AR-Mirror

Jared Weinstein
Robert Gerdisch
jared.weinstein@yale.edu
robert.gerdisch@yale.edu
https://github.com/rgerd/ar-mirror

1 ABSTRACT

In the last couple of years, "smart" mirrors have gained popularity as an easy DIY project. These systems add additional visual information on the mirror's surface. Popular widgets include weather reports, calendars, or a running news feed. We seek to improve on previous work by creating a system that displays content in the mirror's reflected 3-D space. The content moves in real time according to changes in the user's perspective.

This paper describes our proof of concept system. We limit ourselves to same constraints as prior mirror projects. Our system captures information through a single camera source and must render content in real time. With these constraints in mind, we tackle three primary goals: (1) recognize and track a user's position in real world coordinates, (2) display content superimposed on the real world based on the user's perspective, and (3) detail a simple way of interacting with the mirror through head movement.

2 INTRODUCTION

As we move into the era of virtual and augmented reality, we must consider how to provide intuitive means of interacting with interfaces in three dimensions. Mirrors are our first windows into a virtual space. Requiring no special equipment, they have always provided an extremely simple yet crucial service: generating a copy of the environment that contains the observer. Not only are mirrors impeccably intuitive, but they are also used multiple times a day by everyone. This combination of familiarity and central importance makes mirrors a promising platform for enrichment through digital interaction. Our digital mirror system has the potential to provide unique interactions in augmented reality. Most AR/VR systems rely on complicated hardware setups. Our system will more faithfully mix reality with the artificial by allowing users to view content naturally in three dimensions, unencumbered by distracting goggles or hand-held controllers.

3 RELATED WORKS

Our work builds on previous work in AR, face recognition, object tracking, and intelligent mirror technology.

In order to smoothly track faces, Shaik and Asari detail a Kalman filter that tracks the size, position, and velocity of facial observations to address partial occlusion for short periods of times. In order to speed up the computation, Shaik and Asari use a skin segmentation approach to facial detection. [1] We diverge from Shaik and Asari by using Haar cascades for recognition. Foytik, Sankaran, and Asari offer an improvement in face tracking by using Modular Principle Component Analysis and a low-level recognition system to properly distinguish between many trackers. [2] They see increased accuracy in face recognition by performing a temporal average. The implemented solutions of Shaik and Foytik set a solid example for our tracker but are likely more complex and robust than required for our use case.

The hardware required for a rich mirror display has been extensively documented. Robert K. Meine filed a 2003 patent for an "inventive mirror" that lets users see electronic information in an unobtrusive way during morning and nighttime routines. The patent is fairly static and displays widgets of information and allows the user to interact through a touch screen. [3] Dorner et al. improve on this system by using a pattern of illumination and darkness to blend reality with rendered visuals. By tracking eye position, the pattern matches up with the user's perspective to create an AR effect. We render visuals in a manner similar to Dorner et al and improve on their work by not requiring a scan of the environment or a projection onto the individual. [5] Hossain et al. use face recognition to authenticate different users and automatically provide a customized widget-based interface. [4]. Lastly, Gentry et al. provides a detailed breakdown of a full build using raspi and voice recognition to display data and control various smart household items. [6]

4 METHOD

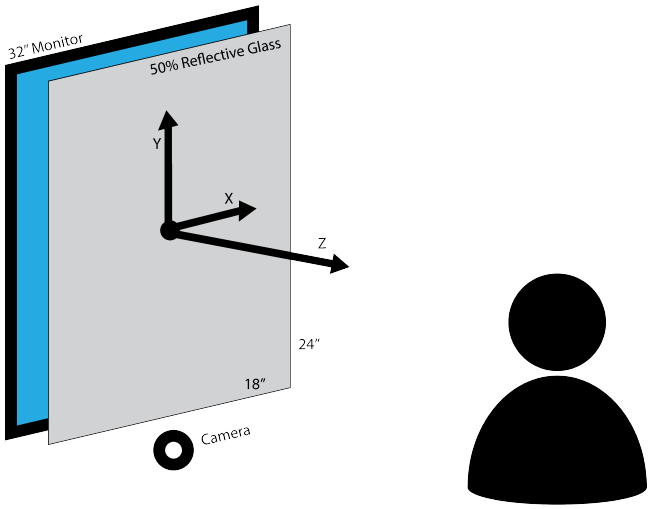

Figure 1: Our hardware setup.

