Secret Messaging System
Design Document

Team Stegosaurus
Caroline Hermans, Snigdha Sharma, Shivani Sheth
October 14, 2018

Table of Contents

Project Description

For our capstone project, we will create a secret messaging system which employs a visually pleasing way to deliver encoded text based messages. Our system will take an initial input message, run an encoding algorithm on it, and print out an encoded image. When the recipient scans the image, they will get back the original message. The goal of this project is to create a more aesthetic encoding mechanism than the QR code while keeping most of the efficiency that the QR code has to offer. Our final image should be able to go undetected by the plain eye as something that has data encoded in it.

Design Requirements

The general design requirements for the project are to create an encrypted, visually pleasing, and undetectable data encoding system. We plan to create both an encoding station and a decoding station that allow participants to type a message, receive a printed encoded image, and then scan that image back at the other station. In order to do this, we need an integrated Raspberry Pi encoding station, which runs our user-facing hardware as well as handling the message encoding and printing. We also need a second Raspberry Pi station that handles the decoding, allowing people to a photo of the encoded message. The decoding station will then decode the image, and display the message back.

For our encoding algorithm, one of our requirements is that the encoding looks beautiful and unrecognizable as data. We plan to use different colors, shapes, and sizes in order to encode our information. We plan for our pattern to be linear, geometric, and colorful. Each letter in the alphabet will have a series of shapes in different colors and sizes, and the order of the shapes and colors will encode the different letters. Because the obscurity of the pattern is important to us, in order to test this, we are going to show a variety of people five different images and have them guess which two have encoded

data. We hope that the encoded data will be identified randomly and unreliably, with a probability of 1 in 5 or 20%.

Additionally, we require that our encoding station is simple to use. The person interacting with the system should not have to do anything more than type their message in and press a button. After they do so, their image should print out automatically. Then, they will be able to take their encoded image and decode it.

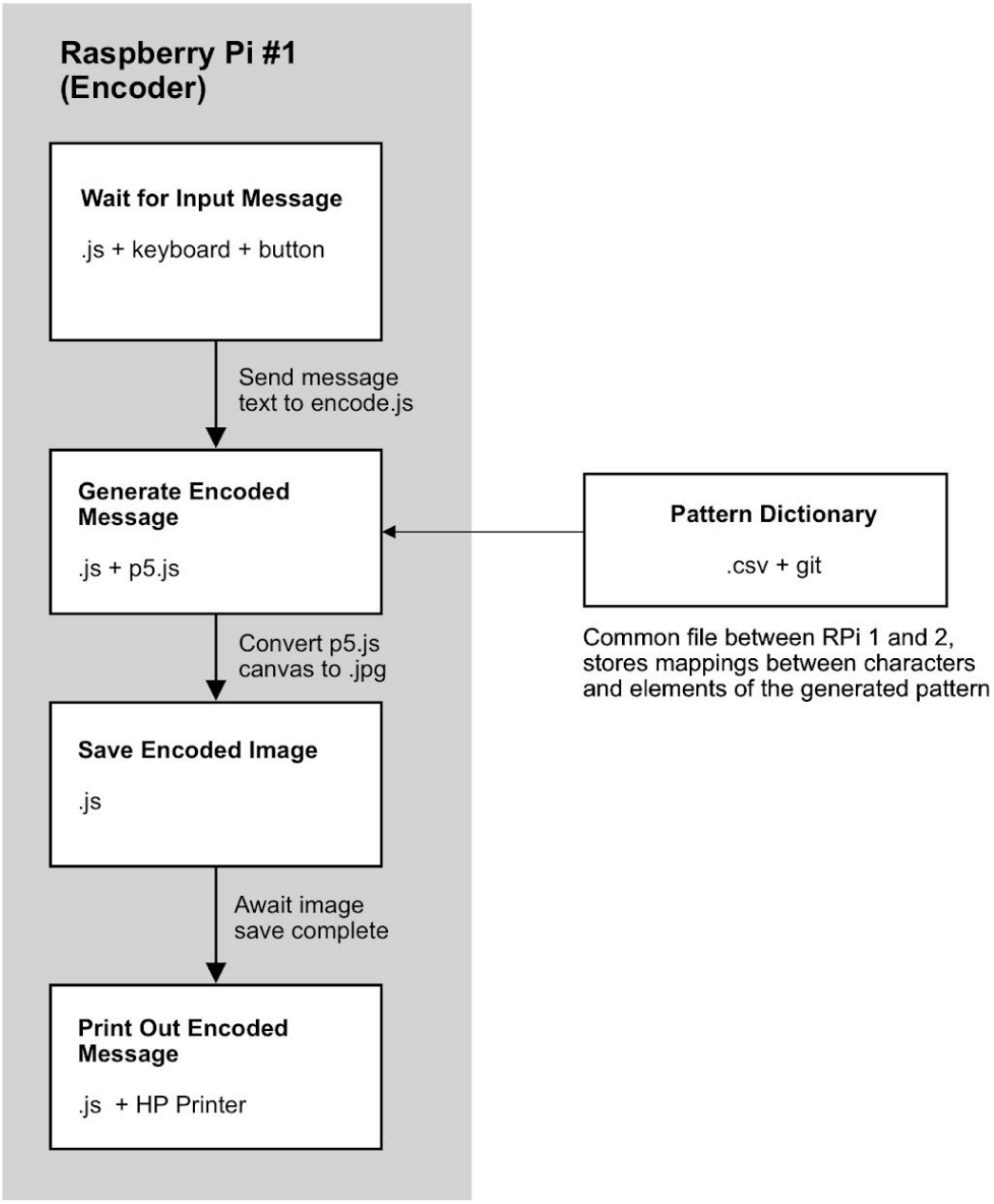
As for our decoding system, we want a simple interaction for receiving the original message back as well. The people using our system should be able to press a single button, have the Raspberry Pi take a photo, and then see their original message displayed on the Raspberry Pi LCD screen. Everything else that is involved should happen automatically and internally within the Raspberry Pi.

