Virtual Keyboard with Kinect

Computer Vision Final Project Fall 2012

Tom Ayotte and Renata Smith

12/7/2012

Table of Contents

Background

The use of interactive displays in computer vision has grown in popularity in recent years. Due to advancements in technology, sensors which aid in interactive display creation such as the Kinect by Microsoft, can be purchased relatively cheaply and can easily be used to gather real-time information thanks to open source programs. These new tools have allowed interactive displays to be used in gaming, education, art displays, advertising, entertainment centers, and numerous other applications. Since the interactive display is able to engage the subject and keep their interest, its use has grown tremendously. Due to the interactive displays ability to engage a subjects interest, its use has grown tremendously in these areas. By responding to the subjects physical cues dynamically, the user is more intimately involved in the program itself.

After a semester of learning about the advances in computer vision, the objective was to use the knowledge we gained and apply it to a working project. Using one of the more recent advances in computer vision, the Kinect, the goal was to create a program that would allow a user with objects to interact with a Kinect to make a virtual music player. Using hand gestures the user would be able to play sounds depending on which object the user interacted with.

Objectives

In order to create this music player, smoothing, thresholding, edge detection, and object recognition techniques that were learned throughout the semester were to be used. Once the objects were identified, differentiating them and finding the user were the next step. With the help of the depth images the objective was to find the user using thresholding and identify color using the HSV color space. Finally, to play sound, an understanding of Matlab’s music functionalities needed to be acquired.

Objectives	
Object Recognition	Identifying different objects would allow us to change the sound that we produce when the user plays the object. Also differentiating objects from the user would be key to playability.
Differ Sounds with Objects	Once we identified objects the goal was to assign those different objects their own unique sound. Creating a library of sounds to choose from was important.
Pitch and Volume Change of Sounds	Being able to identify the motions of the user and objects would allow us to then alter the sounds at our will.
Real Time Playability	It would be more interesting if this project could actually be played with a Kinect.

Preliminary Approaches

The first week was spent trying to read data from a Kinect on a computer. Attempts were made with both Mac and Windows PCs. After many hours of unsuccessful efforts the decision was made to create data with basic Google Drawing images and then video files that were recorded with an iPhone. Once initial progress was made with this data, new data was collected with Colin Lea using his Kinect.

Various techniques for identifying objects were tried before settling on the final process. The first attempt at object recognition was to apply a Gaussian smoothing filter on the RGB image produced by the Kinect then threshold the image and convert it to grayscale before applying a Canny edge detection filter. Many combinations of kernel size for the Gaussian filter and thresholds for the edge detection were tried. This allowed decent detection of objects but often missed objects of similar color to the background and detected shadows as objects. The next attempt was a system of gradient filtering with image opening and closing being applied to the RGB image. Once again many combinations of thresholds and kernel sizes were used. This system gave better object detection but still detected shadows as objects. However, this still did not give consistent enough results, especially with all the shadows in our data.

Methods

Instead of trying to apply filters and detect objects in RGB, the image was first converted to the HSV color space. Once the image was in HSV and the saturation values were isolated, the image could be used to detect objects more easily, as seen in Figure 1:

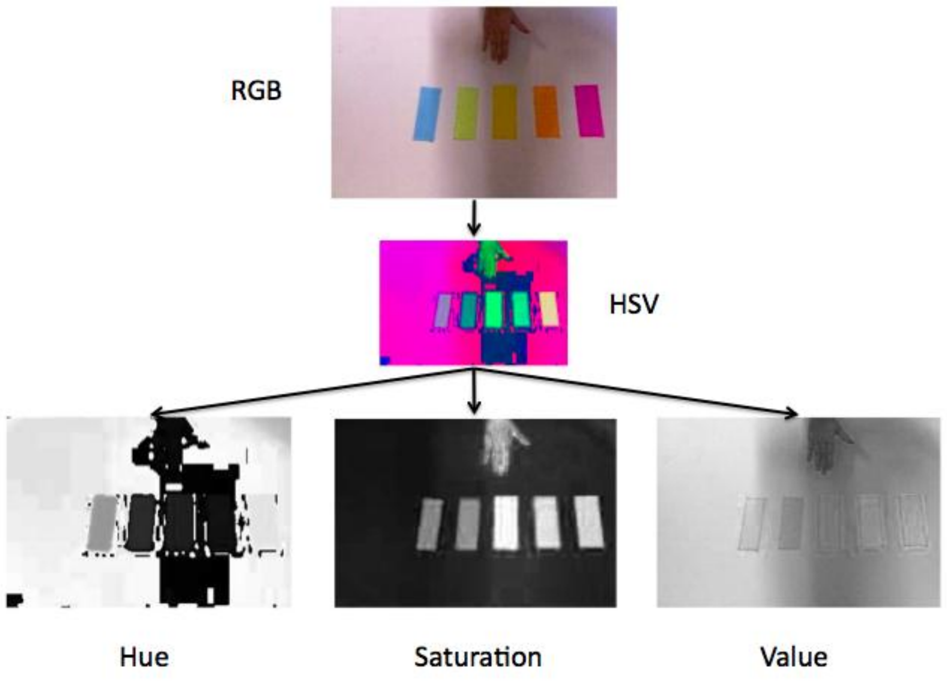

Figure 1

A Gaussian smoothing filter was applied, then a threshold, and finally a closing function. With this image it was possible to use Matlab's bwlabel() function, which detects connected components in a black and white image, as seen in Figure 2:

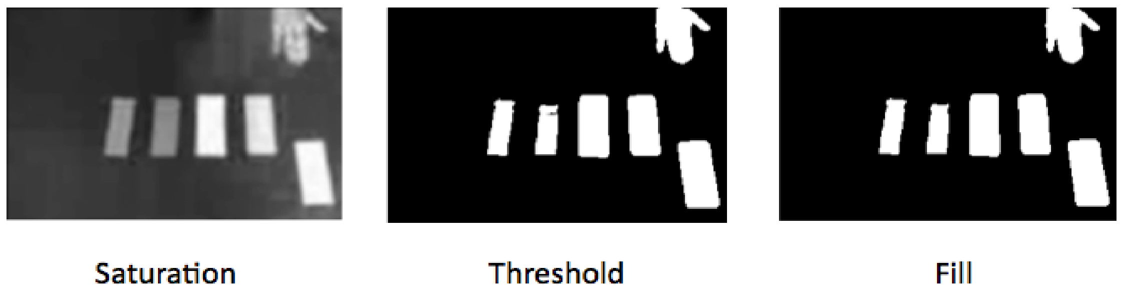

Figure 2

With the output of this function we were then able to threshold the objects to a set that was larger than 10 by 10 pixels. This allowed us to get rid of small noise objects that were not wanted. Once the objects were thresholded with the size restraint a depth restraint was placed on the objects. If the object identified was closer to the Kinect than a certain depth the object identified must be a hand or arm and was not counted as an object. The combination of RGB and depth images made it easier to detect the user and the objects.

Once the objects were detected a collection of objects were created with each object's color and position. The color was determined by averaging the Hue value of 3 pixels in the middle of each object. The position was determined by assuming that the object began at the minimum detected value from the bwlabel function and went to the maximum value that it detected.

With this information playing sounds was just a matter of detecting which objects were being played by the user. It was decided that if the user placed their hand in line with the object in the x dimension above a certain y dimension the user was attempting to play the object's sound. In order to make this happen the depth image was used to detect the hand. If the median of the mean of a rectangle's column of pixels in the depth image corresponding with an object was below a certain threshold then the object's sound is played.

To play the sounds a single 250 Hz pitch .wav file is imported at the beginning of the program. The sampling rate is manipulated by multiplying it by a constant to represent the different colored objects. The volume is then manipulated based on the objects distance from the starting line which is displayed in the output by multiplying the audio signal by a different constant.

In Figure 3, we see the output shown to the user. The first image is a depth image which displays a box where the hand is if it is playing a sound. The second image is a color representation of the objects the program is detecting. The third image is the original RGB image with boxes around each object that the program detects that could go with sounds along with a starting line which shows where the volume of the sound would be softest. The final image contains rectangles corresponding with the color of each object. The width of each rectangle corresponds with the relative size of each object to one another.

