s0916235 – H.Knudsen
s0943941 – S.Hofman

IVR Coursework 1

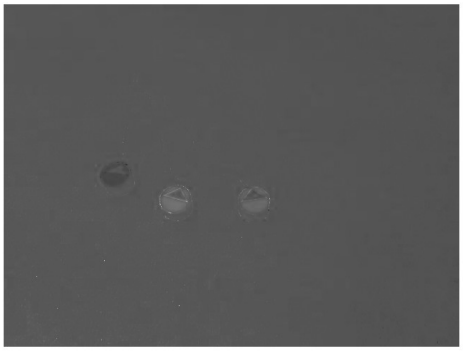
Introduction:

The capture and recognition of an image is a task laden with complications. The solutions, correspondingly, tend to be of the same magnitude. In our approach, we employed a hybridization of ideas. Our first step splits the image into normalized channels and finds an apt threshold value with an image histogram. This is followed by eliminating noise smaller than a certain size, and calculating the centroid of the remaining white objects. The largest of this is assumed to be our robot. Then, we scale the image to focus on the robot and use a similar thresholding process to find the black triangle against the coloured robot (highest peak to the left of the largest peak on the histogram). A centroid is calculated again to give us two points for a directional line. We improve this by using the triangle orientation and the dot product to produce a more reliable angle. Finally, we plot the robot location via its centroid, and connect these points using a errorchecking function (to prevent incorrect leaps). The background image is composed of an average composite of the entire image set, with the effect of the robots minimized.

Detection of Robots

To detect the robots, we tried multiple approaches. We first tried applying Canny Edge Detection to find the edges of the robots in the frames. Despite successfully finding it, we could not use the detected edges to identify the robot's colour without massive computational cost. Our next approach attempted to remove the background, but because the blue robot remains stationary in the data sets, it was classified as background. Instead, we attempted to threshold based on normalized channels. This normalization approach allows us to ignore differences in lighting and emphasize areas with strong levels of colour. First, using two for loops, we normalized the image by setting each pixel value's color to be a relative indicator. Due to our use of Matlab, this operation became too computational expensive, and we instead used an algorithm provided by Bob Fisher. This matrix summation and division allowed us to normalize the image and split the channels with a marked increase in speed. Splitting the image into Red, Green and Blue channels allows us to focus on the individual robots - a red channel will show a red robot brighter than on the normal image. This difference of brightness makes autonomous identification easier in the second step: theresholding.

First, we wanted to improve the strength of the individual channels. Using HSV as a basis, we assumed red would be a value-strong channel, blue would be a value-weak channel, and green would be somewhere in between (an observation confirmed by thresholding – the green threshold would often return the blue robot).



s0916235 – H.Knudsen
s0943941 – S.Hofman

Thresholding returning two light values for both green and blue, despite the threshold being directed toward green.

This formula, gnew = 1.4*g - r/2 – b/2 , allowed us to get a stronger image for all of the colour channels (note the constant ‘1.4’ improved the strengths of the weaker channels. This value was arbitrarily selected, and could be improved. Because red is a stronger channel, we left it as ‘1’ : rnew = r - g/2 - b/2). Below is what we are left with (red channel after improved strength).

This drastic difference in colour allows for global thresholding, rather than computationally slow adaptive thresholding. At this point, only true red, green and blue colours remain. So when we take the histogram of the image (a measure of intensity of pixel values), we can ignore everything to the left of the largest peak (representing the bright image), thus classifying what is background and what is robot. This is calculated with code provided in lectures and located in the sample file dohist. Upon thresholding, we receive a black and white image of a single robot channel.

This is a black and white image after the threshold has been applied

This is repeated for each channel, identifying each colour of robot. Later, a circle with the corresponding colour is drawn around the robot for ease of identification (calculated by a Bounding Box – see below).

A successful identification returns an outlined robot that can also be used to easily identify direction later. A failure results in a number of patches not involved with the robot (such as when the shadows are classified as red in the first data set – see above), although this typically will still find the centre of the robot (calculated in the direction part). For the first data set, all but 6 were correctly identified and of those six, they were toward the end when the robot passed from the frame. For the second data set, all were correct.

s0916235 – H.Knudsen
s0943941 – S.Hofman

Direction of Robot

In our algorithm, the next step is to find the centre of these robots and from the surrounding data, plot the direction. We do this by calculating the centroid, using the Matlab function *regionprops.* However, due to noise in the thresholded image calculated above, the centroid can be wrong. We can attempt to correct this by eliminating the areas where the number of white pixels is not greater than a arbitrary value – we found 30 was a good number. However, the incorrect identification of the centroid is responsible for most of the incorrect classification – the shadow or background will be thresholded incorrectly, and the centroid (centre of mass of data) will be plotted within the largest subsection of this. However, this is an infrequent occurrence in the test data, only coming into effect on the first data set (where the shadows provide issues). The centroid will also be calculated incorrectly when the robot passes from the image. We attempted to stop this by adding a function where, if nothing is found, the rest of singlecolor is aborted and -1 is returned for all values (our error checking number). However, our calculations perform as expected for the majority of cases.