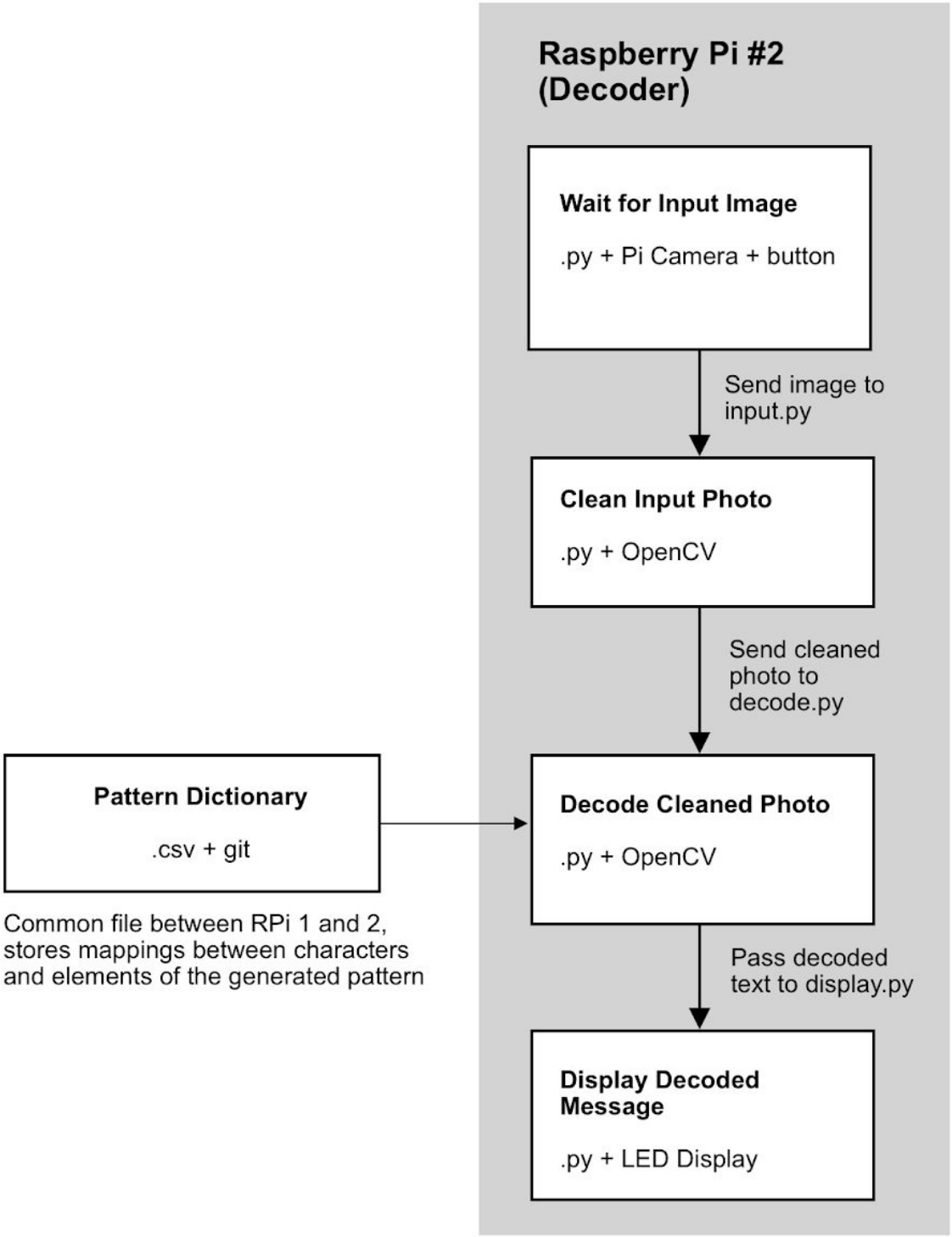
Our decoding CV code needs to be able to recognize all of the data that we are encoding. Our decoding system needs to be able to look at a series of shapes in different colors and sizes, and list out the data that it sees. In doing so, the decoding system will recognize color, shape, size, and whether the shape is filled or unfilled, for each shape in the encoded image. After we use CV to detect the various shape data in the image, we need to reference the original dictionary to figure out which characters were encoded in the image. Our decoding system also needs to be able to turn a photo from a Raspberry Pi camera into a clean image for decoding.

