GNG1103

Design Project User Manual

3D Printer Management System

Submitted by:

District 9, Group 9

[Michael Puma, 300069597]

[Michael Dolan, 300059454]

[TEAM MEMBER 3, STUDENT NUMBER]

[TEAM MEMBER 4, STUDENT NUMBER]

[TEAM MEMBER 5, STUDENT NUMBER]

Date: December 7, 2019

University of Ottawa

Abstract

This user manual will allow the reader to provide, maintain, or modify our design project. The design project is a 3D printer management system that uses Ross DashBoard and a Raspberry Pi microcontroller. The goal of this design project is to automate the CEED Makerspace. The device allows the easy signing out and returning of SD cards, and allows CEED staff to keep track of print times, and collect data on printers and their users.

The Pi is responsible for creating a server, and reading QR codes employing python script. Dashboard controls the user interface in which the user will use to see the active printers as well as initiating the connection with the Pi. The QR codes are attached to pre existing SD cards used in the makerspace, and when scanned log that printer as signed out on Ross Dashboard.

Table of Contents

List of Figures

List of Acronyms

Acronym	Definition
RPI#	Raspberry Pi (version #)
UI	User Interface
CEED	Center for Entrepreneurship and Engineering Design
fps	Frames per second

1 Introduction

The goal for the fall 2019 GNG1103 class was to automate an aspect of any of the CEED areas using the Ross DashBoard software. This design project's goal was to automate the 3D printers in the CEED Makerspace with a 3D printer management system that uses a Raspberry Pi microcontroller interfaced by Ross Dashboard. This user manual will allow anyone to reproduce the original design, troubleshoot problems, or modify and produce an improved design. The user manual will provide the steps taken to build and run the original design, including: how to build the prototype, how to use the prototype, and how to maintain the prototype. In addition, recommendations for next steps will be provided.

Figure 1: Final design of 3D printer QR scanner

The problem statement created for our design project was: a need exists for the CEED employees that optimizes the CEED building through automation using Ross DashBoard and a microcontroller; that is under the $100 budget and is easy to operate and maintain. It is desired

that the automation will reduce the workload on the CEED employee, the student, or both. Our project creates a fast, easy, and expandable system that allows a CEED employee to easily sign out, sign in, and keep track of 3D printers and their corresponding SD Cards. In addition our system allows the easy collection of user feedback.

Our product is based off a Raspberry Pi 4 and employs the use of a RPI Camera. The camera runs custom software that allows the Pi to process and relay information read from QR codes. The QR codes are designed to be printed and attached to an SD card which corresponds to a 3D printer in the makerspace. A CEED employee will have access to both the RPI as well as a dashboard UI. The UI allows for the staff member to easily scan a QR Code, signing it out, or returning it if already signed out. The UI also displays the number of active printers, and the time in which the printer was signed out. The Pi itself also stores a log of active printers, as well as a list of all QR Codes scanned as well as the time the code was scanned in, and out. What makes our project special is the easily expandability, if the makerspace wishes to add more printers, not modification needs to be made to RPI, its software or the Dashboard UI. Simply by printing more QR codes and assigning them a unique value that corresponds to a 3D printer, the scanner will automatically sense the new QR code, and add it to the system without any user modification required.

2 How the Prototype is Made

The Raspberry Pi 4 was chosen due to its performance. For a microcontroller the RPI4 is one of the most powerful for the cost. This was required due to the power hungry QR scanning code that must be running at all times, along with the server that hosts the dashboard. Both a fan and heatsink were added to increase the airflow through the microcontrollers case, and allow it to be cooled adequately. The 1080p, 30fps RPI camera was chosen due to the quality of imaging. This allows for a more accurate scan, and reduces the chances of failure. A cheaper lower quality camera could have potentially been used, but this was not tested due to budget constraints. A fans 2 pin headers were splices and soldered to the Raspberry Pi 4 board. All other electrical components were ran directly through plug and play ports built into the RPI4, including a type-C power connecter, and the camera ribbon port

For the software two codes must run simultaneously on the RPI4 itself, and a third code built into the Dashboard UI. The first code is responsible for analyzing the video process scanning for QR codes, and reading their values, as well as relaying that value to a CSV file on the RPI's desktop. The second code creates a server that allows for the connection between the Dashboard UI running on a separate computer and the Pi itself. This code also is responsible for relaying data between the Pi and dashboard. The third code runs on dashboard, and makes a request to the Pi to send data when a button is pressed.

