AV Assignment 2

Flavius Popescu, Jakub Hampl

Algorithms Used

The bulk of our system relies on background subtraction and colour thresholding. We have achieved very good performance using only very simple techniques making more complicated techniques like model based skin detection somewhat superfluous.

For background subtraction, the first attempt was to use the background image provided. However, we soon realized that there were plenty of artifacts resulting from subtle differences between this image and the training set. We investigated the use of normalized RGB, but many of the differences were probably more due to haze, camera jitter and blur.

We found that taking an average over the entire set of images provides a much better result that can be successfully used in background subtraction, with most of the juggler removed from the image. Figure 1 shows the resulting background image.

Calculating the average background image

To calculate the average background we iterate through all the images and simply average their numerical values.

```
function [avgbg] = avgall()
    files = dir('juggle1/0*.jpg');
    avgbg = zeros(size( imread('background.jpg')) , 'double' );
    for ii = 1:size(files,1)
        Image = imread(['juggle1/', files(ii).name]);
        avgbg = avgbg + double(Image);
    end

    avgbg = uint8(  avgbg / size(files,1)  );
end
```

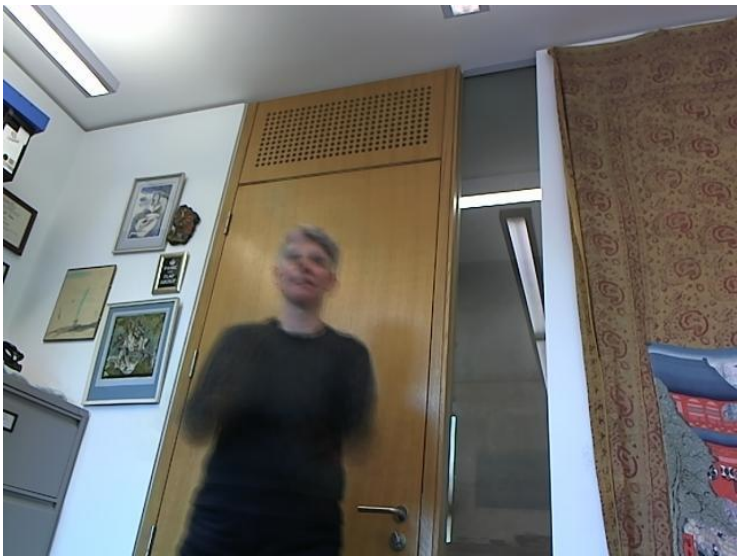

Figure 1: Average background computed over the entire set of images

When subtracting the background, the difference in pixel colour between a given frame and the averaged background is computed and thresholded for each RGB channel. The threshold values were determined empirically. During this step we also need to preserve colour information since we later detect the balls through colour thresholding.

We start by defining threshold values for each colour channel.

```
function [res] = bgdiff(m, bg)
    m = double(m);
    bg = double(bg);

    tred = 20;
    tgreen = 10;
    tblue = 20;
    res = zeros(size(m));
```

Then we loop through each pixel and compare the difference in colour of our image and the background image to the threshold value. If the difference is too big, we set the pixel to black, otherwise we keep the original colour.

```
for i=1:size(m,1)
    for j=1:size(m,2)
```

```
        if abs(m(i,j,1) - bg(i,j,1)) > tred | ...
                abs(m(i,j,2) - bg(i,j,2)) > tgreen | ...
                abs(m(i,j,3) - bg(i,j,3)) > tblue
            res(i,j,:) = m(i,j, :);
        else
            res(i,j,:) =  [0; 0; 0];
        end
    end
end
```

Since the difference is calculated from an average image, there are plenty of artifacts that can confuse the later algorithm. We make a copy of our image and threshold it to black and white. Then we erode the images and use the result as a mask to further get rid of areas with colours.

```
  res = uint8(res);
  n = bwmorph(im2bw(res,0.15), 'erode', 1);
  res = uint8(double(res) .* repmat(n, [1 1 3]));
end
```

Figure 2 shows the result of the background subtraction. Notice that there are still many areas that surface as new foreground, however they are small in area size and can be removed with an erosion operation. Figure 3 reveals a cleaned up version. A large number of skin pixels is also removed.