We want our capstone to be practical to use in terms of the time it takes, both to generate the images and to receive the decoded message back. Our project will to process the text into an image in less than a second, and we aim for it to process the picture back into text in less than three seconds. This is similar to the time a QR code takes to scan and load the link. We also want users to be able to input up to 32 characters in our messaging system, which is the average length of a short sentence. In order to make it practical for real life use, our system needs to be reliable. We plan on ensuring 100% accuracy in decoding the image to the original text. We will do this by adding redundancy for each character and also repeating the pattern multiple times. We also developed unique representations for each character so it is easier to decode.

Functional Architecture

Our system involves two Raspberry Pi computers, one responsible for encoding the message and printing the image, and one responsible for decoding the printed image.




Raspberry Pi #2
(Decoder)
Wait for Input Image
.py + Pi Camera + button
Send image to
input.py
Clean Input Photo
.py + OpenCV
Send cleaned
photo to
decode.py
Pattern Dictionary
.csv + git
Decode Cleaned Photo
.py + OpenCV
Pass decoded
text to display.py
Display Decoded
Message
.py + LED Display
Common file between RPi 1 and 2,
stores mappings between characters
and elements of the generated pattern

Design Trade Studies

While designing our capstone project, we discussed many different ideas about the design of our encoding system, the interaction between our hardware, and what we want to prioritize in our project.

