Sharing Without a Network Through Video

Anthony Wang
xy@mit.edu

Kevin Zhao
kez@mit.edu

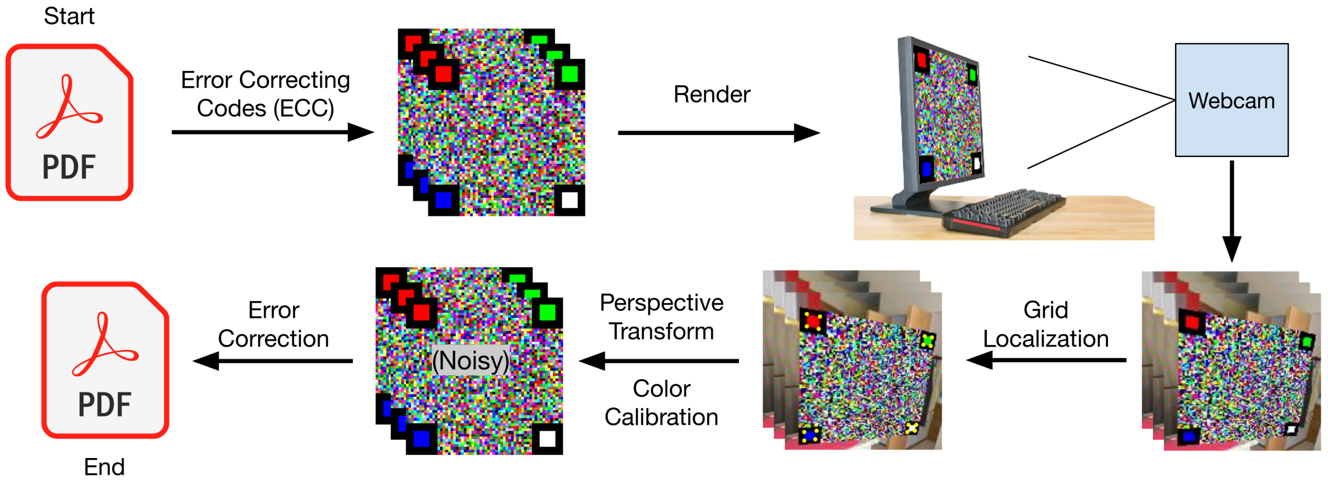

Figure 1. General overview: The sender encodes a file into a sequence of grids and displays these grids at 30 FPS on their screen. The receiver reads the displayed grid using their camera and in real time, recovers the original file by locating the grid corners and performing a perspective transform and color calibration.

Abstract

Wireless data transfer is crucial for many applications, but existing methods typically rely on a network. We present an alternative visual method for data transfer, SWANTV, inspired by QR codes, which displays a sequence of large colored grids on a screen and decodes it using a camera. The primary computer vision challenge is to consistently locate the grid in an image, and we develop both a CNN approach and an algorithmic approach to this task. To achieve high bandwidth, we address several hardware obstacles through methods like color calibration. Our evaluation shows that SWANTV can achieve a bandwidth of over 2 megabits per second, which is 80 times faster than prior work in visual data transfer and comparable to low-quality Wi-Fi speeds.

1. Introduction

Wireless data transmission is essential to many applications, but data transfer methods such as Wi-Fi typically rely on a network. These transfer methods fail when networks are inaccessible, which may occur due to factors such as remote location, server outages, and incompatible hardware. This shortcoming motivates a method of sharing data without a network.

Barcodes and QR codes are two visual methods for wireless data transmission that only require a screen and camera. However, barcodes can only encode short strings and QR codes have a maximum capacity of 2953 bytes [1], so neither method can transfer files larger than one page of text.

However, QR codes are still a good foundation for visual data transfer, and can be improved in three main ways:

1. Split large files into many frames, and display the frames in an animation.
2. Use a significantly larger grid size to encode more data in each frame.
3. Use multiple colors instead of simply black and white to further increase the capacity of each frame.

In this work, we propose *Sharing Without a Network Through Video* (SWANTV), a software pipeline for high-bandwidth data transfer through these enhanced QR codes. The primary computer vision challenge is to determine the location of the grid within an image with high precision, so we develop both a CNN approach and an algorithmic approach to this task. To achieve high bandwidth, we address several hardware obstacles through methods like color calibration. Our evaluation shows that SWANTV can achieve a bandwidth of over 2 megabits per second, which is 80 times faster than prior work in visual data transfer and comparable to low-quality Wi-Fi speeds.