Our setup (see Figure 1) closely matches a typical smart mirror setup [6]. The mirror body consists of two components: a large sheet of partially reflective glass and a monitor mounted directly behind it. When the monitor is black, the glass only reflects light from the room. As a result, the glass becomes fully reflective. When

the monitor is turned on, light from behind the glass mixes with reflected light. Under these conditions, visuals displayed by the monitor appear superimposed on the reflection. Our setup is 18" by 24". We use a Macbook Pro to process camera data and render visuals. We use the built-in Macbook camera. This camera sits 8" below the bottom of the mirror. Besides the vertical translation, all axes are aligned between the camera and the mirror's surface. Our calculations are greatly simplified by this translational and rotational alignment. It is possible to replicate our work with an unaligned setup, but it would require much more calibration of the camera's extrinsic parameters with respect to the mirror's surface.

Brief note on notation: Since we are working in two coordinate systems, 2-D pixel coordinates and 3-D "real-world" coordinates, we will use x, y, and z to denote positions in 3-D and x' and y' to denote positions in 2-D. At one point we will use x'' and y'', and these are to denote normalized pixel coordinates in the screen, ranging from -1 to 1, where $(-1, -1)$ is the top-left of the screen, $(0, 0)$ is the center of the screen, and $(1, 1)$ is the bottom-right of the screen.

4.1 User Tracking

The camera input is used to track the user's face, which in turn is used to both identify the user and resolve the user's point of view. Identifying the user is useful for enabling personalized interactions and efficiently tracking the detected bounding box surrounding the user's face across frames of camera input.

4.1.1 Facial detection. For each input frame, we begin by detecting faces using OpenCV's Haar Cascade classifier, which uses a cascade of image processing kernels to detect bounding boxes of the front of a human face. Each bounding box is cropped and subsequently matched with a user by a trained face recognizer that uses local binary pattern histograms (LBPH). The LBPHFaceRecognizer is trained on around one hundred images collected in a prior video recording.

AR reflections are rendered from a single perspective. As such, only a single user can interact at a time. However multiple users may interact with a shared mirror throughout the course of a day. Recognizing faces allows us to customize the experience for each user. In our case, we save a preferred color with each user. The display color is automatically switched when a new user is detected. Although this is a fairly trivial adjustment, it demonstrates the possibility of a fully customized and user-sensitive experience.

As a final step, the cropped and labeled faces are passed through a trained keypoint detector that labels 15 key locations on the face. The trained architecture comes from the Kaggle Facial Keypoint Detection Competition [8]. The keypoints provide additional information to help resolve the location and orientation of the face.

4.1.2 Depth calculation. We use the height of the facial bounding box to determine the user's "distance" from the mirror. Because none of our applications require us to know exactly how far away the user is from the mirror (in, for example, feet or inches), we can simply rely on the proportionality of perspective to give us a relative position of the user with respect to the mirror. For example, if the height of the user's facial bounding box in pixels (h'_{face}) at time t is 100 pixels, and at time $t + k$ is 50 pixels, we know that the user is twice as far from the mirror (fortunately we cannot control the size of our head–otherwise this would be impossible without a depth camera). When the user's face is centered horizontally and vertically on the screen, and the height of their face takes up the entire screen's height in pixels (h'_screen), we say they are at the origin $(0, 0, 0)$:

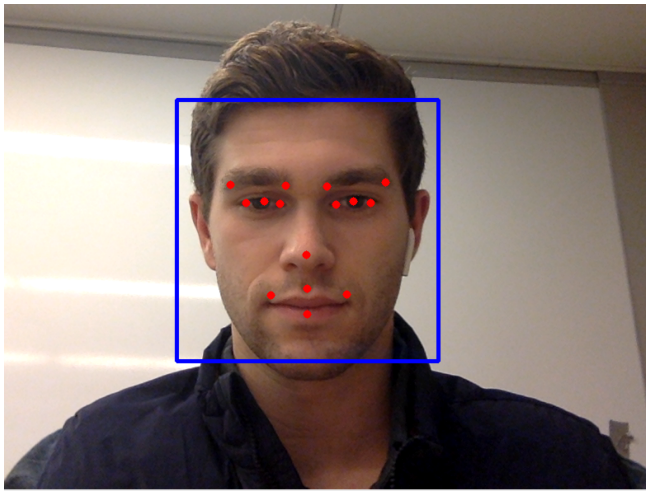

Figure 2: Facial keypoints.

$$z_{face} = (1.0 - \frac{h'_{face}}{h'_{screen}})\mu \quad (1)$$

Where μ is a scaling factor to make the distance correspond to some familiar unit of distance, such as feet or meters. We used 32 in this setup since it is a nice number. We put the origin where $h'_{face} == h'_{screen}$ because once the height of the face exceeds the height of the screen in pixels, we can no longer find a bounding box with Haar cascades, as the user is too close to the camera.

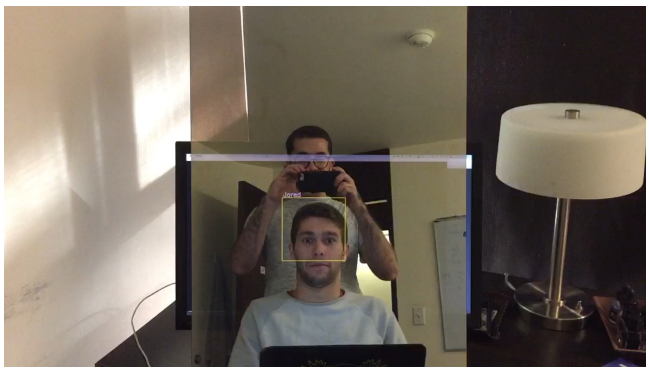

Figure 3: Simple facial bounding box overlay.

4.1.3 Full position calculation. Once we have the depth of the face relative to the mirror, we can also calculate the distance from the camera along the x and y-axes. We use the following relation:

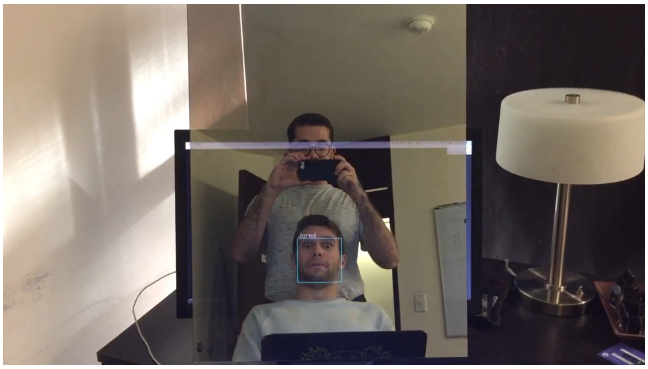

Figure 4: Change in color of bounding box demonstrates a larger measured depth.

$$x = \frac{x'' * z}{F} \tag{2}$$

$$y = -\frac{y'' * z}{F} \tag{3}$$

Where F is the focal length of the camera. While we do not need x, y, and z to correspond directly to any real-world measurement system, we do need them to correspond to each other convincingly. This is why we need the focal length of the camera: to determine how, as the user moves away from the camera, their x and y movements on the screen scale proportionally. Using a method derived from a tutorial online, we used known distances and pixel measurements to directly measure the built-in camera's focal length.[7] Placing the computer two feet from a piece of paper 8.5" by 11", measuring the width of the paper in pixels yields the final unknown in the relation:

$$F = \frac{w'_{paper} * z}{w_{paper}} \tag{4}$$

For example, in the case of our experiment, we had the following values: $z = 44$", $w_{paper} = 11$", and $w'_{paper} \approx 75$, to get $F \approx 300$.