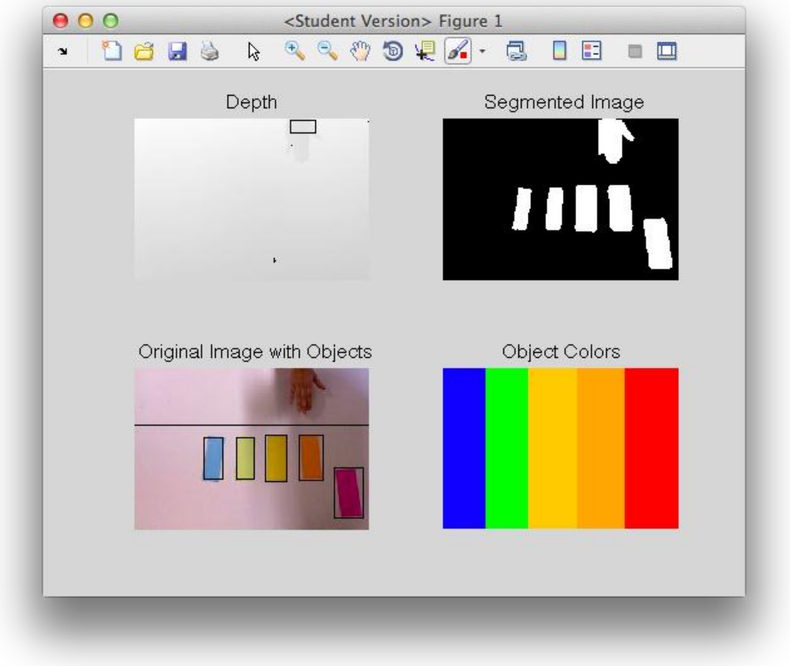

Figure 3

Results

Functionality

The project achieved many of the initial project goals that we outlined in terms of what the user is capable of performing with the programs. However, some of our initial goals had to be modified from our original intentions due to certain problems we encountered during the course of the project.

Achieved Interaction	
Object Recognition	User can place objects ("buttons") in the play space and the program will recognize the button's shape, size, color, and placement
Hand Triggered Sound	User can play sounds through the program by placing their hand below the specific button they would like to trigger
Change Pitch with Object Color	They pitch of each sound is linked to the color of the button the user places in the play space. The user can assign whatever .wav file they would like to 8 different colors
Dynamically Change Volume with Object Placement	User can dynamically change the volume of each sound while the program is running by moving the button farther or closer to the play space.

The program was able to function correctly in the environment it was designed for (a white backdrop with colored, flat buttons). However, it would not work as effectively in different environments, such as one with a noisy background or reflective surface.

Limitations

Kinect Orientation: The Kinect is assumed to be facing straight down on the playing surface, with its view normal to the surface. Due to the way the hand is recognized by the program, any large variation in viewing angle could cause the hand to not be recognized correctly.

Playing Surface: We discovered that the playing surface needed to be a plain, non-reflective surface in order for the program to run effectively. If there was too much background noise caused by the playing surface, the program might falsely identify the background as an object.

Lighting: During testing, it was found that shadows caused by the player could alter the color recognized by the program. This would then alter the sound played with the user would trigger that button.

Speed: During the demonstration, the program was dramatically slowed down by the addition of the volume change functionality. Although results could be achieved, they would not have been as effective in a real-time environment.

Conclusions

The project functions properly in the right environment and completed most of our original goals. Along with a functioning project, a better understanding of many computer vision techniques and Matlab functionalities was gained by the authors due to the hands on experience. In the future, it would be interesting to extent our program

What we learned:

- The Kinect is not just plug and play, it takes a lot of work in order to read data from it with a computer.
- In real situations, using a Gaussian function to smooth does help to reduce noise.
- Shadows can be very disruptive to object and color detection.
- HSV color space is very useful in object and color detection.
- Computers are stupid and won't listen to what you say.

Our Advice to Future Students:

- Don't bite off more than you can chew.
- Don't assume the Kinect will just work with your computer.
- Be creative.
- If you are working with showing real time images in Matlab, use 'pauses' occasionally to give your computer time to show the image.

