Advanced Vision Assignment 1

Moataz Elmasry and Monica Paun

February 13, 2014

Abstract

In this report, we outline our attempt at identifying and tracking a set of marbles moving across a fixed featureless background. The first section of the report discusses the detection of the marbles in each individual frame, the second section discusses the tracking of the marbles across consecutive frames, and the third and final section evaluates our results against the ground truth results. For each of the tasks, we describe the algorithm used, and its performance.

Chapter 1

Marble Detection

1.1 Algorithm

1.1.1 Object-oriented Approach

1.1.2 Background Subtraction

It is assumed that the first image of the dataset represents the background of the scene. The function `sub_background.m` takes a pair of images and a threshold and returns:

- A set of all the pairs of coordinates where the correspond pixels are more different than the threshold;
- A set of all the pairs of the values in the non-background image of all the three color channels of the points mentioned earlier

1.1.3 Post-processing

The `identifyMarbles` function of every `myImage` object places a list of `myMarble` objects in the `marbles` field of the former mentioned class of the two. In order to achieve this, the following steps are taken:

- given the list of pairs where a significant difference between the background image and the current image is found, the surface described by them is smoothed with the aid of the `strel('disk',3)` function
- The resulting blobs are split into connected components with the `bwconncomp` function
- The attributes of the connected components are requested with the `regionprops(obj.CC,'Centroid', 'Area', 'PixelList')`; thus, for each component, its centroid, area and list of pixel coordinates that compose it are noted.
- the centroid of each connected component is added as a `marble` centroid to the list of `marbles` in the `image` object; each one of these detected marbles are assigned an unique id.

1.1.4 Connected component area splitting

As the marbles are colliding at times in their motion, some of the connected components correspond to multiple balls. In order to avoid these, the following procedure has been attempted:

- In the case the ellipsoid axes corresponding to the connected component have a ratio of 1:1.8 or more, assume the component correspond corresponds to more balls
- Make a 50 bin histogram of the black and white values of the pixels from the component's `PixelList` attribute.
- Smooth the histogram by convolution with a Gaussian filter, as indicated in the IVR course in this type of case.
- Take the local minima of the smoothed histogram to represent limits between the range of colour corresponding to each marble in the blob.
- Split the `PixelList` locations in these newly defined bins and compute a centroid for each of them; assign each of these centroids to the center of a ball, whom an unique ID must be given.

  Although this method seemed promising on certain examples where a connected component only encompassed two balls with colours that are quite different, the main problem of this approach were shadows. As the marbles are quite transparent, there will be a noticeable amount of shadows around them; thus, the grey points corresponding to the outside shadows will be easily mistaken with the grey color the transparent part of the marbles looks like. This grey points will usually be all added to one of the bins of the assumed subcomponents, shifting its centroid towards the centroid of the whole connected component. Being able to deal with this kind of the situation, in order to make use of this approach, seems possible, but not in the immediate time frame of this assignment, as a considerable array of variables needs to be tuned.