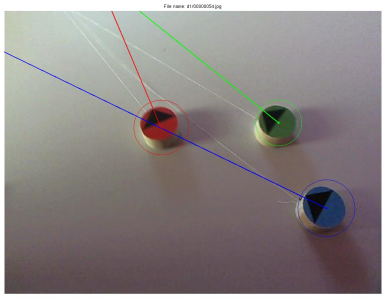
From *regionprops*, we also have a BoundingBox for all the data found in regionprops. This will form a square around the image, with the edges touching the lowest pixel locations of white data. We can use this box to isolate each individual robot and information with greater accuracy. However, the data provided typically gave an oversizing on either height or width. We found the data was improved by forcing the image to be a square box with the lengths to be the averages of the height and width. We can use this to isolate the robot by placing the box around its centroid. After performing an imagecrop on the original image (so we have color to perform a threshold), we get a close-up image of the robot.

ImageCrop of Robot

This is again thresholded, this time on the highest peak to the left of the robot (the largest dark region, which should be the triangle). Calculating the centroid of the triangle, we attain a second point. From these two points (adjusted for the imagecrop), we can form a line which approximates the direction .

Centroids example

However, we found we could improve the direction by trigonometry and the orientation function from imagesprops. Using the orientation line of the triangle as an angle, we add 90 degrees to calculate the norm. We plotted this line across the screen, but this approach only gives an indication of robot orientation, not the forward direction. To fix this, we used the dot product from this line and the vector formed by the two centroid points to determine which edge of the image the object faces. We found this method gave better accuracy and appeared more visually appealing. The plotted line indicates the direction of the robot.



***Orientation Line*

An issue arose during the initial BoundingBox selection. Because of the inclusion of the shadows and the sides of the robots in the thresholding process, the bounding box wouldn't touch the edges of the robot. Thus, when we cropped and thresholded, the centroid calculated wasn't on the object, but rather on the dark carpet in d2. To correct for this, we shrunk our box by a small factor (5 pixels in this case). Possible errors can arise from this method – see the discussion of problems below.

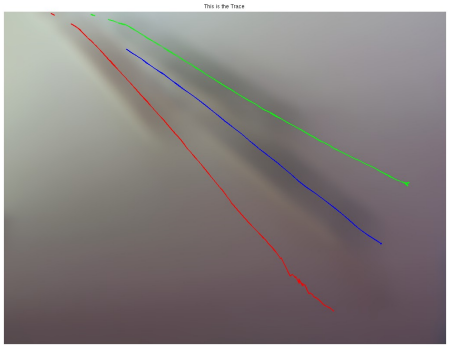
Background and ImageTrace

The final part of our approach is partitioned into three parts – first plotting the centre of the robot and circles around the robots for ease of identification as well as indicating direction, then background synthesis over a series of images, and finally the tracking of the robot path. The first bit is the easiest – use the centroid calculated from the first threshold to indicate the centre of the robot, and a circle, with a radius of half the BoundingBox, centred on the centroid to indicate the general area around the robot. **See figure Orientation Line for example of circles and direction being indicated

Background synthesis is calculated by the FINDrobot function while it loops through the images searching for robots. This calculation used an interesting approach – once we found the robots using thresholding in *singlecolor*, we could assume that everything not white was background. Our function *singlecolor* returns the negation of the threshold – all black (0) for robot and white (1). Then, they are combined together to form a 3D image matrix with multiplication, so the robots are black in each. Then, since multiplying the colour image by 0 will set a pixel to black (where the robots are), and multiplying by 1 will leave a pixel untouched (where the robots aren't), we multiply this by the original image. Thus, we have the original image, minus the robots. Once FINDrobots is finished, the blackened image is averaged with the original image to offset the robots presence. The function then performs error checking and a distance calculation (also from FINDrobots). If the robots haven't been found (value of -1), or if the distance travelled by each of the robots is less than a minimum (i.e. the robot hasn't moved) or is greater than a maximum (error checking for false identification by checking against impossible movement e.g. robot moves from left of plot to right in one frame), then that image is added to the background. This process also uses

a weighting to improve background images, where the minimum distance indicates the strength of an image. If all the distances are large, the image is more valuable than one where an image barely moves at all. Finally, we exit the loop of all the images, and divide the summed image (a massive double) by the addition of the amount of images used and the total weight. This is converted to a uint8 to view, and blurred using a fspecial disk filter to eliminate the ghostly robots.

