**QR SCANNER USING RASPBERRY PI**

**Abstract:**

A QR code (Quick Response code) is a type of two-Dimensional Barcode invented in 1994 by a Japanese automotive company [Denso](https://en.wikipedia.org/wiki/Denso#Denso_Wave). A QR code is a machine-readable optical label that consists of black squares arranged on a white background and can be read by an imaging device like camera. They became very popular due to their fast readability and greater storage capacity compared to standard UPC barcodes. A typical UPC barcode is a series of contrasting strips that encodes a 12-digit number, where as QR code is a small matrix of squares that could represent up to 4296 characters. There is an enormous amount of research on various techniques to decode information from a QR-code. However, it is challenging to scan a QR code when it is tiny and has varied lighting conditions. This project sheds light on how to use a Raspberry pi to scan compact sized QR codes and send the code content to an HTTP API with a network call.

**Task Objective:**

Each solar module and contains a QR code that is printed white on a dark background and contains information about the module’s serial number. Monitoring each module during its production phase is very essential to improve the process. To achieve this, a basic stand-alone equipment is needed that could scan QR codes and is easy to replicate so that it could be installed at several stages in the production.

Main objective of this task to develop a model on raspberry pi that could scan QR codes along with a hardware setup.

**Structure of a QR code:**

The QR code structure is very simple, it is in the form of a square as it is easy to read pixel wise. As shown in the figure 2.1.2, It contains position markings (3 prominent squares) which indicates the way the code is printed. A dotted line attached to these position markers is called timing pattern and it talks about the size of the matrix. Version information would specify which version of QR code is being used out of more than 40 available versions. Format information which engulfs position markings contains the information about error tolerance and data mask pattern and simplifies the scanning. Data and error connection keys zone as the name suggests contains the data and keys to the error connection. Quiet Zone is a blank space that helps in differentiating the QR code from the background.