1.1.5 Marble Identification

1.2 Performance Analysis

Normalized vs non-normalized

1.2.1 Successful Detections

1.2.2 Unsuccessful Detections

Chapter 2

Marble Tracking

2.1 Algorithm

2.2 Performance Analysis

2.2.1 Successful Detections

2.2.2 Unsuccessful Detections

Chapter 3

Final Evaluation and Discussion

Chapter 4

Code Appendix

4.1 Main script

```
%-------------------------------------------------------------------------
%Workspace clean-up

clc;     %Clear the command window.
close all; %Close all figures (except those of imtool).
clear; %Erase all existing variables from workspace.
clearvars; %Remove all stored variables from memory.
clear classes; %Remove all stored class objects.
%-------------------------------------------------------------------------
%Add classes and functions to path
addpath('my_classes', 'my_functions', 'saved_variables');
%-------------------------------------------------------------------------
%DATASET SELECTOR
dataset = 'dataset';

%NUMBER OF IMAGES
num_images = 71;

%TASK SELECTOR
%Select which task you want the code to accomplish:
% 1 --> Marble Detection
% 2 --> Tracking
task = 2;

%DIFFERENCE THRESHOLD
diffThreshold = 40;

%MAX MARBLE CORRESPONDENCE
maxDifference =  30;

%BACKGROUND IMAGE
backgroundImage = myImage();
```

```
backgroundImage.dataset = dataset;
backgroundImage.number = 1;
backgroundImage = backgroundImage.generatePath();
backgroundImage = backgroundImage.readImage();

%--------------------------------Task 1---------------------------------------------
if (task == 1)

    track_image = backgroundImage.data;

    for imageNum = 1 : num_images

        %Initialize target image
        image = myImage();
        image.dataset = dataset;
        image.number = imageNum;
        image = image.generatePath();
        image = image.readImage();

        %Perform background subtraction on image
        image = image.removeBackground(backgroundImage.data, diffThreshold);

        %If this is the first image, use it as the previous image in
        %marble tracking
        if (imageNum == 1)
            prevImage = myImage();
            prevImage.dataset = dataset;
            prevImage.number = imageNum;
            prevImage = prevImage.generatePath();
            prevImage = prevImage.readImage();
            prevImage = prevImage.removeBackground(backgroundImage.data, ...
                diffThreshold);
            prevImage = prevImage.identifyMarbles();
        end

        %Identify location of marbles in image
        image = image.identifyMarbles();
        image = image.locateClosestMarble(prevImage, maxDifference);

        %Initialize final image to be labeled and displayed
        finalImage = image.data;

        %Radius of circles to draw around identified marbles
        radius = 10;

        %Draw circles around each of the detected marbles. Radius of each
        %circle is 10 pixels
        for marble = 1 : size(image.marbles,2)

            finalImage = drawCircle(finalImage,image.marbles(marble).com, ...
                radius,'r',1000);
        end

%           for marble = 1 : size(image.marbles,2)
%               display(image.marbles(marble).ID);
%           end
%           subplot(2,2,1), imshow(image.data);
%           subplot(2,2,2), imshow(image.preprocessed);
%           subplot(2,2,3), imshow(image2.preprocessed);
        imshow(finalImage);
        pause(1);

        prevImage = image;
    end
end

%-------------------------------Task 2----------------------------------------
if (task == 2)

    %Initialize image to contain track of each marble
    track_image = backgroundImage.data;

    %Array of colours
    colours = ['y'];

    current_colour = 1;

    for imageNum = 1 : num_images

        %Initialize target image
        image = myImage();
        image.dataset = dataset;
        image.number = imageNum;
        image = image.generatePath();
        image = image.readImage();

        %Perform background subtraction on image
        image = image.removeBackground(backgroundImage.data, diffThreshold);

        %If this is the first image, use it as the previous image in
        %marble tracking
        if (imageNum == 1)
            prevImage = myImage();
            prevImage.dataset = dataset;
            prevImage.number = imageNum;
            prevImage = prevImage.generatePath();
            prevImage = prevImage.readImage();
            prevImage = prevImage.removeBackground(backgroundImage.data, ...
                diffThreshold);
            prevImage = prevImage.identifyMarbles();
        end

        %Identify location of marbles in image
        image = image.identifyMarbles();
        image = image.locateClosestMarble(prevImage, maxDifference);

        %Draw line between marbles identified as having the same ID between
        %frames
        for marble = 1 : size(image.marbles,2)
            for prevMarble = 1 : size(prevImage.marbles,2)

                if (image.marbles(marble).ID == prevImage.marbles(prevMarble).I

                    if isempty(prevImage.marbles(prevMarble).colour)
                        %Assign marble a coloured track
                        image.marbles(marble).colour = colours(current_colour);
                        %Increment next colour. Start again at zero if we run
                        %out of colours in the array
                        current_colour = mod(current_colour, 1) + 1;
                    end

                    track_image = drawLine(track_image, image.marbles(marble).c
                        prevImage.marbles(prevMarble).com, ...
                        colours(current_colour), 1000);
                end
            end
        end

        prevImage = image;
    end
    imshow(track_image);
end
```

