SmartWall

Gary Qian, Manyu Sharma, Sarah Sukardi, Tony Jiang

Johns Hopkins University, Department of Computer Science

December 15, 2016

Abstract

This paper presents Smartwall, a program that uses computer vision to turn any surface into an interactive board using a standard camera (such as a webcam) and a projector. The program uses a custom calibration matrix and perspective projection to enable object tracking, as well as employs deep learning for hand recognition and gestural board manipulation. Smartwall is an extremely accurate, cost-effective, and simple way to facilitate interactive teaching, brainstorming, and entertainment, at a fraction of the cost of other existing devices.

I. Introduction

Humans have drawn on surfaces for millenia. From primitive pre-historic cave paintings to 17th century frescoes to the chalkboards and whiteboards commonly used in schools and universities today, the usage of surfaces as conduits for brainstorming, depicting information, and even art, have made them long essential to processes of creativity and conveyance.

Current, modern approaches to drawing on surfaces suffer from either requiring physical, depletable media (whiteboards, chalkboards, pens, paint) or expensive equipment (modern-day smartboards, MSRP $1000 - $9000). This paper presents a cost-effective method to turn any wall into a drawable surface (and more generally, a surface on which one can execute full control of one's computer using only gestures) using only two pieces of equipment: a camera and a projector, where the projector can be substituted with any medium capable of displaying digital content (ie. televisions, monitors, etc.). The camera requirement can be filled by readily available webcams present on almost all modern laptops.

Our method is easily adaptable to spaces with unique constraints and requires equipment that most modern rooms already come equipped with, combining both traditional as well as state-of-the art computer vision techniques to allow for sophisticated and accurate gestural recognition.

II. Methods

We separate our discussion of methods into several subsections:

- Calibration
- Detection
- Training
- Recognition
- Output

i. Calibration

Figure 1: *Sample Setup with Camera and Projector*

For camera calibration, the camera used to track hand movement is set to point towards the wall. Note: we will proceed to use the phrase projector to represent any display medium from TV screens to actual projectors. The projector projects the content eventually to be controlled with a human hand in the same direction as the camera. A custom pattern of green dots is displayed onto the wall for the camera to record; this is the setup required for the camera calibration process to begin.

The pattern we found that worked the best was 9 green dots arranged in a x pattern as shown. This pattern results in full coverage of the extremities of the projected screen, ensuring capture of any strange warping effects. Each dot has a unique x position, which will be used later to sort and match the detected dots with their expected locations. Also, each dot's position and size is relative to the projector resolution, ensuring compatibility with a projector of any size and resolution. This method will work under the assumption that the projection surface will be flat and fully visible to the camera.

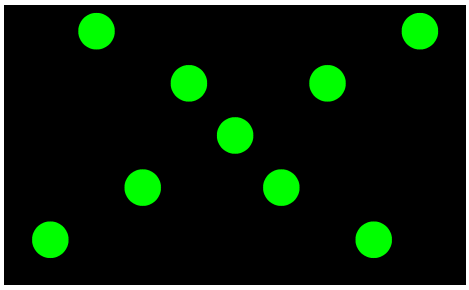

Figure 2: *Custom Calibration Matrix*

The camera then begins the process of calibration. Frames are captured in real-time from the camera. For each frame, a box blur with an 11-pixel diameter is applied to the image (this blur radius is manually adjusted and 11 seemed to work the best; the smaller the radius, the smaller the detectable objects are). Each frame is then converted to HSV color-space to detect hues on the screen. An HSV image can isolate certain colors without regard to the specific saturation or brightness of the patch. The range of color to be detected is then defined manually and thresholded to retrieve a binary image of only the desired colors, where 255 is green areas and 0 is everything else. For calibration, we used green, which was chosen because of the lack of green in skin tones, and the camera's increased green sensitivity due to the Bayer patterns of most cameras today. In the threshold values, we cut off colors that are too unsaturated because at low saturation, the color accuracy is very susceptible to noise. We also cut off areas that are too dark because lowlight performance of cameras is also unreliable. and grainy. Finally, the thresholded mask is eroded to remove noise.