During our brainstorming sessions for the encoding algorithm, we initially came up with a radial pattern, much like a mandala. We liked that it was aesthetically pleasing and was a unique shape unlike the square, pixelated QR code and the linear barcode. However, we rejected this design after experimenting with basic OpenCV and p5.js as generating a radial pattern was exponentially more difficult than with a linear pattern because of size changes and lots of trigonometry. We decided that we would rather spend more time working on adding additional features to our pattern, such as aesthetic noise and encryption, than having a simpler radial pattern.

Another area where we had to make a difficult design decision was the selection of the scanning and decoding mechanism. Initially, we planned on using an iPhone app to scan the printed image, detect the shapes and patterns, and then decode it to retrieve the input text. This involved integrating OpenCV with iOS, which proved to be more complicated than we had anticipated. Using OpenCV with Python on iOS turns out to be extremely challenging, which is what we had been working on for the past month. However, we thought that the portability aspect of using a smartphone to do the decoding was an exciting feature of our project, and wanted to try to make it work. After looking into the best way to do this, we realized this would involve learning Objective-C and rewriting all of our existing OpenCV detection in Objective-C. We decided this was not feasible, especially given our timeline, and decided to run the detection and decoding on a second Raspberry Pi. The Raspberry Pi does, however, give us other UI alternatives, such as an LCD text display to display the decoded message. We made this decision because porting to Objective-C would take time that we hadn't accounted for, and would not add any additional critical functionality to our project.

System Description/Depiction

Raspberry Pi #1 (Encoder)

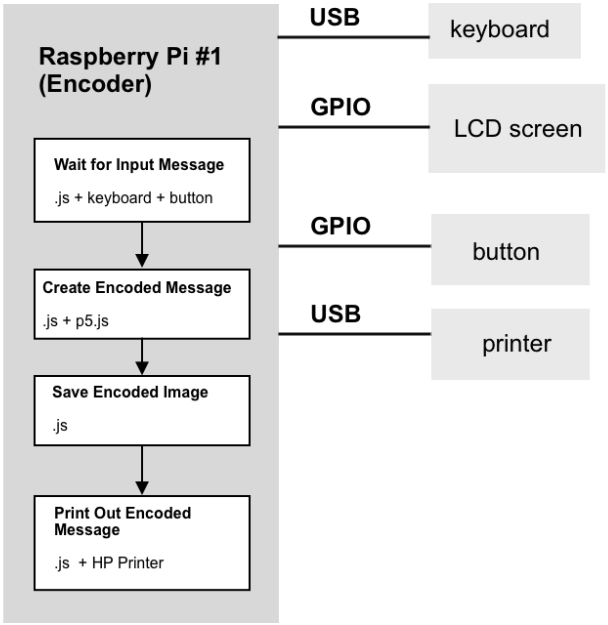


Figure 1. Raspberry Pi #1 System

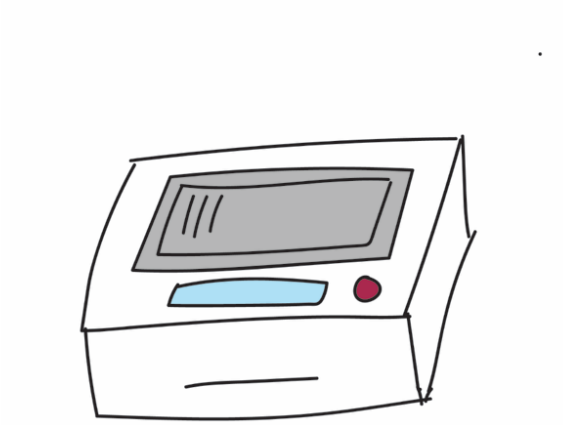


Figure 2. Encoding Station Concept

Our overall encoding system, as seen in Figure 2, will be controlled by a Raspberry Pi. Attached will be a Keyboard (USB), a button (GPIO), a printer (USB), and an LCD screen (GPIO), as seen in Figure 1. When users type onto the keyboard, their message will display on the screen via the GPIO pins. When they hit the button, the message on the screen will print out via USB printer.

Buttons

We are using EG Starts 5V 24mm LED buttons. The Raspberry Pi will listen for the button press on the GPIO pins, and when the button is pressed, it will send the typed message to our encoding block.

