A3 - Creative Report

Linda Yen

yenling12@gmail.com

February 12, 2024

Abstract

This Assignment is the Creative stage. The main objective is to take the knowledge gained from Assignments 1 and 2 and apply it to a specific problem. This report will walk through how I used my Raspberry Pi to solve an inventive problem, the challenges I faced, and the results of my project.

1 Introduction

As a beginner in embedded hardware implementation, I decided to implement a fun project that involved some of the peripherals I explored in Assignment 2. In this assignment, I will walk through how I created a Magic The Gathering (MTG) Card Scanner. For those unfamiliar with MTG, it is a strategic, collectible card game that can be played amongst two to four players. As with any collectible card game, over time, the amount of cards you possess grows exponentially as more games are played. One of the ways to keep your card collection to a manageable amount is to sell your less desired or redundant cards to your local game store. The manual, tedious process would be to review each card you want to sell and look up online how much the cards are worth by searching the card id or title. To automate this process, I decided to create a card scanner that would take a card image, extract the card title or id, utilize a public API to determine it's market cost, and store it's value in a file.

2 Methods

Here is the general execution flow of the MTG Card Scanner:

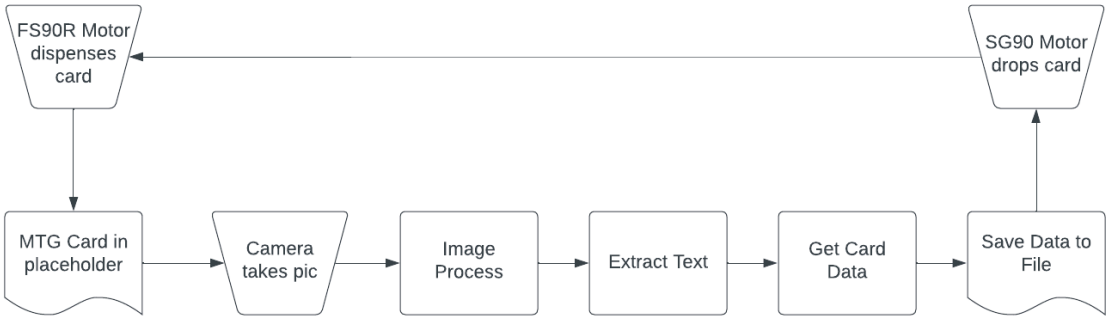

Figure 1. MTG Card Scanner Process

2.1 Materials

Below are the materials I used for the final implementation:

- Raspberry Pi 4 8GB
- Voltage Regulator for Bread Board 5V/3.3V
- 5V 2A (2000mA) switching power supply
- Beffkkip SG90 9g Micro Servo Motor

- FeeTech FS90R 360 Degree Continuous Rotation Micro Servo Motor + RC Tire Wheel
- LEGO - Classic Medium Creative Box
- 5 Megapixels 1080p Sensor OV5647 Mini Camera Module with 6 Inch 15 Pin Ribbon Cable
- Pin Connectors
- Capacitor (350uF 16V)

2.1 Servo Motors

2.1.1 Powering the Motors

Both servo motors are powered by 5V through a Voltage Regulator. The SG90 data port is connected to GPIO12 and the FS90R data port is connected to GPIO13. Both motors' positive and negative ports are connected to the Voltage Regulators positive and ground port respectively. The Rasbperry Pi ground port is also connected to the Voltage Regulator's ground. I added a 350uF Capacitor in parallel to the positive and ground terminal as I've read it improves motor performance by reducing jitter and absorbing voltage spikes. All connections were made on a breadboard. An image of the set up is shown below:

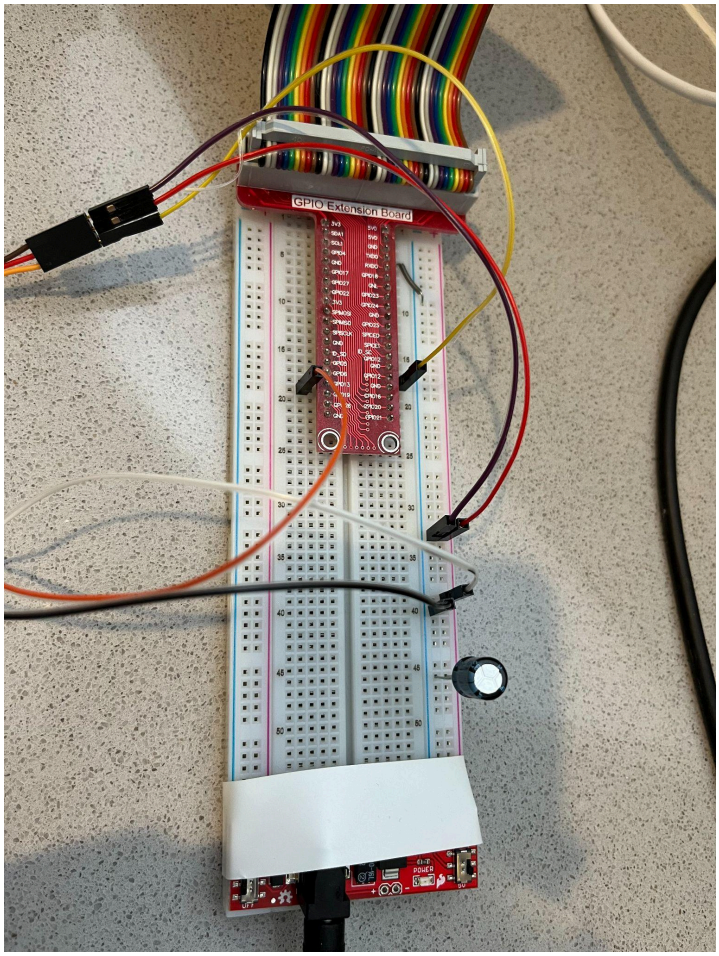

Figure 2. Bread Board Connections

NOTE: If 5V external power supply is connected directly to the motors (without a Voltage Regulator), the motor will not operate. This could be that the power supply can not provide the peak current the motor needs or there could be voltage fluctuations in the power supply output.

2.1.2 Beffkkip Micro Servo SG90

Since there was no accurate datasheet for this micro servo from this manufacturer, I determined the following based on some customer reviews and testing:

- Operates at 50Hz (Period of 20ms)
- PWM: 0.5ms = 0 degrees, 2.5ms = 180 degrees

Therefore, using the equation:

$$Duty\ Cycle\ (\%) = \frac{PW\ (ms)}{Period\ (ms)} x\ 100$$

For 0.5ms and 2.5ms, I get:

$$\frac{0.5ms}{20ms} x\ 100\ =\ 2.5\%\ \text{and}\ \frac{2.5ms}{20ms} x\ 100\ =\ 12.5\%$$

Therefore, the motor operates at a 10% window with 2.5% as the lowest value. From this we can relate duty and angle:

$$Duty\ Cycle\ (\%)\ =\ \frac{Angle\ (deg)}{18} +\ 2.5$$

This motor was used in the card holder section for the image capture process. When the card image has been captured, the motor will rotate 90 degrees to drop the card in preparation for the next one.

2.1.3 FeeTech Continuous Servo FS90R

Unlike the previous servo motor, the FS90R can rotate continuously 360 degrees. Therefore, instead of going to a specific angle, the motor will be static at a specific pulse. A shorter pulse makes it rotate clockwise (CW) and a longer pulse makes it rotate counter-clockwise (CCW). Using the datasheet [2], the below are some important specs to note:

- Rotating direction: 700usec - 2300 usec (CW when 700-1500usec; CCW when 1500-2300usec)
- Rest point is 1.5 ms
- Rotation is 360 degrees

Therefore, using the same equation above for 0.7ms, 1.5ms, and 2.3ms I can get the Duty Cycles needed for 50Hz:

$$\frac{0.7ms}{20ms} x\ 100\ =\ 3.5\%\ \text{and}\ \frac{1.5ms}{20ms} x\ 100\ =\ 7.5\%\ \text{and}\ \frac{2.3ms}{20ms} x\ 100\ =\ 11.5\%$$

Since this motor can continuously rotate 360 degrees, I used it for the card dispenser section. The motor with the RC wheel is used to push out the bottom most card from the deck, into the card holder. After some testing, if I set x < 7, *GPIO.PWM.ChangeDutyCycle(x)*, the wheel will rotate CW. If I set x > 7, the wheel will rotate CCW. Closer to Duty Cycles 3.5% and 11.5%, the wheel rotation is at its maximum speed and when closer to Duty Cycle 7%, the wheel rotation is at its minimum speed.

When the card is pushed out by the wheel, the previous card may also be withdrawn partially due to the friction between the cards. Therefore, to reset the previous card for the next withdrawal after the card has been dispensed, the motor is set to reverse in the CW direction to push the previous card back under the deck.

2.2 Camera Set Up

For operating the camera, I used the Picamera2 Library [1]. To set up the initial position of the camera on the card scanner, I used the QT.preview mode since the board is "headless" (NOTE: I used PuTTy to ssh into the board since it supports X11 forwarding). The camera takes a still picture resolution of 2592x1944. Since the camera was capturing images at a close proximity, the focus had to be adjusted as the image quality was too poor to capture texts properly. Following reference [3], I used a pair of tweezers to rotate the camera lens to improve the focus of the camera. Below are the results before and after focus adjustment:

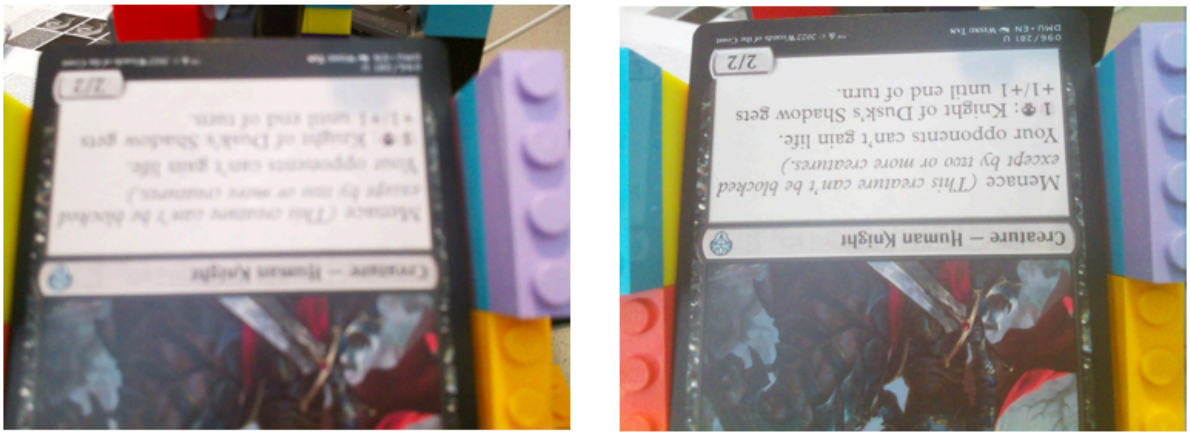

Figure 3. Before and After Focus Adjustment

2.1 Image Processing

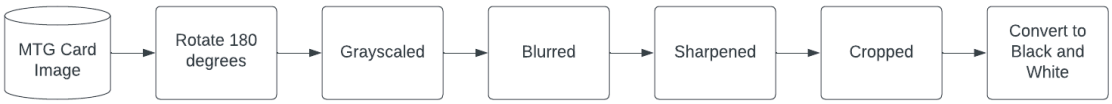

Figure 4. Image Processing Steps

Since the image is captured upside-down, the image is rotated 180 degrees (*cv2.rotate(img_rgb, cv2.ROTATE_180)*)

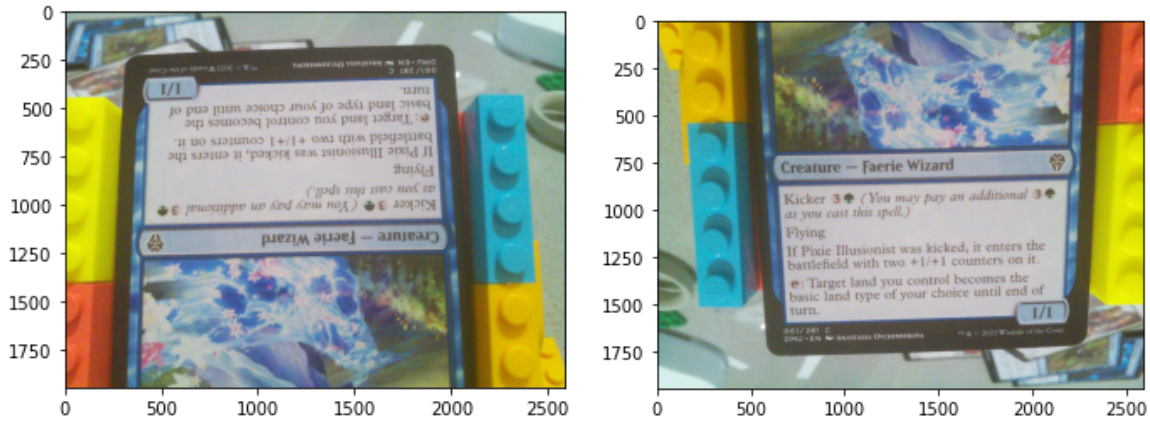

Figure 5. Image Rotation

The image is then converted to Grayscale (*cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)*). This standardizes the image for different lighting conditions, reduces the data size, and improves edge detections.

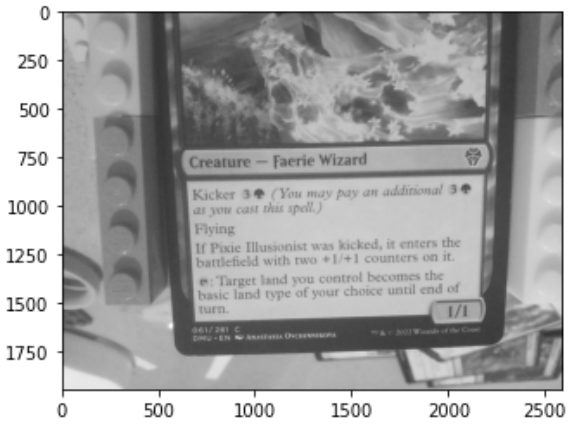

Figure 6. Image Grayscale

To remove noise, the image is blurred (*cv2.GaussianBlur(img_gray, (7, 7), 0)*
) and then sharpened (*cv2.filter2D(img_blur, -1, kernel)*) to accentuate the details and edges.