2. Related Work

2.1. QR Code Extensions

Color QR codes have been extensively studied, such as in HCC2D [8] and HiQ [14]. However, individual QR codes still have a fixed capacity. To bypass this capacity limit, several projects such as TXQR [6] and QRTransfer [13] explored animated QR codes. However, to the best of our knowledge, existing animated QR codes can only reach a bandwidth of 25 kilobits per second (kbps), due to using on standard QR codes which have low information capacity.

2.2. Error Correcting Codes (ECC)

Despite their shortcomings, these existing methods employ useful techniques to detect and correct decoding errors. QR codes use Reed-Solomon error-correcting codes [11] to reconstruct the data even if parts of the image are corrupted. TXQR uses fountain codes [5] to divide the file into redundant frames so the file can be reconstructed after any sufficiently large subset of the frames is decoded.

2.3. Keypoint Detection

Harris [9] and FAST [12] are classical corner detectors. However, these hand-crafted features are may be less robust than simple CNNs. MOSSE [4] learn filters by applying Gaussian smoothing to the targets and minimizing the MSE loss, but a shallow CNN may capture more features than any single layer of filters. Existing CNN methods for keypoint detection [2, 15] are typically complex and do not meet our fast CPU inference demands. [7] do train a hardware-efficient CNN, but they require existing hand-crafted features for training.

3. Methods

The general task and approach is shown in Fig. 1. In short, the primary challenge is accurately transferring data through the inherently imperfect modality of vision. To address this challenge, we develop a robust computer vision pipeline with error correction.

3.1. Encoding

We use Reed-Solomon codes and fountain codes, specifically Raptor codes, to encode the data into a sequence of binary frames. Every three bits of the sequence are mapped to one of the 8 vertices of the RGB color space cube, and the corresponding color sequence is arranged into cells on a rectangular grid with dimensions ranging from 50 by 50 to 300 by 300.

To facilitate the decoding process, we mark the corners of each grid with a predetermined pattern, shown in Fig. 2. We use a different color in each corner for the color calibration step of decoding (Sec. 3.4).

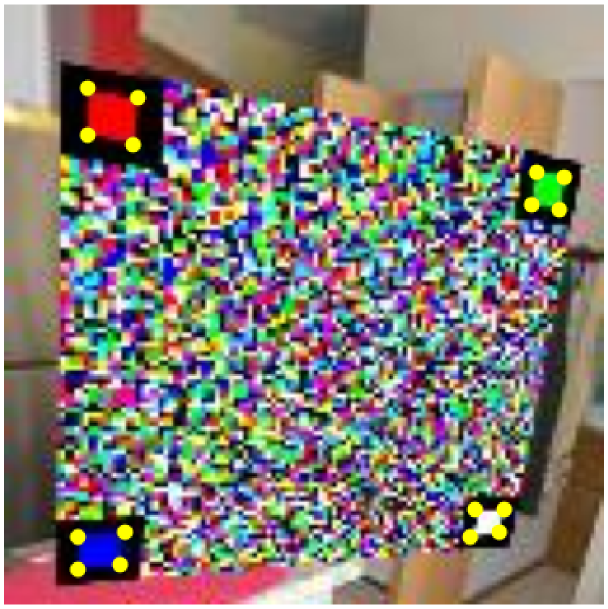

Figure 2. Simulated image and label (overlaid in yellow) pair.

3.2. Data

Our training data is artificially generated through a simulated environment. The grids are randomly initialized and overlaid on a random background sampled from Places365 [16] to approximate the real world backgrounds in the camera frame. The grid is then distorted through image augmentation techniques like perspective transform and color jittering. Labels of desired keypoint locations like corners can be maintained across the transformations. Fig. 2 shows one generated image-label pair.

3.3. Grid Localization

One of the main challenges is accurately localizing the grid, as the location, orientation, scale, and illumination of the grid will vary throughout the camera footage. As the grid is rectangular, we reduce this task to identifying its four corners, which is sufficient to determine the grid layout. We develop two methods for corner localization, both of which are two-stage approaches where the first stage makes coarse predictions of each corner location that get refined in the second stage.