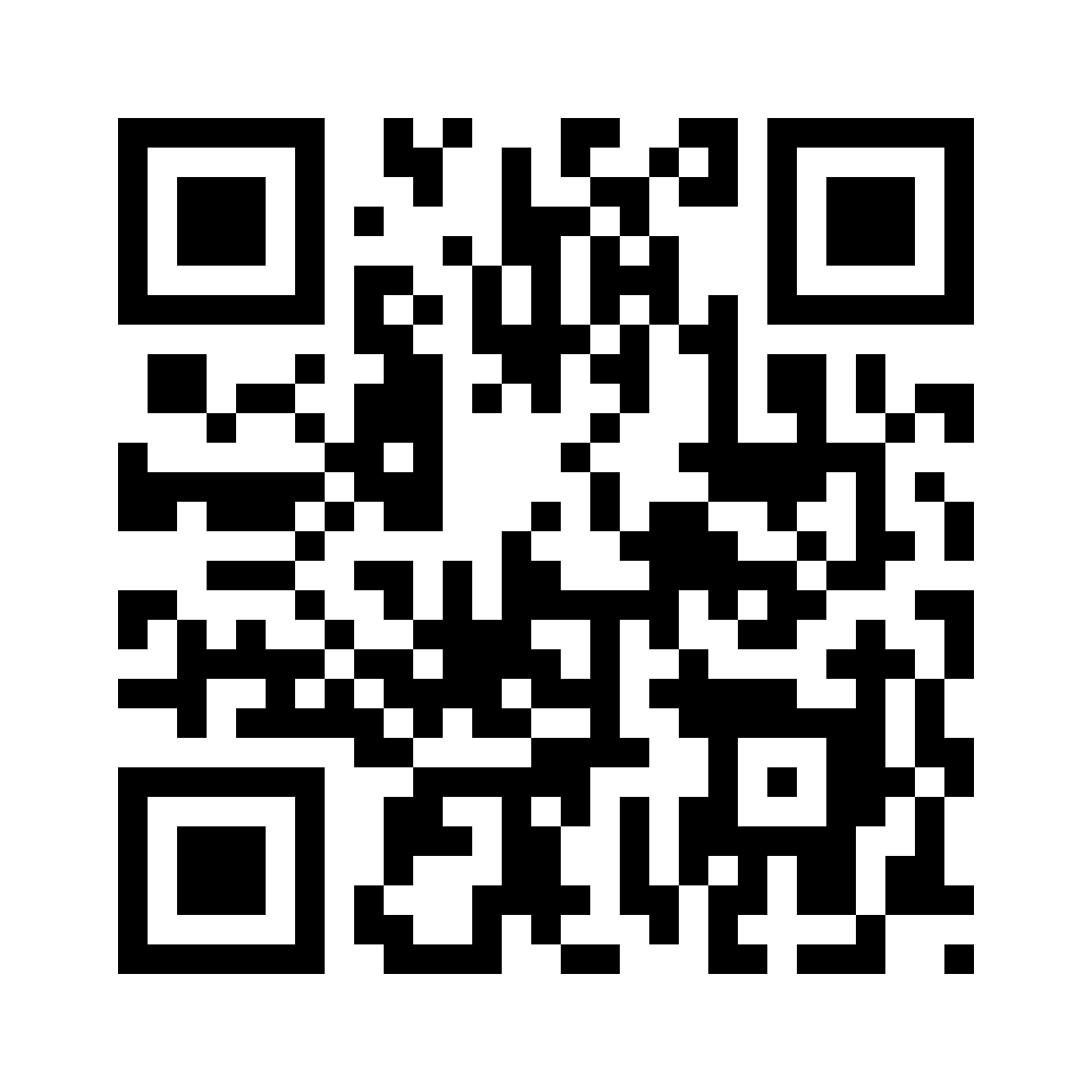