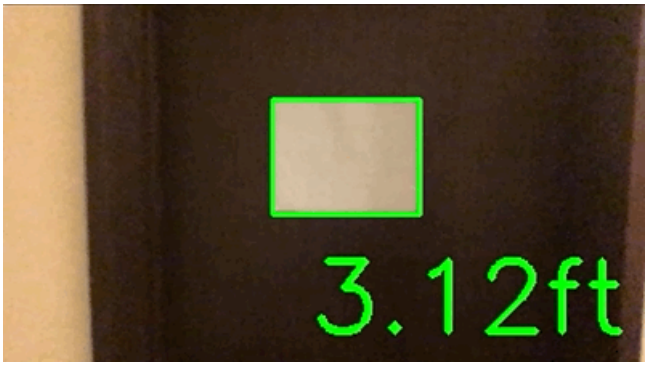

Figure 5: Using the pixel width measurement with the camera's focal length to determine distance from the paper in real time.

4.2 AR Content

With accurate tracking on the user's head, we can render interface elements convincingly in the reflected virtual space by constructing a virtual camera to model the user looking through a screen that matches the user's position and the dimensions of the mirror, respectively.

4.2.1 Mapping to the screen. We use a simple method based on raytracing to map points from virtual space onto the screen. Knowing the position of the observer relative to the origin (the center of the mirror) as well as the position of the virtual object in this same coordinate frame, we can cast a ray from the user's face to the point on the "other side" of the mirror, and the intercept at $z = 0$ is the projected position of the point on the screen.

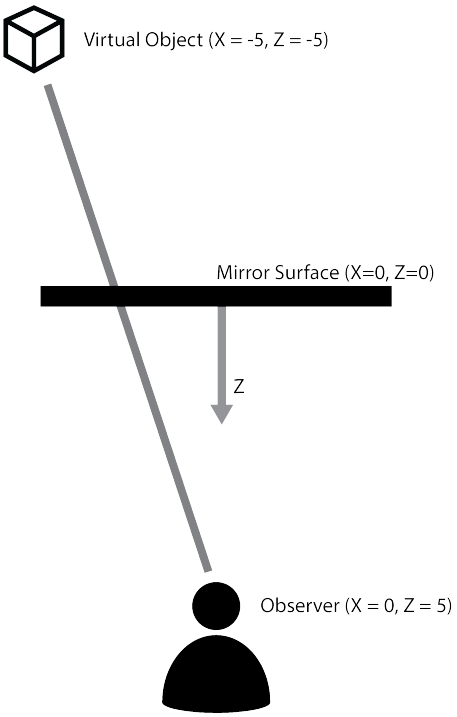

Figure 6: Simple facial bounding box overlay.

4.2.2 Hidden elements. Since a mirror reflection works the same way as if the reflected image were simply the user's view of a scene on the other side of a window, we can treat the reflected scene as a virtual space, and fill it with virtual objects. In our case, we saw opportunity in the space normally hidden behind the edges of the mirror's boundaries. When a user normally uses a mirror, they are mostly aligned with its center. However, peering around the frame of the mirror reveals a large amount of empty space that is typically unused. In the case of a normal mirror, the user could simply turn around to see the entire reflected scene. But with a smart mirror, we could hide UI elements like settings that should be readily available but otherwise hidden out of sight. We emulated this effect by placing a rectangle (the frame of a future rectangular settings panel) in a place that is usually hidden by the edge of the

mirror, but once the user leans to the side, looking around the edge of the mirror, the panel reveals itself:

Figure 7: Peering around the edge of the frame reveals a hidden panel.

4.3 Interactions

We seek to make interactions as simple as possible. Rather than using a touch screen or other physical interactions, we constrain all methods of interaction to the input of our camera. This allows for the possibility of interacting with interface elements without a physical controller.

