**INVENTORY CONTROL WITH PATTERN RECOGNITION AND RFID**

Ryan Metzler

Joe Masar

Nam Do

Paul Scaccia

Pierre Jackson

Nick Cerutti

Department of Electrical Engineering and Computer Science, Cleveland State University

Submitted to—

Faculty Advisor: Dr. Zhao

Sponsor: ZIN Technologies

Sponsor Representative: John T. Zoldak

**Contents**

**1.0 Abstract 4**

**2.0 Introduction 4-7**

2.1 Zin Technologies 4

2.2 Problem Statement and Background 4-5

2.3 Standards 5-6

2.4 Design Constraints 6-7

**3.0 Design Objectives & Introductory Information 7-14**

3.1 Physical Hardware 7-10

3.2 RFID Hardware 10-11

3.3 RFID Software 11-12

3.4 OCR Software 12-13

3.5 Data Acquisition 13

3.6 Delimiting Software 13-14

**4.0 Optimized Inventory Control 14-21**

4.1 Technical Solution 14

4.2 RFID 14-15

4.3 OCR and OpenCV 15-19

4.4 Data Acquisition 19-21

4.5 Delimiting Data 21

**5.0 Obstacles Overcome 21-24**

5.1 RFID 21-22

5.2 OCR and OpenCV 22-23

5.3 Data Acquisition 23

5.4 Delimiting Data 23-24

**6.0 Budget 24**

**Contents Continued**

**7.0 Future Considerations 24-25**

7.1 Multiple Scan Error Detection 24-25

7.2 Improved Lighting 25

7.3 Custom Lexicon 25

**8.0 References 26-27**

8.1 NASA Label Standards 26

8.2 IEEE SPI Standards 26

8.3 RFID NFC 26

8.4 Optical Character Recognition 26

8.5 Miscellaneous References 26-27

**9.0 Appendix A: 28-34**

9.1 Arduino Uno Schematics 28

9.2 Adafruit PN532 NFC/RFID Controller Schematic 29

9.3 Physical Design Drawings and Specifications 30-31

9.5 Miscellaneous Physical Design CAD Drawings 32

9.6 GUI Xml Code 33-34

**10. Team Qualifications 35-40**

10.1 Resumes 35-40

Paul Scaccia 35

Nam Do 36

Pierre Jackson 37

Ryan Metzler 38

Nick Cerutti 39

Joe Masar 40

1. **Abstract**

The aim of this project was to develop an improved method for inventorying medical consumables prior to their delivery to the International Space Station (ISS). The report details the full engineering life-cycle from recognizing the initial problem to a finished product. For the sake of clarity, we have named our product “Optimized Inventory Control System” and it will be referred to as OICS for the remainder the report.

OICS is an overhead scanner that utilizes optical character recognition (OCR) and radio frequency identification (RFID) to increase the accuracy and speed in which medical consumables are able to be inventoried. This eliminates the need for manual documentation and minimizes human error.

To test the OICS we scanned 22 samples. Each sample featured a label with a RFID tag. Of those 22 samples, 16 of them scanned successfully (both the OCR and RFID data was detected correctly) on the first scan. This gives the OICS a 73% success rate on the first scan. The remaining 6 tags were successfully scanned on a second attempt. It is worth noting that the user reviews the results of the scan prior to prompting the program to save the data. For this reason, no errors entered the final output spreadsheet.

1. **Introduction**

**2.1 Zin Technologies**

ZIN Technologies is a small business industry leader providing advanced engineering services and product development solutions for NASA, DoD and private industry. They provide systems design, development, engineering and integration, test and evaluation, advanced aerospace systems modeling and simulation that support the full life-cycle development of aerospace hardware and software [15]. ZIN has over 25 years of experience partnering with NASA management, scientific experts and industry to manage and develop space systems. ZIN’s mission is to provide cost effective innovative solutions, products through advanced technology, superior engineering and management excellence [15].

ZIN Technologies has graciously granted us the opportunity to develop an optimized method of managing inventory of medical consumables. We are grateful for their support in all aspects of this project.

**2.2 Problem Statement and Background**

Aboard the International Space Station (ISS) the astronauts have what NASA calls medical convenience kits. NASA’s pharmacy currently places printed labels with a Radio Identification tag under each label on each item in the medical convenience kit. When packing a shipment for the International Space Station each items label and RFID information is manually entered into an Excel spreadsheet. Obviously this method is not as efficient as it could be. This method is not taking advantage of the RFID tags. However, on the ISS they are taking advantage of the RFID tags. In space each medical pack item is being scanned with a RFID scanner once a month and the excel spreadsheet inventory is automatically updated.

Manual data entry takes valuable time and labor. In today’s highly technologically advancing world, manual data entry should never be used as a long term solution. As mentioned previously, the current system already includes placing RFID tags on each medical item. By autotomizing the process our system will take advantage of these tags and significantly increase the efficiency of the process by decreasing the time required to document the contents of each medical convenience kit. The excel spreadsheet will be formatted accordingly to match the spreadsheet from the ISS inventory. That will allow for easy comparison which will yield the overall usage of the medical items.

**2.3 Standards**

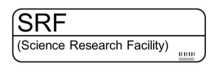
NASA has developed in depth standardization of their medical consumable labeling. Each label meets the following standards set by NASA in accordance with Japan Aerospace Exploration Agency (JAXA) and the European Space Agency (ESA) under Appendix O of the Pressurized Payloads Interface Requirements Document [1].

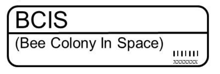
From section O.3.4.3 under part C Labeling Standardization: [1]

Each IMS Combination Label are as specified below.

1. Each label is to contain a horizontal line. A vertical line may be used for vertical space limitations.
2. Only the OpNom for the item to which the label will be applied is to appear above the line or to the left of the line.
3. The payload’s acronym (if applicable) is recommended to be spelled out on the main unit’s hardware ID label. This is recommended to be placed directly below the line.
4. The part number and serial number (if applicable) are below the line, and below the spelled out payload name.
5. If the Inventory Management System (IMS) barcode is integrated with the hardware ID label, it is to fall below the line, and be placed in the lower right hand corner of the label.
6. On control panel name labels, the OpNom is recommended to be above the line. The spelled out name, if needed, is recommended to be applied under the line.
7. No other text, other than that mentioned above, is to appear on the hardware ID label.

Examples of the labels follow in Figure 1:





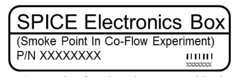


Figure 1 – Examples of NASA standardized labels for medical consumables

**2.4 Design Constraints**

Many different design constraints were taken into consideration throughout the life-cycle of the design process. The constraints considered were largely RFID operation, user friendliness, price, size, modularity, and camera operation. Each of these considerations, and how they effected the final design, will be discussed in more detail in the following paragraphs.

The first concern during the design process was creating a product whose physical construction would not interfere with the signals being exchanged between the interrogator and the tags in the RFID system. Materials with a high conductivity could absorb or block the radio frequency signals. It was for this reason we opted to create the base of our system out of acrylic. The interrogator is embedded in a hollow cavity inside a thick sheet of acrylic. The RFID tag is placed on the surface of the acrylic base. This material will allow the RFID to communicate without any interference.

Our second largest concern was the user friendliness of the final design. We were aware that the product would be used by pharmacists who likely did not have a technical background. For this reason, we put emphasis on a simple graphical user interface that could be used by anyone with no technical experience.

The cost of the OICS largely effected our choices in the software design process. We determined that it was not realistic to design an entire OCR program from scratch. It was at this point we considered an open-source OCR program or purchasing an OCR program. In the interest of keeping costs realistic, we opted to use an open-source OCR program. Opting to use the open-source code may have compromised some of the accuracy of the OCR. Additionally, the open-source OCR was largely catered to operate on a Linux operating system. Ergo, in a sense the cost metric was largely responsible for the system operating on Linux as opposed to Windows.

Next we considered size when designing our product. Initially we wanted to make the OICS hand-held. This would give the user flexibility and make the product portable. However, we were concerned a hand-held unit would rely on the user to remain ridged (no shaking) to take a quality photo for the OCR. It was this concern that ultimately led to our choice to make a static overhead system.

We wanted to keep the system relatively modular. During the design process, we were mindful to use parts that were common and standardized. We wanted a design that could easily be dissembled, or modified for the best user experience. We also gave precedence to parts that could easily be replaced in the event of failure.

Lastly we knew the system need to contain a camera to take a photo for the OCR. It was from this need that arose the call for an “inverted L” overhead design. The camera would be suspended over the main platform and could take a quality photo of whatever medical consumable was placed on the platform.

**3. Design Objectives & Introductory Information**

**3.1 Physical Hardware**

When designing the OICS, several key characteristics were considered.  First, the general functionality of what operations OICS should perform needed to be understood.  The two basic tasks this apparatus must perform are to capture an image, and to scan an object, with the acquired information from each task being imported into a delimited file.  The captured images are taken by a webcam device and must be able to be processed by OCR software.  The scanning of the objects must be utilized by an RFID transmitter/receiver device.  Rather than having to operate two separate devices to achieve this, a major goal of this project was to create an apparatus that could incorporate these two devices and their respective output functions into one apparatus.  A major aspect of our design intent was to create an apparatus that would function reliably.  Also, because the two devices are fragile electrical components, securely attaching or enclosing the two devices into one apparatus was also considered.

