StyleArm: a style-transferring robotic arm.

Vernon Stanley Albayeros Duarte

Abstract

As the paradigm of human-computer interaction shifts to increasingly intelligent systems that require no direct user inputs to provide services, the way we interact with our machines is still primarily "active interactions" as the computer requires direct user input to provide its content.

In this master's thesis, we create a physical proof of concept that implements computer vision algorithms to aid in interaction in the form of a self-built robotic arm with a camera that can interface with a Raspberry Pi, a small computer. This robot aims to provide the user with a way to interact with computer vision processes they might find helpful, like Style Transfer for story-telling and publication on social media accounts. The proof of concept utilizes a feedforward style transfer neural network in near real-time in its hardware, switching the context of the style transfer depending on the user's facial expression.

Index Terms

Computer Vision, GAN, Generative Adversarial Network, Style Transfer, Deep Learning, Neural Networks, Face Detection, Raspberry Pi, Robotics

I. Introduction

HUMAN-COMPUTER interaction has remained somewhat stagnant in terms of how we interact with our computers. Since the invention of modern operative systems, our "workflow" to interact with them has not changed significantly. This general workflow would be to use some input device, like a mouse or a touchscreen, to request a computer to provide some content. There have been alternative input methods proposed, but the overall paradigm has never changed. These alternative input methods, in our opinion, never reached full market potential because of one major issue: they require more work from the user or an alteration of their traditional workflow.

Microsoft's Hololens [1], one of the more recent proposals, is interesting because even though it requires a complete change in the user's interaction pattern with their device, it presents the user with intuitive interaction patterns in the form of "augmented reality" overlays on their field of view. The way this approach integrates use cases into a new "input space inspired this master's thesis.

In this Master's Thesis, we aim to produce a physical prototype slash a proof-of-concept that integrates a few computer vision use cases for easy use by an average user. This project builds and improves upon the author's own Bachelor's final thesis project [2].

This robot aims to provide a simple gateway for any user to take and save a picture after a Style Transfer neural network has treated it for story-telling and publication on social media accounts. The robot can function as a picture frame and takes its pictures for the user whenever it is not in use (and has been allowed to). An emotion recognition system provides an extra layer of interaction to provide relevant styles to the photographs and displays the results in real-time on an attached display.

Neural style transfer (NST) methods manipulate images or videos to adapt to the visual characteristics of another image. NSTs are most commonly implemented through deep neural networks. The first publication using neural networks to perform NST was the work of Leon Gatys et al. [3] in 2015. This paper used a VGG19 architecture pre-trained on the ImageNet dataset.

Just a year earlier, in 2014, Ian Goodfellow's groundbreaking Generative Adversarial Networks paper [4], proposed a new framework for estimating generative models using two, simpler models. Goodfellow's team was looking for an alternative to the, at the time, state-of-the-art undirected graphical models with latent variables such as restricted Boltzmann Machines [5], [6]. The main problem with this approach was that it was intractable for all but the most simple problems. The alternatives that did not involve defining a probability distribution explicitly, meaning that they could be trained by back-propagation, were in the form of generative stochastic networks [7]. This approach, however, still required Markov chains for sampling. Goodfellow's proposed Generative Adversarial Networks (GANs from now on) does away with the need for Markov chains for sampling, and because GANs do not require feedback loops during generation, they can leverage piecewise linear units and improve the backprop performance.

This "family" of networks has, over time, proven useful for NST. *Style transfer* is a field that manipulates images or videos in order to apply the "style" or appearance of another image or video. Using GANs for style transfer was first explored by Tero Karras et al. in their paper: "A Style-Based Generator Architecture for Generative Adversarial Networks" [8]. An example of style transfer can be seen in Figure 1.

Author: Vernon Stanley Albayeros Duarte, stanley.albayeros@gmail.com
Advisor: Fernando Vilariño, Computer Vision Centre, Universitat Autònoma de Barcelona
Thesis dissertation submitted: September 2021

While GANs are on the bleeding edge regarding performance with NST, use cases with reduced computational capabilities might not take full advantage of them.

[h]

