Final Report of Smart Slides Presenting System

Team member:

Xinghan Wang, x2wang@ucsd.edu, A53287481
Yupei Zhu, yuz310@ucsd.edu, A91095937

Project Motivation:

Imagine standing in front of a projection screen and presenting some slides. Now you want to switch page, highlight or magnify something on you slide. Most likely you will have to go back to your computer and do so, but frequently running back and forth can be interruptive. We want to solve this problem with a Raspberry Pi based system. The idea is our portable system will turn any normal projection screen into a virtual touch screen so that we can interact with the slides (switching slides, zooming in certain parts of the slide, drawing on the slide and etc.) in real time with bare hands. It will also keep track of the timings to notify user to either speed up or slow down. The system will help deliver more intuitive, interactive and smooth presentations.

Related Work:

Logitech's Spotlight Presentation Remote [1] is a hand-held device that allows users to interact with the slides. The device works like a traditional laser pointer but the laser point is virtual. It is capable of magnifying, circling and highlighting around the virtual laser point. Our system differs from theirs in that ours allows user to directly interact with the slides with bare hands and therefore more intuitive. Since our system uses hand gestures and location as inputs (instead of buttons with fixed/pre-programmed functions), we could add more functionalities easily.

Another related product is the online presentation software by Emaze [2], which allows users to control slides (play, stop, next and back) using human voice. This product is similar to our system in that it allows users to interact with the slides during the presentation without additional hardware in hand. However, our system has more functions (zoom and draw) and is non-intrusive (no interruption with voice commands).

Intuitively, our system will be very competitive compared to either Spotlight Presentation Remote or Emaze. Our system does not need extra hardware in the user's hand and it will not generate accidental commands due to misrecognition of voice. User only needs to point on the screen to send commands to the host with their bare hands and we have achieved very high accuracy in recognizing user's commands. Meanwhile, the total price of our system is much cheaper than Logitech's Spotlight.

Hardware Components:

a. Models and manufacturers:

i. Raspberry Pi 3: Raspberry Pi Foundation
ii. Raspberry Pi camera module V2-8 Megapixels: Raspberry Pi Foundation
iii. 3.5 Inch HDMI TFT LCD touch screen: UCTRONICS
iv. LED light: Elegoo
v. Button: Elegoo
vi. Active buzzer: Elegoo

b. Reasoning for design choices:

We choose the official camera module because it is more stable and supportive than other camera modules. It also provides enough resolution (maximum 1080P video) for our project. We choose the LCD screen mainly because it is one of the cheapest and most highly-rated LCD screens on Amazon. The sensors are included in the 37-in-1 kit so we do not need to buy them again.

Software Components:

a. Softwares and versions:

i. OpenCV 4.0.0 with Neon [3]
ii. Python 3.5
iii. Pyzbar 0.1.8 [4]
iv. Google Firebase Realtime Database and related Python tools
v. WiringPi library

b. Reasoning for design choices:

OpenCV is one of the most powerful image processing tools available on the internet and it could be configured on Raspberry Pi with support of different languages. Unlike its previous versions, OpenCV 4.0.0 would be built with NEON enabled automatically on ARM processors so it will ensure the ultimate performance on Raspberry Pi.

Pyzbar is a powerful and reliable library to detect and decode barcodes and QR codes that is based on OpenCV. It can also calculate the bounding boxes of multiple QR codes in the same frame which is suitable for the initial calibration process.

We integrate Firebase to ease the setting-up cost of the system. With Firebase, we only need to ensure Raspberry Pi and the host computer are connected to the internet and then the system is ready to go. Other methods may involve Bluetooth pairing or finding IP addresses that are either hard to implement or hard for user to interact with the system.

HW & SW integration:

The workflow of our system is the following:

First, we open up the user's application to show the slides on the host computer. The application will read in the time limit for each slide as well as the time limit for the whole presentation from a text file and upload them to firebase. It will show a calibration frame with 4 QR-codes on the corners to help Raspberry Pi locate the screen boundaries.

Then, we start the application on Raspberry Pi. It will keep finding the 4 QR-codes' locations until the button is pressed. LCD screen will display the camera frames to help user to locate the QR-codes. The active buzzer will keep making sound when QR-codes are not found. Once 4 QR-codes are found, the system will extract only the screen part that is rounded by QR-codes for next-step processing. User can check the sanity of system by checking if only the screen part is showing on the little LCD display. Once user press the button and the system finds all the QR-codes, it will enter presentation mode.

After Raspberry Pi enters presentation mode, it starts to measure 1) remaining time in total 2) remaining time for each slide and show both of them on the LCD screen. If the remaining time is less than zero, LEDs start to flash. An RGB LED will tell users the mode they are using: red for zooming / green for drawing rectangles.