After meeting with Zin Technologies about the specific purpose and application of this apparatus, we learned that it did not need to be mass produced.  This opened the path of custom design and manufacturing for us to follow.  We learned that the application was not mobile, nor did it need to be.  Due to this a mobile hand held apparatus was not necessary.  A table top design would be desirable.  Although a table top design is stationary, this stability we thought would aid in producing consistent and reliable images and scans.  We did not want to create an apparatus in which the user was needing to frequently adjust the position of the webcam, nor the position of the RFID device.  The table top design is not intended for mobile application, although we still aimed to make the apparatus as small and compact as possible, so that moving it to another work station would be easy.

The table top design was a good start but we needed to determine more specific designs for how the two devices would be incorporated, oriented, and attached within one apparatus.  First, we understood that the object (bag with label) could easily be placed in a stationary position.  The bag simply needs to rest on its back or top side upon a flat surface.  Due to this, the apparatus could be designed so that the user places the bag upon a flat base with the two devices mounted overhead to capture images and scan the labeled bags.  I immediately thought of structures I have seen while working in laboratories, which consist of framing rods and various connectors.  I could design the entire apparatus around an inverted L-shape structure attached to a base.  The inverted L-shape structure would consist of two framing rods joined by a connector that orients them perpendicular to one another.  The vertical rod would simply be used as support for the horizontal rod, which would extend to one side of the vertical rod.  The horizontal rod would be used to mount the two devices, while being raised above the labeled bag.

At this stage in the design process I came across difficulty in finding an effective way to mount the RFID device onto the framing rod.  During this time, I learned that the RFID device could scan the labeled bag from underneath it and did not need to be mounted overhead.  This led me to the idea of securing the RFID device to the base.  Then I would construct a platform (positioned slightly above the RFID device) for the labeled bag to rest upon while it would be scanned from underneath.  At the same time the labeled bag would be scanned from underneath by the RFID device, the webcam would capture an image of the label on the bag from above it.  This was a great idea because it meant that the user had to only place the bag one time and both operations could be performed.

With respect to my designs for a base at this stage, I was planning on constructing a rectangular base made of very thin aluminum or steel.  My other team members informed me that the RFID device cannot be in contact with magnetic surfaces.  Not only should it not be in direct contact with magnetic surfaces, but the device is also sensitive to magnetic disturbances in its near proximity.  Due to this material constraint, I needed to reexamine my design for the base and RFID device.

The idea of a platform being positioned above the RFID device would not work.  The platform was intended to be supported by four small posts, leaving all four sides open between the base and the platform.  This would cause the RFID device to be vulnerable to any other nearby magnetic frequencies which could create error during an intended scan.  I was informed that the RFID device was so sensitive that it would be best to enclose it within non-magnetic material as much as possible. This would eliminate any possible external magnetic disturbances that could be detected by the RFID device, and thus, adversely affect the results during intended scans.  A need now existed for the RFID device to be positioned within a specialized case or enclosure.

I have always thought traditionally about machining parts by the act of removing material rather than additive methods.  For the base of the apparatus, I chose to create a recessed cavity in a block of acrylic for the RFID device to be contained within.  The acrylic is non-magnetic and machinable.  By machining a recessed cavity for the RFID device to be placed in, this surrounds the device by a thick layer of dense acrylic around five of its six sides.  These surrounding boundary layers serve as a barrier, created to protect the RFID device from unintended magnetic frequency response signals.  Additionally, a tray will be machined around the outer border of the recessed cavity.  This tray will support an acrylic insert that will rest above the RFID device and allow the labeled bag to be placed upon.  The acrylic insert serves the same purpose as my initial idea of a platform above the RFID device.  The acrylic insert also helps to protect the RFID device by blocking any external dust or debris.

The recessed cavity is designed to width and length dimensions that are close to the width and length dimensions of the RFID device.  For example, the width of the recessed cavity is three inches and the width of the RFID device is approximately two inches.  The same dimensional offset was applied to the length dimensions.  This was done intentionally to achieve two things.  The first, is that the somewhat tight cavity secures the RFID device in place by prohibiting its movement in all directions.  This allows for its position to remain consistent between consecutive scans.  We also did not want the cavity to be too tight.  The dimensional offsets are in place to allow for ease of placing and removing the RFID device from the base when it may be necessary.  Additionally, the dimensional offsets give the user a small amount of adjustability of the RFID device’s exact position within the cavity.  At the back end of the cavity, a small groove has been machined to allow for the USB cable to extend out of.

The acrylic insert has a half-sphere shape cut away from its back side which, when placed in the tray of the cavity, will align with the groove for the USB cable.  This cutaway helps to give more clearance for the protruding USB cable, but also serves the purpose of a finger hole for the user to easily remove the insert whenever access to the RFID device is needed.

In Figure 2 you can see a computer aided design of the final design. For additional detailed photos, CAD drawings, and exploded views of the final physical design please see the appendix.

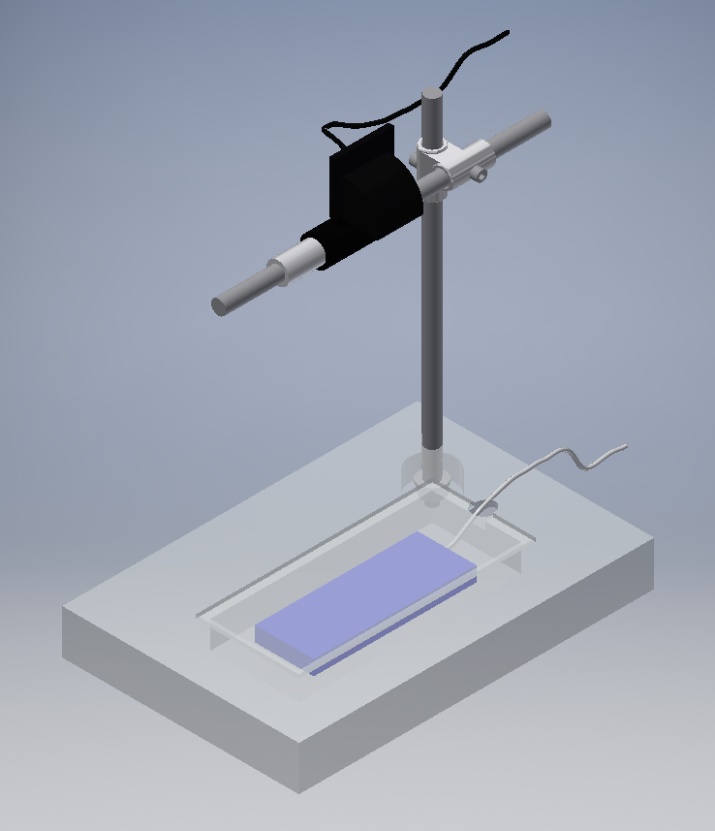


Figure 2- CAD design of the final physical design

**3.2 RFID Hardware**

A radio-frequency identification system is comprised of two pieces, the tags (sometimes called labels) and the interrogators (sometimes called readers). The interrogators are two-way radio transmitter-receivers that send a signal to the tag and read its response. Tags fall into two categories, passive and active. An active tag has an onboard power source that periodically transmits the tag's identification signal. Passive tags use the signal from the transmitter to reflect information to the interrogator.

The RFID system we are using for this application is Near Field Communication (NFC). Near Field Communication is a subset of RFID. NFC is a set of communication protocols that enables two devices to communicate at short-range. The effective range is 10 centimeters or less at a frequency of 13.56MHz [4].

NFC uses [magnetic](https://en.wikipedia.org/wiki/Magnetic_field) [induction](https://en.wikipedia.org/wiki/Electromagnetic_induction) between two [loop antennas](https://en.wikipedia.org/wiki/Loop_antenna) located within each other's [near field](https://en.wikipedia.org/wiki/Near_and_far_field) on the interrogator and tag. This effectively forms and air-core communication link. Most of the RF energy is concentrated in the allowed ±7 kHz bandwidth range, but the [spectral mask](https://en.wikipedia.org/wiki/Spectral_mask) for the [main lobe](https://en.wikipedia.org/wiki/Main_lobe) is as wide as 1.8 MHz [4].

This NFC system uses passive tags capable of read/write operation. In this mode, the NFC enabled interrogator can read or write data to any of the supported tag types in a standard NFC data format. In the case of passive tags, NFC uses a Manchester coding technique to transfer data. NFC devices use full-duplex communication meaning they can send and transmit dating simultaneously. This is only logical considering the interrogator needs to transmit power to the tag and receive the reflected information.

**3.3 RFID Software**

The Arduino Uno consists of a ATmega328 microcontroller with a bootloader made by Arduino. This allows for simplicity of coding with many libraries and protocols already defined and included.

With the addition of the Adafruit RFID/NFC “shield” (attaches to top of Arduino headers) further libraries were needed. Adafruit’s entire product line is all open source. Libraries were available from the Adafruit website. Once these libraries were added, simple functions were accessible for programming. The two main header files used are the Wire.h for SPI communications and the Adafruit\_PN532.h used for the NFC controller [16].