Finally we need to identify the balls, and for that we convert the image into the HSV colourspace and threshold on the hue channel, for each ball. We measured average hues of the balls and chose threshold values in order to detect each of the three balls.

This alone turns out to work remarkably well as discussed below, however to improve performance further, we decided to use our tracking information. One constraint on the domain seems to be that the balls only travel a certain distance in one time step, therefore we measured the mean distance and standard deviation for the ground truth set. We used the information in order to extract a region of the image surrounding each ball position in the current frame. We decided to set the area to 3 standard deviations of the mean of the distances between successive frames of the true data set (the complete data set fits within 2.5 standard deviations).

After thresholding is applied, we simply take the biggest blob from the region and compute its centroid. The largest area remaining after the threshold stage corresponds to each of the three balls, since their colour is quite distinct from the rest of the image. Since the threshold is not applied to the entire image, the risk of falsely detecting a ball far away from the expected position is eliminated.

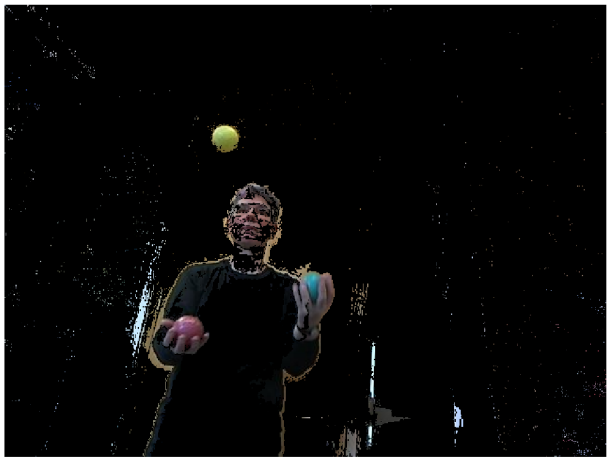

Figure 2: Result of background subtraction operation, without erosion.

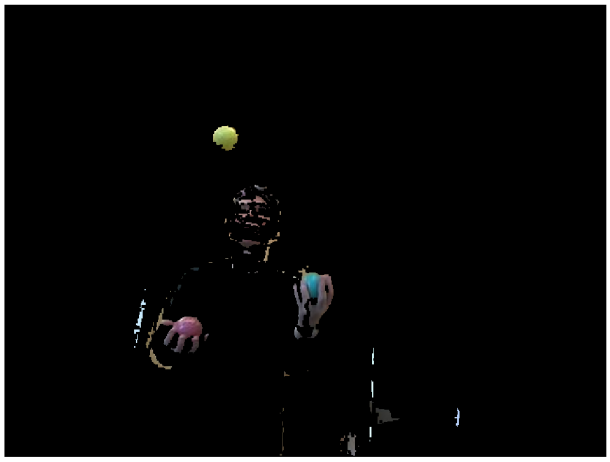

Figure 3: Result of background subtraction operation, with erosion.

This allows us to get rid of some complete misclassifications like detecting the reflection of the ball in the window.

```
function [C] = biggest_center(BW)
```

Find the centroid of the biggest object

```
CC = bwconncomp(BW);
S = regionprops(CC,'Centroid');
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
C = S(idx).Centroid;

end
```

Figures 5, 6 and 7 show the result of applying the threshold on the expected regions for the frame in figure 4.

Figure 4: The detection algorithm applied to a frame. The coloured crosses reveal the computed centroids for each of the three balls.

For visualization, figure 8 illustrates the three thresholding results overlaid.

Figure 5: Thresholding for the red ball pixels, detected and marked in white. Notice how part of the skin on the fingers of the juggler is also removed.

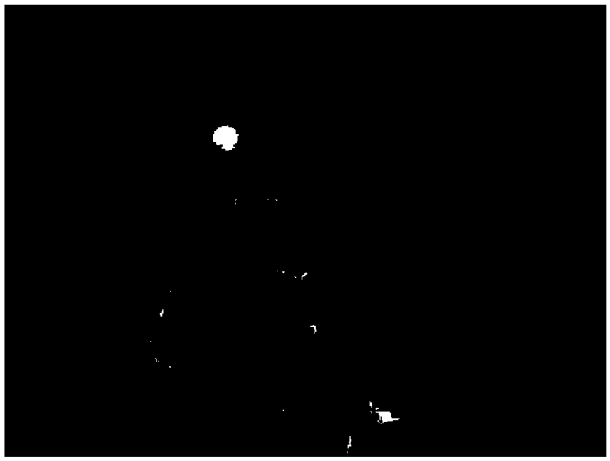

Figure 6: Thresholding for the yellow ball pixels, detected as the round white shape.