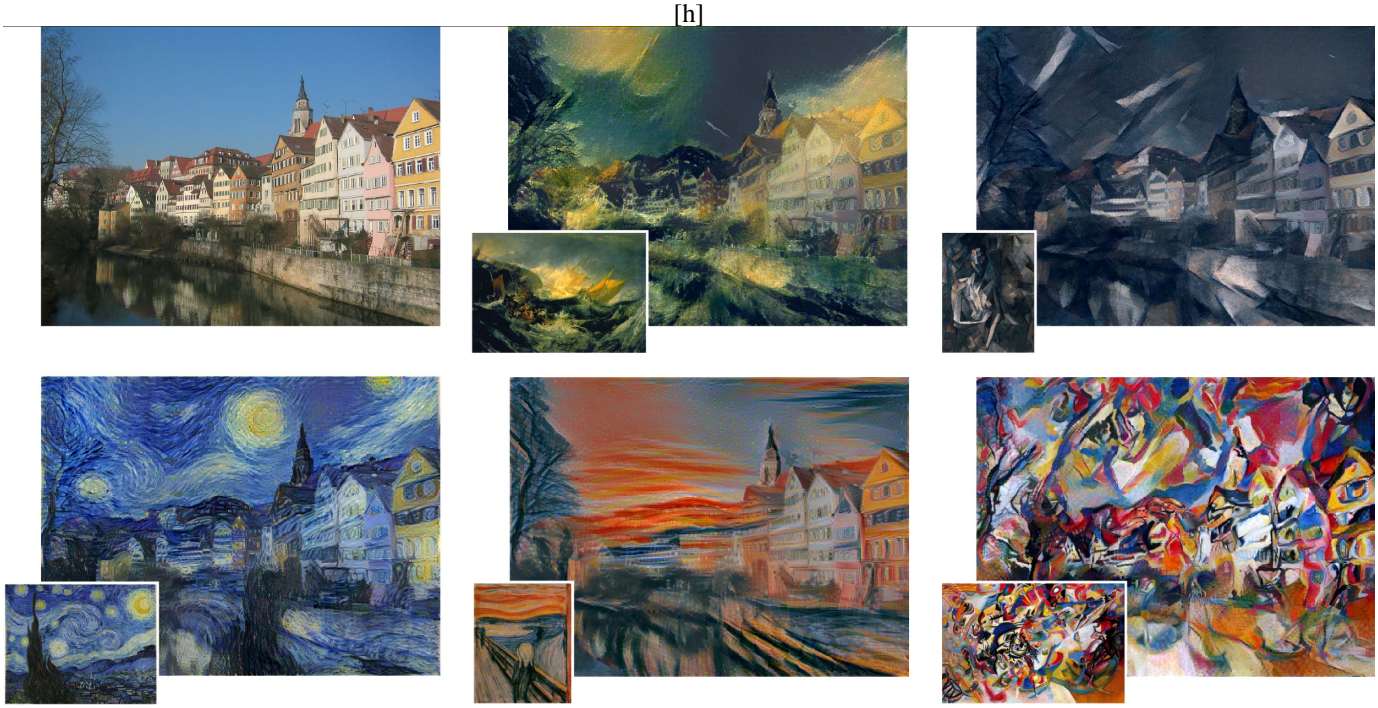


Fig. 1: Style transfer examples

II. Generative Adversarial Networks

Goodfellow's approach was to use two models to solve the problem. The "generator" network generates candidates in the form of randomly sampled noise from true data distribution, while the "discriminator" network evaluates these samples. These two networks play a min-max game where the generator tries to generate "fake" samples that it thinks the discriminator will think are real, and "real" samples that it thinks the discriminator will think are fake, while the discriminator tries to "beat" the generator network by correctly classifying the incoming samples. Over the years after Goodfellow's original GAN paper, the performance of these networks has improved quite a bit. Alec Radford's DCGANs [9] proposed the use of convolutional networks in place of fully connected networks, which improved the results and performance of the models. Further improvements for implementing GANs on large scale images such as Andrew Brock's 2018 [10] paper on the subject, further proved that GANs could be used for larger resolution images.

III. StyleGAN

Tero Karras from NVIDIA proposed in 2018 an alternative GAN architecture to implement a style-transfer network. [11]

While the latent space code is provided to the generator network through its input layer, Karras omits it and starts from a pre-learned constant.

NVIDIA's team provides their novel generator with a direct means to generate stochastic detail using explicit noise inputs, as single-channel images of uncorrelated Gaussian noise. These noise images are broadcasted to all feature maps using the learned per feature scaling factors and then added again to the output of their respective convolutions.

Karras et al. had terrific results using this approach, as seen in Figure 2.

[h] 

Fig. 2: StyleGAN results

In 2019, NVIDIA's team set out to improve StyleGAN by targeting several of it's characteristic artifacts [11]. In this paper, the team sets out to redesign the generator normalization and regularize the generator to encourage good conditioning in the mapping from latent codes to images.

To remove normalization artifacts, NVIDIA's team identifies an AdaIN operation that normalizes the mean and variance of every feature map separately, destroying any information found in the magnitudes of the features relative to each other. They instead remove the normalization step from the generator, causing the droplet artifacts to disappear.

On the other hand, the generator's architecture is revisited completely. The original StyleGAN applies bias and noise within the style block. This modification obtains more predictable results, moving these operations outside the style block and operating on normalized data. They also remove bias, noise, and normalization to the constant input without any observable drawback in performance or quality.

A. *StyleGAN2*

Later in 2020, Karras' team releases the "StyleGAN2" paper [12]. This time around, Karras and his team set out to improve their StyleGAN to work with limited amounts of data. They propose implementing an adaptive discriminator augmentation mechanism that stabilizes training when the dataset used is small. They demonstrate that results are not greatly affected even without modifying the loss functions or the architecture of the network.

IV. Method

In this section, we will describe the methods used for face and emotion recognition. We also discuss the datasets and propose the methods we will use to make use of them. We propose these methods considering that we need to achieve a minimum viable product for our robot. With this project, we aim to produce a physical prototype, so we have to consider that experimental results will impact the final implementation of the features and how much of a compromise between quality and performance we are willing to make.

A. *FERPlus Dataset*

The FERPlus dataset, published by Barsoum et al. [13], is an improvement over the previous FER [14] dataset published for a *Kaggle.com* research competition.

FER, prepared by Pierre-Luc Carrier and Aaron Courville. Each image measuring 48 pixels wide and high the dataset consists of 28k images. The dataset annotations consist of numeric codes ranging from 0 to 6 for the emotion present in

the image file: *(0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).* The FERPlus dataset improves the previous FER dataset by re-tagging the dataset manually3. FERPlus adds more classes, bringing it up to 8 emotions total. Each image has been labeled by ten independent "taggers" (humans), which means that each face image has a distribution of emotions.

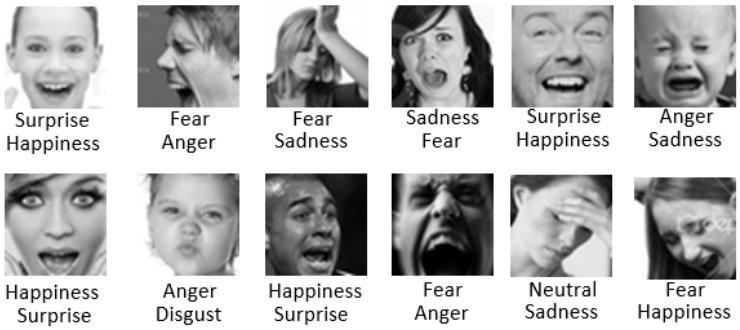


Fig. 3: Comparison of FER and FERplus labels. Image obtained from the FERPlus paper [13].

B. COCO 2014 dataset

Microsoft released their COCO 2014 training dataset as part of a more extensive publication containing training, validation, and test images. Totaling 328K images, it was the largest and most curated dataset at its publication. The dataset contains 80 different classes and multiple captions per image.

C. Target styles for style transfer

Fig. 4: Target Styles. The corresponding assigned emotions, from left to right: Anger, Sadness, Fear, Happiness, Surprise.

We selected five different paintings to be our target styles an assign them an emotion:

- *The Scream* By Edvard Munch[1] for the "anger" emotion,
- *The Starry Night* By Vincent Van Gogh[2] for the "Sadness" emotion,
- *La Musa (Mujer Leyendo)* By Pablo Picasso[3] for the "Fear" emotion,
- *Feathers* By Kathryn Corlett[4] for the "Hapiness" emotion,
- *June tree* By Natasha Wescoat[5] for the "Surprise" emotion.

These particular paintings were selected for various reasons. The first, is that some of them are widely used in other style transfer works, so we could compare qualitative results while we were developing them, like *The scream* or *The Starry Night*. We also found some target styles interesting enough to include them, like *Feathers* and *June tree*.

The emotions assigned to each style were not pre-selected, meaning we did not select a particular style to portray a particular emotion, as the final style transfer image would sometimes not correspond to our pre-selected style. With *The scream*, for instance, logic would have it that it would be selected to represent surprise or fear as that seems to be the motif of the painting, but in our results, we think the black and red hues and aggressive brushstrokes of the style-transferred image confer a more angry emotion rather than the other two.

