$
7      foreach $Z^{j}_{undecoded} \in \boldsymbol{\Lambda^{qd}_{undecoded}}$ do
8          Put four corners of $Z^{j}_{undecoded}$ to $\Psi^{qd}_{undecoded}$
9      $\Psi^{qd}_{aligned} \leftarrow Align(I^{auto}, I^{qd}, \Psi^{qd}_{undecoded})$;
10     Every four points in $\Psi^{qd}_{aligned}$ construct a minimum
         bounding box $Z^{j}_{aligned}$, put $Z^{j}_{aligned}$ to $\boldsymbol{\Lambda^{qd}_{aligned}}$;
11     $\Lambda^{qd}_{detected} \leftarrow$
         $SubregionDetect(d^{s}, Th^{s}_{f}, d^{l}, Th^{l}_{f}, I^{qd})$;
12     $\Lambda^{qd}_{detected} \leftarrow Classify(\Lambda^{qd}_{detected})$;
13     foreach $Z^{j}_{aligned} \in \boldsymbol{\Lambda^{qd}_{aligned}}$ do
14         Select $Z^{k}_{detected}$ with maximum $IoU_{j,k}$;
15         Recode $Z^{k}_{detected}$ into $\Lambda^{qd}_{mapped}$;
16     Decode each subregion in $\Lambda^{qd}_{mapped}$ and replace
         undecoded results of $\Lambda^{qd}_{undecoded}$.
```

part of the image. The auto-focused image is equally divided into 4 quadrants. If the blur or distortion happens in the 2nd quadrant, we will try to locate the blurred or distorted code's backup in the top-left focused image. Thus, it is more likely we can decode the backup code, which contains the same information as the code in the original image. However, accurately locating the blurred or distorted code's counterpart in the backup image is not straightforward, because the hand-shake may happen to the user when taking those 5 images, which incurs displacement of the backup images compared with the original one. We thus need to align the two images before locating the backup code. In our mechanism, we use the fast and memory efficient image alignment algorithm [29] to align those backup images and the original auto-focused one. In particular, we align the bounding boxes in the auto-focused image with that in the backup image. Note that the image alignment algorithm [29] can only align points in the previous image to their current positions in the next image. Thus, when processing the refocus in each quadrant qd (in the Fig. 6, $qd = 2nd$) , we will extract the corners of all the bounding boxes in undecoded bounding box set $\Lambda^{qd}_{undecoded}$ and arrange their corners into Point Set $\Psi^{qd}_{undecoded}$ in order. And pass $\Psi^{qd}_{undecoded}$ into the alignment algorithm. Then we can get the generated Point Set $\Psi^{qd}_{aligned}$ and we will recompose the bounding box set $\Lambda^{qd}_{aligned}$ using the points in $\Psi^{qd}_{aligned}$ in order. Note that we cannot directly decode the bounding boxes in $\Lambda^{qd}_{aligned}$ as the substitution of $\Lambda^{qd}_{undecoded}$. Due to the imperfection of alignment algorithm, the generated bounding boxes in $\Psi^{qd}_{aligned}$ can hardly contain a whole part of one QR code. Thus, these bounding boxes in $\Lambda^{qd}_{aligned}$ can indicate the current positions of the previous bounding boxes.

In order to overcome the above problems, we apply the code subregion detection mentioned in Section III again to these refocused photos, and pass them though the classifier mentioned in Section IV and we can get the bounding box set $\mathbf{\Lambda}^{qd}_{detected}$. We need to map each previously aligned subregion $Z^{j}_{aligned} \in \mathbf{\Lambda}^{qd}_{aligned}$ to each current detected subregion $Z^{k}_{detected} \in \mathbf{\Lambda}^{qd}_{detected}$. In order to utilize both the location and shape information of $Z^{j}_{aligned}$ to guide us to select the most similar bounding box $Z^{k}_{detected}$, here we use Intersection over Union (IoU) as the metrics to measure the similarity between $Z^{k}_{detected}$ and $Z^{j}_{aligned}$. For each region $Z^{j}_{aligned}$ in $\Lambda^{qd}_{aligned}$, we will map it to the region $Z^{k}_{detected}$ with the maximum IoU of Quadrant qd refocused photo to be mapped, and record this mapped subregion into bounding box set Λ^{qd}_{mapped}. Finally, we will decode Λ^{qd}_{mapped} to replace the error decoding subregions in the auto-focused photo to compensate the focus problem.