2.1 Mechanical

2.1.1 BOM (Bill of Materials)

Material	**Cost**
Raspberry Pi 4	46.95
Raspberry Pi 4 Case	6.95
Camera Module	32.98
USB-C cable	4.30
SD Card 16gb	4.99
Final total	**96.17** + tax

2.1.2 Equipment list

- Soldering Iron
- Wires with male-female pin outputs
- Display for the Raspberry Pi (HDMI based)
- Miscellaneous adapters (HDMI to Nano hdmi)

2.1.3 Instructions

Step 1: Building the Pi

First obtain the official RPI4 Case as seen in Figure 1. To mount the fan to the Pi, a 1.5” hole saw was used along with a drill press. Four holes were also drilled to allow for the mounting of the Fan to the Pi itself. Four screws were used to attach the fan to the lid of the case as seen in Figure 2. Cut the end off of the fan seen in Figure 3. Solder the red wire to the 5V header (Pin 4) and the black wire to the ground pin (Pin 6), see Figure 4 for the pin layout. Connect the ribbon cable for the camera to the camera header on the Pi, and run the cable through the USB port opening on the case.

Figure 2: Pi with case and fan mounted

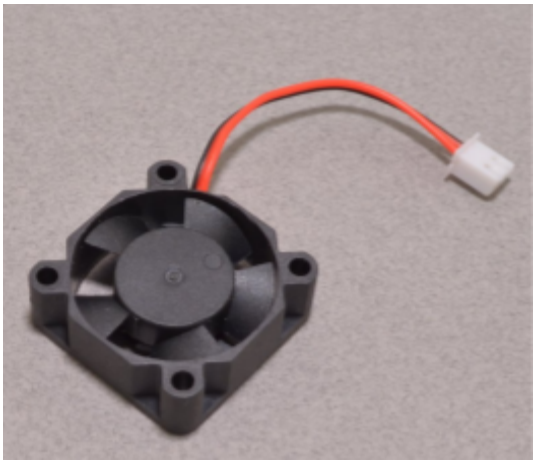


Figure 3 (left): 5V brushless 30mm fan

Figure 4 (right): Raspberry Pi 4 pin layout

Step 2: Programming the QR Code Scanner

This code employs the use of both a virtual environment as well as OpenCV. To create a virtual environment the following commands lines are entered in the terminal on the Pi itself. A tutorial was followed, which can be found here: https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/, with the only modification made being naming the virtual environment 'barcode' instead of 'cv' "mkvirtualenv barcode -p python3". A summarized version of this tutorial is as follows:

- First boot the Pi and launch the terminal, and enter the following commands outlined in this tutorial exactly as they are written, excluding the quotations.
- "Sudo raspi-config"
- select 'advanced options'
- Select 'expand filesystem'
- Select 'Exit'
- In the terminal enter
- "Sudo reboot"
- Wait for the Pi to reboot then relaunch the terminal

- “Sudo apt-get update && sudo apt-get upgrade”
- “Sudo apt-get install build-essential cmake unzip pkg-config”
- “sudo apt-get install libjpeg-dev libpng-dev libtiff-dev”
- “sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev”
- “sudo apt-get install libxvidcore-dev libx264-dev”
- “sudo apt-get install libgtk-3-dev”
- “sudo apt-get install libcanberra-gtk*”
- “sudo apt-get install libatlas-base-dev gfortran”
- “sudo apt-get install python3-dev”
- “Cd ~”
- “wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip”
- “wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.0.0.zip”
- “unzip opencv.zip”
- “unzip opencv_contrib.zip”
- “mv opencv-4.0.0 opencv”
- “mv opencv_contrib-4.0.0 opencv_contrib”
- “wget https://bootstrap.pypa.io/get-pip.py”
- “sudo python3 get-pip.py”
- “sudo pip install virtualenv virtualenvwrapper”
- “sudo rm -rf ~/get-pip.py ~/.cache/pip”
- Add the following lines to the ~/.profile
- “# virtualenv and virtualenvwrapper”
- “export WORKON_HOME=$HOME/.virtualenvs”
- “export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3”
- “source /usr/local/bin/virtualenvwrapper.sh”
- “echo -e "\n# virtualenv and virtualenvwrapper" >> ~/.profile”
- “echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.profile”
- “echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >> ~/.profile”
- “echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.profile”
- “source ~/.profile”
- “mkvirtualenv barcode -p python3”
- “workon barcode”
- “pip install numpy”
- “cd ~/opencv”
- “mkdir build”
- “cd build”