[1]From: https://www.edvardmunch.org/the-scream.jsp
[2]From: https://www.vangoghgallery.com/painting/starry-night.html
[3]From: https://www.slobidka.com/pablo-picasso/158-picasso-la-musa-mujer-leyendo.html
[4]from: https://www.ebsqart.com/Artist/Natasha-Wescoat/1439/Art-Portfolio/June-Tree/717313/
[5]From: http://kathryncorlett.blogspot.com/2012/05/wallpaper-feathers-petals-leaves.html

D. Methodology

To achieve an MVP (Minimum Viable Product) and be worthy of an MSc project, we set our goals as the following:

- Remove the need to use Google's Cloud API for all computer vision tasks, implementing local solutions.
- Implement new routines involving engaging computer vision algorithms like Style Transfer.
- The user should easily be able to obtain the style transferred files.
- The visual representation on the screen should be visually pleasing enough that users will want to keep it on as a background element.
- Make the robot completely autonomous: remove the need to offload computation to a 3rd party service or a local machine.
- Achieve a real-time representation of the robot's image processing.

E. Face detection

Previously, our pipeline for the robot required sending images to Google's Cloud Services (GCS) for emotion recognition. As we have removed GCS, this is no longer the case. To replace the previous face detection and emotion detection parts of the pipeline, we need to implement local face detection and emotion detection methods based on the current state of the art. Face detection allows the arm to follow a subject should it approach the camera's viewing angle limits. We propose using a Haar-Cascade classifier to detect faces.

F. Emotion recognition

Detecting faces in the frame also serves to improve emotion detection. As described in Gunwan et al. [15], a neural network trained on cropped faces performs optimally on similarly cropped images. We propose training a network on the FERPlus [13] dataset.

G. Style transfer method proposal

We will use emotion recognition to change the context of the style transfer network, loosely adapting the target style transfer to a user's visible emotional status.

We propose a local implementation of a neural network capable of producing style transfer images as close to real-time on the Raspberry Pi 4's hardware as possible. The models can be pre-trained in a more powerful environment, but inference should run entirely locally on the device. This way, the robot does not need any external connections or dependencies in run-time. We propose experimenting with two different architectures: a recent model that requires more potent hardware but has better performance (results) and an older model that does not require high-end hardware, producing worse results but potentially giving us faster results and allowing the robot to run inference locally. Thus, we propose adapting Karras' StyleGAN2 representing the "state of the art" method, and since we are already using VGG16 and VGG19 models for our emotion recognition, adapting the method proposed by Justin Johnson et al. on their *"Perceptual Losses for Real-Time Style Transfer and Super-Resolution"* [16] publication representing the "old but fast" method to run on our hardware. While StyleGAN2 was trained using the FFHQ dataset [17], we propose using Microsoft's COCO 2014 training dataset [18] to train our style transfer models, as we are not using only face images on our style transfer.

If one of these two methods can run locally while achieving a good framerate, it should be strongly considered for the final implementation, as it would complete our final two objectives for an MVP.

H. Combining everything to form a product

We also need a way to get our style transferred images from the robot to the user. We propose using the Telegram Python API, implementing a Telegram bot that users can enable to receive their images on their mobile devices.

When we have all of our methods implemented, we will need a way to create a cohesive user experience through our robot. We need to design and implement a state machine that should conform to the following process:

1) User powers the robot on.
2) Robot searches for faces.
3) When a face is found, recognize emotion and perform live style transfer.
4) Whenever the user requests it, provide them with the style-transferred image file.
5) Whenever the face is lost, go back to state 2)

V. Physical prototype description

In this section, we will describe everything about the hardware side of this project.

A. Software package

The following different pieces of software were used to create this prototype:

- Raspbian OS 32-bit [19]: Linux distribution used in the Raspberry Pi.
- Fritzing [20]: Software package used to generate the connection graphics for the electronics components.
- Autodesk Fusion 360 [21]: CAD program used to design printable parts needed for the robot.

B. Hardware used

The following list contains all the hardware we used for our robot:

- **SG-90 servomotors**: chosen for their size and respectable 1.3 Kg-cm torque.
- **Raspberry Pi 4B 8GB**: chosen as the main controller for the robot. The extra RAM memory is needed as the internal GPU shares memory with the CPU.
- **PCA9685 12-Channel PWM Driver**: The Raspberry Pi only has a few hardware PWM enabled pins, with only two PWM channels available. Several software-based solutions use more pins as PWM-enabled pins with more channels, but we found these to negatively affect the robot's performance, and in any case, powering the servos becomes cumbersome. As a result, we used an I2C based PWM generator, the PCA9685 12-Channel PWM Driver. This component can be connected to the Raspberry Pi through the I2C interface, does not rely on the Pi's CPU to accurately generate the pulses needed to drive our three motors, and has the added benefit of having its power lines for the motors.
- **3.3v 1S LiPo Battery + Adafruit Powerboost 1000C**: Step-up voltage regulator with undervoltage protection for our LiPo battery. This will be used to power the servomotors.
- **Qualcomm Quickcharge 3.0/USB Power delivery capable USB battery**: This battery will be used to power the Raspberry Pi. Either QC3.0 or USB PD is needed to provide 3A at 5V of power to the Raspberry Pi.
- **7 Inch HDMI Display**: This display allows us to show users a live style transfer of the camera feed.

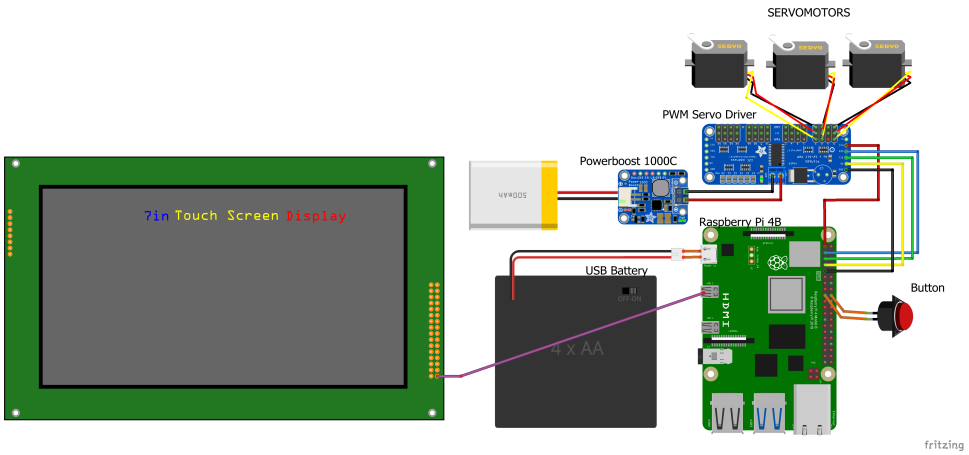


Fig. 5: Connection diagram of the final hardware configuration.

C. 3D printed parts

We redesigned the previous robot completely, as some of the STL files used to print the parts had errors. We took advantage of the occasion to bake in some extra features like consideration for a small USB battery and lipo battery to stop relying on outlets, places to route the cables internally, a button to interact with the robot, and a redesigned that allows for a small fan to be placed above the Raspberry Pi 4B's CPU heatsink.