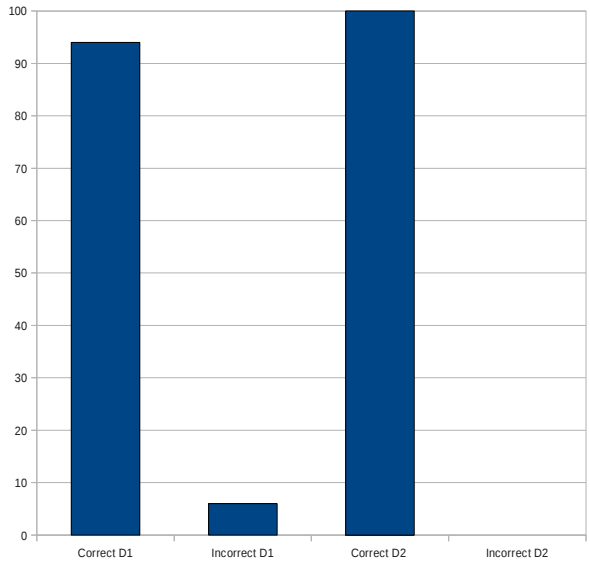
Finally, we take the synthesized background, and plot the centroids of each the red, green and blue robots for each of the frames (stored as an array during the image looping array) as a line. For example, we take the second frame's centroids, and draw a line to the first frame's centroids. This process, continued over the entire image, shows the path of the robots. We also included an error checking method to prevent the robot from jumping all over the place due to incorrect classification – if it jumps a distance greater than a certain distance (arbitrarily chosen), or is not found, then that frame does get plotted. An example of our background with image plot is below.



Trace Example On background

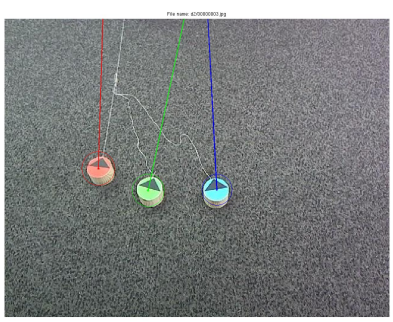
Results:

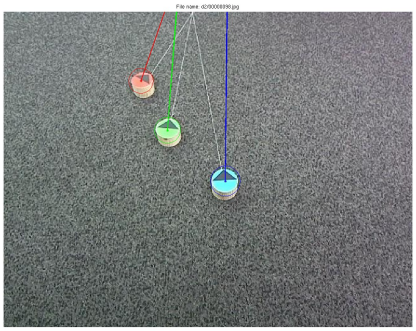
Below is a graph showing the number of incorrect results vs correct results. Please note for dataset 2 there where no incorrect results. Also note that correct classification assumes the line drawn is within 15 degrees of the black arrow. Anything greater than that value, we classified as incorrect.

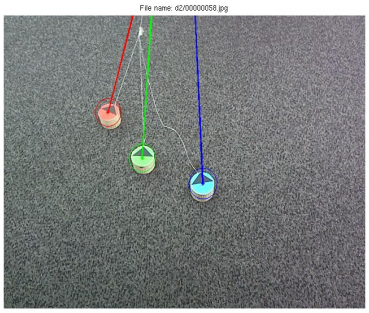


s0916235 – H.Knudsen
s0943941 – S.Hofman

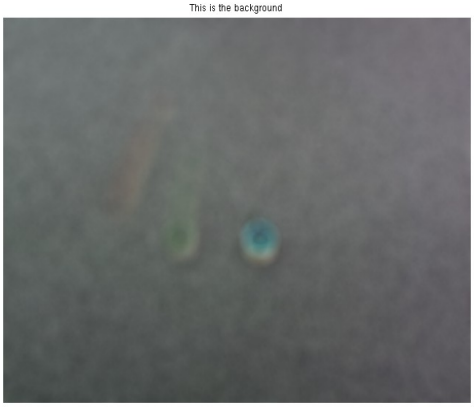
Here are two examples of the dataset D2 which show all the robot correctly detected and identified. Although of note is the red direction is slightly off. We still counted this slight offset as correct.



