Artwork Super-Resolution Scanning Application

ECpE SENIOR DESIGN – GROUP 18

Reece Dodge, Isaac Plambeck, Garrett Powell, Samuel Schaphorst

Reece Dodge

Conceptual Lead

Electrical Engineering

Cedar Rapids, IA

Isaac Plambeck

Software Development Lead

Software Engineering

Sioux City, IA

Garrett Powell

Product Delivery Lead

Electrical Engineering

Marion, IA

Samuel Schaphorst

Testing/QA Lead

Electrical Engineering

Omaha, NE

THE PROBLEM

- Freelance artists with small budgets and limited income
- No current cheap/easy solution to digitizing physical artwork
- Few ways to monetize physical artwork (aside from sale)
- Large-scale prints require high resolution images
- Difficult to create a “backup” of a painting

OUR SOLUTION

- Application that takes in a set of 3-10 user uploaded photos a single canvas
- Automatically produces a digital copy according to a user-determined output resolution
- Velcro corner brackets for mounting to artwork
  - Artwork boundary identification
  - Pantone color reference cards
- Built in Python
  - OpenCV Computer Vision library

REQUIREMENTS

FUNCTIONAL

- Automatic performance of various image processing algorithms
  - Super-resolution scaling
  - Color correction
  - Border detection
  - Perspective correction
  - Noise reduction
- Simplified algorithms to reduce computation time

NON-FUNCTIONAL

- Utilize Python/OpenCV
- Easy to use UI to allow for efficient navigation
- Limited to cheap and accessible hardware
  - Smart phone cameras
  - Pantone Color Match cards
  - ArUco markers
  - Wooden corner brackets

EXISTING SOLUTIONS

ADOBE PHOTOSHOP

- Requires strong photoshop skills and digital image processing background
- Each image processing step is performed manually
- Computationally inefficient
- Adobe subscription required

PURPOSE-BUILT SCANNING MACHINES

- Large
- Expensive


DESIGN OVERVIEW

PROCESS FLOW DIAGRAM

USER INPUT

IMAGE UPLOAD

The user uploads the photos to ArtScan and sets a target output resolution.

IMAGE COLLECTION

The user mounts our corner brackets to their artwork and takes 5-10 photos of the entire canvas in a well-lit enviornment, standing as close as possible.

IMAGE PROCESSING

COLOR CORRECTION

ArtScan uses 4 Pantone Color match cards to ensure the output's color profile is true and accurate.

SUPER-RESOLUTION UPSCALING

The images are upscaled to the target resolution using an OpenCV LapSRN super-resolution scaling model.

BORDER DETECTION

ArtScan uses ArUco markers mounted to corner brackets to find the canvas within the images.

CROP/PERSPECTIVE CORRECTION

ArtScan uses ArUco markers to find the canvas within the images and performs perspective correction. Non-artwork space is cropped.

ArtScan

FINAL OUTPUT

IMAGE SUPERPOSITION

ArtScan overlays the set of images and finds the median of each pixel. The result is the final output .

FILE OUTPUT

ArtScan prepares the output file and delivers it to the user.

TESTING

TESTING DATA – SAMPLE IMAGES

- Client provided sample artwork photos
- Varied lighting conditions
- Artificial noise added images
- Photoshopped ArUco markers (pre-corner brackets)