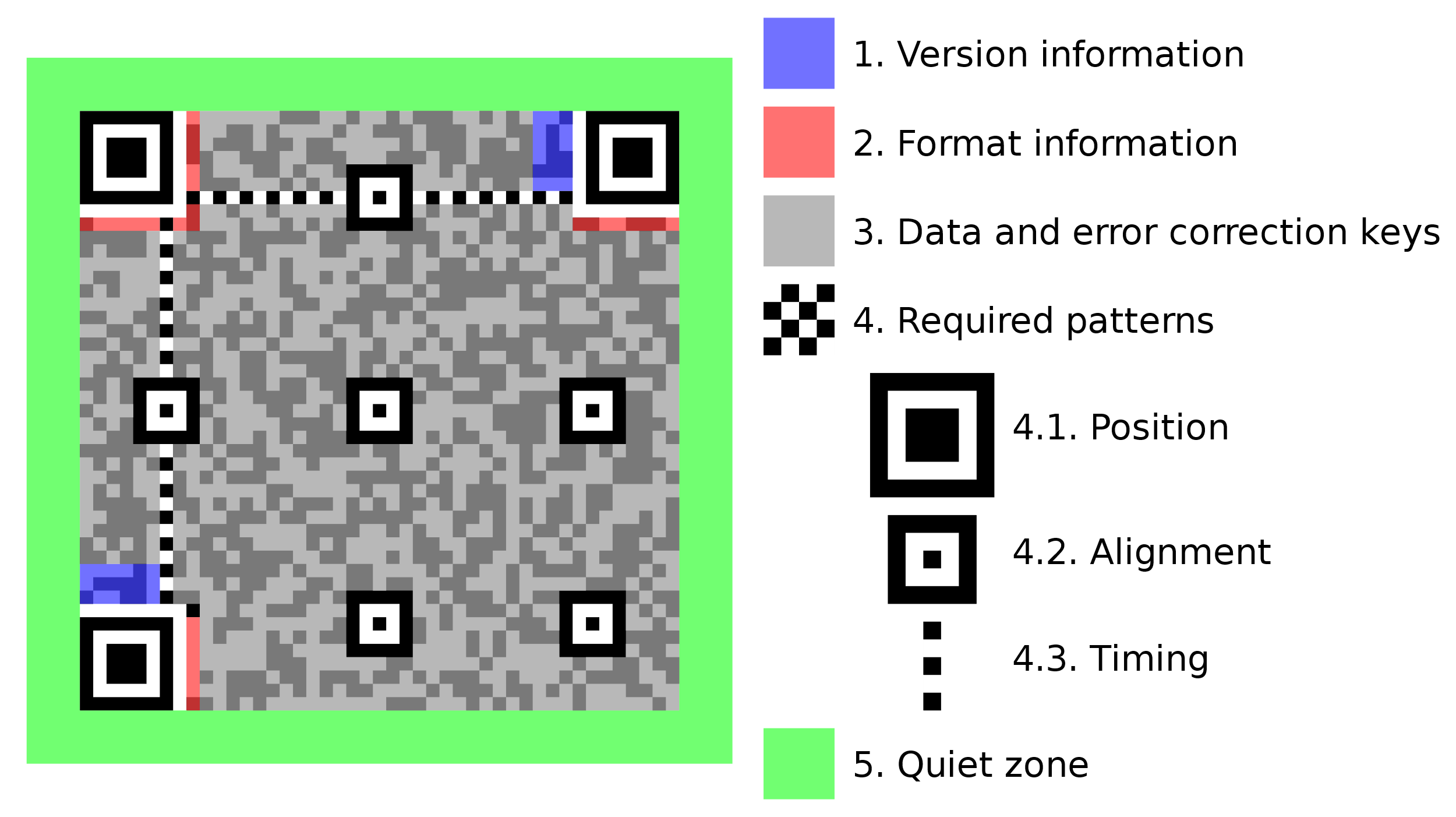
**