Appendix (source code)

Run

```
% Program to start project
clc
close all


% Load images and sounds

[color depth] = videoread();

sounds = cell(1,8);

[s1,Fs1] = wavread('250HZ.wav');
s1 = .7*s1;
s1 = [s1 s1];

p1 = audioplayer(s1,.5*Fs1);
sounds{1} = p1;

p2 = audioplayer(s1,.6*Fs1);
sounds{2} = p2;

p3 = audioplayer(s1,.7*Fs1);
sounds{3} = p3;

p4 = audioplayer(s1,.8*Fs1);
sounds{4} = p4;

p5 = audioplayer(s1,.9*Fs1);
sounds{5} = p5;

p6 = audioplayer(s1,1*Fs1);
sounds{6} = p6;

p7 = audioplayer(s1,1.1*Fs1);
sounds{7} = p7;

p8 = audioplayer(s1,1.2*Fs1);
sounds{8} = p8;

ls = length(sounds);

figure;
backgroundD = median(mean(depth{1}));


% Run program as if reading video:
```

```
for l = 250:475

    % Get background depth
    m = mean(mean(depth{l}(70:120,:)));

    % Get current frame and run it through 'project'
    i = color{l};
    j = depth{l};
    [playCol vol] = project(i,j,backgroundD,m);


    % Stop sounds that shouldn't be playing
    for y = 1:ls
        if ismember(y,playCol)==0;
            stop(sounds{y})
            sounds{y} = setVolume(1,s1,((.5+.1*(y-1))*Fs1));
        end
    end

    % Play sounds that should be playing
    if playCol>0
        for p = 1:length(playCol)
            if(sounds{playCol(p)}.isplaying==0)
                temps1 = s1;
                sounds{playCol(p)} = setVolume(vol(p)*4,s1,((.5+.1*(playCol(p)-1))*Fs1));
                play(sounds{playCol(p)});
                pause(.001);
            end
        end
    end

end
```

Project()

```
% Function that does everything

function [playCol,vol] = project(videoFile,depthFile,backgroundD,m)
% clc
% close all


% Initialize variables
color = zeros(10,3);
Objects = zeros(100,5,4);
Im = videoFile;
depth = depthFile;
disparity = 80;
```

```
Ihsv = rgb2hsv(Im);     %convert to hsv
Is = Ihsv(:,:,2);       %isolate the saturation (s)
Is2 =  imfilter(Is,fspecial('gaussian',[5 5], 5)); %gaussian smoothing filter
Is3 = im2bw(Is2,.4);    %Threshold the image
Is4 = imfill(Is3,'holes');
L = bwlabel(Is4,8);     %Label connected components

%Show the segmented image
subplot(2,2,2)
imshow(L);
title('Segmented Image','FontSize',15)


% Convert to grayscale and get size of image
I = rgb2gray(Im);
[sizeY,sizeX] = size(I);
line = uint16(.35*sizeY);

% Normalize depth image
D2 = im2double(depth);
min_ = min(min(D2));
max_ = max(max(D2));


% Identify Objects
count = max(max(L));    % # of objects it thinks there is
count2 = 0;             % # of objects there really is
for z = 1:count
    [t u] = find(L==z);
    if (max(t)-min(t)>10&&max(u)-min(u)>10)&&(max(t)-min(t)<70&&max(u)-min(u)<70)
        if count2==0&&min(t)>line
            if(depth(t,u)>backgroundD-disparity)
                count2 = count2+1;
                Objects(count2,1) = min(t);
                Objects(count2,2) = min(u);
                Objects(count2,3) = max(t)-min(t);
                Objects(count2,4) = max(u)-min(u);
                Objects(count2,5) = line - (Objects(z,1)+(Objects(z,4)/2));
            end
        elseif min(t)>line&&count2>0
            if (.5*min(u)+.5*max(u))- (Objects(count2,2)+.5*Objects(count2,4))>10
                if(depth(t,u)>backgroundD-disparity)
                    count2 = count2+1;
                    Objects(count2,1) = min(t);
                    Objects(count2,2) = min(u);
                    Objects(count2,3) = max(t)-min(t);
                    Objects(count2,4) = max(u)-min(u);
                    Objects(count2,5) = line - (Objects(z,1)+(Objects(z,4)/2));
                end
            elseif (.5*min(u)+.5*max(u))- (Objects(count2,2)+.5*Objects(count2,4))<10&&count2>0
                if max(t)-min(t)>Objects(count2,3)&&max(u)-min(u)>Objects(count2,4)
```

```
                    if(depth(t,u)>backgroundD-disparity)
                        Objects(count2,1) = min(t);
                        Objects(count2,2) = min(u);
                        Objects(count2,3) = max(t)-min(t);
                        Objects(count2,4) = max(u)-min(u);
                        Objects(count2,5) = line - (Objects(z,1)+(Objects(z,4)/2));
                    end
                end
            end
        end
    end
end



% Show depth image
subplot(2,2,1)
imshow((D2-min_)/(max_-min_))
title('Depth','FontSize',15)

% Draw line in image
for q = 1:sizeX
    Im(line,q,1) = 0;
    Im(line,q,2) = 0;
    Im(line,q,3) = 0;
    Im(line+1,q,1) = 0;
    Im(line+1,q,2) = 0;
    Im(line+1,q,3) = 0;
end


% Show the original image with objects boxed
subplot(2,2,3)
imshow(Im);
title('Original Image with Objects','FontSize',15)
hold on
for z = 1:count2
    plot(rectangle('Position',[Objects(z,2),Objects(z,1),Objects(z,4),Objects(z,3)]))

    hsv = rgb2hsv(Im);

    temp(1,1:3) =
hsv(Objects(z,1)+uint16((Objects(z,3)/2)),Objects(z,2)+uint16((Objects(z,4)/2)),:);
    temp(2,1:3) =
hsv(Objects(z,1)+uint16((Objects(z,3)/2))+1,Objects(z,2)+uint16((Objects(z,4)/2))+1,:);
    temp(3,1:3) =
hsv(Objects(z,1)+uint16((Objects(z,3)/2))+2,Objects(z,2)+uint16((Objects(z,4)/2))+2,:);

    color(z,1:3,:) = mean(temp);
    color(z,1) = round(color(z,1)*100)/100;

    [colorVal playColor] = getObjColor(color(z,1:3));
    if(colorVal>0)
        color(z,1:3) = [colorVal 1 1];
        Objects(z,1,2) = playColor;
    else
        color(z,1:3) = [0 0 0];
        Objects(z,1,2) = playColor;
    end

    color(z,1:3) = hsv2rgb(color(z,1:3));

end
hold off


pos = zeros(count2+1,1);

% Show color display
subplot(2,2,4)
title('Object Colors','FontSize',15)
hold on

for z = 1:count2

rectangle('Position',[pos(z),0,uint16(Objects(z,3)*sizeX/sum(Objects(:,3))),sizeY],'FaceColor',co
lor(z,1:3),'LineStyle','none');
    pos(z+1) = pos(z) + uint16(Objects(z,3)*sizeX/sum(Objects(:,3)));
end
daspect([1,1,1])
xlim([0, sizeX])
ylim([0, sizeY])
axis off


playCol = 0;
c = 0;
vol = 0;

% Find if hand is playing
if m>739
    for z = 1:count2
        m = median(mean(depth(uint16(line - .9*line-5):uint16(line - .9*line)+10,
uint16(Objects(z,2)-10):uint16(Objects(z,2)+20))));
        if m<770&&m>backgroundD-150
            subplot(2,2,1)
            hold on
            rectangle('Position', [uint16(Objects(z,2)-10),uint16(line - .9*line-5),30,15]);
            c = c+1;
            playCol(c) = Objects(z,1,2);
            vol(c) = (Objects(z,1)-line);
            hold off
        end
    end
else
```

```
    pause(.001);
end
```