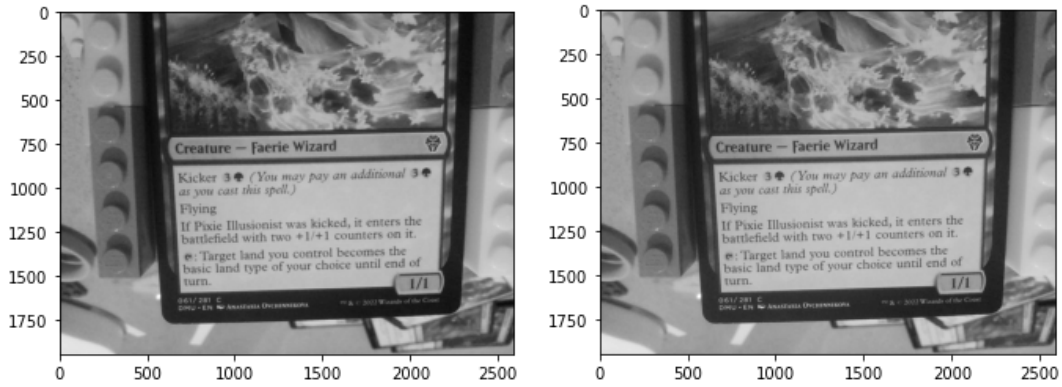

Figure 7. Image Blurred (left) then Sharpened (right)

The image is then cropped (*img_gray[1600:1800, 600:900]*) where the Card ID and Card Set number is located (bottom left corner of the card).

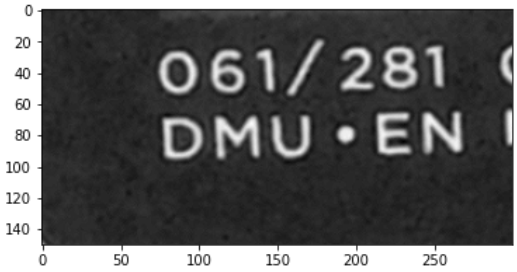

Figure 8. Cropped Image

Finally, the image is converted to black and white, where the text is black and the background is white (*cv2.inRange(cropped_image, lower, upper)*) by applying a binary mask. This makes the text clearer for extraction.

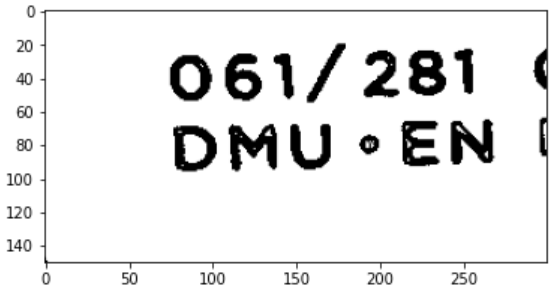

Figure 9. Black and White Image

2.1 Text Extraction

In order to extract the text from the processed image, I chose to use Tesseract, an open-source Optical Character Recognition (OCR) engine [4]. Since I am using Python for this project, I downloaded the PyTesseract librar, which is a Python wrapper for the Tesseract OCR engine.When used on the processed image in Fig 9., I get the following:

O61/ 281 |
DMU °-EN |

The first 3 digits is the Card ID (“061”) and the first 3 characters on the new line is the Set ID (“DMU”). In order to determine if the card is a “foil” type (which has a different USD price associated with it) I implemented a way to check if a “/” character is next to the Card ID. If a “/” is absent, then the card is labeled as a “foil” card.

3 Results

3.1 Structure

LEGOs were used to construct the camera holder, the placeholder for the card, and the card dispenser structure. All code was written in python (Appendix A). An image of the set up is shown below:

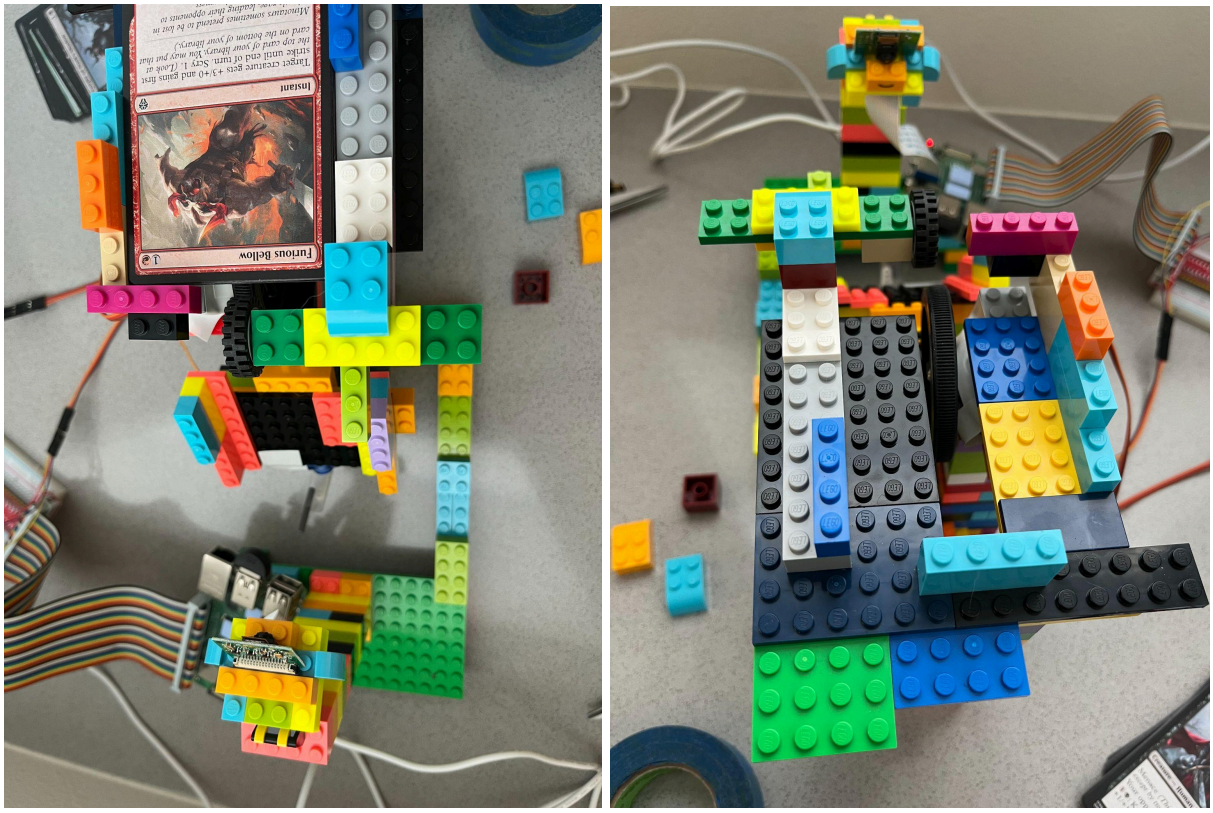

Figure 10. MTG Card Scanner

3.2 Test Run

For a 20 cycle run (20 cards scanned and processed continuously), the elapsed time was 137.07sec (~2.3min). The temperature at the start was 41.3℃ and at the end 45.7℃. Out of the 20 cards scanned, only one card had an error where Tesseract confused a “D” with an “O” for the Card Set value, therefore no data was found. Below is the resulting data table from the run:

Text_Extract_Status	Card_ID	Card_Set	Card_Data	Card_Name	Card_Price_USD	Card_Price_Foil_USD	is_Foil
success	116	DMU	[{'object': 'card', 'id':	Balduvian Berserker	0.02	0.02	FALSE
success	157	DMU	[{'object': 'card', 'id':	Broken Wings	0.02	0.05	FALSE
success	60	DMU	[{'object': 'card', 'id':	Phyrexian Espionage	0.02	0.03	FALSE
success	80	DMU	[{'object': 'card', 'id':	Battle-Rage Blessing	0.02	0.14	FALSE
success	4	DMU	[{'object': 'card', 'id':	Argivian Cavalier	0.02	0.03	FALSE
success	128	DMU	[{'object': 'card', 'id':	Goblin Picker	0.05	0.03	FALSE
success	76	DMU	[{'object': 'card', 'id':	Vodalian Mindsinger	0.16	0.13	FALSE
success	11	DMU	[{'object': 'card', 'id':	Citizen's Arrest	0.02	0.04	FALSE
success	26	DMU	[{'object': 'card', 'id':	Mesa Cavalier	0.02	0.02	FALSE
no data found	96	OMU	0				FALSE
success	95	DMU	[{'object': 'card', 'id':	Gibbering Barricade	0.03	0.06	FALSE
success	183	DMU	[{'object': 'card', 'id':	Tear Asunder	2.23	5.47	FALSE
success	187	DMU	[{'object': 'card', 'id':	Vineshaper Prodigy	0.02	0.06	FALSE
success	105	DMU	[{'object': 'card', 'id':	Shadow Prophecy	0.04	0.24	FALSE
success	41	DMU	[{'object': 'card', 'id':	Academy Wall	0.03	0.08	FALSE
success	182	DMU	[{'object': 'card', 'id':	Tail Swipe	0.2	0.98	FALSE
success	161	DMU	[{'object': 'card', 'id':	Elfhame Wurm	0.02	0.12	FALSE
success	122	DMU	[{'object': 'card', 'id':	Electrostatic Infantry	0.12	0.57	FALSE
success	30	DMU	[{'object': 'card', 'id':	Runic Shot	0.05	0.12	FALSE
success	115	DMU	[{'object': 'card', 'id':	Writhing Necromass	0.04	0.07	FALSE

Figure 11. 20 Cycle Test Run Results

4 Discussions

4.1 Time

During the 20 cycle run, the temperature increased by 4.4℃, but after a few more test runs, the end temperature remained around 45.7℃. Therefore, temperature was not a concern for this project. As for time, the whole process does take some time as only scanning 20 cards took 2.3 minutes. Scaling this to a larger number (100 cards), would take just over 11 minutes, assuming nothings gets shifted during the cycle.

4.2 Lighting

A key factor in the error rate of the card scanner was lighting. In the image processing phase, I used cv2.inRange() to convert the text to black and everything else to white. However, even the smallest change in lighting required me to adjust the lower and upper bounds of cv2.inRange() constantly (e.g. Morning vs Evening made a large difference in error rate if parameters remained the same).