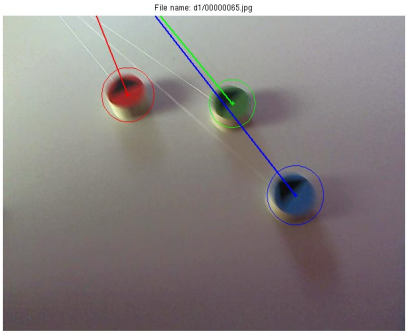


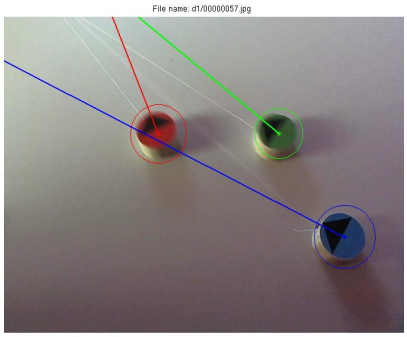


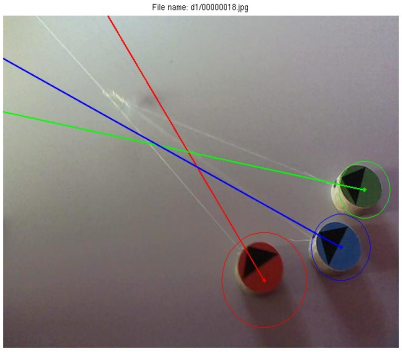
Here is our synthesized background image and the same image with the robot path traced over it for data set 2.



Here are three examples of success in the first data set.

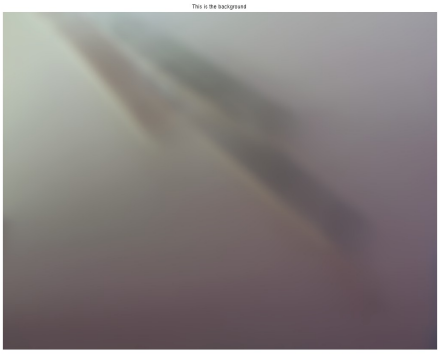


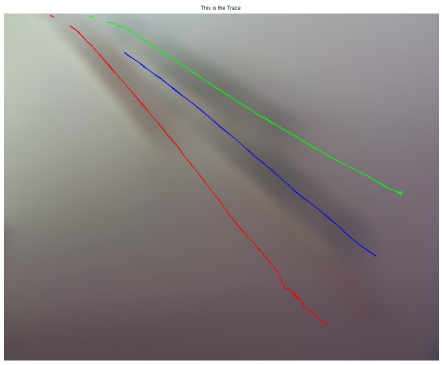




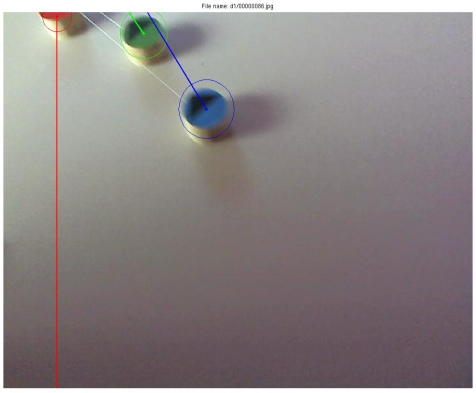
s0916235 – H.Knudsen
s0943941 – S.Hofman

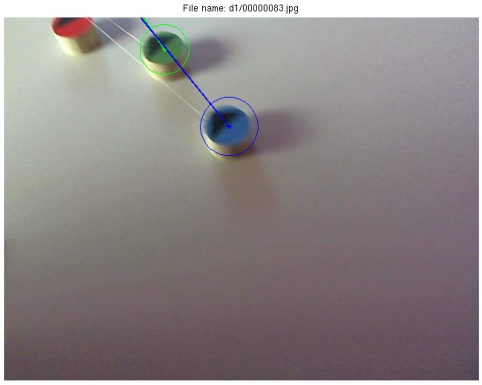
Here is our synthesized background image and the same image with the robot path traced over it for data set 1.



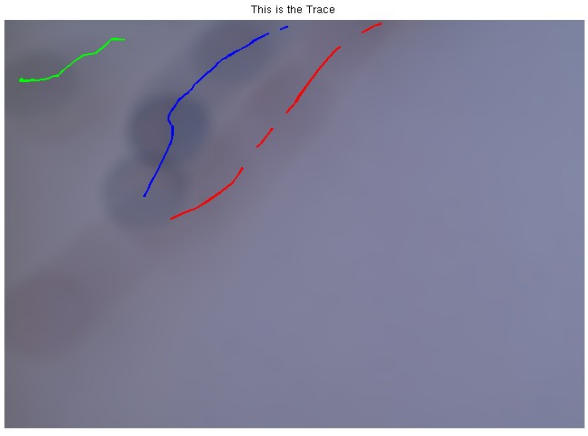


Here are two examples of dataset 1 where red is incorrect. This occurs when we detect a robot, but since the black triangle is off the screen, it finds the closest value. Because of the reddish shadows, if finds it below the object, and so assumes the robot is facing downward. The second error arises when the red shadows are not enough to find a “triangle,” so the image does not classify a robot. The other errors in the first data set were of similar nature.