TESTING METHODOLOGY

- Isolated component testing
- Integration testing

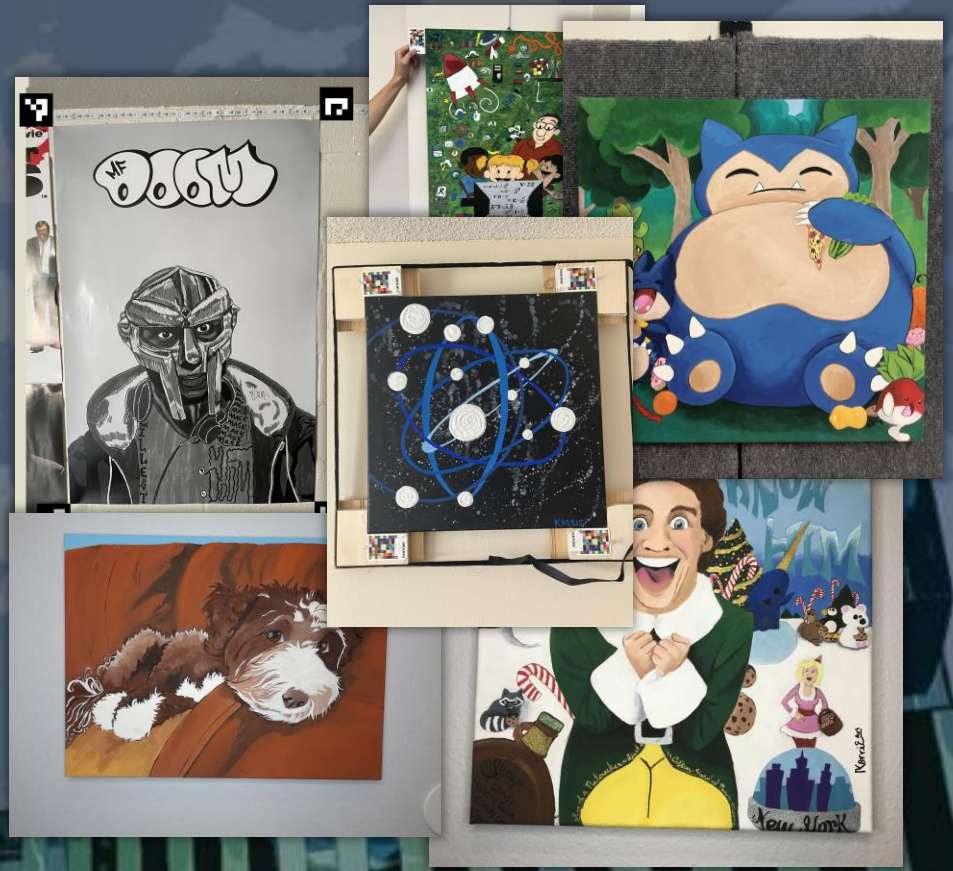
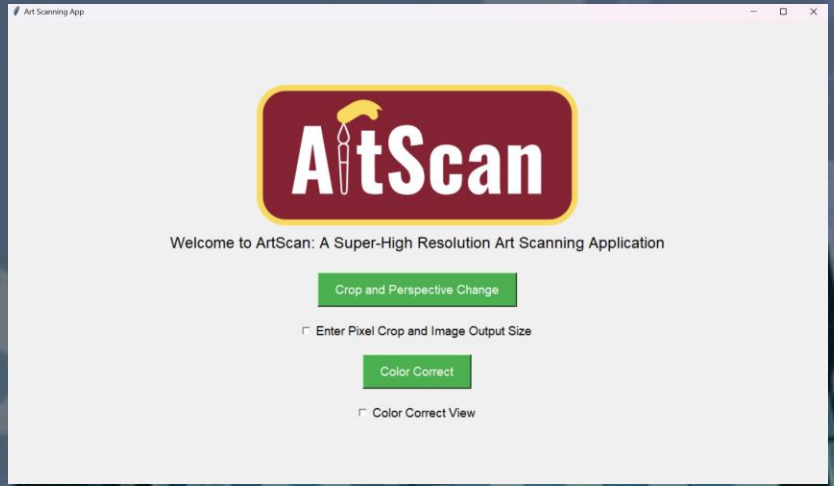


IMAGE COLLECTION

- Utilize four wooden frames for each corner of painting
- Each wooden frame is equipped with a pantone card and an ArUco marker
- Velcro patches around the edges for canvases that are hung on wall