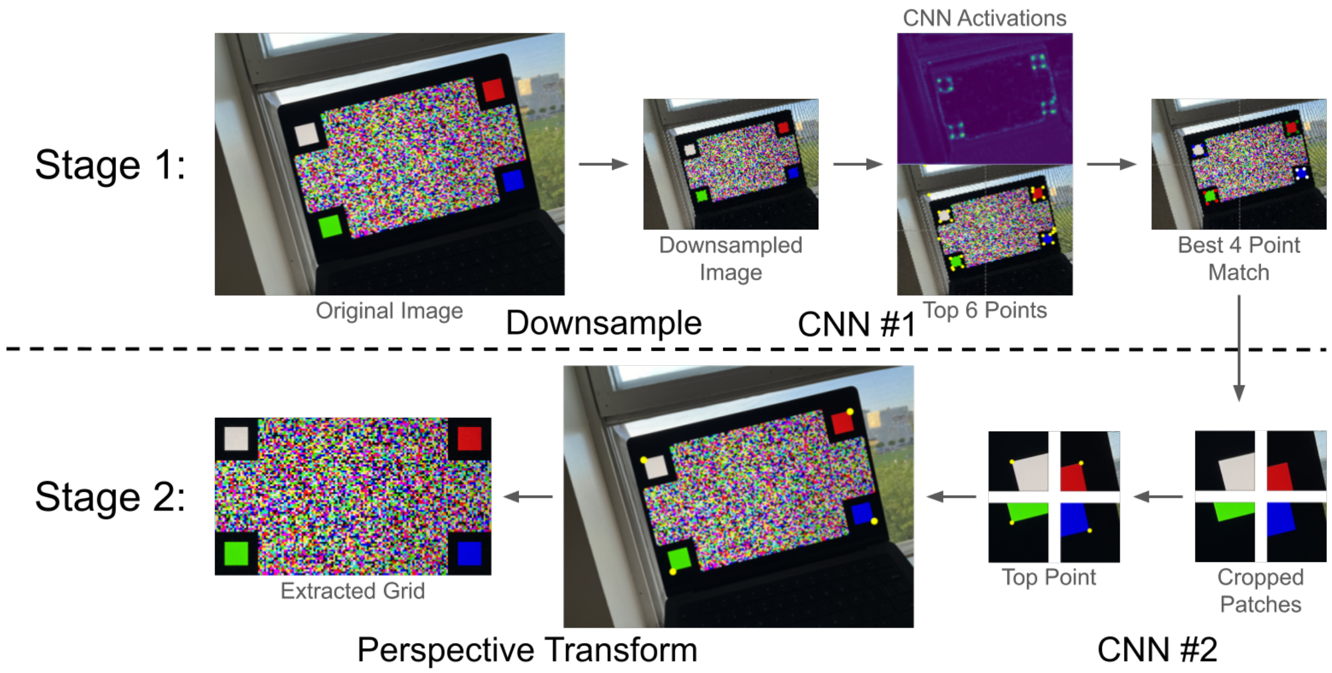

Figure 3. Overview of the two-stage CNN approach for extracting the grid from the webcam frame.

3.3.1 CNN Approach

In our two stage CNN approach, one CNN is first applied to a downsampled version of the webcam frame to obtain coarse predictions of the corner locations. Cropped patches around these regions are extracted from the original image, and the second-stage CNN leverages the high resolution of these patches to make finer-grained predictions. Fig. 3 shows each step of this approach.

As inference must be performed 30 times per second to process each frame in real time, we use shallow, quantized CNNs. Specifically, both our first and second-stage CNNs are composed of only three convolutional layers, with a maximum width of 32 filters.

We train the first-stage CNN to predict four keypoints per corner, for a total of 16 keypoints per grid, shown in Fig. 2, while the second-stage CNN is trained to predict the outer corner keypoint for each patch. Gaussian smoothing is applied to the training targets, similar to [4]. Our CNNs are trained to predict these smoothed labels by minimizing a weighted MSE, where nonzero targets are assigned greater weight to balance the relative abundance of zero targets which could otherwise drive all predictions towards 0. We additionally employ quantization aware training [10] to minimize the drop in performance from quantization.