getObjColor()

```
function [colorVal playColor] = getObjColor(readColor)

% Choose which color the object is based on ranges
    if readColor(1,1)>=0 && readColor(1,1)<.05||readColor(1,1)>.8
        playColor = 1;
        colorVal = 0.025;
    elseif readColor(1,1)>=.05 && readColor(1,1)<.1
        playColor = 2;
        colorVal = .1;
    elseif readColor(1,1)>=.1 && readColor(1,1)<.15
        playColor = 3;
        colorVal = .125;
    elseif readColor(1,1)>=.15 && readColor(1,1)<.2
        playColor = 4;
        colorVal = .35;
    elseif readColor(1,1)>=.2 && readColor(1,1)<.4
        playColor = 5;
        colorVal = .45;
    elseif readColor(1,1)>=.4 && readColor(1,1)<.5
        playColor = 6;
        colorVal = .55;
    elseif readColor(1,1)>=.5 && readColor(1,1)<.62
        playColor = 7;
        colorVal = .65;
    elseif readColor(1,1)>=.62 && readColor(1,1)<.8
        playColor = 8;
        colorVal = .75;
    elseif readColor(1,3)<.12
        playColor = 0;
        colorVal = 0;
    end
```

setVolume()

```
function [AP] = setVolume(vol,s,Fs)

    AP = audioplayer(uint8(vol*s),Fs);

end
```