From each color isolated frame, contours are drawn around each blob and the center of each contour is computed. We say that often times, a large solid green patch could potentially end up with two or more very close but disconnected blobs representing it. To resolve this, any nearby contour centers within a 30-pixel radius are clustered to form a single point to consolidate any single blob that was broken up due to noise. The amount of detected points from the image is then counted.

When the amount of detected points is equal to the amount of points on the custom pattern (in our case, 9) designed for optimum calibration, the points are sorted by x coordinate and matched with the database of known points. Since the points all have unique x positions, we can simply match the lowest x position detected point with the lowest x position known point and so on. We found that this process is actually extremely robust in most conceivable uses and can still reliably match points even under very extreme tilt positions and placement angles of the camera. From the 9 matched point pairs, a homography is found to output a transformation matrix from camera space to projector space. If the pattern is not adequately detected (detected points $\neq$ 9), the system continues the process again for up to a set amount of frames, after which the

program will assume that conditions are not adequate and will prompt the user to readjust so that a proper homography can be found.

We have found that our system accurately detects our custom pattern projected onto a flat white background in less than 10 iterations, or frames, even with the camera positioned at various angles. In good conditions, the detection will often happen in the first frame. Overall, the detection time often depends on the camera auto-focusing and white balancing, which will depend on each individual camera.[1].

ii. Detection

After camera calibration has occurred, the program switches into real-time capture mode for object detection. After a frame captured real-time is converted to HSV, thresholded, eroded, contoured, and filtered to remove unwanted detections (similar to the algorithm used for initial calibration object detection), the colored point detected is transformed using the projective transformation found during the process of camera calibration to find the corresponding location on the screen. To further filter out invalid points, if the contour center's transformed location is out of range of the screen, it is discarded.

Once the location of the detected point in both camera and world coordinates has been found, a point can be directly output onto the screen in the location of the detected point, or the point can be sent for further processing and image recognition. [2]

iii. Training

Training for subsequent object recognition was performed using the Keras library for Python. Over 7000 images of open and closed hands with variation of skin tones and lighting conditions and with differing backgrounds were captured to accumulate a robust training data set. In captured images, a green square with optimal hue for camera calibration was placed onto the hand; a red dot was then superimposed through software onto the hand to allow for different color recognition and color-agnostic training by the neural network by simply covering up the green patch. Each training image consisted of a 32x32 image. The sample space depends on the resolution of the camera. For example, for a 640x480 webcam, the 32x32 image is created by down-sampling a 64x64 area on the webcam image. For smaller screens, the area represented can be dynamically adjusted. One of the primary reasons we chose such a small resolution was because we had to ensure that the system remained very fast to allow for real-time processing. Since Smartwall is interactive, any latency would be immediately noticeable. We ensured a delay in the training image capture rate to ensure that each image was sufficiently different from the previous image. The training image capture system is built into our display.py smartwall system and can be turned on by changing the boolean 'training' to true. This allows for the training data collection system to be the same as the system used to collect real-time detection data.

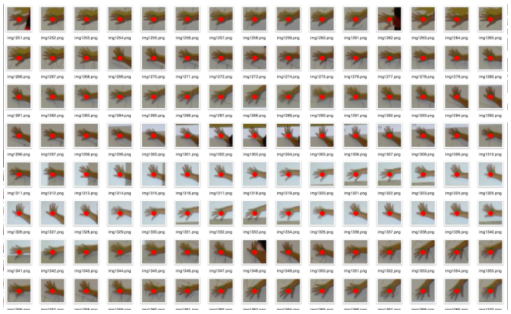

Figure 3: *Custom Training Data*

The images obtained were trained against using a deep convolutional neural network, using a pattern of Convolution, Dropout, Convolution, and Max Pooling layers repeated 3 times with 32, 64, and 128 feature maps. Finally, a larger Dense layer was used at the output of the convolutional neural network to more efficaciously translate the large number

[1]Source code in provided file **calibrate.py**.
[2]Source code in provided file **display.py**

of feature maps to class values.