We use various heuristics to convert the first-stage CNN's activations to probable corners. To identify the four keypoints per corner, we first assume that each corner is fully within one quadrant. Then, the top 8 locations[1] are extracted from each quadrant as potential keypoints. Because the true corner keypoints resemble a square, we enumerate all $\binom{8}{4}$ combinations of four points but only consider the combinations that form a convex quadrilateral with relatively equal diagonal lengths, and then select the combination with the least relative difference in side lengths to be the four corner keypoints. The outer corner is the corner keypoint farthest from the center of the image.

3.3.2 Algorithmic Approach

Alternatively, we develop a two-stage algorithmic approach to locate the corners of the grid, as shown in Fig. 4. The first stage downsamples the four corners of the camera image so that the smaller dimension is 8. This blurs the middle of the grid into shades of brown and gray, so we can locate three of the corners by simply by finding the pixels with the highest red, green, and blue values. For finding the white corner, we find the pixel with the maximum sum of its three color channels and minus twice the standard deviation of its three channels. The first stage returns a point inside each corner, so the second stage performs a flood fill on the image at the original resolution from those four points to determine all the pixels for each corner. We can take the average location

[1]To avoid double-counting keypoints from a single smoothed peak, we iteratively select the point with the maximum activation and suppress all neighbors by subtracting a Gaussian centered at the selected point, which is inspired by the soft NMS [3] used in object detection.

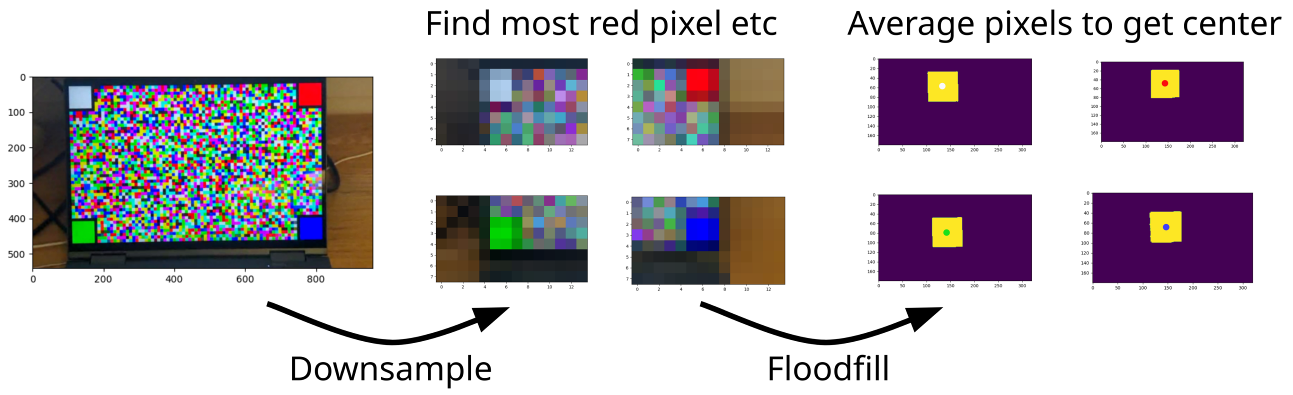

Figure 4. An overview of the algorithmic approach to corner-finding.

of the points in a corner to determine the center, and the average color of the points to determine the color of the corner.

3.4. Decoding and Color Calibration

Given the four corners and the grid dimensions, we can reconstruct the original grid using a perspective transform, assuming minimal camera distortion. However, the colors in the camera image do not perfectly match the original encoded colors due different lighting conditions and properties of the camera and monitor.

We mitigate this problem with color calibration, which leverages the four corner colors c_w, c_r, c_b, c_g to determine a new color space. We define the origin of this color space to be o, with the three basis vectors r, g, b. Thus, we obtain the system of equations $c_w = o + r + g + b, c_r = o + r, c_g = o + g, c_b = o + b$. We can solve this system of equations and transform any color c from the original image to our new color space using the equation

$$\begin{bmatrix} | & | & | \\ r & g & b \\ | & | & | \end{bmatrix}^{-1} (c - o).$$

Finally, we can determine the original color of each cell by rounding to the nearest color out of the 8 possible original colors.