This is an example from a custom data set where the robot is not detected for a brief period of time resulting in a broken trace this is caused by lighting changing as the object are moved through the images. Note this does not occur in image set D2 but does occur in image set D1.



---

s0916235 – H.Knudsen
s0943941 – S.Hofman

Discussion

On average, our code produces successful results. In both pieces of test data, it identifies the robots, plots the direction, runs in good time, and composites an average background image that attempts to deal with the stationary objects. Not only that, but, with a couple of exceptions, performs well on data captured by our own testing. While these capabilities fulfil the requirements of the assignment, several improvements can be made.

A possible issue can arise from the slight cropping of the BoundingBox. This was the result of an arbitrarily picked value which may have been tailored for the specific data sets, as it continues to return valid results. Tested on another image set could prove the value erroneous. This is the possible source of the failure to compute the direction for the image sets created to test.

Once the robots reach the end of the screen, the object will occasionally pick up shadows that are not robots, due to the algorithm still calculating the centroid of any coloured mass. This results in the directional arrow being computed at a contrary angle to the previous directions. Possible fixes for this include an indicator of the previous image's angles to ensure constant direction, or a function that checks the robot's position

Another limitation arises when the image is too dark. Our normalization not only employs the traditional normalization of R/(R+G+B), but an additional subtraction to emphasize the images: r = r- b/2 – g/2. This approach fails to capture images darker than a certain value, as too many channels are set to 0 (black). We attempted to offset this by increasing the strength of the normalization values (R*2)/(R*2+G*2+B*2), but for some dark images it fails. However, as it allows for a fast, easy global threshold, we decided to leave it. If image lighting became an issue, we would have to switch approaches.

Since the background is synthesized as a composite from images, the robots are always visible slightly, despite the weighted averaging. A possible solution for this would be to take the area bounded by the ImageBox (containing a cropped image of the robot) and apply median filtering on this point to try and fully eliminate the robot.

Finally, the path traced by each robot occasionally has gaps where it could not find the robot on darkened data. However, it finds the robot at a later frame, and continues to plot from this point. This, as continuous data, means that the robot would have to pass from the last point of found data to the new point. We could account for this by connecting the unconnected points once all the data is gathered, and applying a smoothing function to mimic a better route.

We also have included extensive error checking in our code to prevent as many errors as possible. These include the robot not being found or being located off the screen. We cannot use points that don't exist to calculate an angle, so we set the variables to minus -1 by default. Every time a value is not found, the code will not plot this point. We debated including an alert for this, but found that too many title on the screen became cumbersome and left it as is.

We have decided to split the work as follows – H Knudsen 45%, S Hofman 55%

Appendix: Code

4 files:
start.m – included for completeness was used for testing of multiple data sets consecutively
main.m – the main function must be called on a dataset
FINDrobots.m – searches image
singlecolor.m – searches specific colour channel
Also Used
dohist.m

s0916235 – H.Knudsen
s0943941 – S.Hofman

----------start.m----------

```
% loads dates sets in directory with the format d#/#######.jpg
% where #is any set of continues integers specfied belwo currently 1 to 8
% int and ####### is 0000001 throught 000000100
for i = 1:2
    i
    main(i);
end
```