The SPI pins, SCK, MOSI, MISO, and SS are defined to the Arduino’s dedicated SPI pins. The Adafruit RFID/NFC shield came without communications protocol connections. Breakout pins go to the header which attaches to the Arduino. The SPI pins were connected via soldering wires from the PN532 SPI pins to the header pins.

Then in the setup the baud rate of the serial monitor is set to 9600 with the function:

Serial.begin(9600);

From here, the code runs the nfc.begin() function (Figure 3) and enters the Main loop. Once a NFC tag is placed near the scanner, the uint8\_t reads a success. Then the identification information is saved to a buffer variable for storing as shown in the code below.

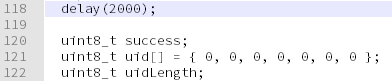


Figure 3 – nfc.begin() code

Executing after a success, the serial print begins printing out serial data (Figure 4) with the following code.

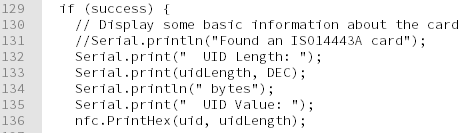


Figure 4 – printing results after a successful scan

For the purposes of the OICS, we omitted all printouts except the UID Value itself. This is the only value we document in the inventory system.

**3.4 OCR Software**

Optical Character Recognition is the conversion of images of typed or handwritten text into machine-encoded textual data. The complete process for OCR falls into three categories, pre-processing, character recognition, and post processing. A discussion of each stage follows.

Pre-processing is the conditioning of images to improve the successful recognition of characters. Within the preprocessing stage there is multiple techniques to improve the quality of the image. De-skewing takes place when the image taken was not property aligned. It may be rotated clockwise or counterclockwise to properly align the image for character recognition. Despeckling can be used to remove negative spots from the photo and smooth edges. Binarization converts an image from color or greyscale to a black-and-white photo. Line removal cleans up non-glyph boxes and lines. These are a few of the most popular technique for conditioning an image. There are several other techniques than can be used to improve image quality. For the sake of being concise we will conclude pre-processing techniques with only the most popular techniques.

Character recognition is the core of the OCR algorithm. There are two types of character recognition algorithms, matrix matching and feature extraction.

Matrix matching compares an image to a stored glyph on a pixel-by-pixel basis. The

method works best with type written text. It often has a library of known existing fonts such as Times New Roman or Calibri. The image is then compared against all the fonts in the libraries. This technique is ideal because of its simplicity but obviously is not useful for handwritten text.

Feature Extraction decomposes glyphs into features like lines, loops, and intersections. The extraction reduces the number of possible features and therefore makes it computationally efficient. The features are compared to a vector representation of the various glyphs in the alphabet. This method often uses a two-pass approach. The second pass is known as adaptive recognition. The letter shapes that are recognized on the first pass are used to better recognize the remaining letters on the second pass. For example, if the first pass detects “The boy threw a \_\_\_\_\_”, the second pass would use this information to intelligently hypothesis the final word if it was not immediately obvious from the feature extraction. This is ideal for uncommon fonts or hand-written writing.

The final stage of the OCR process is post-processing. The accuracy of the OCR can be increased by comparing the output of the character recognition stage to a lexicon. A lexicon is a list of words that can occur in a document. In our particular application, the lexicon would be limited to words in the English language and further refined to words that would occur in a pharmacy application. More powerful post-processing programs use near-neighbor analysis. Near-neighbor analysis uses the already calculated results to make intelligent decisions about future character recognition. For example, it the word “bent” was used 10 times previously in the document, then the word “lent” was detected, it would signal to the program that an error may have occurred.

**3.5 Data Acquisition**

There are two pieces of data the OICS acquires for each medical consumable.

The printed IMS label is captured in a photo by the web camera. The web camera sends the photo via USB cable to the central computer where OCR software determines the text written on each label.

Each IMS tag has a RFID tag. The ID is scanned using an Arduino, an open source ATMega328 microcontroller, with an Adafruit PN532 NFC/RFID Controller Shield. The Scanner antenna communicates with the IMS label tag and receives the ID. The ID is then sent to the PN532 controller and sent out via SPI to the Arduino. From the Arduino, the ID is sent as serial data over a USB cable to a central computer containing our OICS software.

**3.6 Delimiting Software**

Delimiter-separated values are a technique for storing data. Data is stored in a two-dimensional array of data by separating the values into unique rows and columns. Delimited data falls into two categories, comma-separated values (CSV) and tab-separated values (TSV).

A comma separate value file stores tabular data in plain text. Each line in the file is a data record. Each data record consists of one or more fields. Each field is created using a comma in each data record. This method can become complicated when field data contains commas that are intended to be raw data and not the indication of a new field. Some CSV files simply cannot handle such field data. Typically escape characters are utilized to differentiate between commas that are intended to be stored as data and commas that are intended to create fields.

A tab separated value file is very similar to a CSV file. Like CSV each line in the file is a data record. However, a tab is used to create data records within each data record.

Both these file types are an ideal format for data exchange. It is commonly used to move tabular data between programs that would typically have incompatible data formats. This attribute makes tab separated values files ideal for our OICS application.

**4. Optimized Inventory Control**

**4.1 Technical Solution**

The system we developed incorporates OCR via a high-end webcam along with a RFID, specifically NFC, open source Arduino based scanner. Identification from both the NFC tag as well as the printed label are read into the computer and sent to excel after a single keyboard button press.

For consistency, the Arduino scanner is placed less than an inch below the surface where each medical item will be placed. The camera is mounted vertically overhead the item scanning surface with a vertical bar connected to a horizontal bar via an adjustable right angle bracket. This design also allowed us to adjust the camera to the appropriate position both vertically as well as in horizontally within the frame of the item surface.

The physical system was designed and built in a modular fashion. This was done to allow for adjustments during the products life cycle. Being modular also allows for easy tear down, setup, and repair when necessary. The overall system weighs less than 10lbs and takes up 8” x 12” x 13.5” (WxDxH) of space. Thus, the system is portable with the addition of a laptop to run the software for data collection.

**4.2 RFID**

NASA has standardized their medical device RFID tags as NFC. For this reason, we have chosen NFC specialized RFID hardware. The Adafruit PN532 NFC/RFID Controller has the most common NFC chip used today at its core. Most cell phones and embedded devices that are capable of NFC, use the PN532 for interfacing with RFID and communicating information to a central microcontroller (Arduino in our case) or processor. See Section 9 – Appendices for schematic and PCB layout of the PN532 Arduino attachment.

The PN532 is compliant with ISO/IEC 14443, the international standard of identification low proximity (NFC – Near Field) cards. For this reason, we have used the ISO 14443A MIFARE cards for testing purposes.

The PN532 is also capable of communication via SPI, I2C, and Serial UART. We chose to implement SPI over I2C due to the simplicity of SPI as well as the faster transfer speeds.

Below is the Arduino with the PN532 NFC scanner attachment in Figure 5.

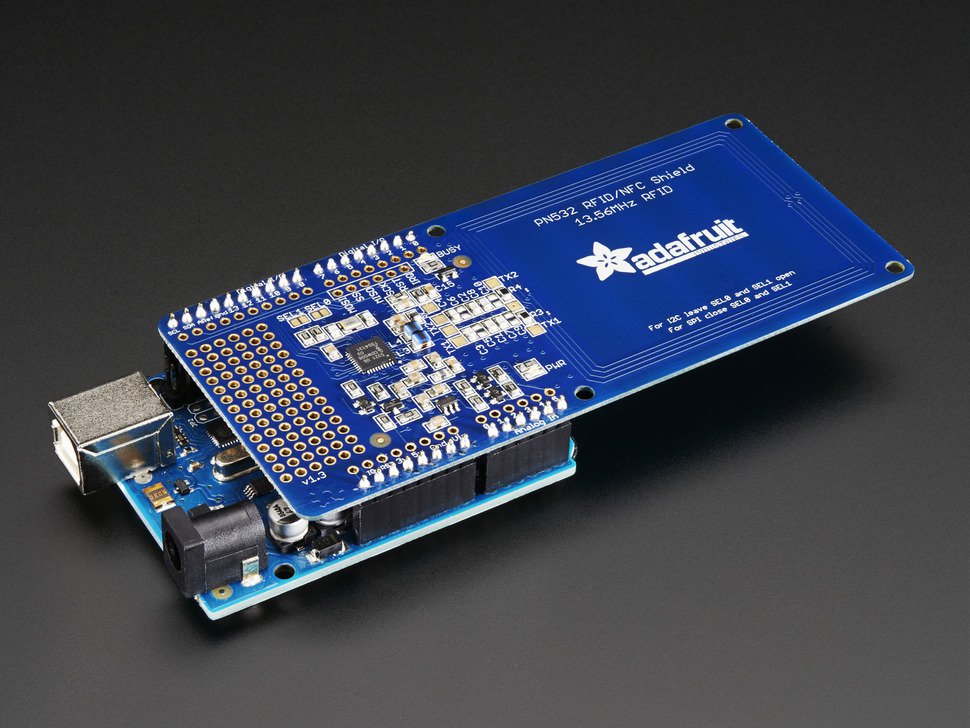


Figure 5 – Arduino Uno with Adafruit RFID/NFC PN532 Shield attached

**4.3 OCR and OpenCV**

Our Application includes a graphical user interface, Tesseract engine, OpenCV engine and RFID engine. The graphical user interface was created using QT5. When installing OpenCV, there is a step that requires the installation of QT creator.