In Figure 6 you can see the final render of the 3D model with mockups of the hardware and the labels for the components inside of the robot.

VI. Experiments

In the next section, we present methods which we believe are optimal for the proper implementation of an MVP. We will summarize the most significant experiments we ran and hint at the results produced. However, we will discuss the quantitative and qualitative results in the results section of this paper.

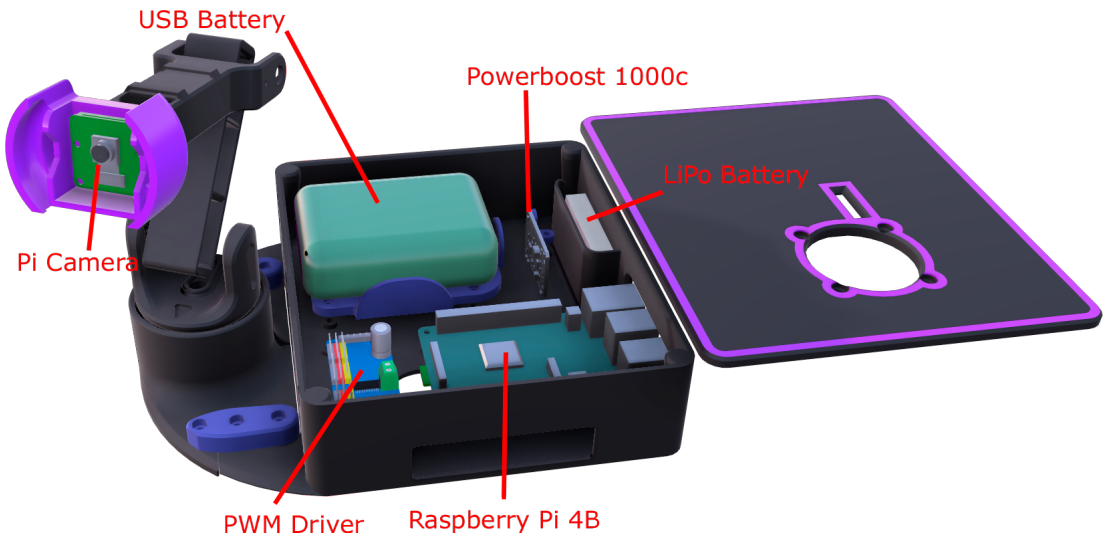


Fig. 6: 3D model with components.

A. *Face and emotion detection*

We used a traditional Haar-Cascade classifier to detect faces within the camera's frame. To test the accuracy and detection rate, we recorded five short videos, 400 frames in length each. We used these frames to test the time it took the classifier to produce a result.

The features we are looking for in our frames are both faces and eye features for two reasons. We want to keep a person within the camera's frame, and we also want to pass a cropped photo of the person to the emotion detector, which performs best when both eyes are visible.

Regarding emotion detection, we used the CNN provided by Octavio Arriaga [22]. In their publication, they report 66% accuracy on the FER dataset on their "sequential fully-CNN."

When training this on FERPlus, we simplified the number of classes to 5 as we found that the robot's behavior was too erratic with more classes. Our chosen emotions are happiness, surprise, fear, sadness, and anger.

We measure the performance of our face and emotion detection algorithms by how much time it takes for them to produce a result, as each passing frame affects the quality of the final product.

B. *Style Transfer*

We started by taking Karras' team StyleGAN2 network description and adapting it to run on the Raspberry Pi 4s hardware.

The purpose of using Karras' model was to minimize the requirements for inference of a trained model following their architecture. Our logic was that by minimizing the model's footprint, the inference would be fast enough to complete in real-time for our robot. In our early testing, reducing the input images to even below 100 x 100 px produced a model with an inference time of around 30 seconds per frame on the Raspberry Pi hardware. We decided that the optimizations required to make this model work at a more reasonable speed were outside of our expertise, as we quickly realized that it required a deeper understanding of the Raspberry Pi's hardware to make use of platform-specific instructions.

We moved on to implementing Justin Johnson's solution on our platform. Jhonson's original publication used Lua with Torch packages to implement a feedforward neural network to improve the speed of the original Gatys et al. paper's results. This implementation takes around 50 milliseconds per frame on a 1200 x 630-pixel image, which seems promising.

Jhonson's team initially implemented their network for Lua with Luarocks and Torch packages. We translated Jhonson's network architecture to Python with Pytorch 1.9 as close to the original architecture as possible. We noticed that, as Jhonson points out in his publication, changing instance normalization in place of batch normalization has a very sizeable impact on speed performance at the cost of very minimal visual performance. We call this network "PiStyle", as it allowed us to implement style transfer in near real-time.

Using the PiStyle network for our implementation, we split the model's functioning into two parts: low-resolution mode and high-resolution mode. When the robot shows the style transfer "live" to the user, we use a low-resolution pre-trained model. This model outputs images at a 125 x 125-pixel resolution, which are upscaled using bicubic interpolation to our display's native resolution of 1024 x 600 pixels. The network switches to the high-resolution mode whenever the user requests an image. This mode is slower than the low-resolution mode but can output images at a 1024 x 600-pixel resolution. These images are then upscaled using bicubic interpolation to 2560x1440 pixels and saved.

C. Providing images for the user

With all of our methods implemented, we need a way to get images to the user. The Telegram Bot API provides us with a simple and effective way to manage a user's data, as we can design it to only ever work with the user that has physical access to the bot.

We implemented the necessary API calls to store the user's chat ID, which is needed to send messages through the Telegram API, and the API calls that send photos as messages to the user. Through the bot's interface on their Telegram app, they can choose any combination of photos they want to receive: original image, low-resolution style transfer, or high-resolution style transfer.

If the user does not want to use the Telegram bot, we save all images to an output folder accessible through the Raspberry Pi's storage card.