VI. Implementation

The settings of above parameters are: $Th_b = 1$, $d^s = 13$, $Th^s_f = 0.6$, $d^l = 75$, $Th^l_f = 0.02$, $Th_l = 11$, $Th_m = 3.5$, $\lambda_{dec} = 2$, $\mu_{inc} = 1.3$, $\lambda_{inc} = 2$, $\mu_{dec} = 2.5$. And we use the famous deep learning framework TensorFlow to train our MobileNetV2 based classifier by setting training batch size to 32, input image size to $32 \times 32 \times 3$, width multiplier α to 0.35 and output vector to two-value vector. Our network is trained from scratch and our learning rate is set to 0.01, weight decay rate is set to 0.99 per 4 epochs. Then we trained about 100 epochs on Nvidia 1080Ti which costs about 6 hours. Finally, we convert the produced TensorFlow model into TensorFlow Lite model which is highly compatible and optimized with Android phones. As for the image refocus part, we use the Camera2 API in Android to automatically force the camera to focus on four quadrant regions. Note that we do not use the preview frames of the camera as the processing photos in our algorithms since the maximum resolution of preview frames is always lower than that of captured photos due to the bandwidth limitation of the hardware bus connecting camera module to the inner circuits of phone. We realize all those mechanisms to create an the BatchQR APP, which is installed in a Google Pixel phone with Android 8.0 system.

VII. Experiments

We conduct experiments to test the performance and robustness of our APP. The default configuration of the experiments is as following: the captured image is fully filled with the QR tags, thus more tags in the image means smaller size of each tag; the environmental luminance level is around 240lx; the QR code is Version 1-L considering the limitation of the space on the cap of the tube. Such code can contain up to 41 numeric numbers or 25 alphanumeric numbers [14], which is equivalent to 10^{41} unique IDs and more than enough for the application scenario.

A. Influence by Number of QR codes

We use three types of frequently used tubes with volumes 50ml, 10ml and 5ml respectively to evaluate performance of the BatchQR. Results are shown in Fig. 8 which are average values of 15 repetitive experiments and their corresponding numerical results are tabulated in Table. I.

TABLE I: Statistics of results in Fig. 8

Tube Type	Attributes	Min	Max	Mean	Median
50ml Tube	Detection Rate (%)	99.69	100.00	99.97	100.00
	Decoding Rate (%)	88.46	100.00	98.01	99.57
	Latency (s)	2.36	10.75	5.13	4.27
10ml Tube	Detection Rate (%)	87.50	100.00	98.33	100.00
	Decoding Rate (%)	75.00	99.50	95.30	97.92
	Latency (s)	2.60	14.83	8.00	7.87
5ml Tube	Detection Rate (%)	92.17	100.00	96.08	96.04
	Decoding Rate (%)	82.91	99.17	92.78	94.40
	Latency (s)	2.37	14.44	7.70	6.61

1) Accuracy Analysis: The subfigures in the upper part of Fig. 8 (a), (b) and (c) illustrate code detection and decoding rates of the three kinds of tubes respectively. Note that the blue dotted curve represents the rate that the QR code can be successfully detected using our code detection mechanism, while other curves represent the ratios that detected codes can be successfully decoded. In this sense, the code detection rate can be regarded as a upper bound of the decoding rate. BatchQR has three important modules as described in previous section, i.e., code detection, false-positive classifier and refocus scheme, and we here examine their functionalities under different numbers of tubes in the batch. "Basic" means the basic code detection procedure with simple parameter adjustment operations but without operations involving the classifier. "BatchQR" means the mechanism with all the components put together. The experimental results indicate the following points:

- Our code detection mechanism can detect the QR codes in batch in 100% most of the time, but the performance degrades on those 5ml tubes. This is because the distance

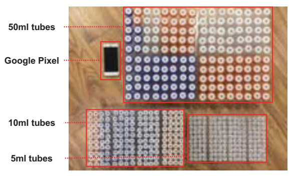


Fig. 7: Experimental Equipments

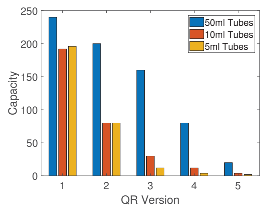
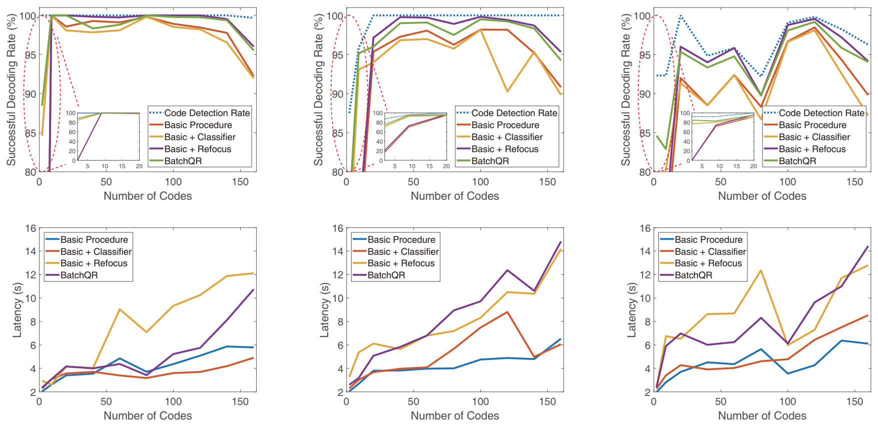


Fig. 9: QR code version v.s. capacity



(a) Results on 50ml Tubes (b) Results on 10ml Tubes (c) Results on 5ml Tubes

Fig. 8: Experimental Results on different total QR tag counts and different kind of tubes

between neighboring tubes in this case is quite small, making our mechanism prone to mistakenly clustering multiple codes into a single bounding box. This will make the detection rate decrease but in a very small scale.

- The proposed image refocus mechanism plays an important role, which could improve the average successful decoding rate by 2-6%. That is, if the user wants to read 160 codes in batch, then about 6 more codes can be successfully decoded with image refocus mechanism. This will save tens of seconds that would been spent on manually locating and reading those 6 codes one after another. The outstanding contribution of the refocus mechanism indicates that the quality of the batch codes is the main bottleneck when batch reading the QR codes. Generally, when more QR tags are captured in one image, this phenomenon is more obvious. Our image refocus mechanism could potentially neutralize the negative impact of the bottleneck.
- The false positive classifier is very useful when there are small numbers of codes in the batch. We can see that when the number of tags is larger than 20, the successful decoding rate drops a little: less than or nearly equal to 1%. This is because the classifier may make mistakes. However, the benefit brought by the classifier over weighs the limited drawback. First, recall that the classifier is used to differentiate the false positive bounding box that contains no QR tag at all, and also used determine appropriate parameters. Note the successful decoding rates when the number of tags is smaller than 20, we can find that the classifier can raise the decoding rate to about 85% on average. Second, runtime saving, which will be discussed in the following subsection.

2) Latency Analysis: The subfigures in the lower part of Fig. 8 (a), (b) and (c) illustrate the latency of batch reading three kinds of tubes respectively. The experimental results indicate the following points:

- In Fig. 8 (a) (c), we can see that the benefit classifier brings is noticeable, which saves up to 5s, accounting for 50% of the total processing time. First, the classifier can save QR code decoding time by removing many false positive samples to save code decoding time where the decoding time for each of the code itself is 2ms-30ms depending to its size. Second, it can save the image refocus procedure falsely triggered by those false positive samples which cost averagely 2s when one quadrant of the photo is triggered. However, we can see that in Fig. 8(b), the classifier may not decrease the time cost to the whole system. This is because due to the specific shape of 10ml tubes, the camera can hardly fully focus on the entire image. It is unavoidable that some tags are obscure in the auto-focused image, and image refocus is often triggered in this way, thus the classification cannot provide an obvious time cost elimination.