4.2 myImage class

```
classdef myImage
    %IMAGEHANDLE Class representing individual image frames
%--------------------------------------------------------------------------
    properties

        %Dataset to which image belongs
        dataset;

        %Image number in the dataset
        number;

        %Path to image
        path;

        %Pre-defined image height and width
        height = 480;
        width = 640;

        %Stores image pixel information in a heightxwidthx3 variable,
        %values are in uint8 format
        data;

        %Stores preprocessed version of the image
        preprocessed;

        %Background subtracted version of this image
        diff;

        %Binary version of the background subtracted image
        binaryDiff;

        %Connected components object of this image
        CC;

        %Region properties of all the components in the image
        rProps;

        %Array containing marble objects currently in the image.
        %Initialized to a pre-defined maximum marbles per image.
        marbles;
    end
%-----------------------------------------------------------------------
    methods

    function obj = myImage()
    %Class construtor. Avoid requiring initialization parameters for
    %greater flexibility

         obj.marbles = myMarble.empty(18, 0);
    end

    function obj = generatePath(obj)
    %Given a number, this function returns the corrosponding image
    %name. dataset should be the name of the dataset's directory


            %Convert image number to string in preperation for
            %concatination.
            str_number = num2str(obj.number);

            %Generate name for image path
            obj.path = strcat(obj.dataset,'/',str_number,'.jpg');
    end

    function obj = readImage(obj)
```

```
%Reads image information and stores it in this object, returns
%error if image path not generated yet.

        obj.data = imread(obj.path);
end

function obj = removeBackground(obj, background, threshold)
%Given a background and a threshold, perform background subtraction
%on this image to obtain the resulting rgb and binary versions.

[obj.diff, obj.binaryDiff] = sub_background(background, ...
                                obj.data, threshold);
end

function obj = identifyMarbles(obj)
%Attempts to identify all marbles in this image. Image must have
%background subtracted versions removed first. Stores list of
%marble objects in this image. Marbles are given an ID starting
%with the frame's number as the prefix.

%for each conencted component, find the center of mass,inialize
%a marble component, add it to the array marbles.

    %Make sure background subtracted versions of image exist
    if (isempty(obj.diff) || isempty(obj.binaryDiff))
        error('Remove background from image first!');
    else

        %Create disk image structuring with radius 3
        se = strel('disk',4);

        %Apply image open and store in this image's preprocess
        %variable
        obj.preprocessed = imopen(obj.binaryDiff, se);

        %Find connected components
        obj.CC = bwconncomp(obj.preprocessed);

        %Compute region properties of all marbles
        obj.rProps = regionprops(obj.CC,'Centroid', 'Area', ...
                'PixelList', 'MajorAxisLength', 'MinorAxisLength');

        %Loop through each connected components, identifying and
        %initializing marble objects
        id = 1;
        for cc = 1 : size(obj.CC.PixelIdxList, 2)

            % If the ratio of the major axis is a certain ratio
            % higher than the minor axis, then treat this as two
            %marbles
```

```
%                       axisRatio = obj.rProps(cc).MajorAxisLength / ...
%                                        obj.rProps(cc).MinorAxisLength;
%                       if axisRatio > 1.35
%
%                           %Do histogram.
%                           pixel_list = obj.rProps(cc).PixelList;
%                           [a,~] = size(pixel_list);
%                           bw_img = zeros(obj.height, obj.width);
%
%                           for iter = 1 : a
%                               bw_img(pixel_list(iter, 2), pixel_list(iter, 1)) =
%                                    sum(obj.data(pixel_list(iter, 2), pixel_list(
%                           end
%
%                           bw_array = reshape(bw_img, obj.height*obj.width, 1);
%                           bw_array(bw_array==0) = [];
%                           first_hist = hist(bw_array, 50);
%
%                           filter = gausswin(50, 6);
%                           filter = filter/sum(filter);
%                           smooth_hist = conv(filter,first_hist);
%
%                           %Find the valley between the two highest peaks.
%                           inv_hist = 1.01*max(smooth_hist) - smooth_hist;
%                           [~, locsmin] = findpeaks(inv_hist);
%
%                           %Distribute points between the peaks.
%                           if (length(locsmin) == 1)
%                               tresh = locsmin;
%                               bin1x = 0;
%                               bin1y = 0;
%                               bin2x = 0;
%                               bin2y = 0;
%                               sum1num = 0;
%                               sum2num = 0;
%                               for iter = 1 : a
%                                   aux = sum(obj.data(pixel_list(iter, 2), pixel
%                                   if (aux > tresh)
%                                       sum1num = sum1num + 1;
%                                       bin1x = bin1x  + pixel_list(iter, 2);
%                                       bin1y = bin1y  + pixel_list(iter, 1);
%                                   else
%                                       sum2num = sum2num + 1;
%                                       bin2x = bin2x  + pixel_list(iter, 2);
%                                       bin2y = bin2y  + pixel_list(iter, 1);
%                                   end
%                               end
%
%                               if (sum1num > 0)
%                                    bin1x = bin1x / sum1num;
%                                     bin1y = bin1y / sum1num;
%                                end
%
%                                if (sum2num > 0)
%                                     bin2x = bin2x / sum2num;
%                                     bin2y = bin2y / sum2num;
%                                end
%
%                                dist = ((bin2x - bin1x)^2 + (bin2y - bin1y)^2)^0.
%
%                                if (dist > 8 && sum2num > 0 && sum1num > 0)
%                                    marble = myMarble();
%                                    marble = marble.assignID((obj.number*100) + id
%                                    marble = marble.assignCOM([bin1y, bin1x]);
%                                    marble = marble.calculateSumRB(obj.data);
%                                    obj.marbles(id) = marble;
%                                    id = id + 1;
%
%                                    marble = myMarble();
%                                    marble = marble.assignID((obj.number*100) + id
%                                    marble = marble.assignCOM([bin2y, bin2x]);
%                                    marble = marble.calculateSumRB(obj.data);
%                                    obj.marbles(id) = marble;
%                                    id = id + 1;
% %                                else
% %                                    marble = myMarble();
% %                                    marble = marble.assignID((obj.number*100) +
% %                                    marble = marble.assignCOM([(bin1y+bin2y)/2,
% %                                    obj.marbles(id) = marble;
% %                                    id = id + 1;
%                                end
%                            end
%                            else
                           %Marble id is the number of the image at which it was
                           %identified concatenated to the order at which it is
                           %identified in a given image
                           marble = myMarble();
                           marble = marble.assignID(obj.number*100 + id);


                           %Assign the center of mass of marble
                           marble = marble.assignCOM(obj.rProps(cc).Centroid);
                           marble = marble.calculateSumRB(obj.data);

                           %Loop through each existant marble object and check for
                           %duplicate ID.
                           for marbleNum = 1 : size(obj.marbles,2)

                                if (marble.ID == obj.marbles(marbleNum).ID)
                                    error('Marble␣with␣that␣ID␣already␣exists!');
```

```
                            end
                        end
                        %Add this marble to our list of detected marbles
                        obj.marbles(id) = marble;
                        id = id + 1;
            end
        end
    end

    function obj = locateClosestMarble(obj, prevImage, maxDifference)
    %Given an image with a detected marbles and this image, the
    %function will attempt to find the corrosponding marble in this
    %image

        %If marbles have been detected in the previous image
        if ~isempty(prevImage.marbles)

            %Loop through each marble in the previous image trying to
            %find its corrospondence in this image
            for marble = 1 : size(obj.marbles,2)

                comMarble = obj.marbles(marble).com;

                for prevMarble = 1 : size(prevImage.marbles,2)

                    comPrevMarble = prevImage.marbles(prevMarble).com;

                    distance = ((comMarble(1) - comPrevMarble(1))^2 + ...
                        (comMarble(2) - comPrevMarble(2))^2)^0.5;

                    if (abs(obj.marbles(marble).sumRB - ...
                            prevImage.marbles(prevMarble).sumRB) ...
                                < maxDifference) && (distance < 35)

                            %Assign this marble the ID of its
                            %correspondence in the previous image
                            obj.marbles(marble).ID = ...
                                prevImage.marbles(prevMarble).ID;

                            %Copy colour used to track previous marble
                            %to this marble
%                                 obj.marbles(marble).colour = ...
%                                     prevImage.marbles(prevMarble).colour;

                            %Find speed of this marble
                            obj.marbles(marble).speed = distance;
                    end

                end
            end
        end


    end
end
end
```