The network topology was defined in Keras and the model trained using 100 epochs. We manually examined the training set images and removed any accidentally collected incorrect images. This ensured that the ground truth we were basing our model off of was correct. This training allowed for subsequent object recognition of open and closed hands with over 99% accuracy under controlled (typical) conditions. We saw a very consistent prediction shift from open to closed and vice versa when the hand transitioned states. When holding the hand clearly in one state, the prediction very rarely ever provides a wrong prediction.[3]

iv. Recognition

In our first implementation, we utilize color detection to find the location of the user's hand. To use the system, the user attaches a green patch roughly 2in x 2in to the back of the hand. Then, the hand can be used to control the mouse pointer by placing the hand in front of the screen. For optimal use, the hand should be close to the screen (to reduce shadows in the projector case, although our system does handle shadows decently well) and keep the back of the hand pointed towards the camera. To click, simply close the hand into a fist.

We used the same algorithm/system used in the calibration step to detect the hand color patch. One of the differences was that the blurring radius used was 9 pixels instead of 11 because the green patch is generally smaller than the calibration dots. The process of object recognition begins when one point (and only one point) has been detected by the system in real-time. This requirement is enforced due the technical limitation that windows computers only register one mouse pointer. If implemented on a device such as a tablet with multi-touch support, multiple points can easily be captured. When only one colored point has been detected, a 32 x 32 pixel window around the detected point is captured (in the same way as the training data) and then sent for processing. This window can be dynamically adjusted based on the resolution of the It is then input into the deep learning model for classification into one of two trained states: open hand and closed hand. A prediction is generated based on the two output probabilities into one of the two states.

To prevent erroneous click events from one or two stray incorrect closed hand prediction states, click events are only sent when two or more previous frames have been predicted by the neural network to be closed-hand frames. This is implemented by retaining a 3 frame buffer, where the majority state (in this case, 2) will determine the current prediction. Only then is the click event deployed by the system. Through experimentation, we found that our classification system was accurate enough that a minimal buffer of 3 was enough to eliminate almost all erroneous clicks while keeping latency down. [4]

v. Output

Closed to Open hand gesture transitions are mapped by the system to mouse button up events, and opend to closed hand gesture transitions are mapped to mouse button down events. A drawing system where one can draw onto the screen only upon closed-hand events is built into the system for demonstration purposes. Additionally, a system for controlling the machine running the program using mouse events is also built into the SmartWall system. This system provides the fundamental functionality of what is known as a smartboard of full computer mouse control.

[3] Source code in provided file **recognize.py**

[4] Source code provided in file **display.py**

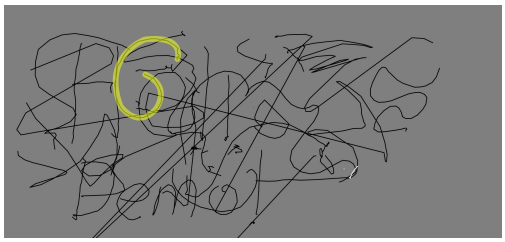

Figure 4: *Drawing from Live Class Demo, with Line by Professor Highlighted in Yellow*

III. Additional Methods

i. Markerless Hand Tracking

Our stretch goal for this project was to be able to recognize and track hands without having to rely on our green target markers. We were not able to implement this in time for our demo, however, we were able to eventually implement this functionality using a background subtractor coupled with a skin tone segmentation algorithm.

The first step in our algorithm involved using a background subtractor to remove all static pixels from the video feed. This method does not infringe on other aspects of our design, as we already require our camera to be stationary during use. The background subtractor is set to "learn" relatively slowly, so as long as users are not completely stationary in front of the camera, the subtractor will be able to work as desired.

The next step involves converting the remaining colors to YCrCb color space. We do this in order to better identify skin-tone colored regions from non skin-tone colored regions, and find contours around all skin-tone colored regions. We apply area constraints on these regions to rule out areas that are too large (such as skin-tone colored clothing) or small (backgrond noise) to be a hand. Finally, we removed regions that had extreme aspect ratios (long lines), as those were almost never hands. We also modified the background subtractor's learning rate based on the number of skintone regions found in the scene. Ideally, the background subtractor will have a low learning rate so that the user won't be subtracted out of the frame, but in the case that too many skin-tone regions were found, we would temporarily increase learning rate to remove static skin-tone regions from the frame.