Keyboard

The keyboard will be attached to the Raspberry Pi via USB. We are using the SR wired 78 key mini keyboard. The Raspberry Pi is always listening to the key presses in our initial input Javascript file, and it displays the information on the LCD screen.

LCD Screen

We are using the SunFounder IIC I2C TWI 1602 Serial LCD Module Display, shown in Figure 3. It will be controlled via Raspberry Pi GPIO pins. It has 32 characters, perfect for our encoding.

Figure 3: LCD Screen for Raspberry Pi 1

Printer

The printer will be attached to the Raspberry Pi via USB. We are using the HP DeskJet1112 Compact Printer. When the encoded message image is complete, the Raspberry Pi will call the printer and have it execute our specific print commands via Javascript.

Generating Encoded Image

The encoded image is generated using p5.js. Once the user input is received, the function will map the input to a sequence of shapes that represent the characters. These shapes are then generated on the p5.js canvas and are repeated at least 4 times for redundancy and to correct for any lighting variation when scanning it back. For example, Figure 4 says “Stegosaurus” and has each letter matched to a sequence for six shapes, repeated horizontally eight times. The letter U for example, is mapped to the sequence of "red star, blue rectangle, blue

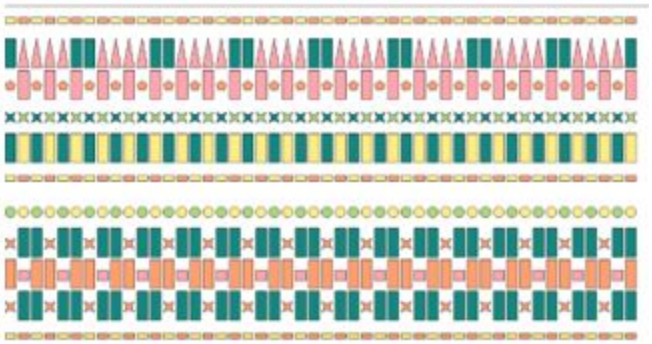


Figure 4: Encoded Image with text “Stegosaurus”

rectangle, red star, blue rectangle, blue rectangle". Another factor added to our encoding is the concept of a red herring and noise. For example, our current model ignores any figure that's pink. This allows us to add noise to our image that provides aesthetic value without interfering with the ability to encoding text.

Saving & Printing Encoded Image

The encoded message will be displayed on a p5.js canvas, which we will then save to a .jpg image on the Raspberry Pi. We will include a callback function to the printer that will be executed after the image is completed saving, done in Javascript.

Raspberry Pi #2 (Decoder)