4.3 myMarble class

```
classdef myMarble
    %MYMARBLE Class representing an individual marble
%-------------------------------------------------------------------------------
    properties

        %Identification number of marble
        ID;

        %Center of mass of the marble
        com;

        %2D histogram of the r/g components from normalised RGB values
        histogram;

        %Sum of red and blue values in this marble's center
        sumRB;

        %Colour used to track marble
        colour;

        %Speed at which marble is travelling. This is basically the
        %distance travelled per frame
        speed;

    end
%-------------------------------------------------------------------------------
    methods


        function obj = myMarble()
        %Class constructor.
        end

        function obj = assignID(obj, ID)
        %Assign an ID to this marble

            obj.ID = ID;
        end

        function obj = assignCOM(obj, com)
        %Assign a center of mass linear coordinate to this marble
```

```
            obj.com = com;
        end

        function obj = calculateSumRB(obj, image)
        %Given an RGB image and this marble with an assigned center of mass
        %calculate the sum of red and green values

            marbleX = round(obj.com(1));
            marbleY = round(obj.com(2));
            obj.sumRB = double(image(marbleY, marbleX, 1)) + ...
                        double(image(marbleY, marbleX, 3));
        end
    end

end
```

4.4 subBackground function

```
function [diffImage, binaryDiffImage] = sub_background(image1, image2, ...
                                                       diffThreshold)
%Given two images and a threshold, this function attempts to subtract
%common pixels from both images, returning the difference image in RGB
%format and in binary format.
%
%image1 is usually a background image and image2 is the one containing
%objects you are after.
%
%Both images must be of the same size.

    %Find image dimensions
    [height, width, depth] = size(image1);

    %Find the difference in values between the two images
    diffValues = sum(abs(image1 - image2), 3);

        %Matrix to hold resulting background subtracted image
    diffImage = uint8(zeros(height, width, 3));
    binaryDiffImage = zeros(height, width);

    %Populate difference matrices
    for iColumn = 1 : width
        for iRow = 1 : height

            %If value greater than threshold, that means a non-background
            %object was detected.
            if diffValues(iRow, iColumn) >= diffThreshold
               %Store results in both unit8 format and binary format
               diffImage(iRow, iColumn, :) = ...
                     image2(iRow, iColumn, :);
```

```
                binaryDiffImage(iRow, iColumn) = 1;
            end
        end
    end
end
```