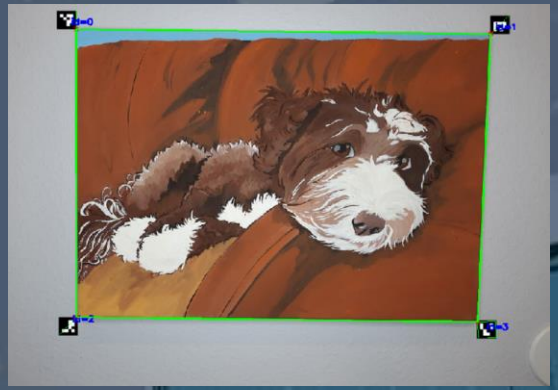
CURRENT USER INTERFACE

- Images are uploaded to user's personal device
- User selects 'Upload' button and selects images
- Final product is displayed



BORDER DETECTION + PERSPECTIVE CORRECTION

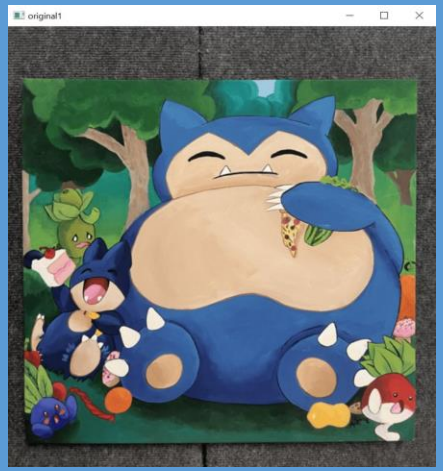
- Program detects each corner based on the 4 ArUco Markers placed in the corner of each painting
- Then Extracts the coordinates of each corner
- Next a perspective transform, and crop is applied using the list of coordinate points, and chosen output image height and width

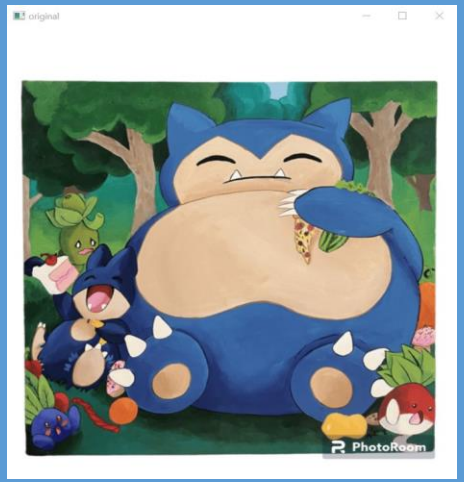


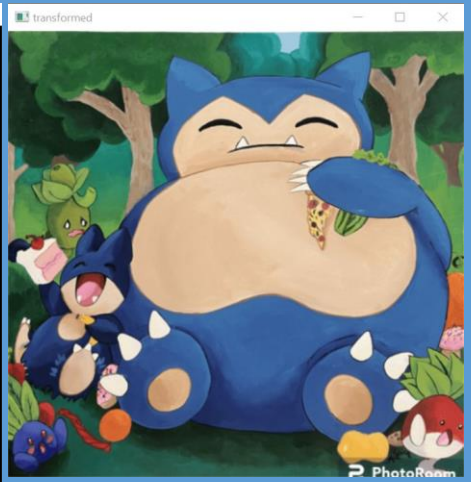
BORDER DETECTION AND PERSPECTIVE CORRECTION

- One major challenge was trying to extract the corner coordinates in the correct order and consistently locate the ArUco Marker.
- A big help to this was getting the specific ArUco Marker ID number, in this case 0, 1, 2, 3. And programming these ID numbers to be Top Left Right and Bottom Left Right corners.
- Marker lighting needs to be good. And Angles need to be straight on.