“cmake -D CMAKE_BUILD_TYPE=RELEASE \

```
 -D CMAKE_INSTALL_PREFIX=/usr/local \
   -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
   -D ENABLE_NEON=ON \
   -D ENABLE_VFPV3=ON \
```

```
-D BUILD_TESTS=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_EXAMPLES=OFF ..”
```

- “sudo nano /etc/dphys-swapfile”
- Set: CONF_SWAPSIZE=2048
- “sudo /etc/init.d/dphys-swapfile stop”
- “sudo /etc/init.d/dphys-swapfile start”
- “make -j4”
- “sudo make install”
- “sudo ldconfig”
- “cd ~/.virtualenvs/cv/lib/python3.5/site-packages/”
- “ln -s /usr/local/python/cv2/python-3.5/cv2.cpython-35m-arm-linux-gnueabihf.so cv2.so”
- “Cd ~”
- Exit and open a new terminal
- “workon barcode”
- “Python”
- “import cv2”
- “Cv2.__version__”

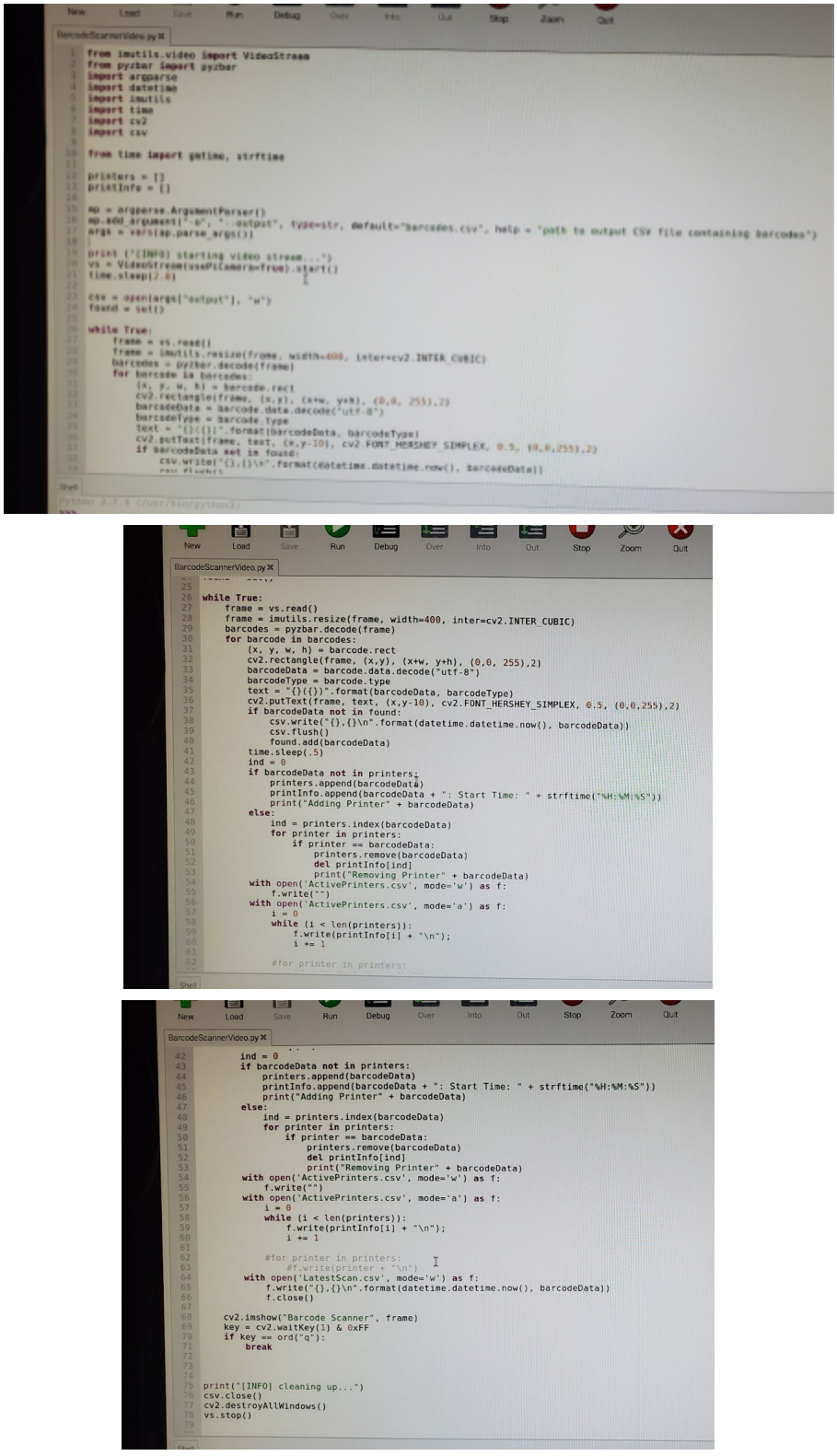
You should see ‘4.0.0’, if so then everything was a success. If not double check the commands to ensure you entered everything correctly, or consult the troubleshooting section found on the website.