QT creator is a cross-platform C++, Javascript and QML integrated development environment which is part of the SDK for the Qt GUI Application development framework. The QT creator was installed using the following commands:

Sudo apt-get install build-essential

sudo apt-get install qtcreator

sudo apt-get install qt5-default

Once QT creator was installed we created a project hierarchy for the graphical user interface. The project hierarchy is seen below in Figure 6 and the final GUI can be seen below in Figure 7. The Xml code for creating the GUI will be attached to the appendix.

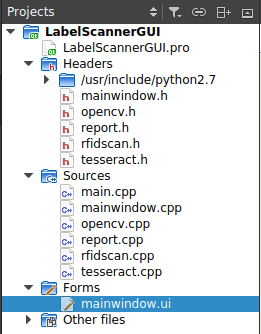
****

Figure 6 – project hierarchy

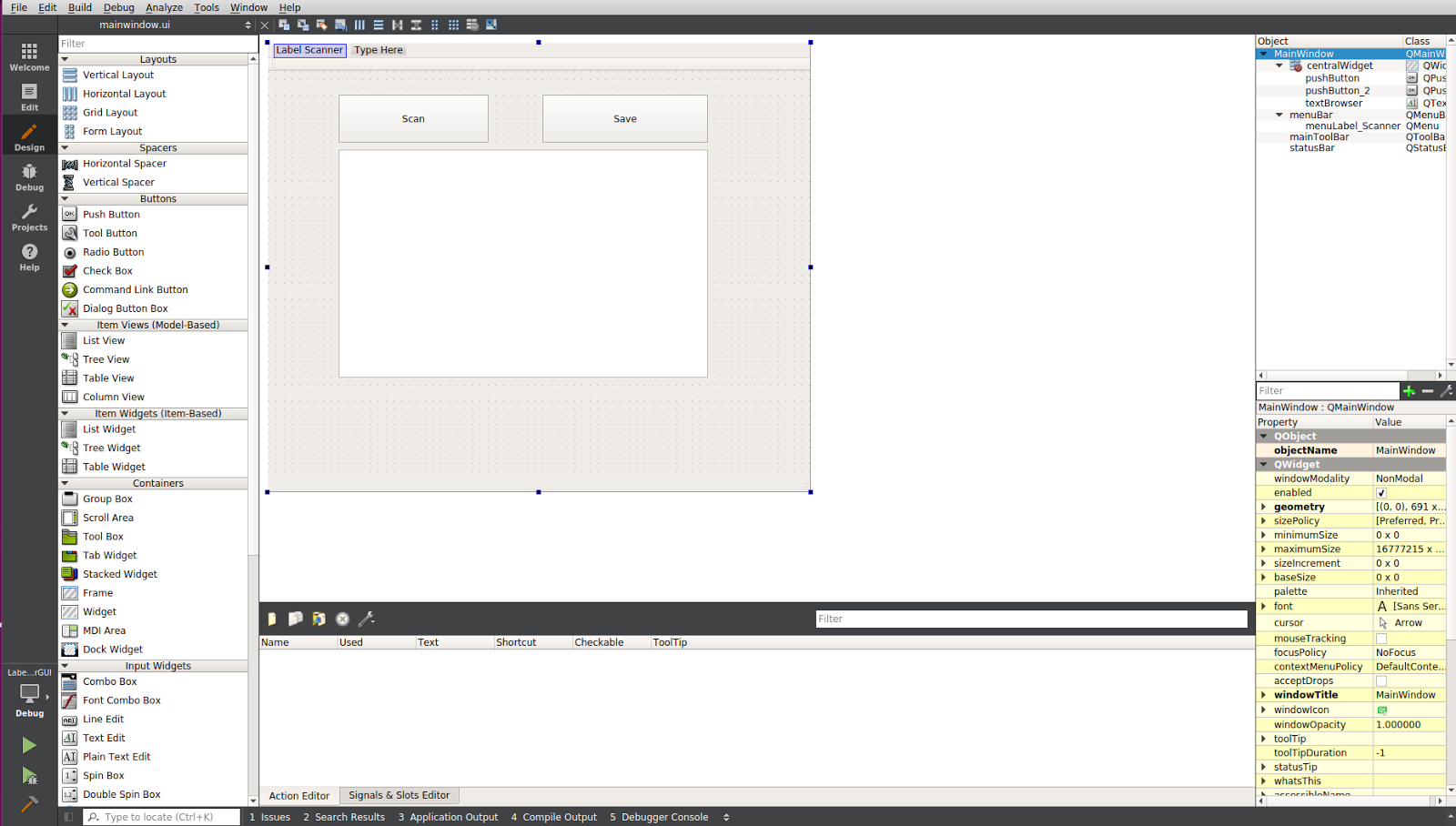
****

Figure 7 – Final Graphical User Interface

Tesseract is an optical character recognition engine for various operating systems. It is open-source software, released under the Apache License. The development has been sponsored by Google since 2006. It is considered one of the most accurate open-source OCR engines which is available.

Tesseract needs the leptonica library for image processing, the tesseract library, and a language package. Installing tesseract and leptonica library in Ubuntu 16.04 requires the following command: sudo apt install libtesseract-dev libleptonica-dev. Next we downloaded the data file for the desired language (English) from GitHub. We moved the language data file into the tessdata directory.

Next we wanted to run a tesseract API example to test the functionality of the program. The code is seen below.

Code:

#include <tesseract/baseapi.h>

#include <leptonica/allheaders.h>  
int main()  
{  
   char \*outText;  
   tesseract::TessBaseAPI \*api = new tesseract::TessBaseAPI();  
   // Initialize tesseract-ocr with English, without specifying tessdata path  
   if (api->Init(NULL, "eng")) {  
       fprintf(stderr, "Could not initialize tesseract.\n");  
       exit(1);  
   }  
   // Open input image with leptonica library  
   Pix \*image = pixRead("/usr/src/tesseract/testing/phototest.tif");  
   api->SetImage(image);  
   // Get OCR result  
   outText = api->GetUTF8Text();  
   printf("OCR output:\n%s", outText);  
   // Destroy used object and release memory  
   api->End();  
   delete [] outText;  
   pixDestroy(&image);  
   return 0;  
}

The above example uses C++ and link with tesseract and leptonica through linker files ( #include <tesseract/baseapi.h>, #include <leptonica/allheaders.h>). It takes in a phototest.tif image file with text image and translate the text into a text file in the same directory.

OpenCV is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform [17].

In order to install OpenCV we excetued the following commands.

$ sudo apt-get update

$ sudo apt-get upgrade

$ sudo apt-get install build-essential cmake pkg-config

$ sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-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 libatlas-base-dev gfortran

$ sudo apt-get install python2.7-dev python3.5-dev

Once OpenCV was installed we used the following code specific to our project.

opencv.h:

#ifndef OPENCV\_H  
#define OPENCV\_H  
void shootit();  
#endif // OPENCV\_H

opencv.cpp:

#include "opencv.h"  
#include <iostream>  
#include <exception>  
#include <stdexcept>  
#include "opencv2/imgproc/imgproc.hpp"  
#include "opencv2/highgui/highgui.hpp"  
using namespace std;  
using namespace cv;  
void shootit(){  
   VideoCapture stream(1); //use 0 if there is only 1 webcam  
   stream.set(CAP\_PROP\_EXPOSURE,2);  
  //stream.set(CAP\_PROP\_SHARPNESS ,20);  
   if(!stream.isOpened()){  
       cout<< "No camera :(\n";  
   }  
   Mat image;  
   stream.read(image);  
   imshow("pic",image);  
   imwrite("/home/comp/build-LabelScannerGUI-Desktop\_Qt\_5\_7\_0\_GCC\_64bit-Debug/label/img.png",image);  
}

The result from the integration of OpenCV and Tesseract into the project is seen below in Figure 8.