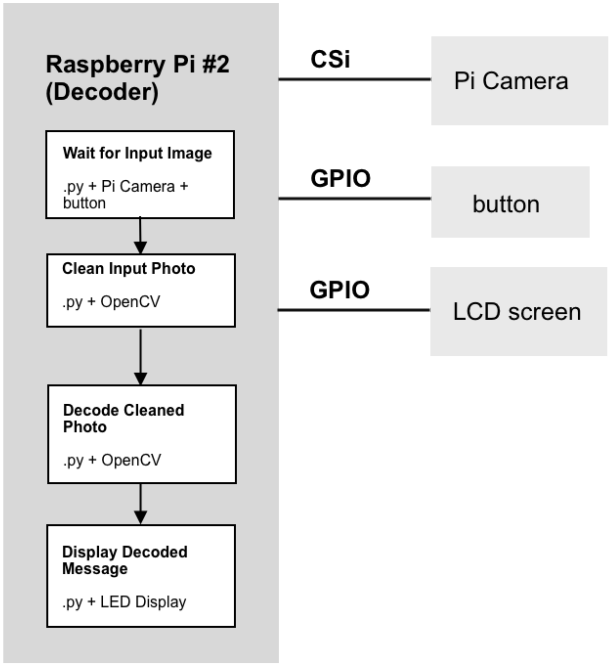


Figure 5. Raspberry Pi #2 System

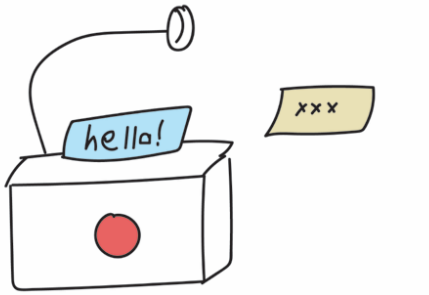


Figure 6. Decoding Station Concept

Our overall decoding system, seen in Figure 6, will be controlled by a Raspberry Pi. Attached will be a Pi Camera (CSi), an LCD screen (GPIO), and a button (GPIO), as seen in Figure 5. When users hit the button, the Pi camera takes a photo. We then

processes the photo and decode it in Python with OpenCV. The button and the LCD screens are the same kinds we are using for Raspberry Pi #1.

Camera

We are using the Raspberry Pi Camera Module V2-8 Megapixel,1080p camera. The camera communicates with the Raspberry Pi via the CSi interface. We can trigger the camera via Python to take a photo, and then send the photo to our decoding blocks.

Scanning and Preprocessing Image

We will scan the image using OpenCV in Python. We will detect the edges of the paper and then use those edges to find the outline representing the picture being scanning. After this, we will apply a perspective transform to get a top-down view of our pattern.

Decoding Image

Once the image is processing, we use OpenCV in python to do our pattern detection. To detect the edges and shapes, we are using the Ramer-Douglas algorithm to do contour approximation and then using hue and RGB values to detect the color. To detect if a shape is filled or unfilled, we convolve it with a gaussian filter and classify the output. Finally, all of these attributes are combined into a csv file and sent over to the decoding algorithm. This parses the csv file, decrypts the data, and then matches the pattern with the dictionary of our encoding algorithm and prints out the original text.