3.5. Practical Considerations

The basic ideas of SWANTV are simple, but physical hardware is complicated and many tricks are required for high bandwidth. One recurrent obstacle is CPU overheating, since our pipeline is fairly compute-intensive. Overheating is particularly problematic for longer videos, so we used a powerful desktop computer instead of a laptop for those experiments. In addition, our camera is capable of recording at 4K resolution, but we choose a lower resolution for better quality and lower CPU usage.

Another problem is the slow refresh time of LCD monitors. We used a 60 Hz LCD monitor for testing, but the monitor displays a new frame line-by-line over the span of $\frac{1}{60}$ seconds instead of near-instantaneously. We displayed the SWANTV code at 30 FPS, so each frame is only fully shown on the monitor for $\frac{1}{60}$ seconds rather than $\frac{1}{30}$ seconds. Originally, we also ran camera at 30 FPS, but sometimes the camera captures an image while the monitor is in the process of displaying a new frame. The solution is to run the camera at 60 FPS and skip the next frame if the current frame was successfully decoded.

4. Results and Discussion

4.1. CNN Approach

Using our simulation method, we generated two datasets with 40,000 128x128 images each for for stage 1 and stage 2, where the two datasets differ in the scale of the grid – the entire grid is visible in stage 1 images, while only part of the corner is visible in stage 2.

CNNs for both stages were initially trained on a Satori V100 GPU to evaluate the performance of different model architectures, but the final quantizated models were trained on a Google Colab GPU due to the lack of any quantization engine on Satori. In all cases, we train for 10 epochs with a batch size of 256, learning rate of 0.01, Noam inverse learning rate scheduler with 30 warmup steps, and the AdamW optimizer. The overall inference time on 1920x1080 frames is under 30 ms on a laptop, so the CNN approach should theoretically be fast enough to decode in real time.

One complicated step of our CNN approach is the heuristics to identify the four keypoints for each corner. Instead of our method that finds the best combination among top $k = 8$ points, a more naive approach is to simply use the top 4 points as the corner keypoints, which is equivalent to our method with $k = 4$. To test the effectiveness of our method, we can vary the number k of top points to extract and observe the change in model performance. The results of this experiment are shown in Fig. 5. The model accuracy initially improves when increasing k because becomes more likely that all four true corner keypoints are in the top k predictions, but increasing k beyond 8 is detrimental to model performance because the extraneous keypoints increase the probability of identifying the wrong corner square.

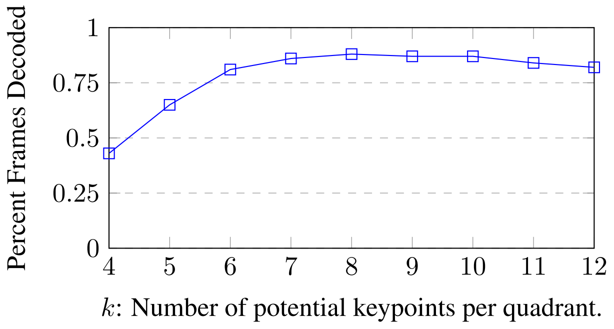

Figure 5. Trade-off between k and model accuracy on a random test video. Note that roughly 10% of the frames cannot be decoded due to inconsistent camera colors, even if the corner detection is correct.

Ultimately, due to simplicity and comparable performance offered by the algorithmic approach, our main results were achieved using the algorithmic corner localization approach, but similar results could be achieved in theory by the CNN approach.

4.2. Algorithmic Approach

We performed several experiments to maximize bandwidth using the algorithmic approach. We used typical consumer hardware for evaluating SWANTV, specifically a 1920x1080 60 FPS Google Pixel 6 camera and a 3840x2400 60 Hz LCD screen. We tested SWANTV in a variety of lighting conditions to demonstrate its robustness.