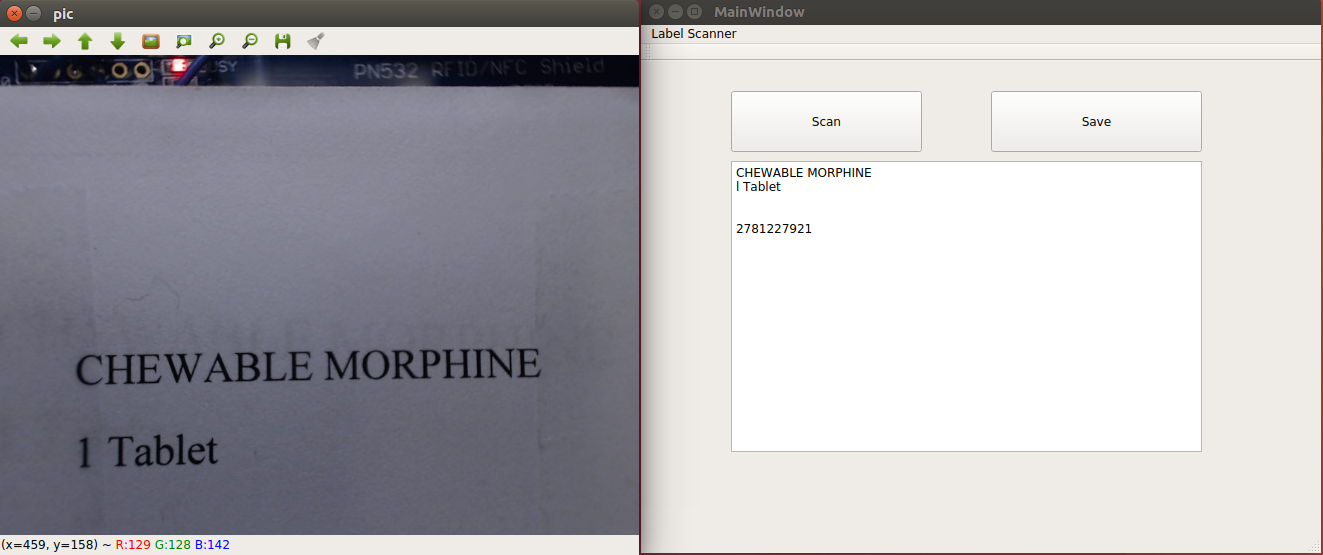


Figure 8 – Results of OpenCV and Tesseract working in unison.

**4.4 Data Acquisition**

Originally, our first approach to communicating with the Arduino RFID scanner was using a Microsoft library called “Winsock”. Winsock is an API which can be used in Visual C++ that handles sockets easily [11]. This solution was straight forward and our program could look for the Arduino to send data as a client, and process it the second it is received as a server. This was a very simple solution but these libraries only work on Windows. This meant when the project switched to a Linux operating system, they were no longer a valid solution.

When first moving to Linux (Ubuntu 16.10), we had to think of a new solution on how to import data from the Arduino. At first, it was thought that we could try to do what Winsock was doing using a similar library, Linux’s famous “Socket.h” [12]. This solution, while would have been equivalent, was much too complicated for the application’s needs. The Winsock API handled a lot of the backend of the process, that on Ubuntu would needed to be manually set up and required more knowledge on the subject. After a few tries and failing to find any similar applications of Socket.h, we decided to seek out a different simpler solution.

The Arduino community is very active when it comes to Python, and we decided to use this to our advantage. The python community itself is also very active, and this helped as well. The open source library provided by the RFID hardware, makes use of serial writing for its outputs, which we found can be handled with a python library called “Python-Serial” [13], as this is common practice for an Arduino application. Using this code in (Figure 9) we can open the serial port easily, and being in Python we can easily parse the data as needed.

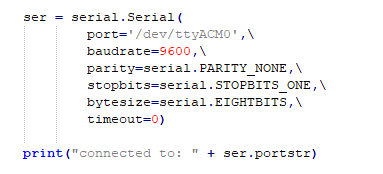


Figure 9 - Opening a serial connection in Python

The first thing we noticed with this approach is with our hardware, the open source libraries meant to be used by the manufacturer, sent out the data in a certain way. We had to remove some of the data sent as it was debugging information and we didn’t want to have to handle it in our Python script. After slightly modifying the code, we could move onto writing the script. Our script can read bytes until we reach the stop bit, and then return out the tag in decimal.

This script is flawed in two minor ways which were out of our control, but handling them is not much of an issue. At the time of development, we had not received any RFID tags, so we had to use the sample tag provided by Adafruit, the manufacturer of the RFID hardware. These sample tags are in hexadecimal, so the current script must account for hexadecimal numbers not decimal like ZIN mentioned. This is not much of a problem as we return decimals ultimately and Python’s integer data type can be told to be in any base, default being ten. The solution for this would be removing the base 16 modifiers on the string to integer

After creating a Python script, it had to be added to our main application in a way it can interact with everything smoothly and efficiently. Fortunately, our Linux distribution is Debian based and we could add the dependencies easily. The package used was the Python-dev package which includes the “Python.h” which we needed to embed the script in C++ [4]. These API functions can be used to give the program its own personal Python console and setting up everything needed to call the Python script, along with some basic error checking. Since the API calls the scripts in a certain way, we had to modify the script to be a function and have a return value. We add this code to a C++ function that then calls the API and runs the Python Script, which turns on the RFID and scans the tag to return a “Double” value and then we can read this using the API and then to return that value to the output of our program. This process can be seen in (Figure 10) parsing function. Figure 11 is a screen shot of the graphical user interface with a scan button that ultimately calls the C++ function that calls the Python script. The larger of the two flaws is when the RFID scanner is active but not reading, it does nothing at all. This means we cannot exit safely using serial timeout as then as soon as it timeouts will just try to read again. This puts us in a situation where we can either do an attempt count or handle this issue in the C++ application, in which we favor the second approach

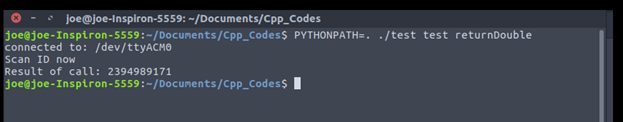


Figure 10 - Embedded Python ran in C++ program from Bash on Ubuntu 16.10

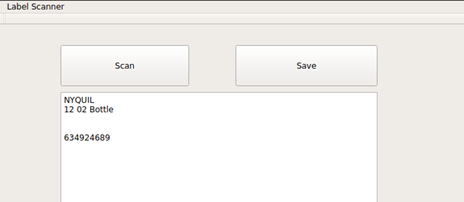


Figure 11 – The GUI that ultimately allows the user to call the C++ function and Python script that reside behind the scenes.

**4.5 Delimiting Data**

A C++ program was written to transfer the text from the output of the tesseract OCR to a tab delimited text file. The chosen language C++ was used for compatibility with the rest of the software.

The program is a console application that takes a defined .txt file from the tesseract OCR software containing the read label information, copies its data, and saves it to a tab delimited formatted .txt file. That text file then is viewable in excel.

**5. Obstacles Overcome**

**5.1 RFID**

**5.1.1 PN532 NFC Controller**

On the PN532 board there is a labeling mistake on the silk screen. The selector points for choosing between UART and SPI are inappropriately labeled. This initially created problems for us because we were attempting to use SPI communication but the hardware was no properly configured.

**5.1.1.1 PN532 NFC Controller Solution**

After reviewing documentation, we noticed that the labeling was swapped so we simply bridged the two pads labeled UART for SPI to be enabled.

**5.2 OCR and OpenCV**

**5.2.1 Problem: Open-Source Linux Operation**

The open-source OCR software we selected was designed to run on an open-source operating system. We attempted to make the software run on a Windows platform but were unsuccessful.

**5.2.1.1 Solution: Open-Source Linux**

To remedy this problem, we decided to change to a Linux based system. The OCR program was intended to run on Linux so it immediately remedied the problems we created by attempting to force the program to run on Windows. Additional, because Linux is open-source we have full control over the operating system. Linux is also free. Some of the programs we wanted to use on Windows were restricted unless a complete version was purchased.

**5.2.2 Problem: Multiple Version of OCR Software**

The OCR software was poorly organized. There are multiple versions of the software online. It was unclear exactly which version was the most ideal and best suited for our needs.

**5.2.2.1 Solution: Trial and Error Solution**

Initially the only solution to this problem was trial and error. We compiled different versions of the program and experimented with their functionality. Every time we experimented with a new version of the software we gleaned information from its operation.

**5.2.2.2 Solution: A La Carte Software**

As we experimented with different versions of the OCR software we found desirable elements in various versions. We took pieces of functionality and created our own final program that best fit our needs.

**5.2.3 Problem: Limited and Poorly Detailed Documentation**

The documentation for the OCR code was poor. It was unclear how the code works on a line-by-line basis. It was also unclear what changes needed to be made to the code for it to work with our specific circumstances.

**5.2.3.1 Solution: Debug Software for Functionality**

To solve this problem, we debugged the code on a line-by-line basis. We looked at the output and data types at various points throughout the program that helped us to understand how the program was working. Once we determined how the functionality of the program worked, we could make intelligent changes to the code.

**5.2.4 Problem: Installing Linux on Laptop with SSD**

As stated previously, we made the switch to a Linux based system as opposed to Windows. At this point we needed to install Linux on the central laptop that the OICS runs on. Typically installing Linux on a personal computer is a trivial task. Our laptop has a solid-state drive as opposed to a more tradition hard disk drive. The solid-state drive is connected to the mother board through a PCI Express port. This created issues with Linux recognizing the solid-state drive during the installation processes. Additionally, the boot loader was failing.

**5.2.4.1 Solution: Linux Installation on Laptop with SSD**

We remedied this by disabling secure boot. Then we entered the BIOS and disabled UEFI. We then used the Legacy mode to install. This allowed Linux to recognize the SSD. However, this created a new problem because the boot loader failed during the installation process. This was due to UEFI being disabled.