Fig: 2.1.1 fig: 2.1.2

**Selection of Hardware:**

Raspberry Pi: Raspberry Pi 3 B+

Camera: Raspberry Pi Camera Module 8MP v2.1

Battery: 10,000 mAh

Selection of Raspberry pi 3 is selected solely because it can run live video stream at a sufficiently higher frame rate, and camera is a pi camera module because it provides better hardware integration along with cheaper power consumption compared to an USB camera. Any battery pack above 8000 mAh runs a Pi conveniently longer than 8 hours.

**Code development:**

The code is developed in Python. OpenCV an Pyzbar are the necessary dependencies along with python 3.7. Installing OpenCV is tedious, if it is not already preinstalled.

We can install OpenCV on a raspberry pi by first updating the Pi using.



Then install the required OpenCV dependencies using:

and finally installing the OpenCV using



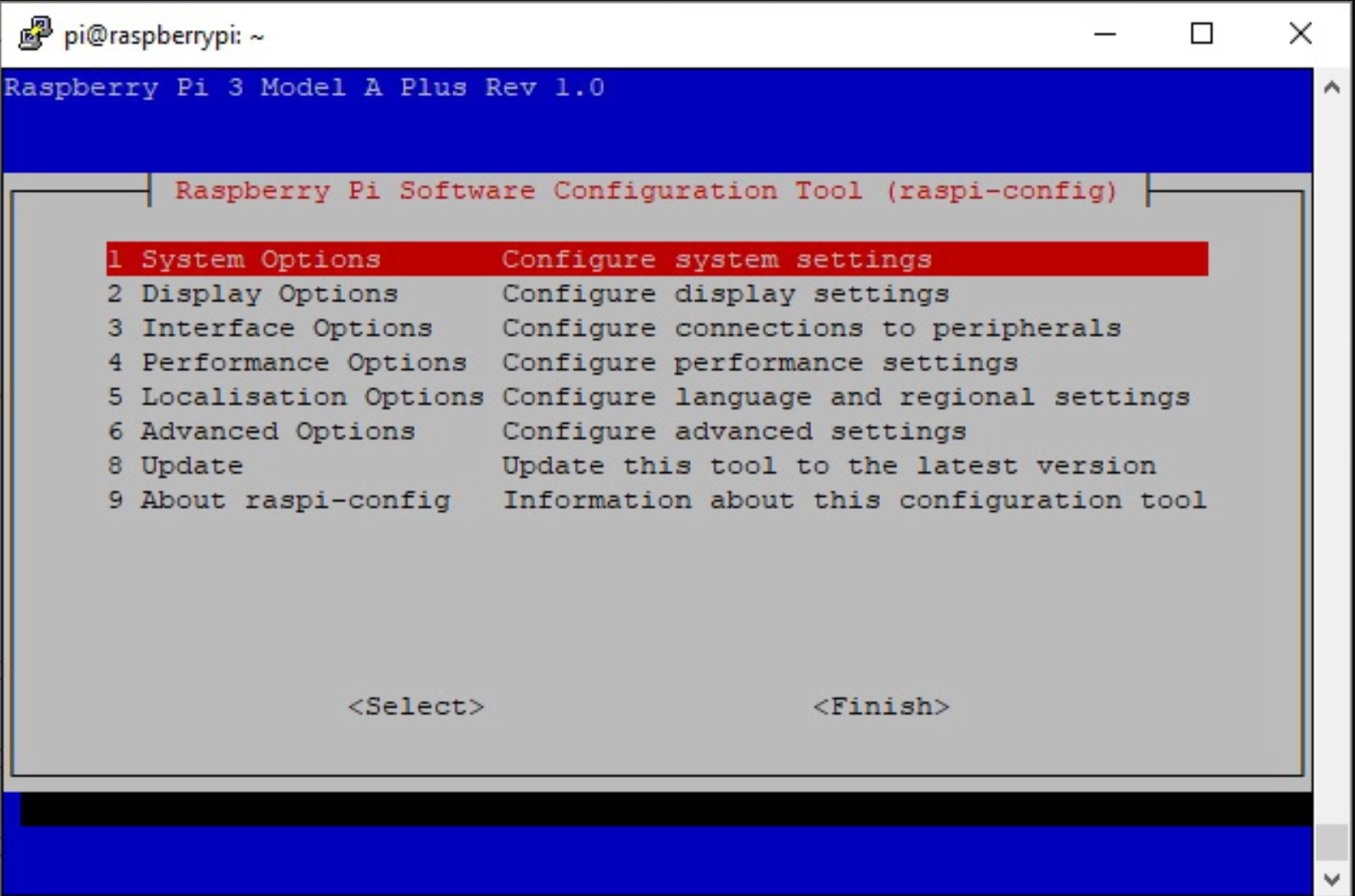
**Installing Zbar**:

Zbar is an open-source barcode reading library that could detect and decode QR codes and works with python. We can use pip installer to get Zbar library.



**Enabling the camera:**

****

 **From the interface options (3), camera can be enabled. Upon enabling the camera, the system asks for a reboot. After which, the camera module could be used.**

**Code Development:**

**Importing Libraries:**

**Importing the required libraries. Most of them are built-in and the rest are downloaded already. Picamera allows us to access the camera module and provides a large number of additional features.**

****

**Camera Setting:**

**By default, the resolution of the Picamera is set to the resolution of the monitor, the maximum resolution is 2592x1944 for still photos and 1920x1080 for video recording, while the minimum is 64x64. A resolution of 640x480 selected in this case so the tiny QR codes are clearly visible and supports higher frame rate. Shutter speed is set to 10000 microseconds or 1/100 of a second to allow enough time to for the sensor to expose to the light and provide a clear picture. The frame rate is set to 32 to be able to capture the QR code on a moving frame. The shutter speed is double the frames per second to prevent any blurring of images. Switching off the auto white balance mode disables the camera’s automatic white balance. PiRGBArray produces a 3-dimensional RGB array from an RGB capture. This custom output class can be used to easily obtain a 3-dimensional NumPy array, organized (rows, columns, colors), from an unencoded RGB capture. Sleep time of 0.1 seconds allow the camera to warm up before starting the capture.**