FIRST SEMESTER CROPPING ALGORITHM







SUPERPOSITION – MEDIAN OVERLAY

- Images run through median loop
- Loop converts each image to an array to calculate the median of each pixel value
- New image is created from array, converted back to an image and displayed to user
- Initial issues with converting image to and from array

```python
def find_median_image(setOfCroppedImages):
    # Open the images

    setOfNPImages = []
    for i in range(len(setOfCroppedImages)):
        #img = Image.fromarray(setOfCroppedImages[i])
        nparrayImage = np.array(setOfCroppedImages[i])
        setOfNPImages.append(nparrayImage)

    median_array = np.median(setOfNPImages, axis=0).astype(np.uint8)

    # Create a new image from the median array
    median_image = Image.fromarray(median_array)

    return median_image
```

MEDIAN OVERLAY

Color Correction with Pantone

- Color correction performed using Pantone color match cards
- Total of 4 cards used in design
  - Each card placed in a different corner
- Correction contributions of each card for all relevant pixels is based on distance
- Pantones detected by ArUco markers present on the card
- Reference Pantone image used to comparatively adjust color histograms

COLOR CORRECTION WITH PANTONE

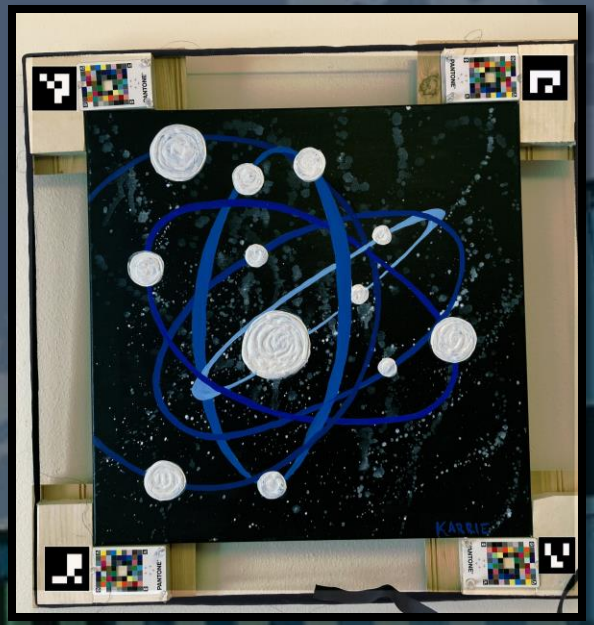
Reference

Input

Output

Color Correction with Pantone

Input

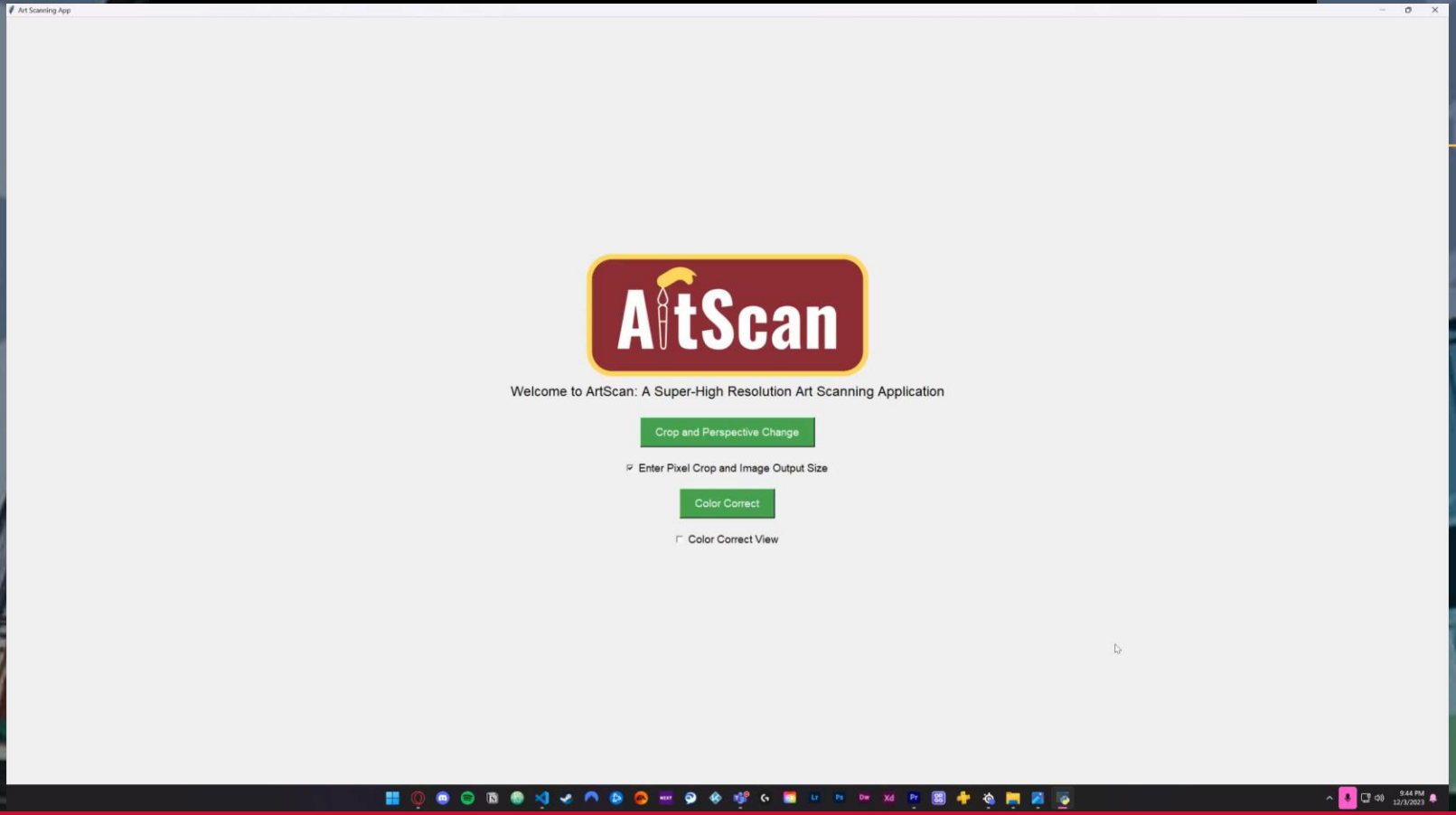


Output