Figure 7: Thresholding for the blue ball pixels. The detected shape is less clear than in the yellow ball case since it is covered by the juggler's fingers.

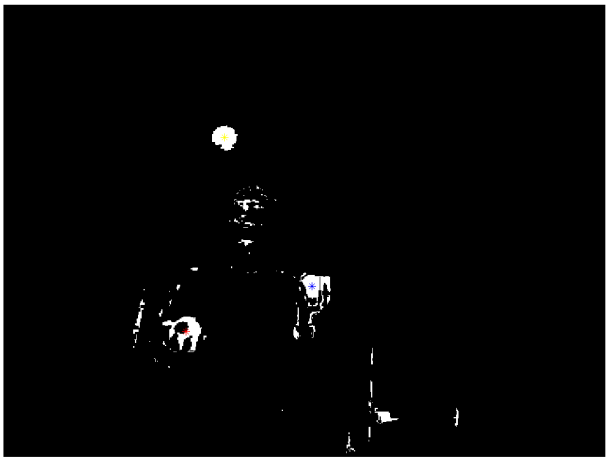

Figure 8: Result of the thresholding stage, overlaid for each colour segment of interest.

Performance

To evaluate performance two metrics are used. Firstly we count the number of misclassifications, which we define as any detection that is more then 10 pixels away from the true data set. Secondly, we track the average (Euclidean) distance from the true data set.

The full system achieves 99.327% within 10px of true center (97.980% for the red ball, 100.000% for the yellow ball, 100.000% for the blue ball). Average distance from true center was 1.836px (SD=1.830).

Without using the position tracking history, the system performs slightly worse, but still achieves 97.306% within 10 pixels of true center and in average is 5.975 pixels (SD=36.486799) from the true center.

Figures and illustrate examples for which the error distance is larger than 10 pixels.

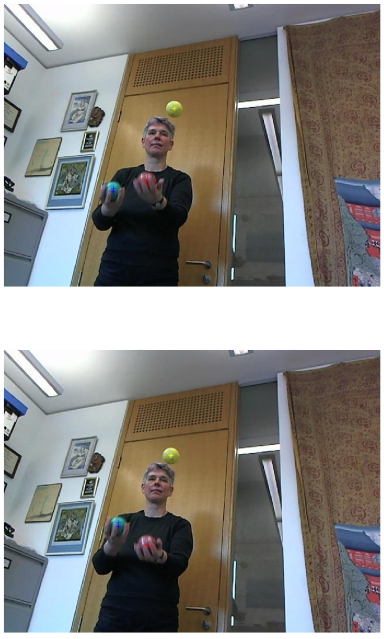

Figure 9: Examples of unsuccesfull recogniton

Figures 10, 11 and 12 show the plotted trajectories of each of the red, yellow and blue balls, respectively. Figure 13 shows the trajectories overlaid.

Figure 10: Trajectory for the red ball.

Figure 11: Trajectory for the yellow ball.

Figure 12: Trajectory for the blue ball.

Figure 13: Trajectories overlaid.

Discussion

One of the problems with using an average image is that if the system should be used as an online system, finding the balls in the first couple of images can be quite difficult. As a simulation of how this behaves we start with the basic background image and as more images come in we iteratively improve our background image. This makes detection in the first roughly five images rather inaccurate, but converges quite quickly to fairly smooth detection.

Iteratively we grow the store variable with all the images we have seen. Store gets initiated to: store = zeros(width, height, 3, 0); Then we apply this function with every image we see.

```
function [store, bg] = avg_adaptive(store, curr)
    store(:, :, :, size(store, 4) + 1) = double(curr);
    bg = zeros(size(store, 1), size(store, 2), 3);

    for i = 1:size(store, 4)
       bg = bg + double(store(:, :, :, i));
    end
    bg = uint8(round(bg ./ size(store, 4)));
end
```

The results we get from doing this online decrease to 88.215% of detected balls within 10px of true center.