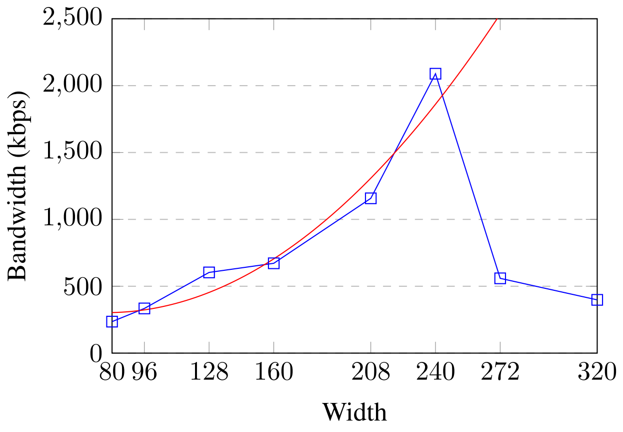

Figure 6. Bandwidth of SWANTV on different grid sizes.

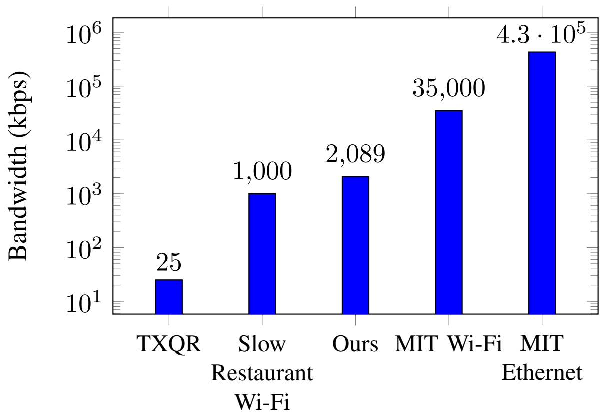

Figure 7. A comparison of SWANTV with other data transfer methods.

Fig. 6 shows that bandwidth roughly scales quadratically with the side length of the grid, as expected. On a grid size of 150 by 240 with an error correction level of 0.2, we achieved a bandwidth of 2089 kbps, which is comparable to low-quality Wi-Fi and fast enough to transfer the text of a book in one second. Fig. 7 compares SWANTV with other common data transfer methods.

4.3. Scaling Challenges

A natural approach for maximizing bandwidth is to increase the grid size. However, cameras have a limited resolution and cannot take clear images of grids above a certain size. For our camera, this maximum size was around 200 by 320. Another approach is to use more colors instead of just 8. We tried using 16 or 256 colors, but our camera could not accurately capture colors in large grids. Finally, we considered running the animated code at a higher frame rate, but that requires expensive hardware like 144 Hz monitors.

5. Conclusion

When we started this project, we were unsure if we could even surpass TXQR's bandwidth, but SWANTV has proven that visual data transfer in the range of Wi-Fi speeds is viable. Our pipeline is currently compute-intensive and unsuitable for real-world applications, but we focused on maximizing bandwidth rather than efficiency. Consequently, we believe that with more work, SWANTV can become a fast, reliable, and efficient method for transferring data between any two devices with a screen and camera.

Description of Contributions Kevin generated the simulated datasets, performed quantization-aware training of the CNNs, and ran the experiment shown in Fig. 5. Anthony developed the algorithmic approach and ran the experiments in Fig. 6 and Fig. 7.

Our code can be found at https://git.exozy.me/k/6.8301-Project.

References

[1] Information capacity and versions of the qr code. 1

[2] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Key. net: Keypoint detection by handcrafted and learned cnn filters. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5836–5844, 2019. 2

[3] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017. 3

[4] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2544–2550. IEEE, 2010. 2, 3

[5] John W Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review*, 28(4):56–67, 1998. 2

[6] Ivan Danyliuk. Transfer via qr, 2019. 2

[7] Paolo Di Febbo, Carlo Dal Mutto, Kinh Tieu, and Stefano Mattoccia. Kcnn: Extremely-efficient hardware keypoint detection with a compact convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 682–690, 2018. 2

[8] Antonio Grillo, Alessandro Lentini, Marco Querini, and Giuseppe F. Italiano. High capacity colored two dimensional codes. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 709–716, 2010. 2

[9] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, pages 10–5244. Citeseer, 1988. 2

[10] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 3

[11] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. 2

[12] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 430–443. Springer, 2006. 2