D. Robot state diagram: how it comes together

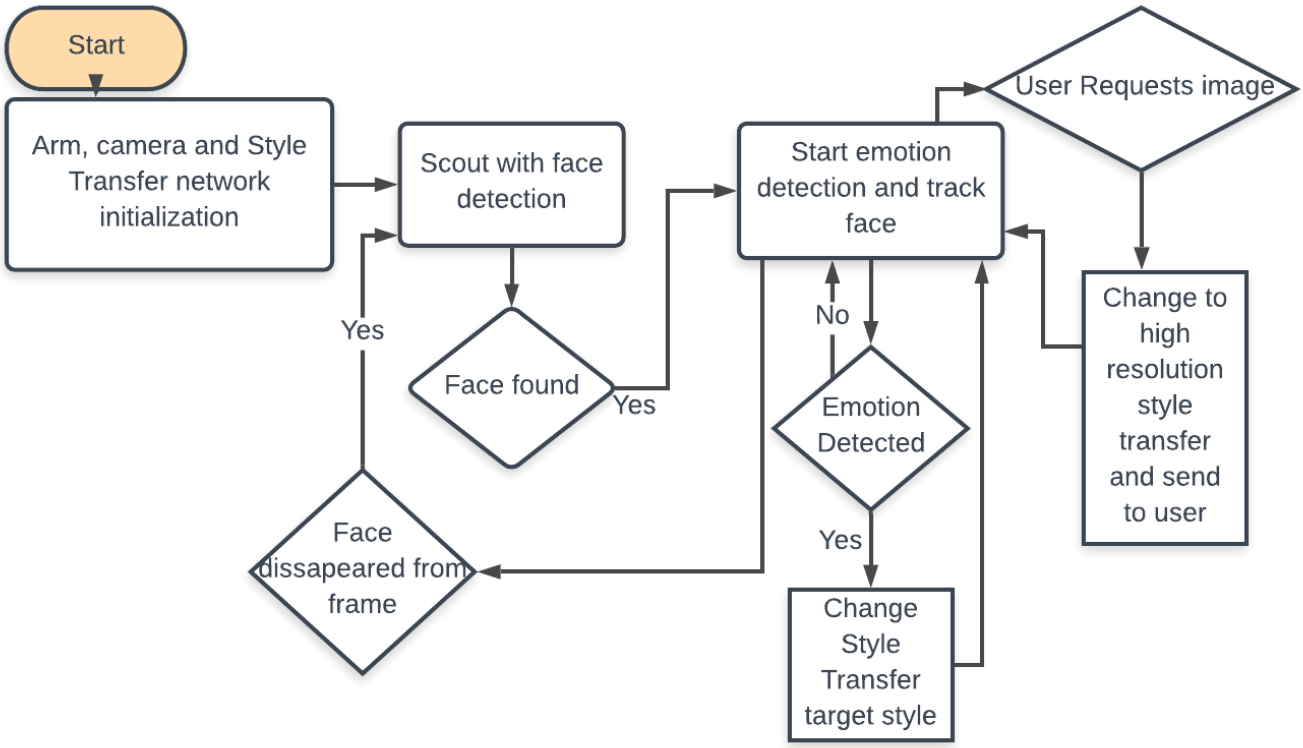


Fig. 7: Robot state machine diagram.

With all of our methods in place and working correctly, the only thing missing is the logic that connects them to form a product. In fig. 7 you will find the state machine that describes our implementation of the robot's operating process.

We tried to keep the diagram simple for clarity, so it requires a few clarifications.

If we follow an execution where a user starts the robot, walks into the field of view of the camera, and requests an image, these are the more in-depth descriptions of what each part of the process does:

1) **User powers the robot on:** A script starts when the Raspberry Pi turns on, switching to the python virtual environment that contains all the necessary libraries for our scripts to work. The camera feed is started. The neural network for style transfer is initialized in low-resolution mode with a default style transfer, and its output is routed to the HDMI display attached to the robot. The arm initializes the position of the servomotors on fixed coordinates, pointing the camera towards the front of the robot.
2) **Robot searches for faces:** The arm starts panning around its horizontal field of view in 10º degree increments until it reaches its movement limit to one side and starts panning on the same 10º degree increments until it reaches the other side. Each time the arm completes a 10º movement, it captures an image frame and runs face detection on it. If there is no face found, it starts the next movement.
3) **When a face is found, recognize emotion and perform live style transfer:** The face is cropped and fed into our emotion recognition script. The result of the emotion recognition engine is saved on a circular buffer of 5 frames, and

the script outputs the average of these five results. This average is used to select one of 5 different styles for the style transfer algorithm. Suppose there is a change in emotion from the previously set style. In that case, the neural network is reinitialized with the corresponding style's pre-trained model (2-5 second overhead, depending on the source and target model).

4) **Whenever the user requests it, provide them with a style transferred image file:** When the user uses the "capture" button, the last full-sized frame on the image is saved, along with its corresponding low-resolution style transfer. The camera feed is ignored throughout the following process. The full-sized frame is then fed to the high-resolution style transfer network. Then, the output is upscaled to 2560 x 1440 pixels using bicubic interpolation. All three files (original photo, low-resolution style transfer, high-resolution style transfer) are sent through the telegram bot to the user. After the robot sends the image, the user can cycle through the next high-resolution model for style, transferring the same original photo or going back to live-view, which reactivates the low-resolution style transfer mode.
5) **Whenever the face is lost, go back to state 2.**
6) **At any point:** The user can use a keypress for cycling through the available pre-trained models for the style transfer. If the user cycles to the next model, emotion recognition and style context switching are put on hold until they start it again.

VII. Results

In this section, we present the results obtained from our experiments and the performance benchmark, which was the time needed to generate a frame for live-view and the time it took for an image in the high-resolution mode to be processed and sent to the user. We also present qualitative results of the implemented style transferring network and a general overview of the final prototype's functioning.

A. Final physical prototype

We 3D printed all the necessary parts, made the necessary connections and assembled our robot. You can find the finished MVP in Figure 8.

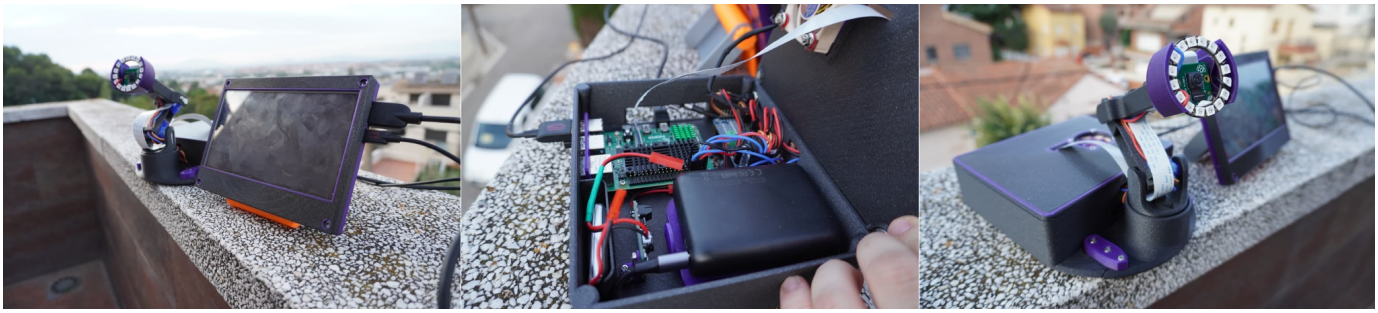
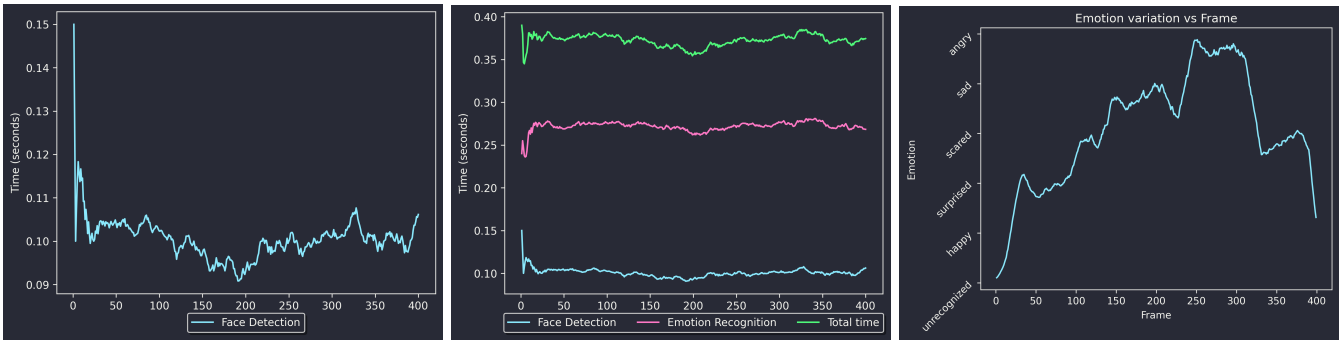