There are three buttons on the left of the screen: mode, up, down. If user points to "mode" button, the mode will switch between zoom and draw. If user points to "up" button, the previous slide will show up. If user points to "down" button, the next slide will show up.

Raspberry Pi will keep tracking skin color segments on the extracted frame and records the rightmost point in all the skin color segments. Once there is no skin color segment on the frame, our algorithm will analyse the recorded points in this user interaction session and send commands(next, prev, zoom, draw) to the host computer through firebase accordingly. We will go over this core part of our system in the next subsection.

Press the button again to end the session.

On the host computer, the user's application will change the frame, draw rectangles or zoom in certain parts according to the commands and coordinates it reads from the Firebase database. Press ESC to exit the user's application at any time.

Algorithm for determining the commands:
Inputs: *w*, *h* (width and height of the screen frame size), *n* (threshold for ignoring some points)
Variables:

```
session_points = []
mode = ZOOM
Begin Algorithm
While TRUE:
    Get an image img from camera.
    frame = extract the screen part of img using the QRcodes' location and map it into a w*h image.
    Convert frame from BGR coded mode to HSV coded mode.
    Set any pixel in frame that are not in the range of skin color to black.
    Erode and dilate frame.
    Find all the contours in frame and add the rightmost point of all the contours to list session_points
    If we cannot find any contours, do the following steps:
        If most points in session_points are in one of the buttons, send the corresponding
        command or change mode to the other one. Empty session_points and continue to the
        next loop.

        Else if mode is DRAW, remove the first n points and the last n points enter session_points
        and find a bounding rectangle for the remaining points. Send draw command with the
        rectangle. Empty session_points and continue to the next loop.

        Else, mode is ZOOM. Find the rightmost point in session_points and send zoom
        command with the point. Empty session_points and continue to the next loop.
End Algorithm
```

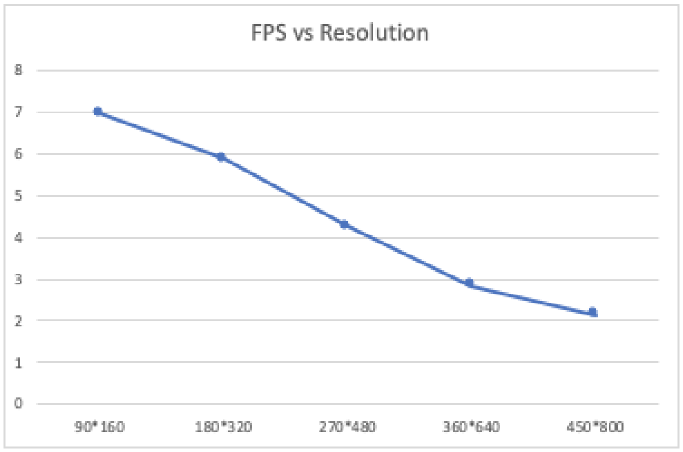
We have explored and tried many ways to find the position of user's hand. Each of them has different pros and cons. For example, OpenPose [5] is a very powerful machine learning based library that can detect poses in real time. However, its frame rate is even less than 0.5 FPS on desktop level CPUs so it is clearly infeasible. Using motion detection techniques based on thresholding is one of the other methods we tried. We talked about it in progress report. It actually gives very good result in dark condition but it also has its disadvantages: when the lighting is good, the thresholding method may not be able to differentiate our skin colors and the white background. Also, it will be a disaster when the slide changes since it will read a lot of motion by comparing two frames. Detecting skin segments is a good turnaround method for motion detection. It will convert the image's format to HSV in which the brightness of color is one of the channels so the other two channels can be analysed without too much influence of the lighting condition. It will not be affected by changing slides and it works very well under good lighting condition. The tradeoff is that it sometimes will ignore the finger parts of the hand because wrinkles will be read as other irrelevant colors and thus the fingers will be eroded to black. However, the accuracy of skin color method is still very high and acceptable. We will present the accuracy in the next section.

In our algorithm, the screen frame is extracted from the output of the camera to a smaller image array to reduce computational intensity and thus speed up the frame rate. However, if the screen frame is too small, we will lose accuracy since the size of the hands will be reduced and combined with noise on the image. After several experiments, we determined 360*640 pixels to be one of the optimal resolutions for the screen frame.

The other problem we need to overcome is that since we are recording user's motion using a 2D camera, it is hard to determine when the user starts to draw the rectangle. We found that in normal cases, given a 40-to-50-inch television, user will spend ~0.5-0.8 seconds to reach the starting point to draw the rectangle and ~0.5-0.8 seconds to leave the screen. So we can remove some points in the front of the list and some points in the end of the list to estimate the bounding rectangle. In our case, the frame rate is 2.85, so we will remove 2 points in the front and 2 in the end. This estimation is surprisingly accurate. We will see the result in the next section.