----------end start.m ----------

---------- singlecolor.m----------

```
function [xcenter, ycenter, xline, yline, radius, rinverse] =
singlecolor(loaded, justColor)

%Error checking - default values
xcenter = -1;
ycenter = -1;
xline = -1;
yline = -1;
radius = -1;

%Perform thresholding
hist = dohist(justColor,0); %dohist based on function from Bob Fisher
filter = fspecial('gaussian', [50 1], 6);
tmp1 = conv(filter,hist);
peak = find(tmp1 == max(tmp1));
bw = justColor > max(peak);
%Eliminate noise
robot = bwareaopen(bw, 50);
rinverse = (~(robot));
[l,w] = size(robot);

%Calculate of robot in this color channel by finding the size of the
%maximum connected object in a black and white image, also calculate a
%bounding box for the robot
s  = regionprops(robot, {'centroid','area','BoundingBox'});

% check an object has been found
if not(isempty(s))
    %gets the id of the maximum found object hopefully a robot
    [~, id] = max([s.Area]);
    %varaible to adjst the bounding box
    boundingboxadjust = 5;
    radius = (s(id).BoundingBox(3)+s(id).BoundingBox(4))/2;

    %Error checking to see if the adjusted box would exceed the edges of
    %the graph
    if(s(id).BoundingBox(1)+radius-boundingboxadjust>l ||
s(id).BoundingBox(2)+radius-boundingboxadjust>w)
        boundingboxadjust = 0;
    end

    % crop the image to just the boundintg box then convert it to graysacle
    % do a histogram on the grayscale and look for peaks so that the arrow
    % can be filterd out in black and white
    arrow = imcrop(loaded,[s(id).BoundingBox(1)+boundingboxadjust,
s(id).BoundingBox(2)+boundingboxadjust, radius-boundingboxadjust, radius-
boundingboxadjust]);
    arrow = rgb2gray(arrow);
```

```
    hist = dohist(arrow, 0);
    tmp1 = conv(filter, hist);
    peak = find(tmp1 == max(tmp1));

    % find highest peak to left
  xmaxl = -1;
  pkl = -1;
  for i = 2 : peak-1
      if tmp1(i-1) < tmp1(i) & tmp1(i) >= tmp1(i+1)
          if tmp1(i) > xmaxl
          xmaxl = tmp1(i);
          pkl = i;
          end
      end
  end
  if pkl == -1
      pkl = 1;
      xmaxl = 1;
  end

  %use threshold to convert crop box to black and white so arrow can be
  %identified
  athresh = arrow < pkl;
  athresh = bwareaopen(athresh, 30);

  %identify the arrow and its orentation
  tri = regionprops(athresh, {'centroid','area','Orientation'});
  if not(isempty(tri))
      [~, id2] = max([tri.Area]);

      xcenter = s(id).Centroid(1);
      ycenter = s(id).Centroid(2);
      % add the arrow's lcoation inside the boudnign box to the bounding
      % box so its location in the overall image is known
      xtriangle = s(id).BoundingBox(1) + tri(id2).Centroid(1);
      ytriangle = s(id).BoundingBox(2) + tri(id2).Centroid(2);

      % get the angle that the trianlge is on
      angle = tri(id2).Orientation;
      refpoint = [xcenter,ycenter];
      %refpoint = [ycenter,xcenter];

      % roate the triangle angle so its it point in the correct direction
      alpha = angle+90;

      % get size of the image so that at line can be drawn right up to the
      % edge of it
      [x y z] = size(loaded);
      %below code calculate 2 point on either side of the overall image at the
angle given
      %through the center of the main robot
      horizontal_distance = ((refpoint(2)-1)/tand(alpha));
      if ((refpoint(1) + horizontal_distance) > y)
          vertical_distance = ((refpoint(1)-1)/cotd(alpha));
          verctical_distance_reversed = ((y-refpoint(1))/cotd(alpha));
          xline = [(refpoint(2) + vertical_distance), (refpoint(2) -
verctical_distance_reversed)];
          yline = [1, y];
```

```
      else
          horizontal_distance_reversed = ((x-refpoint(2))/tand(alpha));
          xline = [1, x];
          yline = [(refpoint(1)+ horizontal_distance), (refpoint(1)-
horizontal_distance_reversed)];
      end

      %working 2point - direction vector of a line joining the two
      %centroids
      w = [(xcenter-xtriangle),(ycenter-ytriangle)];

      %1 direction vector of 1 half of the region prop orientatation
      d1 = [xcenter-yline(1),ycenter-xline(1)];

      % work out the angle between the orientation line and line between
      % two point centroid's
      angle = (180/pi)*(acos((dot(w,d1))/((norm(w,2))+(norm(d1,2)))));

      %choose which half of the orentation line is point forward and pass
      %the point at the edge of the frame back so it can be plotted as a
      %line
      if (angle <= 90)
          xline = xline(1);
          yline = yline(1);
      else
          xline = xline(2);
          yline = yline(2);
      end
   end
end
```

---------- End singlecolor.m----------

----------FINDrobots.m----------

```
function [rxc, ryc, gxc, gyc, bxc, byc, background] = FINDrobots(image)
%this function takes in an image and attempts to find all 3 robots and
%there direction. as well as computing the image minus the robots for the
%background

% loads the image from the file this is passed in it will be in the from of
% d#/00000###.jpg
loaded = importdata(image, 'jpg');

%below is matrix normalazation by bob fishcer as a replacment for out for
%loop preforming the same functions this greatly acclerates our code
[R,C,D]=size(double(loaded));
NM = sum(double(loaded)*2,3);
NM3 = zeros(R,C,3);
NM3(:,:,1) = NM;
NM3(:,:,2) = NM;
NM3(:,:,3) = NM;
IN = imdivide(double(loaded)*2,NM3);

%this sperates the image channels into thre respective color spaces
r = double(IN(:,:,1));
g = double(IN(:,:,2));
b = double(IN(:,:,3));

% this computes a greysale image of e.g. color space bye subtracting the other
```