The unsuccessful detections illustrated above occur due an inaccurate calculation of the blob centroids, after the thresholding stage. This happens when the balls are partially concealed by the juggler's fingers which deteriorates the visible round shape. In addition, not all skin pixels are removed. A system based on histograms ([1]) or Gaussian models ([2]) could be employed for skin detection. Then, partial shape matching can be performed on the remaining shape for circular segments corresponding to the ball edge, which can be used to better approximate the centroid.

Appendix: Remaining Source Code

Task 1: Detection

The algorithm starts by calculating the average background from all the images gathered (see `avgall` bellow), and doing some setup. Also an option for skipping this step is presented.

```
function main(skip_detection)
    avgbg = avgall;

    fg = figure(1);
    files = dir('juggle1/0*.jpg');
    tracks = zeros(size(files,1), 6);
    wb = waitbar(0, 'Initializing');
    count = size(files,1);

        thresh_reg = 40;

        kinit = 0;

    if nargin == 1 & skip_detection
        tracks = load('track.mat');
        tracks = tracks.tracks;
    else
        for ii = 1:count
            tic;
            Image = imread(['juggle1/', files(ii).name]);
```

The algorithm processes each input image. It substracts the average background image, keeping the colours intact in places where the difference is large enough. Then we calculate thresholds for the individual color values.

```
        diff = bgdiff(Image, avgbg);

                    diff = double(diff);
                    if kinit == 1
                        yc = round(tracks(ii-1, [1]));
                        yr = round(tracks(ii-1, [2]));
                        bc = round(tracks(ii-1, [3]));
                        br = round(tracks(ii-1, [4]));
                        rc = round(tracks(ii-1, [5]));
                        rr = round(tracks(ii-1, [6]));

                        y_rows = [max(yr - thresh_reg, 1) : min(yr + thresh_reg, size(diff,1
                        y_cols = [max(yc - thresh_reg, 1) : min(yc + thresh_reg, size(diff,2
                        b_rows = [max(br - thresh_reg, 1) : min(br + thresh_reg, size(diff,1
                        b_cols = [max(bc - thresh_reg, 1) : min(bc + thresh_reg, size(diff,2
                        r_rows = [max(rr - thresh_reg, 1) : min(rr + thresh_reg, size(diff,1
                        r_cols = [max(rc - thresh_reg, 1) : min(rc + thresh_reg, size(diff,2

                        y_mask = zeros(size(diff));
```

```
                b_mask = zeros(size(diff));
                r_mask = zeros(size(diff));

                y_mask(y_rows, y_cols, :) = 1;
                b_mask(b_rows, b_cols, :) = 1;
                r_mask(r_rows, r_cols, :) = 1;

                ydiff = double(diff .* double(y_mask));
                bdiff = double(diff .* double(b_mask));
                rdiff = double(diff .* double(r_mask));
            else
                ydiff = diff;
                bdiff = diff;
                rdiff = diff;
            end

            kinit = 1;

        Y = thresh_yellow(ydiff);
        B = thresh_blue(bdiff);
        R = thresh_red(rdiff);
```

Then we calculate the centroid of the biggest continous blob in our thresholded image and store them in our tracking matrix.

```
        cy = biggest_center(Y);
        cb = biggest_center(B);
        cr = biggest_center(R);
        tracks(ii, :) = [cy, cb, cr];

        set(fg, 'name', files(ii).name);
        imshow(R + Y + B);
        hold on
        plot(cy(1), cy(2), 'y*', cb(1), cb(2), 'b*', cr(1), cr(2), 'r*');
        drawnow;


        pause(1 - toc)
        perc = ii/(size(files,1) * 2 + 4);
        waitbar(perc,wb,sprintf('%d%% completed...',round(perc * 100)));
    end
    save('track.mat', 'tracks');
end
```