As we mentioned above, keypoint tracking allows us to determine the orientation of the face. We use the position of the nose relative to the center of the facial bounding box to approximately determine the face's rotation about the x and y axes:

Figure 8: Observing the nose's position on the face to determine which way the user is looking.

From this, we can look at the nose's position over time to detect gestures such as nodding and shaking the head. We do this by detecting when the nose crosses into regions of the facial bounding box, and timing how long it takes to get to the opposite region. For example, if the nose crosses into the left 20% of the facial bounding box, and then within the next 500 milliseconds the nose crosses into the right 20%, we detect this as a shake of the head. We can then tell the UI to cancel, change selection, or any possible action that may correspond to "no". In the case of a nod, we might want to select the element under the selection cursor, aknowledge an information pop-up, etc. In our case, nodding and shaking advance and retreat in the selection of colors for the time interface on the mirror.

Figure 9: Nodding to change the color of the clock

5 EVALUATION

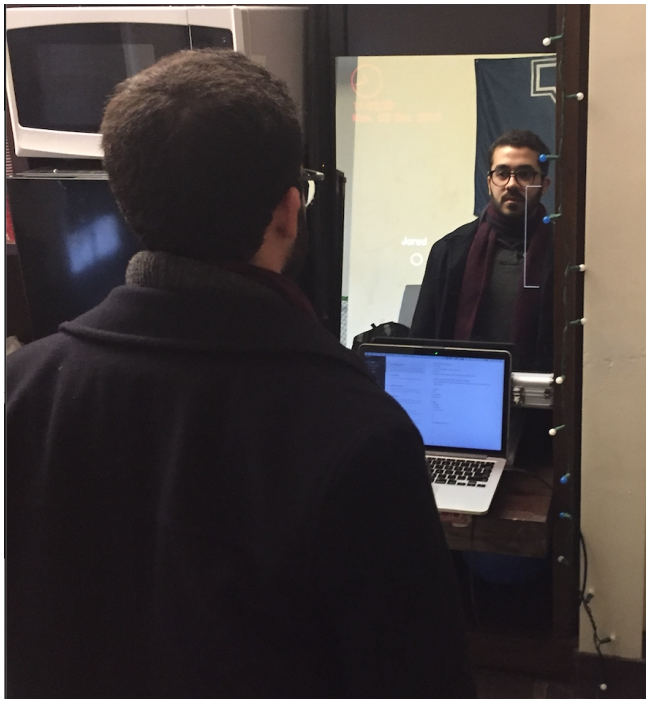

Figure 10: Setup for user feedback

We sampled 10 users for feedback. Individuals were all college undergraduates with ranging levels of technology experience. Each participant was shown the three key features of our system: face