- As the number of tags increases, the final processing latency of BatchQR increases; latency is generally in the order of seconds with Google Pixel, which is satisfactory on mobile.

B. Influence by QR Code Version

The pattern and size of the QR code is mainly determined by the code version, which could impact the image processing mechanism in the BatchQR. In the experiment, we fix the width of the QR code, and define *capacity* of BatchQR as the maximum number of QR codes that can be successfully decoded in at least 90%. Figure 9 shows results, where it is straightforward that the higher version of the code pattern results in lower capacity. This is because higher versions contain more information, and the black-and-white pattern is more complicated. To successfully decode such codes require the image of each single code to be with comparatively higher quality. This trend is more obvious for small size tubes as shown in the figure. We want to emphasize that our code detection mechanism can detect those codes even if they are very small, because the black-and-white pattern still present no matter how complicated it is.

C. Influence by Luminance Condition

To evaluate how the environmental luminance conditions affect BatchQR, we do these experiments on 160 50ml tubes.Results are tabulated in Table II. We can see that the performance is good in 150-3500 lux, where 150lx is like the condition of a room with lights on in the nighttime and 3500lx is like the condition of a room with sunlight. The results indicate that our APP can cover almost all the practical environments. Also, we can see that when the light is darker, i.e., 3lx or 25lx, the decoding process of each QR code is the obvious bottleneck rather than our proposed detection algorithm, simply because in such environment, the distortion and obscurity of each QR tag will be more obvious; therefore, the low successful decoding rate is unavoidable. However, we can see that image refocus algorithm can largely remedy this situation in 25lx environment which is not totally dark, which averagely increase the successful decoding rate by nearly 10%. This is because in such environment, to accurately capture the details of each QR codes is a difficulty due to reasons mentioned above. However, refocus can provide better results because one image only needs to focus on 1/4 regions.

TABLE II: Performance Under Different Illumination

Lux	Detected	Before Refocus	After Refocus	Total Time (s)
3	47.20	0.00	0.60	12.23
25	156.38	98.54	113.00	14.17
150	158.91	145.09	148.82	11.14
300	159.18	147.29	152.94	10.11
540	152.73	144.27	149.87	12.33
1500	151.57	147.79	150.43	10.68
3500	154.14	148.43	152.64	10.56

VIII. Conclusion

This paper has presented BatchQR, a mobile APP that can batch read densely arranged QR codes attached to caps of the tubes and vials in clinical and biological labs. We have proposed code subregion detection algorithm with high speed and reliability. Moreover, we have proposed a false-positive classifier to sift out the false positive subregions to save needless processing time. At last, we have designed a image refocus scheme to remedy the distortion and obscurity in the corners of captured image. Combining the three proposed modules constitutes the overall architecture of BatchQR. Further, we have analyzed the accuracy and robustness of our system. It turns out that BatchQR can read and decode 160-180 Version 1-L QR codes with 90%-95% accuracy in 10-14s under practical illumination, which is only 4% of the time consumed by the regular QR code reader in the same situation.

IX. Acknowledgement

The work in this paper is supported by the National Natural Science Foundation of China (NSFC) 61572319,61872233, 61829201, 61532012, 61325012, 61428205.

References

[1] Clinical and Laboratory Standards Institute (CILS), Global Laboratory Standards for a Healthier World, Online:https://clsi.org/.

[2] Clinical and Laboratory Standards Institute (CILS), Specimen Labels: Content and Location, Fonts, and Label Orientation, 1st Edition, Online:https://clsi.org/standards/products/automation-and-informatics/documents/auto12/, Apr. 2011.