Fig. 8: Finalized assembled hardware.

B. Pre-processing: face and emotion detection



(a) Face detection time per frame.

(b) Face, emotion detection and total time per frame.

(c) Emotion variation by frame of a sample.

Fig. 9: Performance of face and emotion recognition.

Our final implementation of face detection can produce results at an average of 0.1 seconds per frame on the Raspberry Pi's hardware, as seen in Figure 9a.

Our overall performance almost matches the original "sequential fully-CNN" model's performance at 65% accuracy, but our implementation can complete the recognition of a cropped face's emotion in 0.28 seconds on average, as seen in figure 9b. We also captured a sample for the detected emotion variation for the same sample as our benchmarking run, as seen in figure 9c.

In Table I you can see the performance of the emotion recognition per emotion.

These frames vs feature detection figures show that our robot not only produces these results with low average time, but also that it does so in a very stable manner.

Emotion	Accuracy
Anger	68%
Sadness	67%
Fear	60%
Surprise	64%
Happiness	68%

TABLE I: Accuracy per emotion

We produced these results by running the network through a subset of images reserved for testing.

C. Style transfer

As discussed in the experiments section, we implemented StylePi using instance normalization instead of batch normalization. For the most part, this does not affect the quality of the images, but it does ocassionally generate interesting artifacts that produce a clearly different visual result as an effect of instance normalization, as seen in an example on Figure 10. We believe that this difference, or "pattern-ization" of the image is the product of how instance normalization works, as we can see in the same figure how the "swirling" pattern of Van Gogh's starry night repeats itself in a regular manner. It is not clear to the author under what conditions this happens, as all models have exhibited this behaviour at some point of the experimental phase.

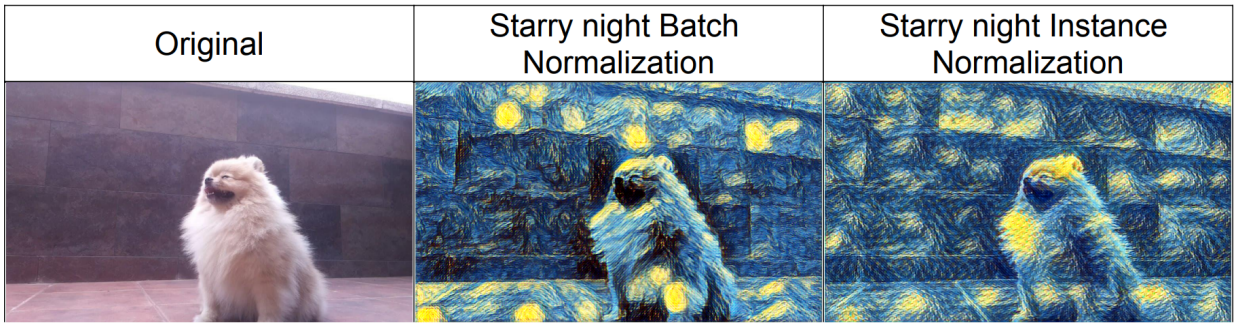


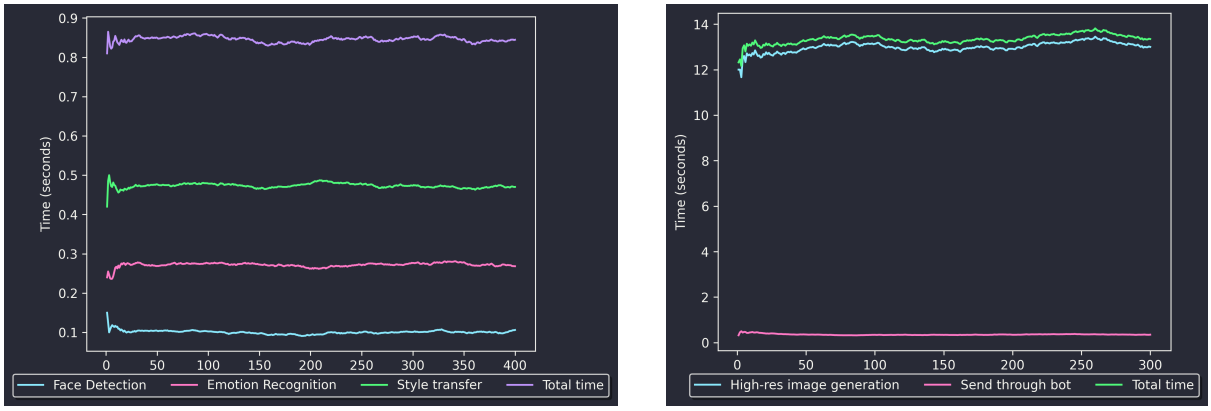
Fig. 10: Comparison of instance vs batch normalization

Ultimately, however, instance normalization was able to provide an average processing time of 0.47 seconds per frame over a 400 frame sample, as seen in Figure 11a.

D. Final MVP performance

In figure 12 you can see the available styles and their associated emotions for a sample image. This figure also includes the output of both modes of the StylePi network: a high resolution and a low-resolution output.

The user can request all images generated by the MVP through the provided Telegram bot. The average time needed to generate a high-resolution style transfer of the camera's captured frame and send it through the telegram bot to the user can be seen in Figure 11b. We found that selecting the minimum number of images (one) vs. selecting the maximum number of images (five: Original, Lowres, Lowres upscaled, Highres, Highres upscaled) did not impact the time it takes for the bot to send them.



(a) Performance of face/emotion detection and style transfer per frame.

(b) Total time needed to generate a high resolution image and send it through the bot.

Fig. 11: Final performance per frame and time needed to generate high-resolution images.