- “exit()”

The virtual environment has been made and the OpenCV software has been installed. Now a python script can be made to utilize the software, and scan the QR codes. The following command lines were entered, and python code was made.

- “sudo apt-get install libzbar0”
- “brew install zbar”
- “mkvirtualenv barcode -p python3”
- “workon barcode” - this code allows for the working on the barcode virtual environment
- “pip install pyzbar”

Next Create a python script on the Pi and label it “BarcodeScannerVideo.py”, and enter the following code:



```
from imutils.video import VideoStream
from pyzbar import pyzbar
import argparse
import datetime
import imutils
import time
import cv2
import csv

from time import gmtime, strftime

printers = []
printInfo = []

ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", type=str, default="barcodes.csv", help = "path to output CSV file containing barcodes")
args = vars(ap.parse_args())

print ("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

csv = open(args["output"], "w")
found = set()

while True:
    frame = vs.read()
    frame = imutils.resize(frame, width=400, inter=cv2.INTER_CUBIC)
    barcodes = pyzbar.decode(frame)
    for barcode in barcodes:
        (x, y, w, h) = barcode.rect
        cv2.rectangle(frame, (x,y), (x+w, y+h), (0,0, 255),2)
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        text = "{}({})".format(barcodeData, barcodeType)
        cv2.putText(frame, text, (x,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),2)
        if barcodeData not in found:
            csv.write("{},{}\n".format(datetime.datetime.now(), barcodeData))
            csv.flush()
            found.add(barcodeData)
    time.sleep(.5)
    ind = 0
    if barcodeData not in printers:
        printers.append(barcodeData)
        printInfo.append(barcodeData + ": Start Time: " + strftime("%H:%M:%S"))
        print("Adding Printer" + barcodeData)
    else:
        ind = printers.index(barcodeData)
        for printer in printers:
            if printer == barcodeData:
                printers.remove(barcodeData)
                del printInfo[ind]
                print("Removing Printer" + barcodeData)
    with open('ActivePrinters.csv', mode='w') as f:
        f.write("")
    with open('ActivePrinters.csv', mode='a') as f:
        i = 0
        while (i < len(printers)):
            f.write(printInfo[i] + "\n");
            i += 1

        #for printer in printers:
            #f.write(printer + "\n")
    with open('LatestScan.csv', mode='w') as f:
        f.write("{},{}\n".format(datetime.datetime.now(), barcodeData))
        f.close()

    cv2.imshow("Barcode Scanner", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break


print("[INFO] cleaning up...")
csv.close()
cv2.destroyAllWindows()
vs.stop()
```

Figure 5: Python code for the Barcode Scanner

The following python script must be launched through the virtual environment. To do this a terminal is opened a the following commands are entered.

- “Source ~/.profile”
- “Workon barcode”
- “Cd Desktop”
- “Python BarcodeScannerVideo.py”

Step 3: Programming the Pi server

Using python on the Pi, the code below is entered with the Port variables value set to an open port on your windows device. This is found by opening the powershell on your computer and entering the command “netstat -an”. This code is run by clicking the play button on python, which initiates the Server on the Pi.