4.3 Tesseract Issues

Another key factor in error rate was the Tesseract OCR engine accuracy. Below are the processed

images of the “error” card (left) and a “success” card (right).

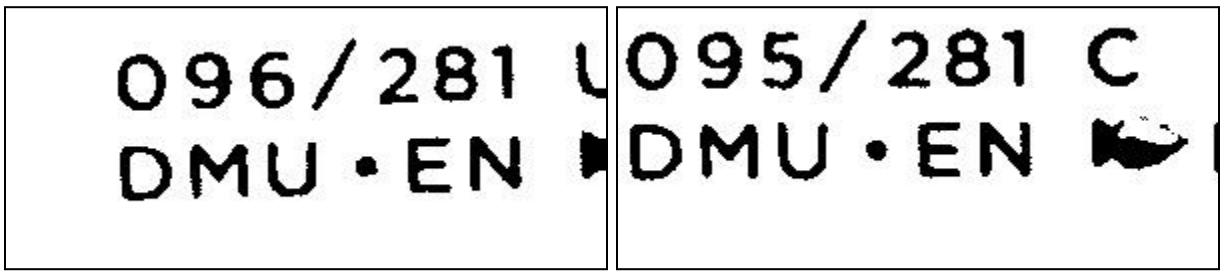

Figure 12. Error Card (“OMU”) (left) and Success Card (“DMU”) (right).

As you can see, they both look almost identical except their relative positions, however Tesseract recognized the text on the left as “OMU” (incorrect) and on the right “DMU” (correct).
Other corner cases include Tesseract recognizing the zero in front of the “061” as an “O”. Since I know the Card ID has to be a number, I used the Re Python Library for regular expression conditions to replace any “O”s or “o”s with a zero. Then I converted the Card ID value to an integer to remove any leading zeros. Additional code was added to ensure the Set ID included only three characters and the Set ID included no more than three digits.

4.4 Functionality Issues

Some other contributors to error rates with the card scanner is the placement of the card dispenser relative to the card holder and the weight on top of the next card to be dispensed. Tiny shifts in movement can cause the card to improperly fall into the card holder at an angle with one side sitting on the edge.

For the structure development itself, I referenced [6] to get started. One of the major challenges for the physical structure was the card dispenser. Using LEGOs for the structure actually limited my ability to fix components with finer adjustments as I was limited by separation length of the LEGO pegs. It took hours of different designs to finally get my card dispenser to output a card at a time. Another aspect of the physical design was the weight on top of the deck. If there was not enough weight on the deck, the card would bounce on top of the wheel and then be blocked by the LEGO piece that holds the rest of the deck back during the card dispense phase.

4.5 Future improvements

For future extensions to the project, I would try to improve the structure and image processing

parameters to make this system more robust. For the image processing piece, I've been manually updating the upper and lower bounds to create the mask to isolate the texts using Tesseract. Therefore, finding an alternative way to extract the texts would definitely save a lot of these small adjustments (e.g. Using a different OCR engine or developing an automated way to determine the ideal parameters). As for the structure, since it is made of LEGOs, it is sensitive to changes and things can fall apart if impacted with a slight force.

5 Conclusion

The exploration I did in Assignment 2 greatly contributed to getting this project to work as a large portion of my time was spent figuring out how to get the motors to operate in a specific way and getting the camera to operate with the correct library and ideal quality. I spent many hours just trying to figure out the correct PWM parameter (as the datasheet was wrong or missing information) to adjust the motor angle and speed. I also spent a lot of time trying to improve the camera image quality through software until I found out the lens could be rotated to adjust focus. In terms of baselines from Assignment 2, that did not have a significant impact on this project since it did not push the raspberry pi to any limits in terms of memory or temperature.

I definitely gained a lot of appreciation for hardware products as getting all the components of this project to work together took a lot of time. Something as simple as a card dispenser, which I assumed would be straight forward, took me days to figure out the appropriate design.

In terms of production, whether limited or mass, this product would need to have a more robust design as it is too sensitive to changes in its environment (e.g. physical shift, lighting, etc) to be re-created.

6 References

1. PiCamera2 Manual. https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf
2. FS90R Servo Motor Continous Rotation Datasheet. https://www.pololu.com/file/0J1867/FS90R.pdf
3. Adjusting the camer focus. Raspberry Pi Foundation. https://projects.raspberrypi.org/en/projects/infrared-bird-box/6
4. Tesseract OCR. Github. https://github.com/tesseract-ocr/tesseract

5. Build Live Text Recognition with the Raspberry Pi. Tutorials for Raspberry Pi. https://tutorials-raspberrypi.com/raspberry-pi-text-recognition-ocr/
6. Portera, M. (May 15 2018). Trading Card Scanner/Organizer. hackster.io. https://www.hackster.io/mportatoes/trading-card-scanner-organizer-84399a
7. Molloy, D. Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux. (2016). Wiley.
8. Raspberry Pi Pinout. https://pinout.xyz/pinout/i2c
9. RPI vcgencmd usage. Elinux.org. https://elinux.org/RPI_vcgencmd_usage
10. Miller, L. (Jul 16 2020). How to Control a Servo with Raspberry Pi. Learn Robotics. https://www.learnrobotics.org/blog/raspberry-pi-servo-motor/