Project Management

Schedule

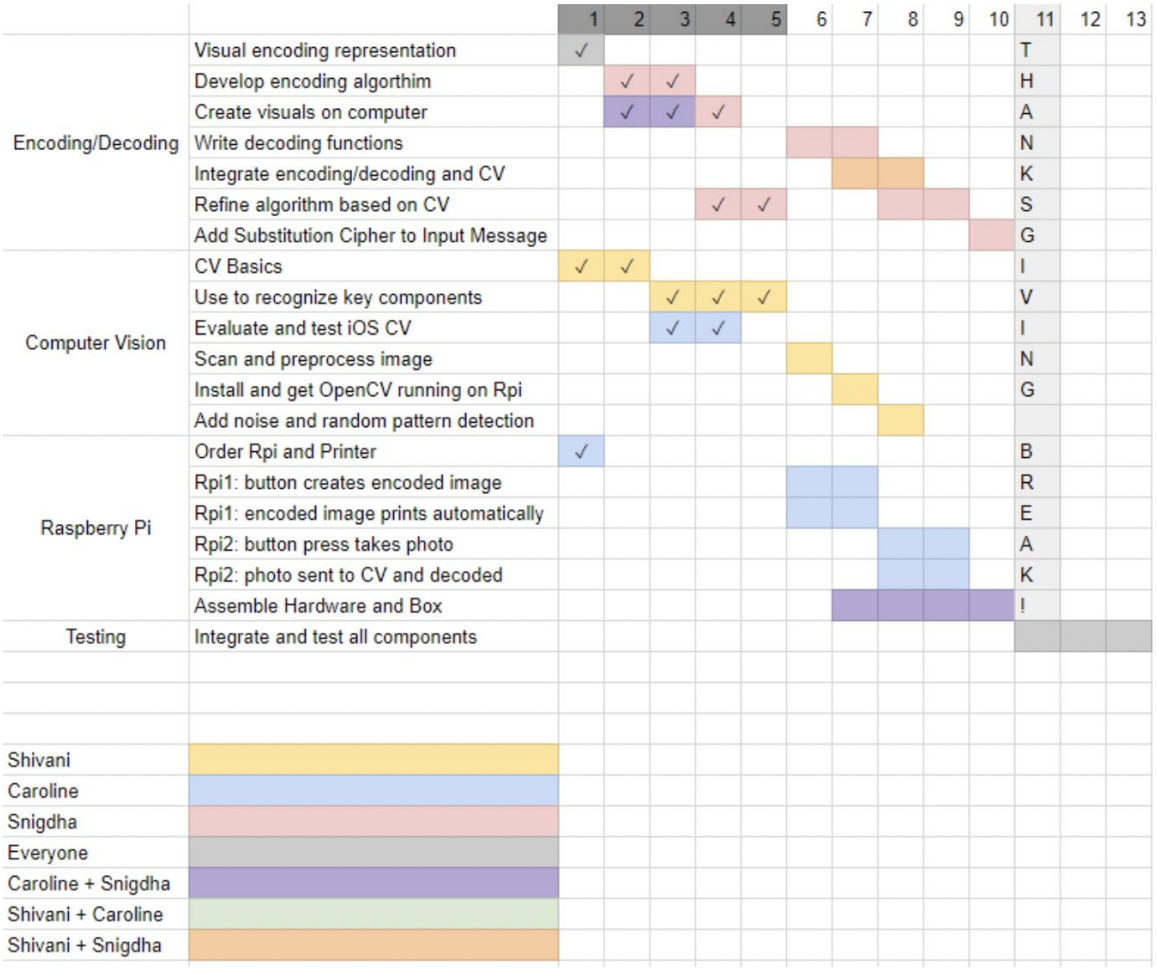


Figure 7. Gantt Chart of project timeline

Team Member Responsibilities

Shivani is responsible for the pattern detection that is part of the decoding step. She will use OpenCV to recognize key components to the project, including but not limited to shapes, filled/unfilled blocks, color, noise, and ordering. In addition, she will be responsible for scanning and preprocessing the image before the CV processing. This includes controlling for various orientations and lighting conditions. Finally, she will port the OpenCV to the second Raspberry Pi and do user testing for the scanning.

Snigdha is working on the encoding and decoding algorithm for this project. This includes working with the rest of the team to brainstorm ideas for the visual representation of the input message, creating the initial encoding algorithms, and generating the pattern using p5.js. This is an iterative process as the encoding patterns will be continually revised and refined based on the CV processing. The second part of the project is the decoding, which will begin once we have a CV algorithm that can properly detect and classify shapes and their individual features. Once the encoding and decoding are complete, Snigdha will work on making the visual representation further unrecognizable and also on adding some level of obscurity to the encoded input itself using substitution ciphers and randomization.

Caroline is responsible for the hardware and integration. She will be setting up the Raspberry Pis with the required buttons, keyboards, sensors, and hardware. Her job is to make sure that Shivani's OpenCV code and Snigdha's encoding algorithm both run according to the user inputs and triggers that we have specified. This includes working with GPIO, saving images, setting up the printer, and integrating the overall system. It also includes working with the camera, and building boxes and environments for the Raspberry Pis to work in. She has also done some work on exploring the image generation, as well as evaluating alternatives to our Raspberry Pi scanning system such as iOS.

Parts List & Budget

Item	Count	Amount Budgeted
Raspberry Pi 3 Canakit	2	$149.98
HP DeskJet 1112 Compact Printer	1	$29.89
SunFounder IIC I2C TWI 1602 Serial LCD Module Display	2	$19.98
SR Mini Keyboard Wired 78 Key Keyboard	1	$12.99
EG STARTS 5 Piece 24mm Full Color LED Illuminated Push button Built-in Switch 5V Buttons	1	$9.99
Raspberry Pi Camera Module V2-8 Megapixel,1080p	1	$25.00
Acrylic for laser cutting	2	$50
Paper for printing	1	$20
HP Printer ink color cartridges	2	$80

Total Cost: $397.75

Risk Management

While we have designed our project and resources to mitigate risks, we still have a few unknowns and risk factors. One of our risks is the image scanning process. We are planning on using a camera connected to a Raspberry Pi to scan the printed image and process it on the Raspberry Pi. However, while we have tried to account for lighting conditions, orientation, and low resolution because of printing, we have not tested for it yet. If the image scanning does not work for some reason, we will try to make our

pattern less detailed and make it easier to scan. If that doesn't work, we can show the image on a screen and then send it to the second Rpi for decoding.