s0916235 – H.Knudsen
s0943941 – S.Hofman

```
%color  spaces to remove the diffrent robots from the diffrent channels the
channels
% are also weighted when doing this becuase some are stronger than others
rnew = r - g/2 - b/2;
gnew = 1.4*g - r/2 - b/2;
bnew = 1.4*b - r/2 - g/2;

%convert back to an intgeral from a partial fraction and convert to a unit
%so image can be displayed
justRed = uint8 (rnew*255);
justGreen = uint8 (gnew*255);
justBlue = uint8 (bnew*255);

%Call the function single color which actually preforms the calulations for
%finding the center of the object and the point to which a line will be
%drawn and the radius for the circle to hightlight the the object
%single color also produces a background with the the background filterd
%from the robot
[rcentx,rcenty,rxline, ryline, rradius, backr] = singlecolor(loaded, justRed);
[gcentx,gcenty,gxline, gyline, gradius, backg] = singlecolor(loaded, justGreen);
[bcentx,bcenty,bxline, byline, bradius, backb] = singlecolor(loaded, justBlue);

%Combines the black robot images with multiplication identity
darkenrobots = (backr);
darkenrobots = immultiply((backg), darkenrobots);
darkenrobots = uint8(immultiply((backb), darkenrobots));

%Multiplies the boolean darkenrobots with the color channels and sets them
%into a background without the robots.
background(:,:,1)=immultiply(darkenrobots, loaded(:,:,1));
background(:,:,2)=immultiply(darkenrobots, loaded(:,:,2));
background(:,:,3)=immultiply(darkenrobots, loaded(:,:,3));

%send centers to be captured in centarray
rxc = rcentx;
ryc = rcenty;
gxc = gcentx;
gyc = gcenty;
bxc = bcentx;
byc = bcenty;

%show the image with the robot circled and a line showing its direction

imshow(loaded);
hold on

rc = [rcentx,rcenty];
gc = [gcentx, gcenty];
bc = [bcentx, bcenty];

%Circle angle
a = [0:2*pi/30:2*pi];

title(strcat({'File name: '}, image));
%Plot the centers and direction the robots
% -1 protection if the robot is not found
if(rc(1) ~= -1 && rc(2) ~= -1 && rradius~= -1)
    plot(rc(1), rc(2), 'r*');
```

s0916235 – H.Knudsen
s0943941 – S.Hofman

```
    plot((rradius/1.5)*cos(a)+rc(1), (rradius/1.5)*sin(a)+rc(2), 'r');
    if(rxline~= -1 && ryline~= -1)
        plot([rc(1), ryline], [rc(2), rxline], 'r','LineWidth',2);
    end
end

if(gc(2) ~= -1 && gc(2)~= -1 && gradius~= -1)
    plot(gc(1), gc(2), 'g*');
    plot((gradius/1.5)*cos(a)+gc(1), (gradius/1.5)*sin(a)+gc(2), 'g');
    if (gxline~= -1 && gyline~= -1)
        plot([gc(1), gyline], [gc(2), gxline], 'g','LineWidth',2);
    end
end

if(bc(1)~= -1 && bc(2)~= -1 && bradius~= -1)
    plot(bc(1), bc(2), 'b*');
    plot((bradius/1.5)*cos(a)+bc(1), (bradius/1.5)*sin(a)+bc(2), 'b');
    if (bxline~= -1 && byline~= -1)
        plot([bc(1), byline], [bc(2), bxline], 'b','LineWidth',2);
    end
end

hold off
pause(1);

end
```

----------end FINDrobots.m----------

----------main.m----------

```
function main(passeddir)
% this is the controlling function for a complete image set
% it will load all images in a directory which has been passed to it
% the directory name must start with a d and be followed by and int
% e.g. 'd1' or 'd2' where 1 or 2 may be specified

% concats to open dir
passeddir = strcat('d',int2str(passeddir),'/');
%loads all images from the dir into a flat array 1x100
files = dir(strcat(passeddir,'*.jpg'));
%centarry is a matrix 100x6 which stores the centers of all 3
%objects in x,y format so they can be plotted later
centarry = double(zeros([(length(files)) 6]));

%below code makes a background based upon weighted movement
%if no movement or to little movement is detcted in makeing a background
%it produces a background based upon average over all the images

% weighted background generating variables
%total number of background images
total = 1;
%number of images in weighted background
counter = 1;
%if weighted background is used e.g. movment threshold has been reached
found = 0;
%the amount of movment that is being used to weight this background
weight = 0;
H = fspecial('disk', 10);
pause on;
```