Experiments:

a. **System FPS:**
We tested the FPS the system can process under different resolutions of extracted frame. The camera is recording 720P video at 30 FPS. Intuitively, as the frame resolution increases, the amount of computation increases as there are more pixels and thus the processing time increases for each frame. Our experiment plot and data shown below validates this hypothesis. We finally settled on 360*640 resolution after trading off accuracy against latency.



Resolution	**FPS**
90*160	**6.97**
180*320	**5.88**
270*480	**4.25**
360*640	**2.85**
450*800	**2.15**

b. **Accuracy under "Draw" mode:**
We test the accuracy of the "draw" mode by first drawing a 500*500 pixels green rectangle in the middle of a 1500*2667 slide, then activate the "draw" mode and try to encircle the green rectangle. The blue rectangles shows the test result over 10 trials. The closer the blue rectangles are to the boundaries of green rectangle the better. Our results are demonstrated below.

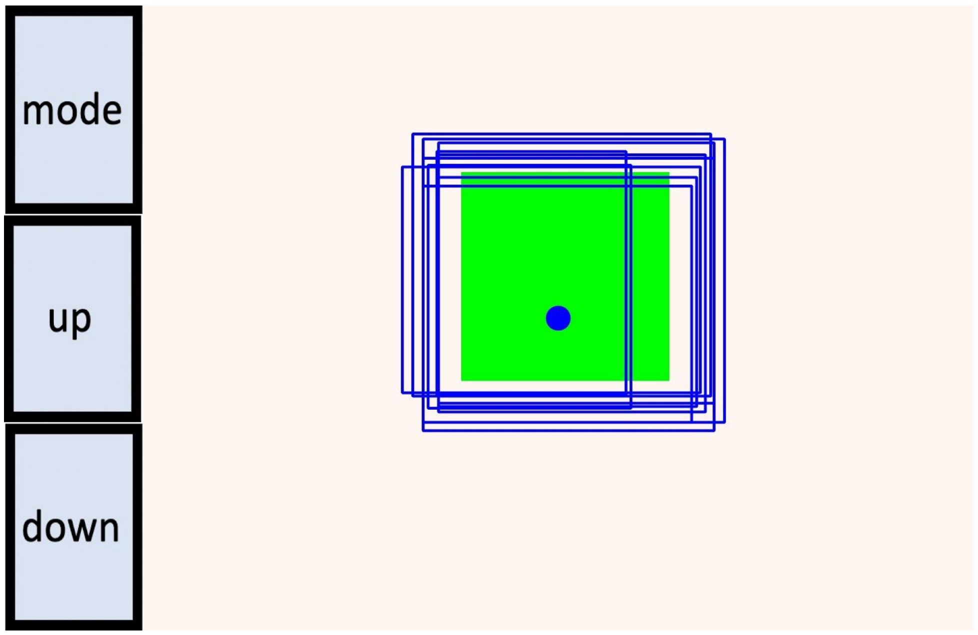


Figure 1(a): experiment setup for part b) and c). Green rectangle is the ground truth

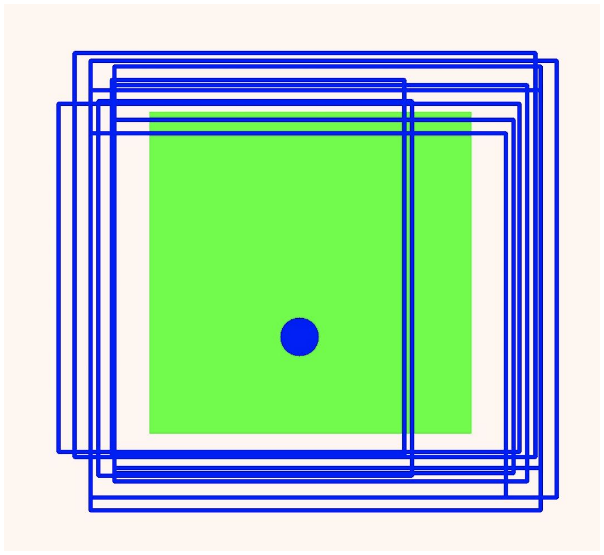


Figure 1(b): zoomed in version of **Figure 1(a)**

As shown in Figure 1(a)(b), the 10 blue rectangles all closely resembles the green rectangle, with only 4 crossing slightly into the ground truth. All boundaries drawn in "draw" mode are within 100 pixels of the ground truth. This accuracy is acceptable for our application.

c. **Accuracy under "Zoom" mode:**
We test the accuracy of the "zoom" mode by first drawing a blue circle with radius of 30 pixels in the middle of a 1500*2667 slide, then activate the "zoom" mode and point to the circle to see how far the detected pointing location is from the ground truth. The experiment setting is again

illustrated by Figure 1(a)(b). The experiment result of zoom is shown in Figure 2.