7 Appendices

Appendix A

```
########################################
# Import libraries
import RPi.GPIO as GPIO
import time
from picamera2 import Picamera2
import numpy as np
import cv2
import os
import pytesseract
from pytesseract import Output
import requests
import re # Regular expression
import csv
import subprocess
########################################
# Initializations
# Set GPIO numbering mode based on GPIO pin
GPIO.setmode(GPIO.BCM)
# Set Motor Pins
fs90r_pin = 13
```

```
sg90_pin = 12
# Set pin 13 as an output
GPIO.setup(fs90r_pin,GPIO.OUT)
# Set pin 12 as an output
GPIO.setup(sg90_pin,GPIO.OUT)
# Set up Camera
picam2 = Picamera2()
capture_config = picam2.create_still_configuration(raw={}, display=None)
picam2.configure(capture_config)
# Initialize variables for file
########################################
# Functions
def rotate_fs90r(pin):
    '''
    Sets the FS90R Servo Motor. Range is
    0 to 360 degrees. Rotates motor CCW for
    1 sec, then rotates CW for 0.5 sec.
    Args: pin
    Returns: none
    '''
    # set pin as PWM. 50 = 50Hz pulse
    motor_fs90r = GPIO.PWM(pin,50)
    # Rotate FS90R Motor
    motor_fs90r.start(0)
    time.sleep(0.5)
    # Rotate motor to push out card
    motor_fs90r.ChangeDutyCycle(7.6)
    time.sleep(1.2)
    # Reverse motor to pull back previous card
    motor_fs90r.ChangeDutyCycle(6.5)
    time.sleep(0.8)
    motor_fs90r.stop()

def setangle_sg90(angle, motor_PWM):
    '''
    Sets the SG90 Servo Motor angle.
    Angle range is 0 to 180 degrees.
    Args: angle, motor_PWM
    Returns: none
    '''
```

```
    motor_sg90 = motor_PWM
    duty = angle / 18 + 2.5
    print (f"rotating sg90 motor {angle} degrees..")
    motor_sg90.ChangeDutyCycle(duty)
    time.sleep(0.5)
    motor_sg90.ChangeDutyCycle(0)

def rotate_sg90(pin):
    '''
    Rotates the SG90 motor to 0 degrees, then back
    to 90 degrees (its reset position)
    Args: pin
    Returns: none
    '''
    motor_sg90 = GPIO.PWM(pin,50) # set pin as PWM. 50 = 50Hz pulse
    motor_sg90.start(0)
    setangle_sg90(0, motor_sg90)
    time.sleep(0.7)
    setangle_sg90(90, motor_sg90)
    motor_sg90.stop()
    print ("sg90 Done")

def process_image(img,count):
    '''
    Processes the inputted image for text extraction.
    Rotates the image, converts the image to grayscale,
    blurs the image, sharpens it, crops the
    image to isolate the text of interest, then puts a
    mask over the image to accentuate the text.
    Args: img
    Returns: Processed Image
    '''
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Rotate image
    img_rgb = cv2.rotate(img_rgb, cv2.ROTATE_180)

    # Convert image to grayscale
    img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
```

```
    # Blur image. Remove noise using a Gaussian filter
    img_blur = cv2.GaussianBlur(img_gray, (7, 7), 0)

    # Sharpen the image
    img_gray = cv2.filter2D(img_blur, -1, kernel)

    # Crop image to grab Ids
    cropped_image = img_gray[1690:1820, 600:900]

    # Save cropped image
    cv2.imwrite(f'./cropped_image/cropped_img_{count}.jpg', cropped_image)

    # Change image to black and white
    # Day Time 160-260
    # Night Time 100-200
    lower = 100
    upper = 200
    img_inrange = cv2.inRange(cropped_image, lower, upper)
    neg_img = ~img_inrange

    cv2.imwrite(f'./bw_image/bw_image_{count}.jpg', neg_img)

    return neg_img

def get_text(img):
    '''
    Extracts any text using pytesseract from
    the processed image.
    Args: img
    Returns: Data outputted by pytesseract
    '''
    data = pytesseract.image_to_string(img, lang='eng')
    print(data)
    return data

def process_string(string):
    """
    Replaces all "O" characters with "0" in a string
    regardless of case.
    Args: string
```

```
    Returns: The modified string with "O" replaced by "0".
    """
    new_string = re.sub(r"O", "0", string, flags=re.IGNORECASE)

    # Remove any erroneous characters after the first 3 chars
    if len(string) > 3:
        new_string = new_string[:3]
    return new_string

def check_id_valid(string):
    '''
    Checks if all chars in the string ar digits.
    Args: string
    Returns: True if all chars are digits. False otherwise.
    '''
    # Check if chars are digits
    isNum = all(char.isdigit() for char in string)

    if isNum and len(string) == 3:
        return True
    else:
        return False

def extract_card_id_set(text):
    '''
    Extracts the card id and card set from
    the tesseract results.
    E.g. 134/281
         DMU -EN
    card_id=134, card_set=DMU
    Args: text
    Returns: status, card_id, card_set, isFoil
    '''
    status = "N/A"