Another risk we have is having the two Raspberry Pis working and set up with all of the components hooked up. We have not tested running the detection and decoding algorithm on a Raspberry Pi and the installation process for OpenCV. We also need to run p5.js on the other Raspberry Pi to do our encoding. If the software is not able to integrate with the Raspberry Pis for some reason, we can always run our project on our own computers and have them communicate with one another. Finally, we might have some issues getting our Raspberry Pis onto CMU Secure WiFi so that they can have access to our global dictionary with the pattern. If this does not work, we can copy the file over via ethernet.

Related Work

Our project is based off the idea that a QR code could be made to look prettier and more obscure, such that it is not immediately obvious that the image has data encoded in it. As such we've taken inspiration from the QR code as well as two other projects, the Penguin Barcode and Darkbyte, to shape our design requirements.

Short for the Quick Response code, QR codes are an efficient way to represent up to 4,296 characters of encoded data, over 100 times the maximum of a one dimensional barcode. They also contain error detection or correction mechanisms, which allow for creation of more artistic codes that can still scan correctly despite intentional errors, such as the code in Figure 9. It scans in a matter of seconds and can be recognized at any angle. However, as multiple articles point out, QR codes are not pleasing to look at and are not always as convenient for users as they are for the developers.

Figure 8: A more aesthetic QR code with intentional errors

The Penguin Barcode operates under the assumption that as long as we can count the number of figures, lines, shapes, etc., enclosed in a region, the shape of the

actual region is irrelevant. While this idea is functional, it doesn't serve our purpose of encoding text, in which the order of the data is as important as the data itself. Darkbyte, on the other hand, does retain this ordering and encodes text into a pattern of RGB values. However, the random grid of colors and gradients is hardly more visually pleasing than a QR code.

As a result, our project offers something new that delivers the best of each of these approaches. Our algorithm is not only as fast as a QR code, but also more visually appealing than the encoded image from Darkbyte. While not as freeform as the Penguin Barcode, our approach is pretty to look at and can be extended to include different shapes and even animals if desired. This customizability of our encoding algorithm and the efficiency of the CV scanning makes our project equally as competitive and much more obscure and pleasing to look at than other related projects.

References

Data Encoding - QR Code Tutorial, www.thonky.com/qr-code-tutorial/data-encoding.

"About Darkbyte." *Text to Image Encoder/Decoder*, xcode.darkbyte.ru/.

"The Penguin Barcode". Computerphile, director. *YouTube*. *YouTube*, YouTube, 14 Jan. 2014, www.youtube.com/watch?v=kW39Mt5kscQ.

Dang, Henry. "Color Detection in Python with OpenCV." *Henrydangprg*, 7 Feb. 2018, henrydangprg.com/2016/06/26/color-detection-in-python-with-opencv/.

"Detect Complex Shapes with OpenCV Python." *Stack Overflow*, stackoverflow.com/questions/44387077/detect-complex-shapes-with-opencv-python.

"How to Build a Mobile Document Scanner in Just 5 Minutes." *PyImageSearch*, 3 July 2018, www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/.

"Information Capacity and Versions of the QR Code." *Information Capacity and Versions of QR Code | QRcode.com | DENSO WAVE*, www.qrcode.com/en/about/version.html.

Mallick, Satya. "Blob Detection with OpenCV." *Learn OpenCV*, 17 Feb. 2015, www.learnopencv.com/blob-detection-using-opencv-python-c/.

Matthews, Cate. "Should Barcodes Be Penguin-Shaped? (We Think The Answer Is Clear)." *The Huffington Post*, TheHuffingtonPost.com, 5 Feb. 2014.