1 iOS Application Specifications

Colorblind Aid consists of three main features: “color tagging,” “color filtering,” and “local tag storage” (see feature details in §2). Largely intertwined, these three features are presented in a 3-pane slidable view, which enables seamless scrolling and transition between the three panes (see UI details in §3). A brief description of each feature:

1. “Color Tagging” – a feature that allows you to mark and label the colors of objects seen through your camera in real-time (§2.1)
2. “Color Filtering” – a feature that applies a filter to a marked color/object, altering the color to “counteract” the deficiency to appear more like what a non-color blind person would see and can be used to transform commonly confused colors into easily discriminable colors (§2.2)
3. “Local Tag Storage” – a feature that allows you to save rooms of marked and labeled colors for access in the future (§2.3)

These features work closely with the native Photos and Camera applications of the iPhone. Furthermore, the application is independent of internet connection. An important specification as it enables the application to truly help colorblind individuals without the burden of first locating a cellular or wireless connection.

2 Features

As mentioned in the above section, Colorblind Aid consists of three main features: “color tagging,” “color filtering,” and “local tag storage.” Each of these features takes advantage of a separate iOS API, with color filtering being the most complex.

2.1 Color Tagging

This feature allows the user to mark and label the colors of objects seen through the camera in real-time. In essence, color tagging must convert the color at the center of the image from the inherit RGB values into a name, such as “blue” or “red”. This feature builds on Apple’s ARKit, Chirag Mehta’s Name that Color algorithm, and Github user @jathu’s UIImageColor to draw augmented reality nodes with appropriately named color hues.

Using Apple’s ARKit provided a straightforward method to create augmented reality nodes. The difficulty in implementing this feature was in converting RGB values to color names.

2.1.1 Initial Approach

Our initial approach to solving this problem was to convert the RGB values of the pixel at the center of the camera to XY color values, which can be mapped onto a parabolic 2D graph of colors. Then, a switch statement could be used to determine the hue of the color. For example, (0.3, 0.5) might map to red.

Following code provided by Philips Hues, the steps to convert RGB to XY are as follows:

1. Get the RGB values from your color object and convert them to be between 0 and 1
2.2 Color Filtering

2. Apply a gamma correction to the RGB values, which makes the color more vivid and more like the color displayed on the screen of your device.

3. Convert the RGB values to XYZ using the Wide RGB D65 conversion formula.

4. Calculate the xy values from the XYZ values.

5. If the found xy value is not within the color gamut of the light, calculate the closest point on the color gamut triangle and use that as xy value.

6. Use the Y value of XYZ as brightness.

Given the CIE Chromaticity Diagram, I mapped the xy values to the appropriate colors.

Unfortunately, this process provided, dare I say, awful results. Due to imprecision in both the mapping process as well as the conversion process, the compounded errors resulted in incorrect labeling.

2.1.2 Final Approach

Given this initial failure, I pivoted to a completely different approach. Building off code provided by Chirag Mehta’s Name that Color algorithm, I compiled a list of about sixteen hundred common RGB and HSL color values, names, and hues (e.g. red, blue, green, orange, grey, brown, violet, white, black, yellow). For example, one such entry is colorName: "Baby Blue", hueName: "Blue", red: 111, green: 255, blue: 255, hue: 127, sat: 255, lum: 183.

Now, given the RGB values for a color, the HSL could be calculated and a straightforward minimum distance algorithm run through the 1600 entries to find the closest would return an approximate color and hue. Of course, this approach relies upon the compiled list of colors to be more or less extensive, which appears to be the case in application.

That being said, there were occasional blips in accuracy, presumably caused by the deviation and unreliability of any single pixel. Therefore, rather than use the pixel at the center of the canvas, @jathu’s UIImageColor algorithm was applied to a small square around the center which extracted the most prominent color. The conversion from color to name was then applied to this color.

These results were substantially better than the initial approach and held up during texting when under good lighting.

2.2 Color Filtering

This feature allows the user to apply a filter to a selected region, altering the color to counteract the deficiency of color blindness to appear more like what a non-color blind person would see and can be used to transform commonly confused colors into easily discriminable colors. In essence, color filtering applies a transformation to the RGB values of the pixels of an image.