    isFoil = False
    # Extract MTG Card ID on the first line
    card_id = text.split("\n")[0].strip()
    # Check to see if there is a '/'. Foil cards do not have a '/'
    if re.search('/', card_id):
        card_id = card_id.split('/')[0]
```

```
    else:
        isFoil = True

    # Replace "O" and "o" with "0"
    card_id = process_string(card_id)
    # Check if all chars in card_id are digits
    id_valid = check_id_valid(card_id)
    #print('id_is_digits: ', id_is_digits)

    # Extract the MTG card set
    card_set = text.split("\n")[1].split(" ")[0]
    # If card_set length is greater than 3, extract first 3 characters
    if len(card_set) > 3:
        card_set = card_set[:3]

    if id_valid and len(card_set) == 3:
        # Convert ID to int type to remove any leading zeros
        card_id = int(card_id)
        print('card_id:', card_id)
        print('card_set:', card_set)
        status = 'success'
    elif not id_valid and len(card_set) == 3:
        status = 'Issue with Card ID'
    elif id_valid and len(car_set) != 3:
        # Convert ID to int type to remove any leading zeros
        card_id = int(card_id)
        status = 'Issue with Card Set'
    return status, card_id, card_set, isFoil

def get_card_data(id, set):
    # Create API info to search on
    info = {
    "identifiers": [
        {
        "set": set,
        "collector_number": str(id)
        }
    ]
    }
```

```
    # POST API URL
    url = "https://api.scryfall.com/cards/collection"

    # Send Request
    response = requests.post(url, json=info)

    # Handle response
    if response.status_code == 200:
        print("Request successful!")
        data = response.json()
        if not data["data"]:
            return 0
        else:
            return data["data"]
    else:
        print(f"Error: {response.status_code}")
        print(response)
        return 0

def empty_folder(folder_path):
    '''
    Clears out image folders.
    Args: folder_path
    Returns: none
    '''
    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        os.remove(file_path)
    print(f"{folder_path} emptied successfully!")

def get_cpu_temp():
  """Gets the CPU temperature of the Raspberry Pi using vcgencmd."""
  output = subprocess.check_output(["vcgencmd",
"measure_temp"]).decode("utf-8")
  temp = float(output.split("=")[1].split("'")[0])
  return temp
########################################
# Create new card.csv file with headers
with open('card.csv', 'w', newline='') as file:
    writer = csv.writer(file)
```

```
writer.writerow(['Text_Extract_Status','Card_ID','Card_Set','Card_Data','C
ard_Name','Card_Price_USD','Card_Price_Foil_USD','is_Foil'])
    print('card.csv created')

# Empty folders
bw_image_path = './bw_image'
cropped_image_path = './cropped_image'
empty_folder(bw_image_path)
empty_folder(cropped_image_path)

# Intialize count for scan cycle
count = 20
cpu_temp = get_cpu_temp()
print(f"CPU temperature at start: {cpu_temp:.1f}°C")
start_time = time.time()  # Capture start time
while count > 0:
    # re-Intialize variables
    data = ''
    card_name = ''
    usd_price = ''
    usd_foil_price = ''
    # Rotate fs90r motor to push out card
    rotate_fs90r(fs90r_pin)

    # # Capture card image
    picam2.start(show_preview=False)
    time.sleep(1)
    picam2.capture_file("test.jpg")
    picam2.stop_()

    # Read and process image
    image = cv2.imread("test.jpg")
    img_processed = process_image(image,count)

    # Extract text from image
    text = get_text(img_processed)
    print('text: ', text)

    # Get ID and Set from extracted text
```

```
    status, id, set, isFoil = extract_card_id_set(text)

    # Drop card in preparation for next card
    rotate_sg90(sg90_pin)

    if status == 'success':
        # Get card data
        data = get_card_data(id, set)

        # If card data is found, get Card Name and Price
        if data != 0:
            card_name = data[0]["name"]
            usd_price = data[0]["prices"]["usd"]
            usd_foil_price = data[0]["prices"]["usd_foil"]
            print(f'card title: {card_name}; price is {usd_price} usd,
{usd_foil_price} usd_foil')
        else:
            status = 'no data found'

    # Write card information to card.csv
    with open('card.csv', 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([status, id, set, data, card_name, usd_price,
usd_foil_price, isFoil])

    count -= 1

end_time = time.time()  # Capture end time
elapsed_time = end_time - start_time
print("Elapsed time:", elapsed_time, "seconds")

cpu_temp = get_cpu_temp()
print(f"CPU temperature at end: {cpu_temp:.1f}°C")

# #Clean things up at the end
GPIO.cleanup()
```