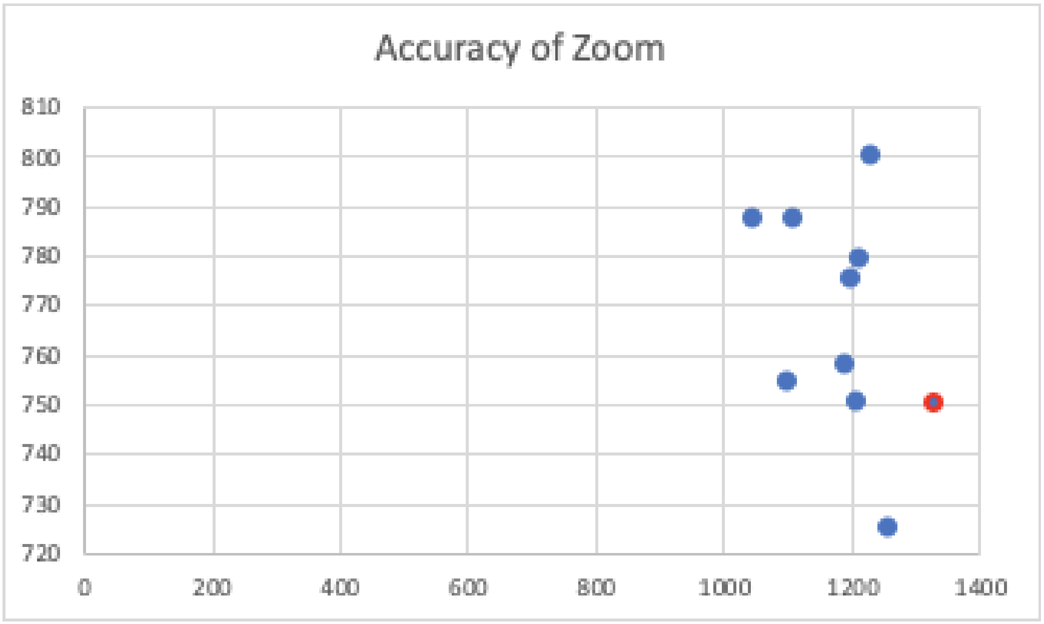


Figure 2: detected pointing location (blue) and ground truth (red).

The average error (L2 distance from ground truth) is 163.65 pixels. Given the slide size of 1500*2667, this error is acceptable, though sometimes user has to point more than once to accurately zoom in on one point.

Conclusions & Future steps:

We implemented a smart slide presenting system based on Raspberry Pi. On the hardware side, we have utilized hardwares including LEDs, active buzzer and button to change/indicate the running state of the system. We use LCD screen to notify users the remaining time for each slide and the whole presentation. We utilize the high resolution Raspberry Pi camera to acquire user's movements. On the software side, we deploy OpenCV library on both Raspberry Pi and user application to recognize hand movement and enable slide interaction. We use Firebase database to communicate between the Raspberry Pi and host computer. Combining the software and hardware components, we successfully implemented a reliable slide presenting system that can change slides, zoom in and draw rectangles only using user's bare hands.

During the development process we have to make several design decisions along the way, such as frame resolution and algorithm to use for hand detection. We have to give up OpenPose because of limited computing power. These decision exemplifies the typical decisions to make when designing an application based on embedded systems under cost and hardware constraints. We learned how to achieve our goal by trading off different aspects.

For next steps, we can try to draw not only rectangles but also random lines on slides. To accomplish this a more accurate hand recognition algorithm should be used. We can also add more interactive functionalities such as moving items around in a slide, which will require developments on the user application. Furthermore, we could port the Raspberry Pi part of the system to mobile phone/tablets so that a user can use the system anywhere if they have a mobile device. This is possible since modern mobile devices have high resolution cameras, Wi-Fi capabilities and sufficient computation power.

References:

[1] Spotlight Presentation Remote. https://www.logitech.com/en-us/product/spotlight-presentation-remote. Accessed Feb 2019.
[2] Emaze online presentation software. https://www.emaze.com/presentations/. Accessed March 2019.
[3] OpenCV 4.0.0. https://opencv.org/opencv-4-0-0.html. Accessed Feb 2019.
[4] Pyzbar. https://pypi.org/project/pyzbar/. Accessed Feb 2019.
[5] OpenPose. https://github.com/CMU-Perceptual-Computing-Lab/openpose. Accessed March 2019.