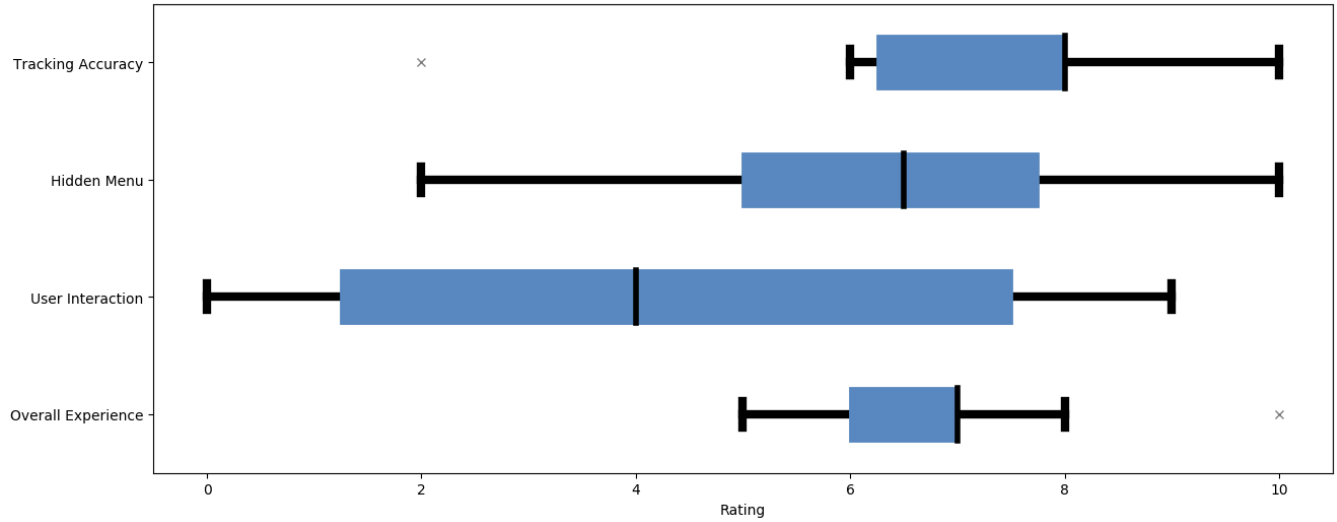

Figure 11: Participant's responses on a 1-10 scale after a session of interaction

tracking, a hidden menu element, and how to trigger a color change through a head nod. They were then instructed to sit at a reasonable distance and get an understanding of each feature. After a brief interaction (approximately 2 minutes), the participants were asked the following questions. Users were also asked to provide explanatory comments on their experience for fuller context.

(1) How accurately was the mirror able to track your face?
(2) Rate your experience with the hidden AR visual?
(3) Rate your experience triggering a color change.
(4) Rate your overall experience.

The results (Figure 11) show that each implemented feature varied widely in its reception. Tracking accuracy was consistently the highest rated. Participants felt it closely matched the position of their face; although, many noticed a substantive delay. Our outlier participant struggled to have his face recognized by the mirror. He believed this to be a fault of his glasses and had success after removing them. Other participants had no issue with their glasses. Further work is required to figure out whether facial accessories inhibit the Haar Cascade from being a good general-purpose solution.

Our hidden AR rectangle element received an average rating of 6.3. According to the participants, the rectangle appeared to stay in the same position in relation to the reflected background regardless of where they moved. Despite this success, no participant felt that the AR element was "part of the room". We identify two potential reasons for this failure. First, for an element to truly appear as if it were in real space it must move accordingly at all times. But our system contains a small delay. This inevitably breaks down the illusion. In addition, the AR element failed to appear embedded in reality when the user was stationary. We believe this to be an artifact of our binocular vision. See the section below on future work for a discussion of this issue.

The user interaction had the greatest deviation in responses. At best the interaction worked flawlessly. Two users were able to consistently and repeatedly change the color with a simple nod of their head. Other participants, however, were never able to trigger a color change. The interaction lacked substantial robustness. However, as a proof of concept, the system successfully demonstrated the possibility of interactions through simple gestures based only on facial keypoints.

6 FUTURE WORK

6.1 Even faster tracking

One of the most important capabilities of smart mirrors is their ability to track the user's perspective as they move. This is crucial for making the virtual reflections realistic, as they should move with the reflections of the real-world environment.

We managed to bring our system up to about 40 frames per second on a Macbook Pro, and this seemed sufficient for convincing real-time rendering. The greatest performance gain was acquired when we reduced the resolution of the image captured by the camera to about 30% of the original before it went to our image processing algorithms.

Further improvements to tracking performance could come from the use of a Kalman filter on the user's position, which can both account for noise from the facial size measurement and predict the user's future position to account for lag. Further work is required to determine if this is a feasible solution. The stochastic nature of head movement, might prevent the filter from being effective.

6.2 Better facial detection

As convolutional neural networks improve over time, not only will facial recognition become more accurate, but facial detection will also become more robust. Our method uses a simple Haar cascade to find faces from camera input, but it is fairly sensitive to lighting conditions and extremely sensitive to facial rotation (specifically pitch and roll). Using a convolutional neural network to find facial bounding boxes would certainly be a better approach in terms of

accuracy and robustness, and with the right hardware, one could avoid a significant performance hit.