s0916235 – H.Knudsen
s0943941 – S.Hofman

```
% below code generates background and fills the center array and runs find
% robots which generates the data to be used below
for i = 1:length(files)
    string = strcat(passeddir,files(i).name);
    load = double(importdata(string, 'jpg'));
    %this line goes into a function which computers robots and there
    %direction as well as computing the image minus the robots
    [a,b,c,d,e,f,g] = FINDrobots(string);
    %saves centers into centarry
    centarry(i, 1) = a;
    centarry(i, 2) = b;
    centarry(i, 3) = c;
    centarry(i, 4) = d;
    centarry(i, 5) = e;
    centarry(i, 6) = f;

    %averages image and image minus robots from find robots
    avgback = (imadd(load,double(g)))/2;

    %amount of movment to look for variables
    mindist = 1.5;
    maxdist = 80;

    if(i==1)
        im = double(avgback);
        im2 = double(avgback);
    else
        if(i>5)
           %-1 is used to indicate that the robot or object was not found
           %this value is used throughtout the code fro protection purposes
           if((a && b && c && d && e && f)~=-1)
               % looks for movment to compute background
               rxdist = centarry(i, 1) - centarry(i-4, 1);
               rydist = centarry(i, 2) - centarry(i-4, 2);
               gxdist = centarry(i, 3) - centarry(i-4, 3);
               gydist = centarry(i, 4) - centarry(i-4, 4);
               bxdist = centarry(i, 5) - centarry(i-4, 5);
               bydist = centarry(i, 6) - centarry(i-4, 6);
               rdist = sqrt(rxdist^2 + rydist^2);
               gdist = sqrt(gxdist^2 + gydist^2);
               bdist = sqrt(bxdist^2 + bydist^2);
               total = total + 1;
               %adds and image to the background if enough movement is found
               if(rdist > mindist && bdist > mindist && gdist > mindist && rdist
< maxdist && bdist < maxdist && gdist< maxdist)
                   lowest = rdist;
                   if(rdist>bdist)
                     lowest = bdist;
                   end
                     if(lowest>gdist)
                         lowest = gdist;
                     end
                     counter = counter + 1;
                     weight = lowest + weight;
                     found = 1;
                     avgback = avgback * lowest;
                     im = imadd(im,avgback);
```

```
                else
                    im2 = imadd(im2,avgback);
                end
            end
        end
    end
end

%choose wether to use movment based background or to use a total average
%background based upon wether enough movment based images where selected or
%a complete avearge is liekly to be better
if(found && counter>20)
   im = uint8(im/(counter+weight));

else
    im = uint8(im2/total);
end

% this filter blurs the image slightly producing a nicer background
% reducing robot ghosting
im = imfilter(im, H, 'replicate');

%show background and waits
imshow(im);
title('This is the background');
pause;
hold on;

%this code draws the trace ontop of the background with a filter to stop
%the trace from jumping erraticly when impossible movment is deteced
%between frames
jumptoll = 40;
for i = 1:length(files)
    % ignors the first frame as there is no movement from a previous frame
    if(i==1)
        ;
    else
        %check for -1 protection and plots the line beween frames
        if ((centarry((i-1),1)||centarry((i),1)||centarry((i-1),2)||
centarry((i),2))==(-1)||abs(centarry((i-1),1)-centarry((i),1))>=jumptoll||
abs(centarry((i-1),2)-centarry((i),2))>=jumptoll)
            ;
        else
        plot([centarry((i-1),1) centarry((i),1)], [centarry((i-1),2)
centarry((i),2)], 'r','LineWidth',2);
        end
        if ((centarry((i-1),3)||centarry((i),3)||centarry((i-1),4)||
centarry((i),4))==(-1)||abs(centarry((i-1),3)-centarry((i),3))>=jumptoll||
abs(centarry((i-1),4)-centarry((i),4))>=jumptoll)
            ;
        else
        plot([centarry((i-1),3) centarry((i),3)], [centarry((i-1),4)
centarry((i),4)], 'g','LineWidth',2);
        end
        if ((centarry((i-1),5)||centarry((i),5)||centarry((i-1),6)||
centarry((i),6))==(-1)||abs(centarry((i-1),5)-centarry((i),5))>=jumptoll||
abs(centarry((i-1),6)-centarry((i),6))>=jumptoll)
```

s0916235 – H.Knudsen
s0943941 – S.Hofman

```
            ;
        else
        plot([centarry((i-1),5) centarry((i),5)], [centarry((i-1),6)
centarry((i),6)], 'b','LineWidth',2);
        end
    end
end
title('This is the Trace');
pause;
end
```

----------end main.m----------

Credit:

Additional testing dataset images provided by Jibran Khan
DoHist, ImageNormalization and Thresholding Code provided Bob Fisher.
Image viewing (imshow), image filtering (fspecial), image arithmetic (immultiply, imadd, etc) and image manipulation (bwopenarea, regionprops) provided by Matlab Image Toolkit
DrawSlope Code modified from http://www.mathworks.com/matlabcentral/fileexchange/25370-draw-an-angled-line-with-a-slope-relative-to-a-reference-point/content/drawslope.m, by user Elad