Finally, we output the result of our program in real time. The following is one of our ideal outputs from our program:

Figure 5: *Camera Input*

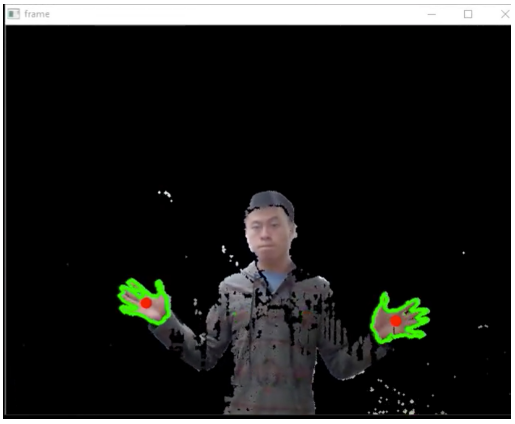

Figure 6: *Output of Markerless Hand Tracking Algorithm*

In order to integrate this with our system, we would have to add a third prediction for our deep learning model, which would be "not a hand". Getting training data for this would not be difficult, as we could easily collect thousands of 32x32 frames

from our camera that are not hands. Once the model is re-trained, we would use the product as normal, and our model input would change from a 32x32 frame around the green target to a 32x32 frame around the center of detected contours from markerless hand tracker. It would only take action if it detected a single open or closed hand, otherwise, it would do nothing. Since our deep learning model has been extremely accurate up until now, we see no reason to believe it would not work after adding a third prediction.

One of the issues we faced with this method is that it requires the user to not wear short sleeves, since the skin regions detected would include the arm. To solve this, the user could wear a watch or thick bracelet to segment the hand from the arm. A more advanced solution would involve using a body segmentation algorithm that could extract the skeleton of a user on camera similar to those outputted by kinect sensors. However, this may not be very easy to implement using just a single webcam.

To conclude, we feel that our markerless hand tracking algorithm is a good foundation to begin tracking hands without markers, however, there are many edge cases faced in testing that would need to be controlled for such as short sleeves, skin colored clothing, shadows from strange lighting, and issues with users who stand still too long. However, in the face of these issues, we have achieved good control outputs in the perfect conditions.

IV. Results

The system proved successful in a variety of environments under both daylight and nighttime lighting conditions. Our system performed better under conditions with fluorescent, or true-white light, rather than those with incandescent, or tinted-light sources, due to the algorithm used for color value thresholding.

Initial training runs for gestural prediction were performed on a simple convolutional neural network with 3 layers achieving 87% prediction accuracy. With this performance we were able to achieve reasonable click accuracy with a buffer of 5 frames, although this led to some latency. When a larger, "deep" convolutional neural network with increasing amount of feature maps was employed, hand gestures were recognized with over 99.8% accuracy.

V. Discussion

In this project, we developed a simple system to intuitively control a computer that is projected or displayed a large screen using hand gestures. The traditional and expensive hardware requirements are avoided using computer vision techniques, which has the added features of ease of use, easy deployment to any situation, and extremely low cost.

Future steps for research on object recognition using a camera and projector include dynamic color thresholding that uses white balance to account for different lighting conditions. Additionally, support for different color markers (currently, SmartWall only supports green) can be built into the system. Eventually, gathering more training data to recognize hands without the need for initial color object detection coupled with background subtraction algorithms for increased robustness can be integrated into the system to allow for drawing that does not require colored markers of any sort for initial detection before recognition algorithms are applied.

References

[Belongie et. Al, 2002] Belongie, Serge, Jitendra Malik, and Jan Puzicha. "Shape Matching and Object Recognition Using Shape Contexts." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.4 (2002): 509-22.

[Basilio et. Al, 2011] Basilio, Jorge Alberto Marcial, Gualberto Aguilar Torres, Gabriel Sanchez Perez, Karina Toscano Medina, and Hector Perez Meana. "Explicit Image Detection Using YCbCr Space Color Model as Skin Detection." *Applications of Mathematics and Computer Engineering* (2011): 123-28.

[Horn and Schunck, 1981] Horn, Berthold K.p., and Brian G. Schunck. "Determining Optical Flow." *Artificial Intelligence* 17.1-3 (1981): 185-203.

[Mikolajczyk et. Al, 2004] Mikolajczyk, Krystian, and Cordelia Schmid. "Scale & Affine Invariant Interest Point Detectors." *International Journal of Computer Vision* 60.1 (2004): 63-86.

[Shi, 1994] Shi, Jianbo. "Good Features to Track." *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* CVPR-94 (1994).