[3] Clinical and Laboratory Standards Institute (CILS), Barcode Labelling: The Basics, Online:https://www.cils-international.com/en/information-centre/blog/articles/barcode-labelling-the-basics/.

[4] ThermoFisher, Abgene 2D Barcoded 2mL Screw Cap Storage Tubes, Online:https://assets.thermofisher.com/TFS-Assets/LCD/Specification-Sheets/D17415~.pdf.

[5] QRStaff, Creating QR Codes In Bulk, Online:https://blog.qrstuff.com/2017/12/10/how-to-create-qr-codes-in-bulk, Dec. 2017.

[6] Barcodez, Bulk Barcode Generator, Online:http://www.barcodez.net/.

[7] ZXing, barcode scanning library for Java, Android, Online:https://github.com/zxing/zxing.

[8] EuroCPS, Smartlab, Sample Management With RFID Tags for Laboratories, Online:https://www.eurocps.org/innovators-projects/ongoing-projects/smartlab-sample-management-with-rfid-tags-for-laboratories/.

[9] Techno Medica, RFID Specimen Management System: TRIPS, Online:http://www.technomedica.co.jp/t01/EnglishPage/products/solution/3/trips.html.

[10] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," in *Proc. IEEE ICCV*, 2017, pp. 2980-2988.

[11] S. Ren, K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. NIPS*, 2015, pp. 91-99.

[12] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges,", in *Proc. ECCV*, 2014, pp. 391-405.

[13] J. R. R. Uijlings, K. E. A. V. D. S, T. Gevers and A. W. M. Smeulders, "Selective search for object recognition," *IJCV*, vol. 104, no. 2, pp. 154-171, 2013.

[14] ISO White Paper, QR Code Bar Code Symbology Specification, Online:https://www.iso.org/standard/62021.html.

[15] Thonky, QR Code Tutorial: Introduction, Online:https://www.thonky.com/qr-code-tutorial/introduction.

[16] Xminnov, How Much Is RFID Tags What is RFID tag cost, Online:http://www.rfidtagworld.com/news/RFID-Knowledge-IOT-Knowledge_555.html.

[17] RFID Journal, RFID Frequently Asked Questions, Online:https://www.rfidjournal.com/faq/show?85.

[18] Barcoding, RFID Frequently Asked Questions, Online:https://www.barcoding.com/resources/frequently-asked-questions-faq/rfid-faqs/.

[19] Amazon, Micro Centrifuge Lab Tubes, Online:https://www.amazon.com/Micro-Centrifuge-Lab-Tubes/b/ref=dp_bc_5?ie=UTF8&node=318099011.

[20] H. K. Chu, C. S. Chang, R. R. Lee and N. J. Mitra, "Halftone QR Codes," *ACM TOG*, vol.32, no.6, pp. 217, 2013.

[21] S. S. Lin, M. C. Hu, C. H. Lee and T. Y. Lee,"Efficient QR Code Beautification With High Quality Visual Content," *IEEE TOM*, vol.17, no.9, pp. 1515-1524, 2015.

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE CVPR*, 2018, pp. 4510-4520.

[23] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. ACM SIGKDD*, 2016, pp. 785-794.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Online:http://arxiv.org/pdf/1409.1556.pdf.

[25] C. Szegedy, S. Ioffe, V. Vanhoucke and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI*, 2017, pp. 12.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Online:http://arxiv.org/pdf/1704.04861.pdf.

[27] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin and J. Matas, "DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks," Online:http://arxiv.org/pdf/1711.07064.pdf

[28] M. Maggioni, E. Sánchez-Monge and A. Foi, "Joint removal of random and fixed-pattern noise through spatiotemporal video filtering," *IEEE TIP*, vol. 23, no. 10, pp. 4282-4296, 2014.

[29] J. Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10 , pp. 4, 2001.

[30] Wikipedia contributors, Image noise, Online:https://en.wikipedia.org/wiki/Image_noise