Using Apple’s Core Image API provided a clear method through which to manipulate the RGB values of a pixel. Building upon the project of Onur Fidaner, Poliang Lin, and Nevran Ozguven who did an analysis of color blindness, in simple terms the process is as follows:

1. Simulate color blindness using a commonly established transformation matrix.

2. Isolate invisible colors to color vision deficiency (i.e. the difference between the original and the simulated pixel).

3. Shift colors towards visible spectrum by applying an error modification matrix (not all values should be affected equally).
4. Add back the compensated error values to the pixel’s original values
5. Range bound the RGB values
6. Return the transformed values

Two versions of this algorithm were developed, the difference being that in the second version the RGB values are first converted to LMS before color blindness is simulated. Simulation matrices for twelve types of color blindness are used and can be found in the appendix (4). The three ending in LMS are for the LMS version of the algorithm.

Using a custom Core Image Filter written in kernel code, this transformation is applied to an image to alter and counteract color blindness.

In limited testing, the results of this transformation were incredible (although hard to show to someone who is not colorblind)! As is displayed in the three images below, applying this algorithm allows colorblind individuals to see previously invisible numerals in images that are used to test for color blindness, by altering the image through the above transformation algorithm.

Furthermore, these simulation matrices can instead be used to simulate color blindness, rather than correct for it. By applying the transformation without the subsequent adjustment and error modification, the pixels are adjusted to mimic what is seen by someone with color blindness. This application has many uses in application development where the developer would be wise to ensure elements of the application are clearly visible to both colorblind and non-colorblind users.

*Left to right: An example from a color blindness test - the numeral is not visible to deuteranopes
The same image with the deuteranopia algorithm applied. The 8 is now visible!
The same image with the LMS deuteranopia algorithm applied*
2.3 Local Storage

This feature allows the user to save rooms of marked and labeled colors for access in the future. By combining the augmented reality nodes with Apple’s NSCoding protocol and local KeyedArchiver, rooms labeled in the AR view (§3.2) can be locally saved on a user’s device.

Keeping with the goal of being internet-independent, using the local KeyedArchiver enables local storage without the necessity of the cloud. Furthermore, it opens the possibility for future development of sharing saved rooms between users of the app by sharing these already saved objects over a connection.

3 User Interface

The user interface of Colorblind Aid strives to be as simple and seamless as possible. Consisting of three side-by-side panes, the user can slide between the three much like as is possible through Snapchat. The panes, from left to right, are Saved Rooms, AR Labeling, and Filtering (Figure 1). The swipe animation, about half way between each pane, is displayed below in Figure 2.

3.1 Saved Rooms View

The leftmost pane is the Saved Rooms view. This view consists of a collection view of locally stored rooms of augmented reality nodes (§2.3). Arranged two across, the user may tap one a view to load those nodes into the AR Labeling view. The associated image for each room is screenshot of the camera taken upon saving the room.

The user may also long-press (defined as a press lasting longer than 1 second) a particular room to be prompted by an action sheet where they may load (the same as tapping) or delete a room.

Along the bottom toolbar of this view is a button labeled “AR” which will transition to the AR Labeling view (§3.2). This same transition is also possible by swiping to the left.
3.1 Saved Rooms View

Figure 1: Saved Rooms, AR Labeling, and Filtering views

Figure 2: UI swipe animations
3.2 AR Labeling View

The middle pane is the AR Labeling view. This view contains the app’s augmented reality capabilities (§2.1). After the user has enabled access to their camera, they will be displayed with a view consisting of what is visible through their camera, onto of which the augmented reality nodes will be placed.

By tapping on the screen, the user will create a node at the center of the screen, whose location is designated by a light grey crosshair. This node, as visible in the middle image of Figure 1, is made up of a cyan sphere and orange bubble text. The sphere marks the location and the text is the color at that location. As this node is placed via ARKit’s API, it will reside in the 3D space of the room, not the 2D space of the screen. Furthermore, the text of the node will only appear parallel along the Y axis, allowing it to rotate, and thus be readable, as the user moves about the physical room.

Along the bottom toolbar of this view are three buttons, a cog icon, a capture photo icon, and a color filter icon. The cog icon will transition to the Saved Rooms view (§3.1) while the color filter icon will transition to the Filtering view (§3.3). These same transitions are possible by swiping to the right or left.

The capture photo icon will transition to the Filtering view and set the image to be filtered to a screenshot of the camera. This enables the user to quickly filter an object seen through the camera, rather than undergo the typical procedure of loading an image from Photos.