4.5 rgbnormalize function

```
function output = rgbnormalize(image_uint)
    %Given an image, this function returns the an rgb normalized version
    %of the image.

   image = double(image_uint);
   [H,W,D] = size(image);

   %Temporary holder initialized to zeros
   temp = zeros(H,W,D);

   %Loop going through every pixel in the image
   for i=1:H
       for j=1:W

           %Find average of all colours for
           %that pixel
           csum = image(i,j,1) + image(i,j,2) + image(i,j,3);

           temp(i,j,1) = (image(i,j,1) / csum);
           temp(i,j,2) = (image(i,j,2) / csum);
           temp(i,j,3) = (image(i,j,3) / csum);
       end
   end
   output = uint8(temp .* 255);
end
```

4.6 drawCircle function

```
function image = drawCircle(image,center,r,color,npoints)
%Function that takes in an RGB image, a 2x1 array representing a center, a
%radius, a colour and a specified number of points, it will produce an
%image with the specified circle superimposed.
%Suggested value for npoints is 100-2000, the more the better!

    %theta value to find all points on the circumference of the circle
    theta=linspace(0,2*pi,npoints);

    %Array containing just the radius value, needed to compute all of the
    %points
    points = ones(1,npoints) .* r;
    %Find the coordinates that form the circle's circumference
    [X,Y] = pol2cart(theta,points);

    %Adjust all coordinates so that they are consistent with the specified
    %center and round them. Furthermore, change any value that is 0 to 1 to
    %avoid calling a zero or negative index.
    X= round(X+center(1));
    X(X <= 0) = 1;
    Y= round(Y+center(2));
    Y(Y <= 0) = 1;

    %Colour specifier
    colourSpec = zeros(1,3);
    if (color == 'r')
        colourSpec(1,1) = 255;
    end

    %Go through computed XY coordinates, changing them to the specified colour.
    for i=1:npoints
        image(Y(i),X(i),1) = colourSpec(1);
        image(Y(i),X(i),2) = colourSpec(2);
        image(Y(i),X(i),3) = colourSpec(3);
    end
end
```

4.7 drawLine Function

```
function image = drawLine(image, point1, point2, color, npoints)
%Function that takes in an RGB image, two 2x1 array representing points
%that need to be connected, a colour and a specified number of points. It
%will produce an image with the specified circle superimposed.
%Suggested value for npoints is 100-2000, the more the better!

    %Find linear space for x coordinates of the two points
    yLin = linspace(point1(2), point2(2), npoints);

    %Find linear space for y coordinate of the two points
    xLin = linspace(point1(1), point2(1), npoints);

    %Adjust all coordinates so that they are consistent with the specified
    %center and round them. Furthermore, change any value that is 0 to 1 to
    %avoid calling a zero or negative index.
    xLin = round(xLin);
    xLin(xLin <= 0) = 1;
    yLin= round(yLin);
    yLin(yLin <= 0) = 1;

    %Colour specifier
    colourSpec = zeros(1,3);
    if (color == 'r')
```

```
        colourSpec(1,1) = 255;
    elseif (color == 'g')
        colourSpec(1,2) = 255;
    elseif (color == 'b')
        colourSpec(1,3) = 255;
    elseif (color == 'w')
        colourSpec(1,1) = 255;
        colourSpec(1,2) = 255;
        colourSpec(1,3) = 255;
    elseif (color == 'y')
        colourSpec(1,1) = 255;
        colourSpec(1,2) = 255;
    elseif (color == 'p')
        colourSpec(1,1) = 127;
        colourSpec(1,3) = 127;
    elseif (color == 't')
        colourSpec(1,1) = 64;
        colourSpec(1,2) = 224;
        colourSpec(1,3) = 208;
    elseif (color == 'k')
    end

    %Go through computed XY coordinates, changing them to the specified colour.
    for i=1:npoints
        image(yLin(i),xLin(i),1) = colourSpec(1);
        image(yLin(i),xLin(i),2) = colourSpec(2);
        image(yLin(i),xLin(i),3) = colourSpec(3);
    end

end
```