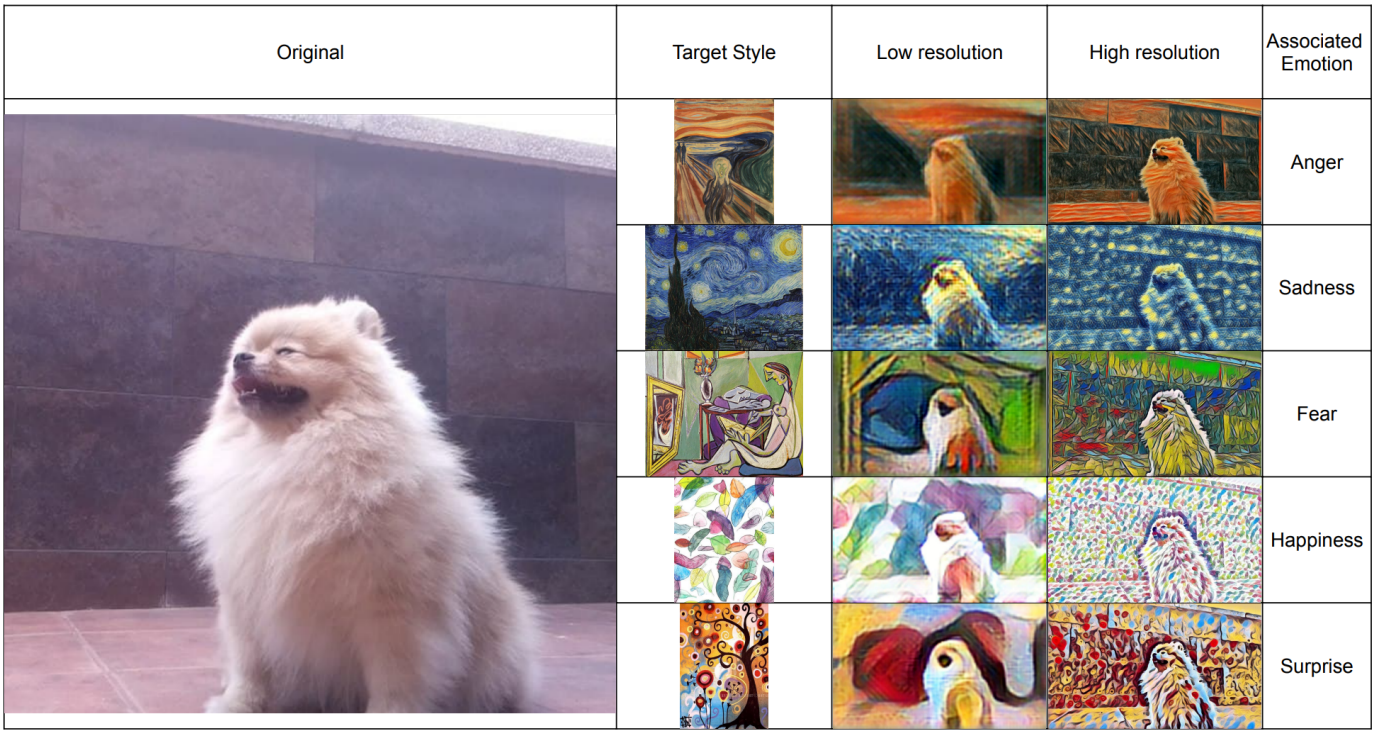


Fig. 12: Available style transfers for emotion context switching sample. From top to bottom: *The scream* By Edvard Munch, *The Starry Night* By Vincent Van Gogh, *La Musa (Mujer Leyendo)* By Pablo Picasso, *Feathers* By Kathryn Corlett, *June tree* By Natasha wescoat.

VIII. Conclusions

In this section, we discuss what we achieved and the conclusions of the project.

The initial goal of this thesis was to create a physical proof of concept that implements computer vision algorithms to aid in interaction in the form of a self-built robotic arm with a camera. This robot's purpose was to allow users to easily interact with computer vision processes like style transfer for story-telling and publication on social media.

In this Computer Vision Master's thesis, we have implemented several computer vision methods using machine learning and traditional methods on a physical prototype capable of functioning in real-time. We have produced a physical prototype at the MVP stage of development with all initial objectives accomplished. We have adapted face and emotion recognition that works in real-time on the Raspberry Pi's hardware.

Even though we failed at adapting a Generative Adversarial Network to run inference at a reasonably fast framerate on the Raspberry Pi's hardware, we finalized a style transfer feedforward neural network that works on it in real-time.

We have also integrated our methods into a cohesive software and hardware package that can serve results to the end-user in a fast and effective manner, without relying on external services to perform its primary functions.

IX. Future work

In this last section, we would like to introduce a short discussion about future improvements and the next steps needed to improve the quality of the MVP.

We believe that the StylePi network is capable of producing results at a faster rate. A deeper understanding of the Raspberry Pi architecture and the use of a 64-bit OS instead of the public, stable release of Raspbian OS 32-bit used in this project should allow for a modest boost in performance for both framerate and resolution of the images in the "low resolution" mode. Our feedforward architecture based on VGG19 is not the only method to implement a neural network that can work in real-time on this hardware. A deeper understanding of our target platform and more experience designing and training neural networks should also result in better performance for our robot.

We also wish we could have had implemented a way to record and style transfer a video using this prototype, but our implementation did not allow us to do so without turning off the live-view feature of the low-resolution style transfer mode.

After going through a few test cycles and having other users use the robot, we realized we had a few oversights in the design decisions for the interface. A Telegram bot and directly accessing the Raspberry Pi's storage should not be the only ways to access the produced images. We propose other methods with familiar third parties like Google and Apple account log-in, allowing users to upload the generated images to their services.

References

[1] Microsoft, 2021. [Online]. Available: https://www.microsoft.com/en-us/hololens/

[2] S. Albayeros Duarte, "Face-following robot arm with emotion detection," 2019. [Online]. Available: https://ddd.uab.cat/pub/tfg/2019/tfg-151968/Final.pdf

[3] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," 2015. [Online]. Available: http://arxiv.org/abs/1508.06576

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, p. 1527–1554, Jul. 2006. [Online]. Available: https://doi.org/10.1162/neco.2006.18.7.1527

[6] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, 1986.

[7] Y. Bengio, E. Thibodeau-Laufer, and J. Yosinski, "Deep generative stochastic networks trainable by backprop," *CoRR*, vol. abs/1306.1091, 2013. [Online]. Available: http://arxiv.org/abs/1306.1091

[8] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *CoRR*, vol. abs/1812.04948, 2018. [Online]. Available: http://arxiv.org/abs/1812.04948

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434

[10] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," *CoRR*, vol. abs/1809.11096, 2018. [Online]. Available: http://arxiv.org/abs/1809.11096

[11] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," *CoRR*, vol. abs/1912.04958, 2019. [Online]. Available: http://arxiv.org/abs/1912.04958

[12] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training generative adversarial networks with limited data," *CoRR*, vol. abs/2006.06676, 2020. [Online]. Available: https://arxiv.org/abs/2006.06676

[13] E. Barsoum, C. Zhang, C. Canton Ferrer, and Z. Zhang, "Training deep networks for facial expression recognition with crowd-sourced label distribution," in *ACM International Conference on Multimodal Interaction (ICMI)*, 2016.