6.3 Stereo vision problem

The most concerning problem with rendering artificial content on a reflected surface is caused by human stereo vision. Any binocular system recieves depth cues based on the displacement between images. The intuitive measure of distance is proportional to the perceived horizontal translation of the object between the left and right images. Because of this, humans will only "feel" like an object is close or far away if the left eye receives a different image than the right eye.

Because we cannot render different reflections for the left eye and the right eye, we cannot generate an intuitive representation of depth for our virtual reflections. Therefore, all of our virtual "reflected" objects appear to sit right behind the mirror, instead of appearing at varying depths.

One method we did not explore of giving the user a sense of depth is to use more parallax. That is, have many objects rendered at varying depths so that the varying amounts of parallax between them as the user moves their head gives the user a sense that some elements are closer than others. Humans have a complex, almost redundant system of sensors to indicate things like depth and motion, and so perhaps more accurately stimulating some aspects of our visual system is enough to at least weakly provide for a convincing illusion.

6.4 Better gesture recognition

A benefit of our method for interacting with interface elements is its accessibility. Anyone can communicate with the mirror's interface by moving their head, and so those missing limbs or lacking motor function below the neck are just as capable of manipulating the system as any other human.

We found that our gesture recognition technique, while often accurate, was rather flimsy. Part of this is due to low sampling rates (limited by the frames per second of the system), the noisiness of the keypoint data, and the inaccuracy of the keypoint network itself. Either a better-trained network or a different approach altogether to detecting nods and shakes might be better equipped for gesture recognition in future systems. We recommend preserving the accessible approach to interaction, but look forward to seeing better techniques for recognizing attempts to interact, given noisy inputs from the camera.

7 CONCLUSION

This project serves as a sort of baseline. It is a working smart mirror that augments reality by overlaying static information on top of reflections as well as moving interface elements in a combined real-virtual space. We are calculating three-dimensional position from a single camera input, and rendering a three-dimensional scene from the point of view of this position, so as to give the effect of holograms moving with the reflected environment while the user moves their head. Finally, the system both recognizes individual users from each other to provide personal customization, and allows the user to interact with it via the most intuitive possible gestures meaning 'yes' and 'no.'

The functionality is all there, but not ready for the public. The purpose of this project is to encourage improvement on top of our design. We believe that future smart mirrors should take advantage of each component of this project (tracking, hidden ui, gesture recognition), while reimplementing it to be more robust and more efficient. Our hope is that eventually the design seen here will not suffer from the weaknesses that we describe, and we will have smart mirrors that are almost as intuitive as normal mirrors.

REFERENCES

[1] Z. Shaik and V. Asari, "A Robust Method for Multiple Face Tracking Using Kalman Filter," *36th Applied Imagery Pattern Recognition Workshop* (aipr 2007), Washington, DC, 2007, pp. 125-130. doi: 10.1109/AIPR.2007.2

[2] Foytik, Jacob, et al. "Tracking and Recognizing Multiple Faces Using Kalman Filter and ModularPCA." *Procedia Computer Science*, vol. 6, 2011, pp. 256âĂŞ261., doi:10.1016/j.procs.2011.08.047.

[3] Meine, Robert. *System and Method for Displaying Information on a Mirror.*

[4] Hossain, M & Atrey, P.K. & El Saddik, Abdulmotaleb. (2007). Smart mirror for ambient home environment. *IET Conference Publications*. 589 - 596. 10.1049/cp:20070431.

[5] Dorner, Charles, et al. *Blended Reality Systems and Methods.*

[6] Gentry, Justin, et al. "Smart Mirror: A smart home solution for increased productivity during those busy mornings" *http://www.eecs.ucf.edu/seniordesign/sp2016fa2016/g24/docs/SD1FinalDoc.pdf*

[7] A. Rosebrock, "Find distance from camera to object using Python and OpenCV," PyImageSearch, 19-Jan-2015. *https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/*

[8] Frolian, FacialKeypointsDetection, (2017) *https://github.com/flothesof/posts/blob/master/20170914_Facia*