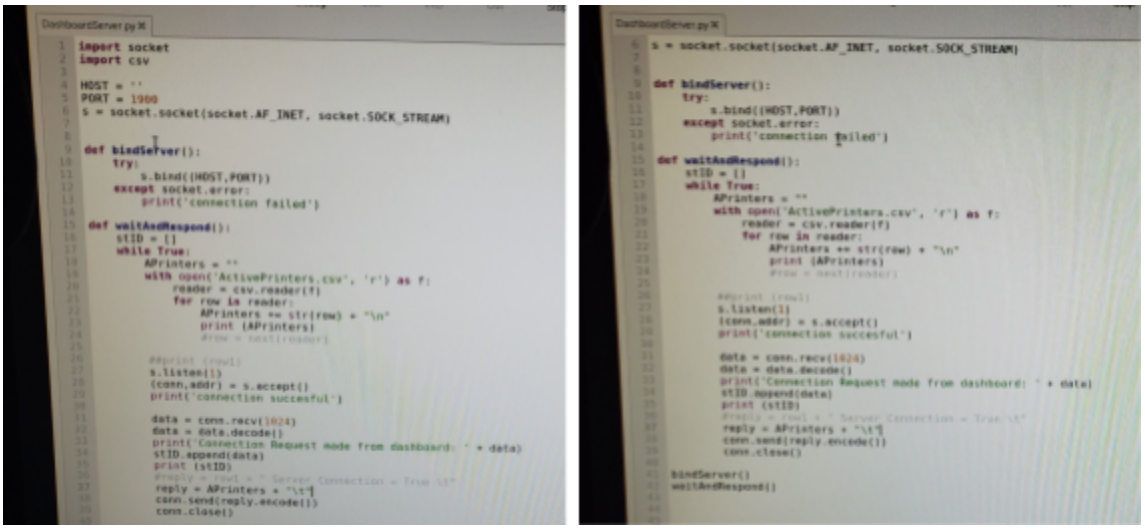


Figure 6: Python code for the Dashboard Server

Step 4: Creating the DashBoard UI

First step is to create 3 text fields which contain the parameters as followed, makerspace ID, Pi's IP address, and finally the Port. Labelled StudentID, PiAddr and piPort respectively. Next create a read only text field named PrinterStatus. Then create the sign out and return button. This button is responsible for calling the Pi and requesting a list of active 3D Printers. The script for the button can be seen in Figure 7. The script references 3 of the text fields created, including the PiAddr, piPort and StudentID parameters. The function then changes the PrinterStatus text field to show the response from the raspberry Pi. The functionality of the Dashboard is now complete, the color and layout can be changed to make it more esthetically improving, as seen in Figure 8.

```
<button buttontype="push" flat="true" height="140" left="40" name="Sign Out and Return" style=
"bdr#FFFFFF;bdr:round;font:mono;size:Big;fg#FFFFFF;" top="20" width="340">
        <task tasktype="ogscript">/*! block id=1009,1010,1011 !*/
//rosstalk.sendMessageWithResponse(params.getValueAsString('PiAddr', 0), params.getValueAsString('piPort', 0), "", "", null);
rosstalk.sendMessageWithResponse(params.getValueAsString('PiAddr', 0), params.getValueAsString('piPort', 0), params.getValueAsString('StudentID'
, 0), "\t", function callbackfunction(success, sentData, resultString, exception) {
  params.setValue('PrinterStatus',0,resultString/*params.getValue('PrinterStats')*/);
});

/*!!
 &lt;block id="1009" type="ogscript_sendtcpstringwresponse" x="449" y="74" w="318" HOST="ID:1010" PORT="ID:1011" STRING="" ENDOFMESSAGE="" CALLB
ACK="null" /&gt;
&lt;block id="1010" type="param_getstringvalue" x="142" y="51" w="318" PARAM="[root, Params for Project.grid, Not In Menu, PiAddr (PiAddr)]" /&g
t;
&lt;block id="1011" type="param_getstringvalue" x="149" y="123" w="318" PARAM="[root, Params for Project.grid, Not In Menu, piPort (piPort)]" /&
gt;
!!*/
/*!!&lt;checksum&gt;ec0e682d2af9e0621d226989a14a10e1&lt;/checksum&gt;!!*/</task>
        </button>
```