To remedy the boot loader problem, we again entered the BIOS. We disabled fast boot and disabled CSM compatibility. We then selected override boot options. We set the bootable USB as the default boot loader. This allowed Linux to both recognize the SSD and successfully utilize the boot loader. From this point forward the installation process went smoothly.

**5.3 Data Acquisition**

**5.3.1 Initial Use of Winsock**

Winsock is an API that supports the input/output requests for socket data. It was ideal for handling serial data output from the Arduino.

**5.3.1.1 Solution**

**5.4 Delimiting Data**

**5.4.1 Non-uniform Data Format**

Data is not uniform in format in terms of how they are read from tesseract.

**5.4.1.1 Data Format Solution**

Data is transferred line by line. When necessary, labels of longer length will populate multiple lines.

**6. Budget**

To best display the details of our budget we have put the data into Table-1 below. The clear majority of our costs went to the business laptop that runs the OICS. The cost and hardware capabilities of this laptop are excessive for this application. However, there was a limited number of PC’s available for purchase that were approved by the Information Services and Technology department as CSU. For this reason, we were obligated to purchase this model.

|  |  |  |
| --- | --- | --- |
| Items | Vendor | Price |
| Dell Precision Mobile 7510 | CDW-G | $1648.00 |
| Logitech HD Pro C920 Webcam | Amazon | $66.94 |
| Arduino Uno R3 | Amazon | $23.75 |
| Adafruit PN532 NFC/RFID Shield | Amazon | $49.65 |
| Optically Clear Cast Acrylic Sheet 1-1/2" Thick, 12" x 12” | McMaster-Carr | $86.00 |
| Camera Mount | Amazon | $14.40 |
| Delrin Precision Shim Bushing | McMaster-Carr | $40.56 |
| 24” Zinc Frame Rod | Grainger | $35.32 |
| Clamp Rod End Connector | Grainger | $26.41 |
| Machining Services | CSU | $0.00 |
| Total |  | **$2015.94** |

**Table -1**

**7. Future Considerations**

**7.1 Multiple Scan Error Detection**

To increase the accuracy of the system, additional checks could be implemented to ensure accurate data. If more time were to allow, we would implement a multiple scan system.

When the user initiates a scan, the system would automatically execute two photos and two RFID scans. The results from the two scans would be compared. If the data from the two scans match, then we can have increased confidence that the data is accurate. If the data from the two scans does not match, then the data from both scans will be thrown out. The system would execute another two scans and repeat the process until matching data is detected. At this point the data would be delimited and output. The user would be notified that as successful scan had taken place.

This same notion could be extended to any K-number of scans. If the notion was expanded to require 3 or 4 matching scans it would result in a higher degree of accuracy. However, this would be at the cost of average time per successful scan.

**7.2 Improved Lighting**

To improve the quality of the image taken by the webcam we would have liked to add a LED light source. The LED’s would be illuminated when a scan is initiated and stay on until the system indicates a successful scan was completed. This would improve the quality of the photo, and potentially improve the accuracy of the OCR.

**7.3 Custom Lexicon**

The labels that the OCR software reads are generally similar in layout and content. We would like to have added a library of common medical consumables names. Those names could be cross referenced with the results from the character extraction in the post-processing stage. In the event a name is detected that is very close to a name deemed common in the lexicon, it could be corrected.

**8. References**

**8.1 NASA Label Standards**

[1] "Pressurized Payloads Interface Requirements Document". *International Space Station Program* R.15 (2015): n. pag. Print.

[2] "NASA Aviation Medical Certification Standards". *Office of the Chief Health and Medical Officer* (2009): n. pag. Print.

**8.2 IEEE SPI Standards**

[3] Tianxiang Liu and Yunfeng Wang, "IP design of universal multiple devices SPI interface," *2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification*, Xiamen, 2011, pp. 169-172.

**8.3 RFID NFC**

[4] Near Field Communication - Interface and Protocol (NFCIP-1)". *ECMA International* 3 (2013): n. pag. Print.

[5] ISO/IEC 18092:2013". *International Standards Organization* 2 (2013): 0-44. Print.

**8.4 Optical Character Recognition References**

[6] Smith, Ray. "Motivation And History Of The Tesseract OCR Engine". (2016): n. pag. Print.

[7] Smith, Ray. "Architecture and Data Structures” (2016): n. pag. Print.

[8] Smith, Ray. "Features and Character Classifier". (2016): n. pag. Print.

[9] Smith, Ray. "Character Segmentation, Language Models and Beam Search". (2016): n. pag. Print.

[10] Smith, Ray. "Page Layout Analysis". (2016): n. pag. Print.

**8.5 Miscellaneous References**

[11] Windows Sockets 2 (Windows)". *Msdn.microsoft.com*. N.p., 2017. Web. 29 Apr. 2017.

[12] Ubuntu Manpage: Sys/Socket.H - Main Sockets Header". *Manpages.ubuntu.com*. N.p., 2017. Web. 29 Apr. 2017.

[13] Pyserial 3.3 : Python Package Index". *Pypi.python.org*. N.p., 2017. Web. 29 Apr. 2017.

[14] 5. Embedding Python In Another Application — Python 2.7.13 Documentation". *Docs.python.org*. N.p., 2017. Web. 29 Apr. 2017.

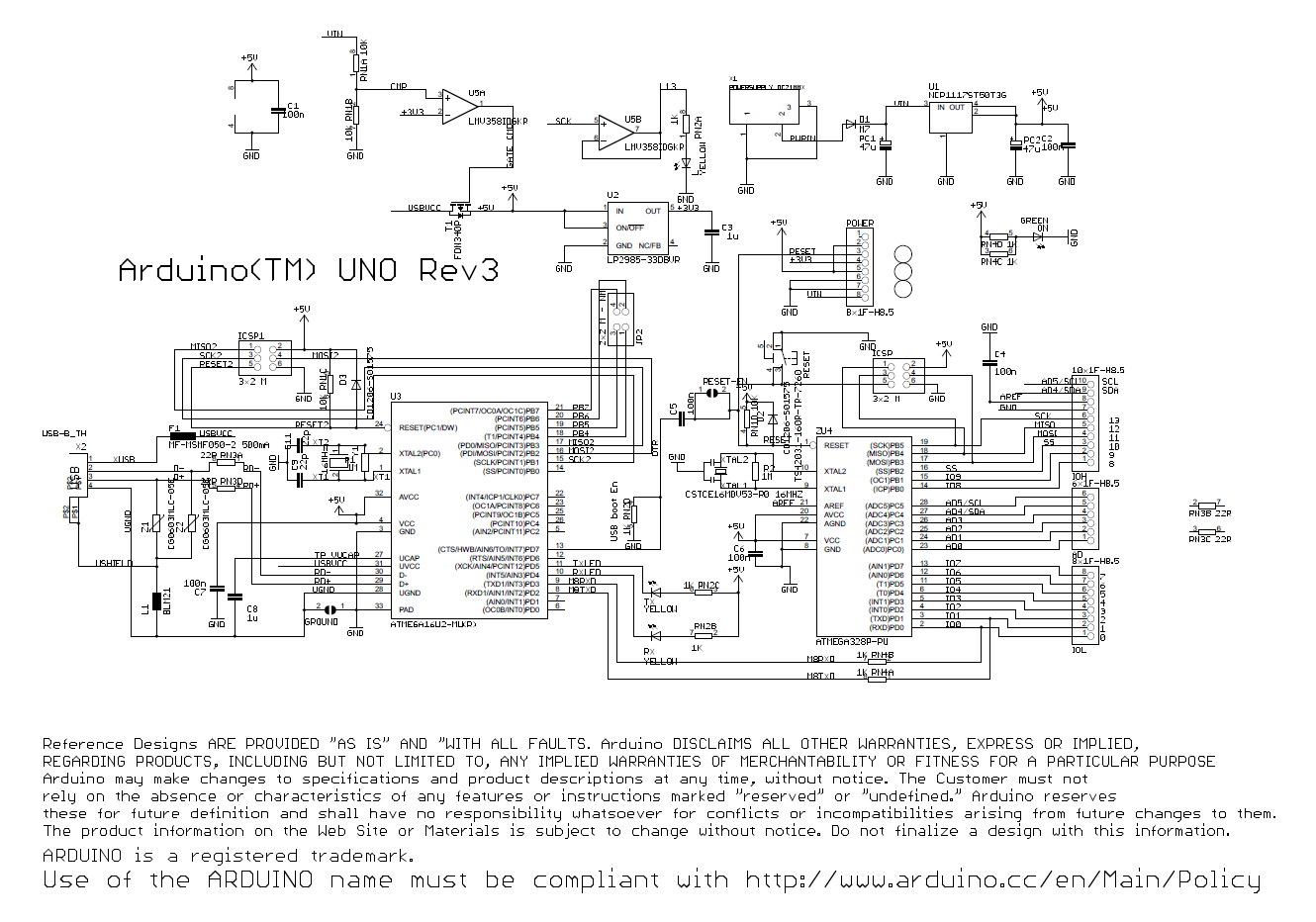
[15] "OUR COMPANY - Zin Technologies". *Zin Technologies*. N.p., 2017. Web. 29 Apr. 2017.