Task 2: Tracking

Tracking is largely done in the previous step, here we visualise the individual required images as well as an overall image of the juggling.

```
figure(1);
imshow(imread('background.jpg'))
hold on
plot(tracks(:, 1), tracks(:, 2), 'y', tracks(:, 3), ...
    tracks(:, 4), 'b', tracks(:, 5), tracks(:, 6), 'r')
print('-dpng', 'report/tracking')

figure(1);
imshow(imread('background.jpg'))
hold on
plot(tracks(:, 1), tracks(:, 2), 'y')
print('-dpng', 'report/tracking-yellow')

figure(1);
imshow(imread('background.jpg'))
hold on
plot(tracks(:, 3), tracks(:, 4), 'b')
print('-dpng', 'report/tracking-blue')

figure(1);
imshow(imread('background.jpg'))
hold on
plot(tracks(:, 5), tracks(:, 6), 'r')
print('-dpng', 'report/tracking-red')
```

Task 3: Evaluation

We load the true data and calculate euclidian distance from the true data and our tracked data. We then count the number of images that were tracked more then 10px off as well as the average error.

```
ev = load('gt1.mat');
gt = ev.gt1';

yellow_d = sqrt(sum(gt(:,[7,6]) - tracks(:,[1,2]), 2) .^ 2);
blue_d   = sqrt(sum(gt(:,[5,4]) - tracks(:,[3,4]), 2) .^ 2);
red_d    = sqrt(sum(gt(:,[3,2]) - tracks(:,[5,6]), 2) .^ 2);
```

```
yellow_correct = sum(yellow_d < 10);
blue_correct   = sum(blue_d < 10);
red_correct    = sum(red_d < 10);

overall_d = [yellow_d; blue_d; red_d];
overall   = (yellow_correct + blue_correct + red_correct) / (count * 3);

disp(sprintf('%f%% within 10px of true center (R=%f%%, Y=%f%%, B=%f%%)', ...
    overall * 100, red_correct / count * 100, yellow_correct / count * 100, ...
    blue_correct / count * 100));

disp(sprintf('Average distance from true center %fpx (SD=%f)', ...
    mean(overall_d), std(overall_d)));

fg = figure(1);
delete('fixme/*.png');
```

Finally we will display each image with the tracked center and the real center, saving to disk those that are more then 10px off.

```
for ii = 1:count
    tic;
    Image = imread(['juggle1/', files(ii).name]);
    set(fg, 'name', files(ii).name);
    imshow(Image);
    hold on
    plot(tracks(ii, 1), tracks(ii, 2), 'y+', ...
        tracks(ii, 3), tracks(ii, 4), 'b+', ...
        tracks(ii,5), tracks(ii, 6), 'r+');
    plot(gt(ii, 7), gt(ii, 6), 'y*', ...
        gt(ii,5), gt(ii, 4), 'b*', ...
        gt(ii,3), gt(ii,2), 'r*');
    drawnow;
    pause(1 - toc)
    perc = (ii + size(files,1))/(size(files,1) * 2 + 4);

    [pathstr, name, ext] = fileparts(files(ii).name);

    if yellow_d(ii) > 10
        print('-dpng', ['fixme/' name 'yellow']);
    end
    if blue_d(ii) > 10
        print('-dpng', ['fixme/' name 'blue']);
    end
```

```
    if red_d(ii) > 10
        print('-dpng', ['fixme/' name 'red']);
    end

    waitbar(perc,wb,sprintf('%d%% completed...',round(perc * 100)));
end

close(fg);
close(wb);

end
```

Thresholding functions

```
function [res] = thresh_blue(m)
  mhsv = rgb2hsv(m);
  h = mhsv(:,:,1);
  res = h > 0.47 & h < 0.65;
end


function [res] = thresh_yellow(m)
  mhsv = rgb2hsv(m);
  h = mhsv(:,:,1);
  res = h > 0.13 & h < 0.25;
end


function [res] = thresh_red(m)
  mhsv = rgb2hsv(m);
  h = mhsv(:,:,1);
  res = h > 0.9;
end
```

References

[1] Jones, Michael J., and James M. Rehg. "Statistical color models with application to skin detection." In Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on., vol. 1. IEEE, 1999.

[2] Yang, Ming-Hsuan, and Narendra Ahuja. "Gaussian mixture model for human skin color and its application in image and video databases." In Proc.

SPIE: Storage and Retrieval for Image and Video Databases VII, vol. 3656, pp. 458-466. 1999.