[13] Lihang Xu. Transfer data with animated qr codes, 2021. 2

[14] Zhibo Yang, Huanle Xu, Jianyuan Deng, Chen Change Loy, and Wing Cheong Lau. Robust and fast decoding of high-capacity color qr codes for mobile applications. *IEEE Transactions on Image Processing*, 27(12):6093–6108, 2018. 2

[15] Xiaoming Zhao, Xingming Wu, Jinyu Miao, Weihai Chen, Peter CY Chen, and Zhengguo Li. Alike: Accurate and lightweight keypoint detection and descriptor extraction. *IEEE Transactions on Multimedia*, 2022. 2

[16] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. *Advances in neural information processing systems*, 27, 2014. 2

A. “Bad Apple!!” Experiement

To showcase the power and robustness of SWANTV, we present here an application involving Japanese anime. “Bad Apple!!” is a 2008 Japanese song and grayscale 8 MB music video that achieved internet notoriety through being ported to unconventional hardware such as graphing calculators and washing machines. For our application, we encoded the original video of “Bad Apple!!” into a SWANTV code and overlaid the edges of the “Bad Apple!” video onto our code, taking advantage of the error correction during decoding. An example frame is shown in Fig. 8.

To extract the edges from “Bad Apple!!”, we simply used all pixels with values between 1 and 254 inclusive. However, Reed-Solomon codes are not as magical as they appear and operate at the byte level, so an incorrect bit ruins the entire byte.[2] In addition, a Reed-Solomon code with a k fraction of redundant bytes can only correct up to $\frac{k-1}{2}$ byte errors. Thus, we used the error correction rate $k = 0.4$, which is much higher than our other experiments. We used a 120 by 160 grid size, so the theoretical maximum bandwidth achievable is 1005 kbps.

In the experiment, we achieved 375.144 kbps, which means less than half of the frames were successfully decoded. However, this bandwidth is still high enough to decode “Bad Apple!!” in less time than the length of the original video, which implies that SWANTV is able to encode more information in each frame than in the original video. Also, it’s just insanely cool that this is possible.

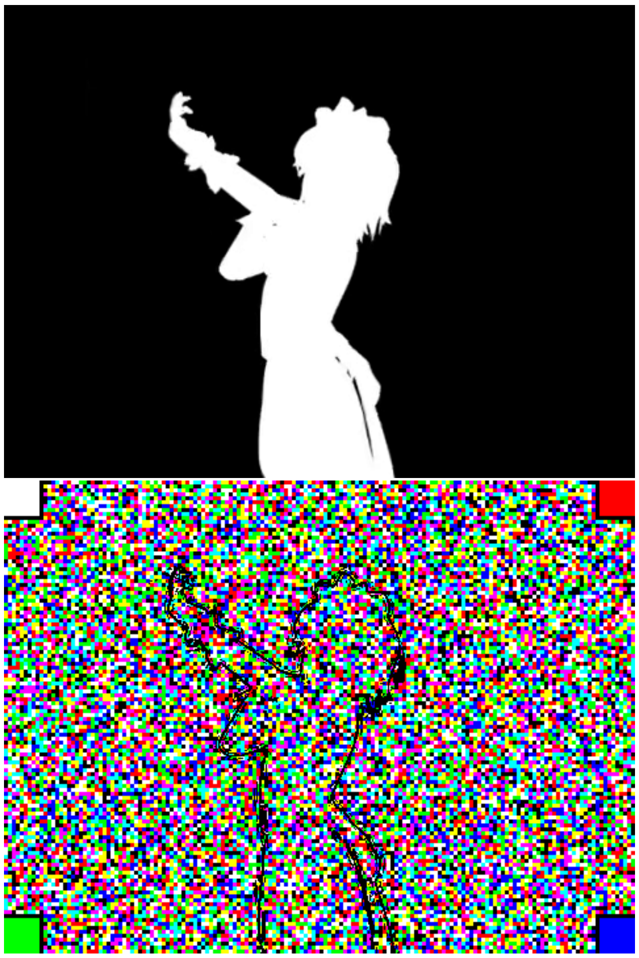

Figure 8. A frame from the original “Bad Apple!!” and the corresponding frame in the SWANTV code, featuring the character Sakuya Izayoi.

[2]Just like how the name of the song “Bad Apple!!” comes from the idiom “one bad apple can spoil the whole barrel.”