[16] Downloads | Adafruit PN532 RFID/NFC Breakout And Shield | Adafruit Learning System". *Learn.adafruit.com*. N.p., 2017. Web. 29 Apr. 2017.

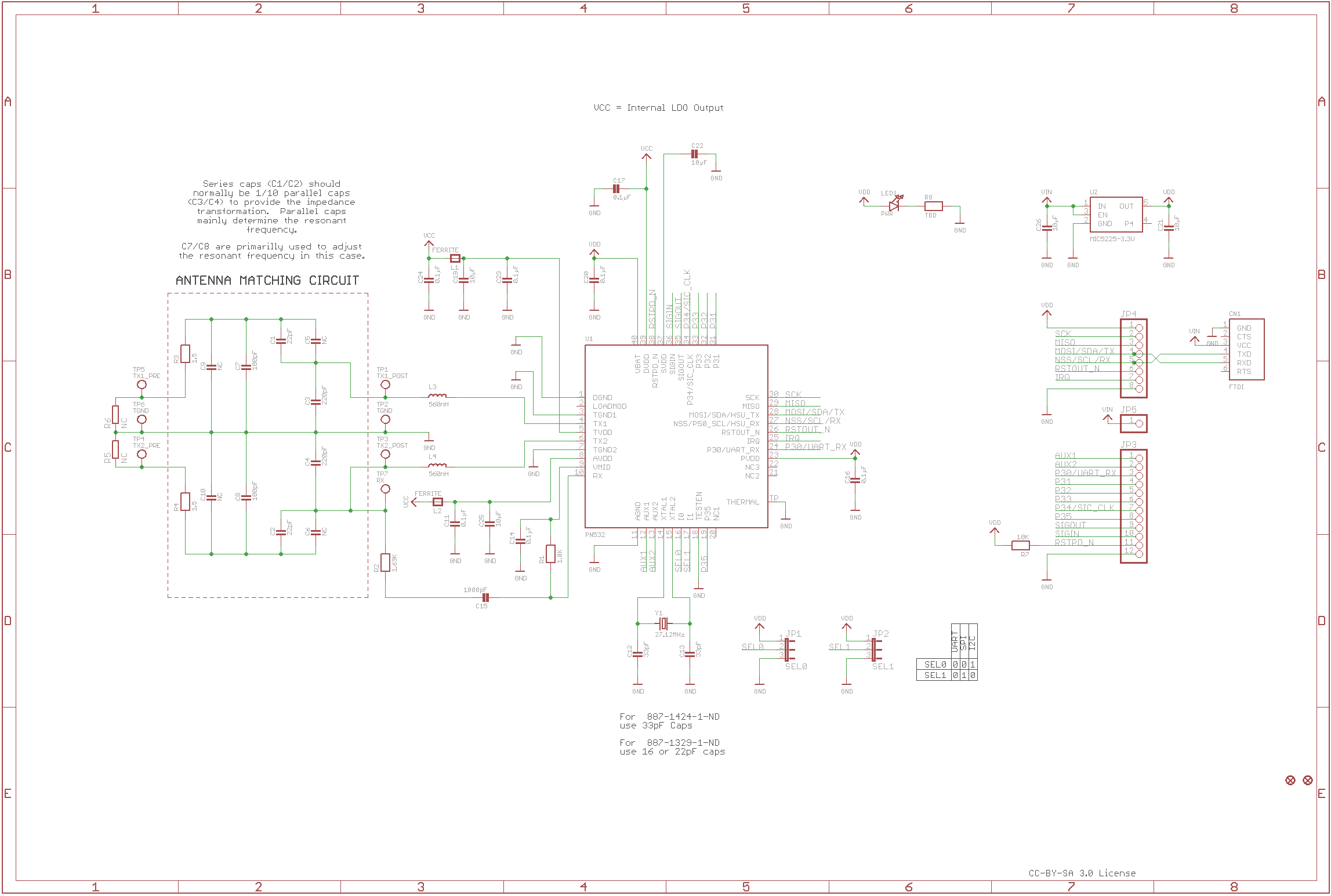
[17] "Opencv Library". *Opencv.org*. N.p., 2017. Web. 7 May 2017.

**9. Appendix**

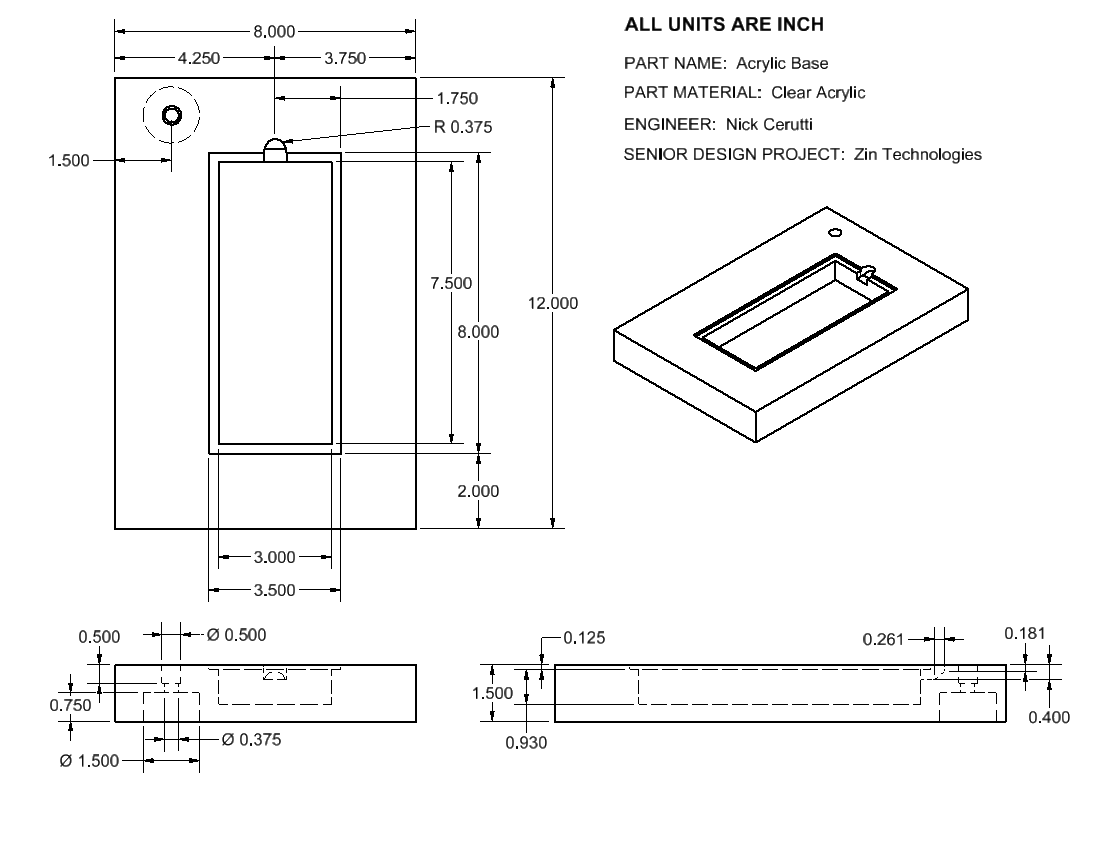
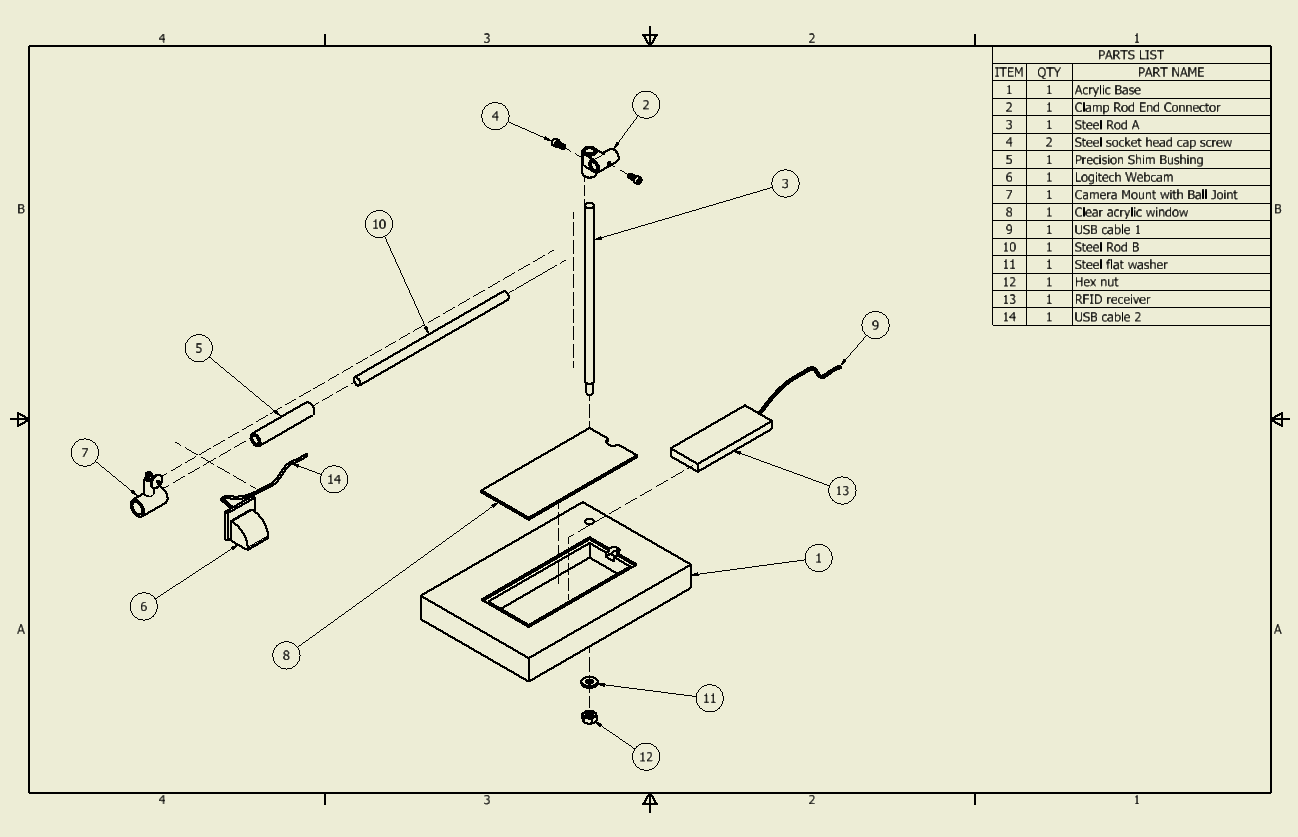
**9.1 Arduino Uno Schematic**