****

**Bitwise not:**

**The 3D frame thus captured, is processed to extract the QR-Code information. First, the image is converted into grayscale, this reduces the number of channels to one and thereby reduces the computational time needed. As mentioned earlier, the QR-code on the module is printed in white on a dark background which is contrary to the general case where the code is printed in black. Hence, we use bitwise not operation to flip the pixel values and therefore obtain the QR-code in black.**

****

**Thresholding:**

**Now, there is a need to obtain the image into black and white before to pass it into the decoding function. Selecting a thresholding technique is the most important aspect of the project. The global thresholding techniques like Otsu thresholding, Binary thresholding produce wonderful results when the light condition is optimal. However, it will be not the case every time, as is evident from the following two examples. Adaptive thresholding techniques performs better in different lighting conditions. In adaptive mean thresholding the threshold value is the mean of the neighborhood area minus the constant C. whereas in the adaptive threshold gaussian the threshold value is a gaussian-weighted sum of the neighborhood values minus the constant C. The weights keep reducing for the values away from the center. In both cases, the block size determines the size of the neighborhood area and C is a constant that is subtracted from the mean or weighted sum of the neighborhood pixels. Therefore, Adaptive thresholding is selected.**

****

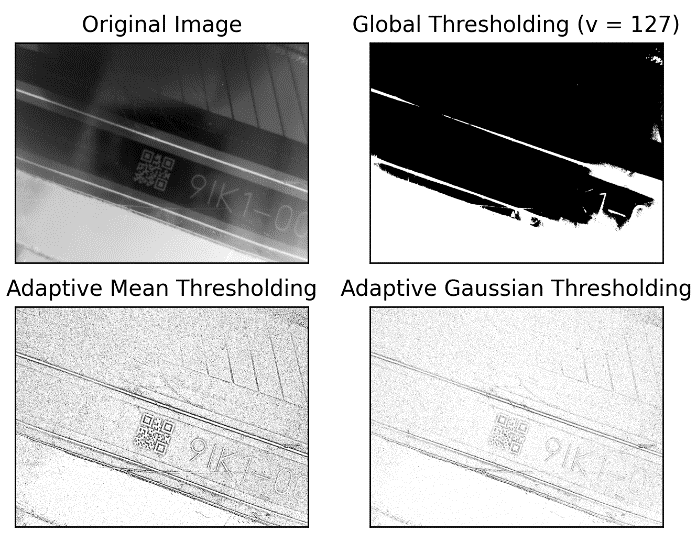
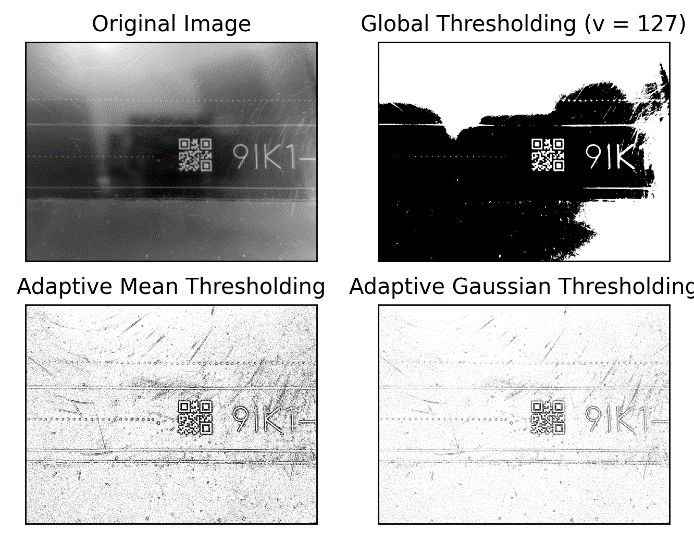


Fig: 2.3.1

Fig:2.3.2

**Decoding using Pyzbar:**

**The last part is decoding the threshold image. The image is passed into the function pyzbar which detects the QR-code and decodes the information, the data obtained from the QR-code is then decoded and saved as ‘mycode’.**

****

**Still to mention.**

* **HTTP API**
* **Photo of equipment.**
* **-----**

****