Along the top toolbar of this view are two buttons, a save icon and a clear icon. The save icon locally saves all of the augmented reality nodes into a room object and prompts the user to give the room a name. This room is then visible in the Saved Rooms view. The clear icon removes all of the augmented reality nodes from the screen.

3.3 Filtering View

The final pane is the rightmost, the Filtering view. This view is the interface through which the user performs the color filtering feature (§2.2). Initially blank, the user selects an image using the crosshair button along the bottom toolbar. Tapping the button prompts an action sheet where the user can either select a photo from their photo library or take a new one using the camera. As mentioned above, they may also capture a screenshot from the AR Labeling view using the capture button.

Once an image is selected, as long as the bottom right button says “Done” the user may filter the photo. Tapping that button enables and disables filtering.

While in filter mode, the user may draw rectangles over the image, creating a region over which they can filter the image. By tapping on a filter rectangle, the user is presented an action sheet allowing them to move, filter, or delete the rectangle. Selecting move prompts the user to tap on the image to change the center of the rectangle. Selecting filter filters the region of the image using the current filter. Delete removes the rectangle altogether.

The default current filter is “normal” or no filter. To change the current filter, the user taps the color filter icon on the right of the top toolbar. Upon tapping this icon the user is presented with a table of the twelve available filters (as described in §2.2). Now, choosing to filter a rectangle will apply this filter.

The user can also simulate color blindness, rather than correct for it, by long pressing the color filter icon and selecting “Simulate” (versus the default of “Daltonize”).

Finally, the user can save the filtered photo by tapping the save icon on the left of the top toolbar. Tapping this icon will blend the filtered rectangles into the original image and save it to the users photo library.

Additionally, along the left of the bottom toolbar of this view is a button labeled “AR” which will transition to the AR Labeling view (§3.2). This same transition is also possible by swiping to the right.
Appendix

4 Simulation Matrices

<table>
<thead>
<tr>
<th>Condition</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>(1, 0, 0)</td>
<td>(0, 1, 0)</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>protanopia</td>
<td>(0.567, 0.433, 0)</td>
<td>(0.558, 0.442, 0)</td>
<td>(0, 0.242, 0.758)</td>
</tr>
<tr>
<td>protanomalred</td>
<td>(0.817, 0.183, 0)</td>
<td>(0.333, 0.667, 0)</td>
<td>(0, 0.125, 0.875)</td>
</tr>
<tr>
<td>deuteranopia</td>
<td>(0.625, 0.375, 0)</td>
<td>(0.7, 0.3, 0)</td>
<td>(0, 0.3, 0.7)</td>
</tr>
<tr>
<td>deuteranomalred</td>
<td>(0.8, 0.2, 0)</td>
<td>(0.258, 0.742, 0)</td>
<td>(0, 0.142, 0.858)</td>
</tr>
<tr>
<td>tritanopia</td>
<td>(0.95, 0.05, 0)</td>
<td>(0, 0.433, 0.567)</td>
<td>(0, 0.475, 0.525)</td>
</tr>
<tr>
<td>tritanomal</td>
<td>(0.967, 0.033, 0)</td>
<td>(0, 0.733, 0.267)</td>
<td>(0, 0.183, 0.817)</td>
</tr>
<tr>
<td>achromatopsia</td>
<td>(0.3, 0.59, 0.11)</td>
<td>(0.3, 0.59, 0.11)</td>
<td>(0.3, 0.59, 0.11)</td>
</tr>
<tr>
<td>achromatomal</td>
<td>(0.6, 0.3, 0.06)</td>
<td>(0.16, 0.78, 0.06)</td>
<td>(0.16, 0.3, 0.5)</td>
</tr>
<tr>
<td>protanopiaLMS</td>
<td>(0.0, 2.02, -2.53)</td>
<td>(0.0, 1.0, 0.0)</td>
<td>(0.0, 0.0, 1.0)</td>
</tr>
<tr>
<td>deuteranopiaLMS</td>
<td>(1.0, 0.0, 0.0)</td>
<td>(0.494, 0.0, 1.248)</td>
<td>(0.0, 0.0, 1.0)</td>
</tr>
<tr>
<td>tritanopiaLMS</td>
<td>(1.0, 0.0, 0.0)</td>
<td>(0.0, 1.0, 0.0)</td>
<td>(-0.396, 0.801, 0.0)</td>
</tr>
</tbody>
</table>