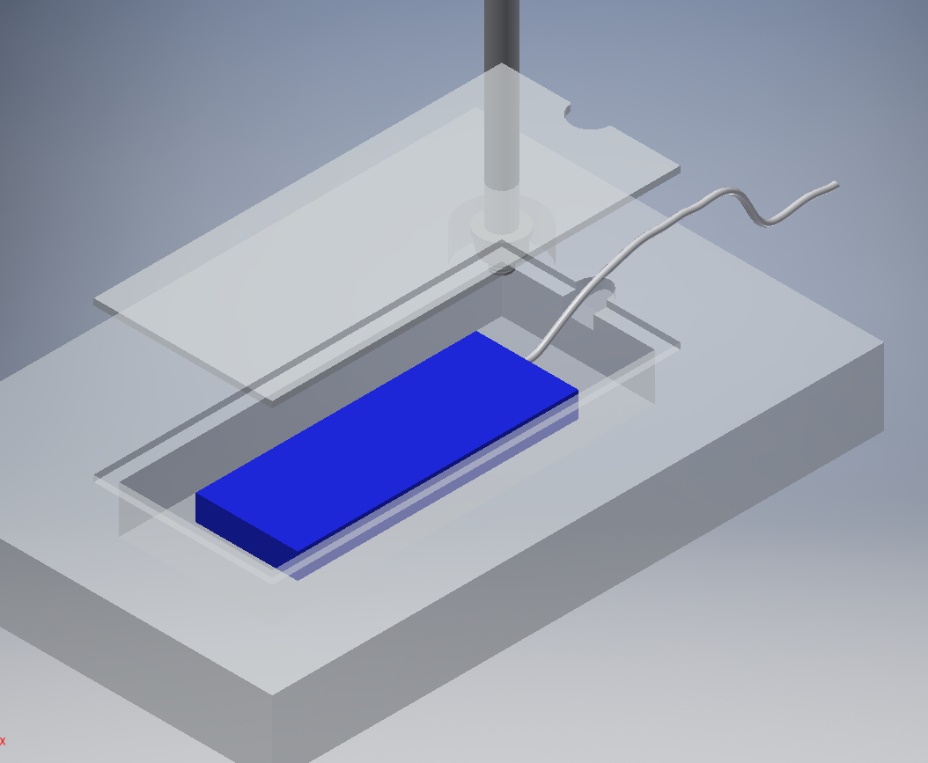
**9.2 Adafruit PN532 NFC/RFID Controller Schematic**

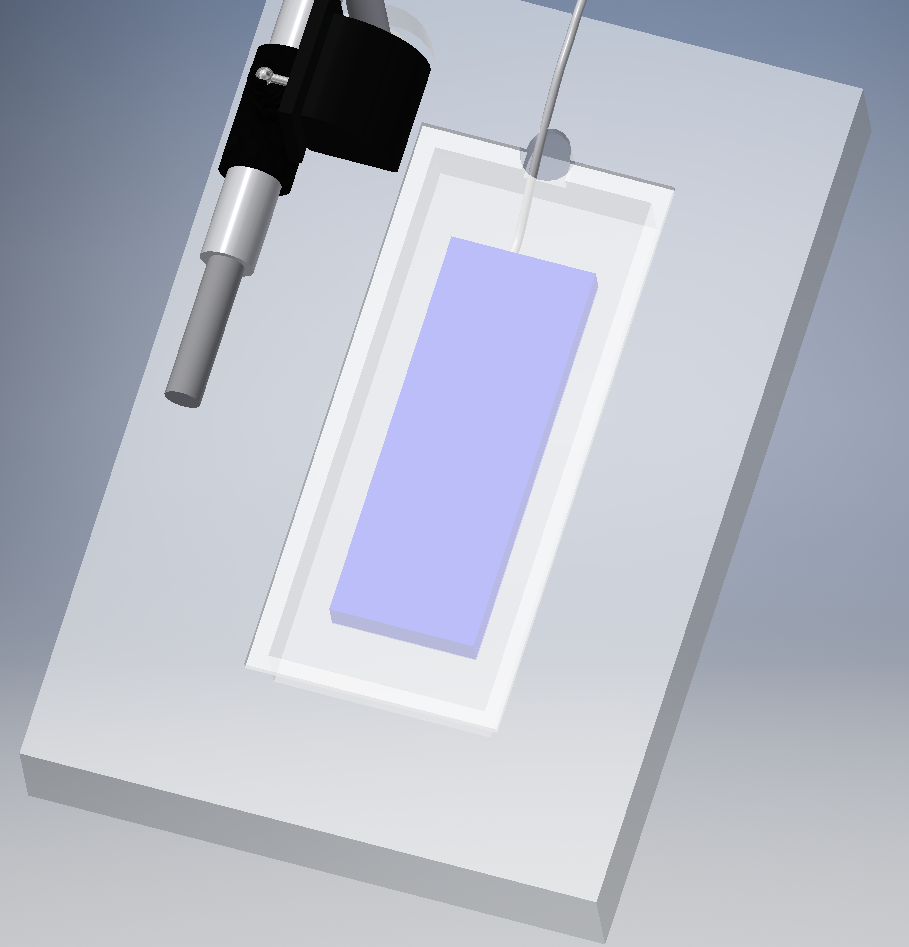


**9.3 Physical Design Drawings and Specifications**



**9.4 Miscellaneous Physical Design CAD Drawings**





**9.5 GUI Xml Code**

Mainwindow.ui:  
  
<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0">  
<class>MainWindow</class>  
<widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
  <rect>  
   <x>0</x>  
   <y>0</y>  
   <width>691</width>  
   <height>572</height>  
  </rect>  
 </property>  
 <property name="windowTitle">  
  <string>MainWindow</string>  
 </property>  
 <widget class="QWidget" name="centralWidget">  
  <widget class="QPushButton" name="pushButton">  
   <property name="geometry">  
    <rect>  
     <x>90</x>  
     <y>30</y>  
     <width>191</width>  
     <height>61</height>  
    </rect>  
   </property>  
   <property name="text">  
    <string>Scan</string>  
   </property>  
  </widget>  
  <widget class="QTextBrowser" name="textBrowser">  
   <property name="geometry">  
    <rect>  
     <x>90</x>  
     <y>100</y>  
     <width>471</width>  
     <height>291</height>  
    </rect>  
   </property>  
  </widget>  
  <widget class="QPushButton" name="pushButton\_2">  
   <property name="geometry">  
    <rect>  
     <x>350</x>  
     <y>30</y>  
     <width>211</width>  
     <height>61</height>  
    </rect>  
   </property>  
   <property name="text">  
    <string>Save</string>  
   </property>  
  </widget>  
 </widget>  
 <widget class="QMenuBar" name="menuBar">  
  <property name="geometry">  
   <rect>  
    <x>0</x>  
    <y>0</y>  
    <width>691</width>  
    <height>19</height>  
   </rect>  
  </property>  
  <widget class="QMenu" name="menuLabel\_Scanner">  
   <property name="title">  
    <string>Label Scanner</string>  
   </property>  
  </widget>  
  <addaction name="menuLabel\_Scanner"/>  
 </widget>  
 <widget class="QToolBar" name="mainToolBar">  
  <attribute name="toolBarArea">  
   <enum>TopToolBarArea</enum>  
  </attribute>  
  <attribute name="toolBarBreak">  
   <bool>false</bool>  
  </attribute>  
 </widget>  
 <widget class="QStatusBar" name="statusBar"/>  
</widget>  
<layoutdefault spacing="6" margin="11"/>  
<resources/>  
<connections/>  
</ui>

**10. Qualifications**

**10.1 Resumes**