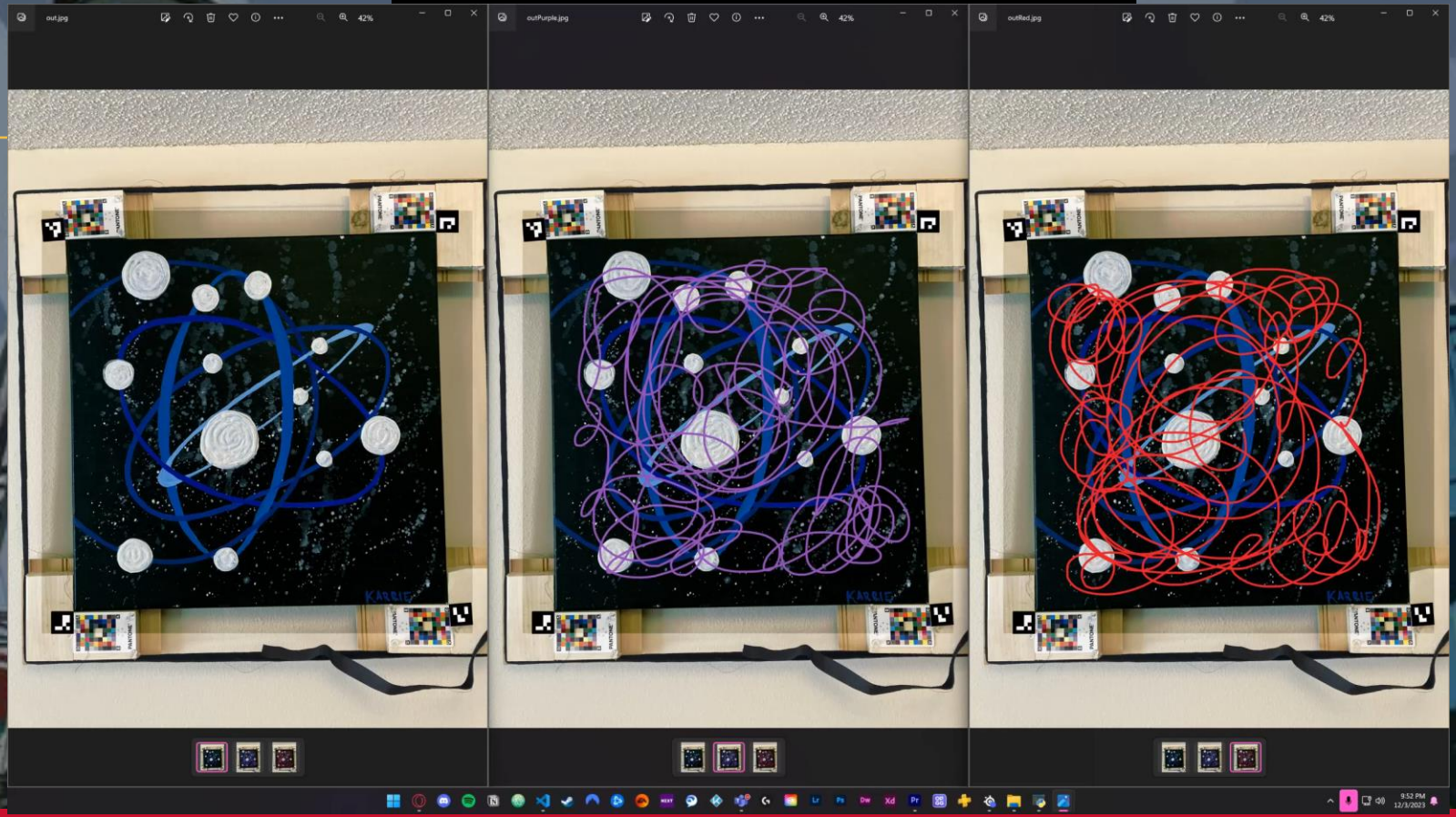
CHALLENGES WITH COLOR CORRECTION

- Program currently detects the Pantone cards at a rate of ~90%
  - Consistency could be increased by fine-tuning ArUco detection parameters
  - Images captured in a higher quality will increase number of pixels associated to the Pantone's ArUco markers, thus improving consistency
- Difficult to quantitatively determine accuracy of color correction
  - Accuracy largely determined by user preference
  - Reference Pantone card can be fine-tuned to the user's preference.
- Color correction takes ~4 minutes to process a single image
  - Process could be sped up by incorporating faster methods of cycling through each pixel