Figure 7: Ross DashBoard Sign In/Out Code

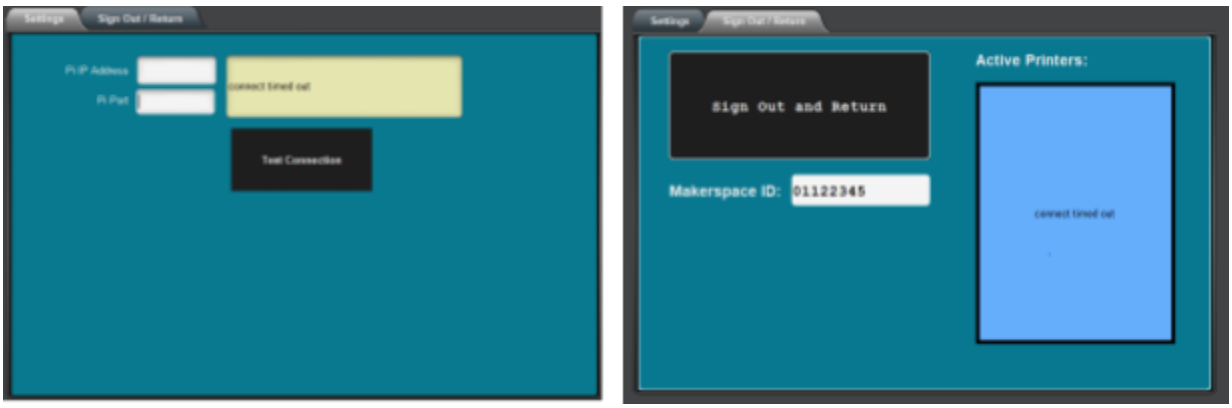


Figure 8: Finished Ross DashBoard UI

Step 5: Creating the QR Codes

To create the QR codes a free online generator can be used. We chose https://www.qr-code-generator.com/. When creating the QR code the value that it contains is a text value, with the value being the label desired for the printer. This label will be printed in the active printers list. For the sake of simplicity we assigned the value a number that corresponded to the 3D printer ie. QR code for printer 1 contains the text value 1. The QR code should be printed to the size of the SD card but made as large as possible to ensure that it can be easily read by the Pi's Camera.

3 How to Use the Prototype

The prototype works by moving a QR code in front of the Raspberry Pi's camera. After a few moments, the value of the code will be sent over the network to the Ross Dashboard. Ross Dashboard will log the scanned code and time, and the 3D printer will be recorded as in use (signed in) or available (signed out).

The prototype does not have any special installation nor safety steps, as it is essentially a small computer. As long as the prototype has a power outlet, access to fresh air, and is away from liquid, it will function fully.

4 How to Maintain the Prototype

From our testing we found that the Pi did not overheat when the fan component was added along with a small heatsink. Although to avoid failure the Pi should be shut off after an extended period of time and allowed an adequate amount of time to cool. Our prototype produces a large amount of heat, which over time can wear out the components. The QR Codes should be reprinted when damaged to ensure that they are clear for the camera to read. The camera itself should be cleaned whenever possible to ensure that the lens is not covered, which would impede with scanning accuracy.

5 Conclusions and Recommendations for Future Work

In conclusion, our task of automating the CEED Makerspace using Ross DashBoard and a Raspberry Pi was successful. The prototype worked by reading specially made QR codes linked to each 3D printer (via their SD card). With DashBoard, the scanned QR code would be linked to a user and logged when the SD was taken and returned.

Some lessons learned include how the optimization of code is crucial when using low processing power devices such as a single core Raspberry Pi. Just because something will work on paper does not mean it will work in actuality, hence the failed Pi 0W. Start planning earlier; starting during the beginning weeks would have allowed this projects end result to be more polished. Next steps include adding functionality to the makerspace ID box, allowing for emails to be sent directly to users to gather feedback. As well as adding a link between active printers and the makerspace website in order to present users with information on the status and availability of 3D printers.

6 Bibliography

Rosebrock, Adrian. “Install OpenCV 4 on Your Raspberry Pi.” *PyImageSearch*, 28 June 2019, www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/.

QR Code Generator, www.qr-code-generator.com/.

APPENDICES

APPENDIX I: Design Files

There were no design files required. Our MakerRepo website linked below contains all the deliverables for this project as well as photos of the finished product.

https://makerepo.com/MDola059/gng1103-group-a9-sd-card-qr-code-scanner