[14] 2013. [Online]. Available: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

[15] T. Gunawan, A. Ashraf, B. Riza, E. Haryanto, R. Rosnelly, M. Kartiwi, and Z. Janin, "Development of video-based emotion recognition using deep learning with google colab," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 18, pp. 2463–2471, 10 2020.

[16] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European Conference on Computer Vision*, 2016.

[17] NVlabs, "Nvlabs/ffhq-dataset: Flickr-faces-hq dataset (ffhq)," Dec 2019. [Online]. Available: https://github.com/NVlabs/ffhq-dataset

[18] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[19] R. P. Foundation, "Raspbian os," https://www.raspbian.org/.

[20] Fritzing, "Fritzing." [Online]. Available: http://fritzing.org/home/

[21] Autodesk, "Fusion360." [Online]. Available: https://www.autodesk.com/products/fusion-360/overview

[22] O. Arriaga, M. Valdenegro-Toro, and P. Plöger, "Real-time convolutional neural networks for emotion and gender classification," *CoRR*, vol. abs/1710.07557, 2017. [Online]. Available: http://arxiv.org/abs/1710.07557

ANNEX

Alternate 3D view of the 3D model

Fig. 13: Top view.

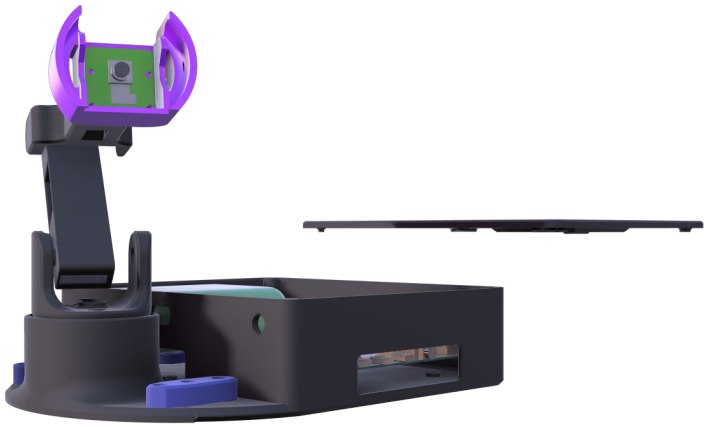


Fig. 14: Frontal view.

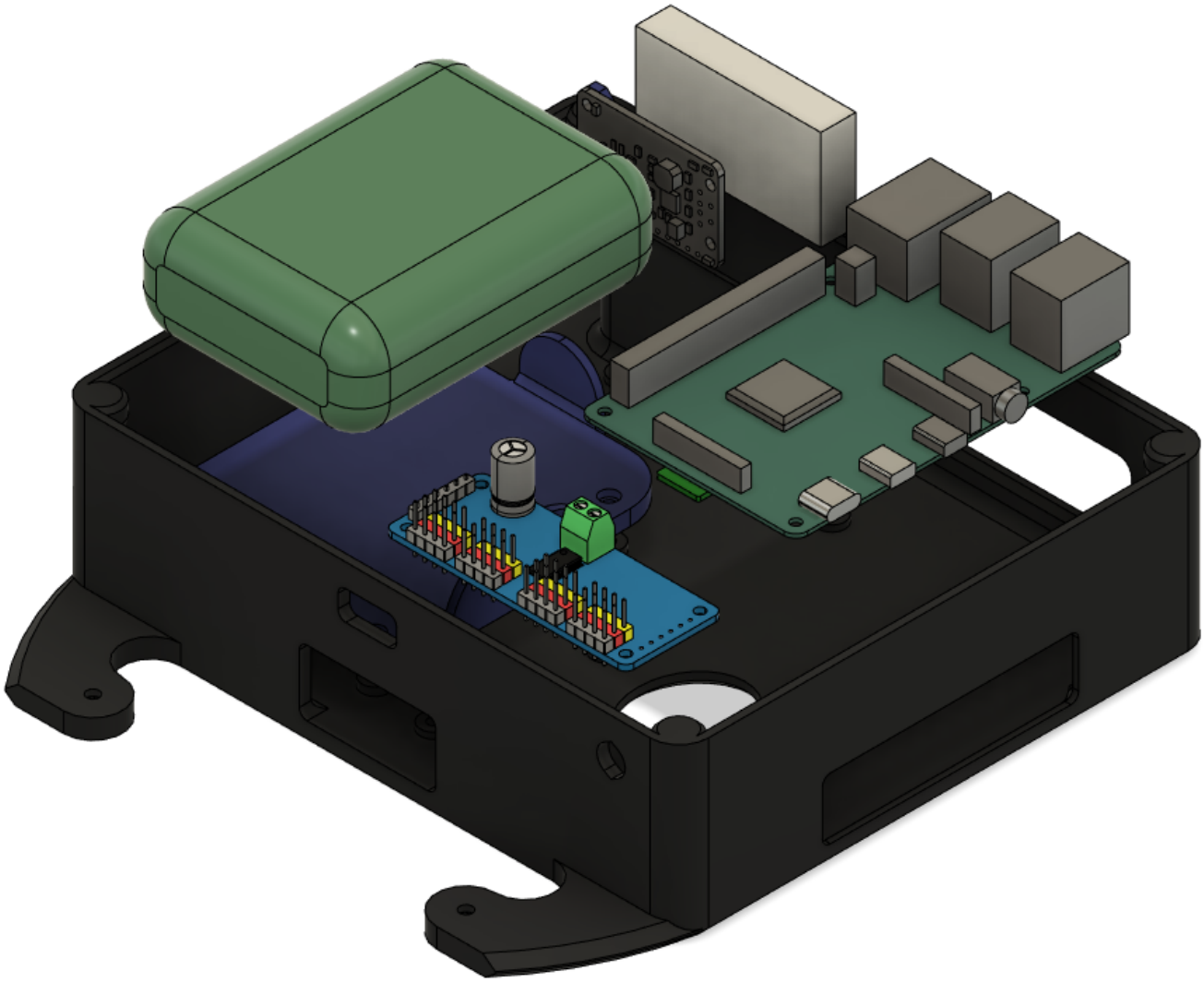


Fig. 15: Exploded view of the electronics.

Fig. 16: View of the arm assembly.

Fig. 17: LCD Display Case.

Fig. 18: Internal Wiring.