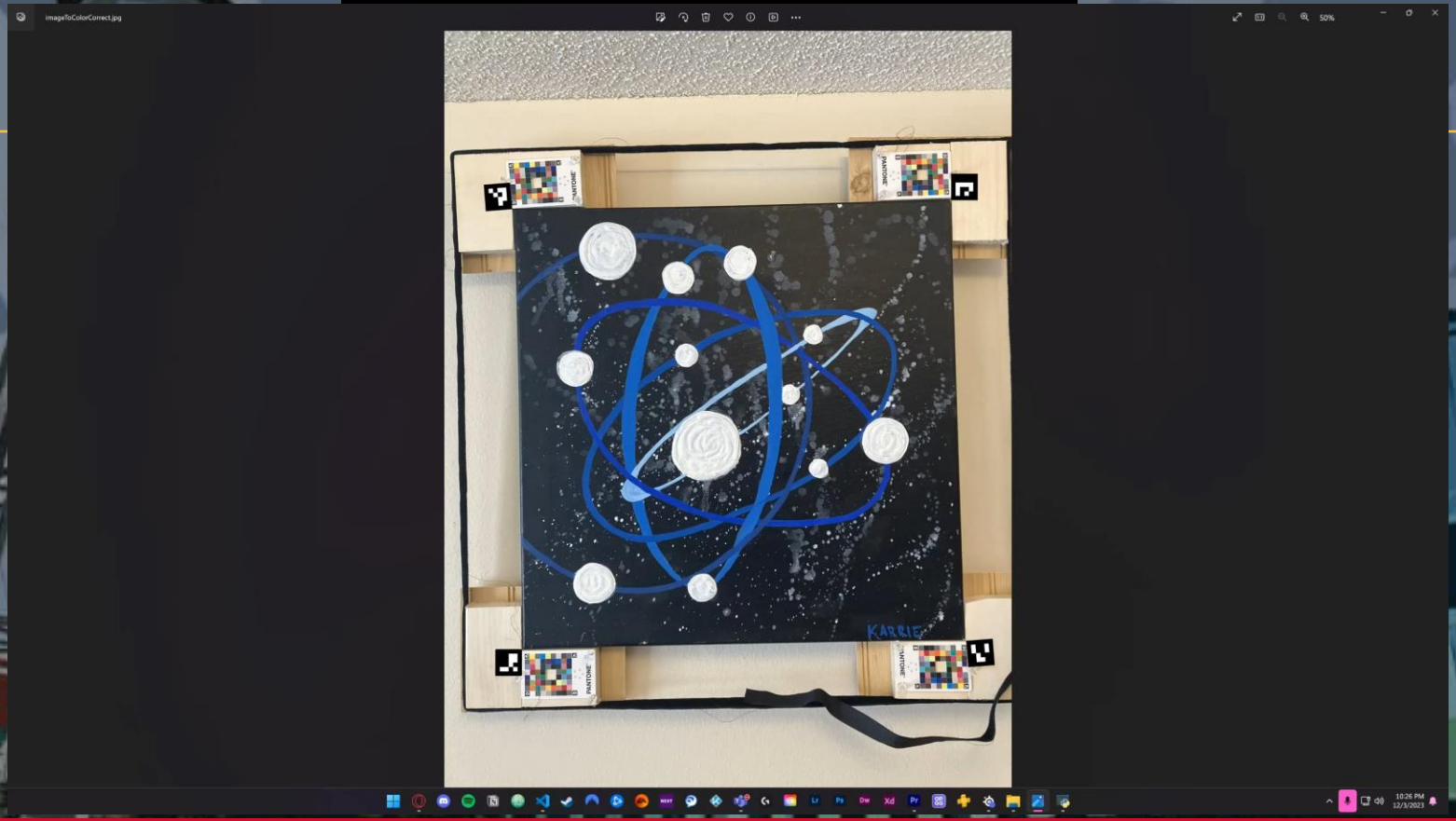
Demo of Crop and Perspective Change



Demo of Median Overlay



Demo of Color Correction



DESIGN CHALLENGES

- Had some issues with data quality and not being able to detect pantone cards
  - Pictures taken with older phone could have affected quality
- Believe better ArUcos can be printed on thicker paper
- Personnel backgrounds
  - Three EEs and one SE
  - Unfamiliar with Python and OpenCV

FUTURE IMPROVEMENTS

- Front-end application improvements
- Integration improvements for seamless automatic processing
- Refinements to functions to improve speed/consistency
  - Fine-tune processing parameters to further improve consistency
  - Speed up color correction
  - Utilize both ArUco markers and Hough transforms in border detection
- Add user confirmation to ensure results up to the user's standards

CONCLUSION

- Our program provides a convenient way to digitalize artwork
- Program utilizes Python and OpenCV to efficiently produce high resolution image
- Use of pantone color cards and ArUco markers allow program to color correct and border detect
- Wooden brackets and Velcro patches allow for collection of data on mounted canvases
- Program is intended to produce results in a quick efficient manner