Introduction to Vision and Robotics: Assignment 1

Chris Swetenham: Student Num, Daniel Mankowitz: S1128165

3/11/2011

1 Introduction

An overview of the main ideas used in the approach

2 Methods

Describe the vision techniques used

2.1 Linking Algorithm

The linking algorithm is used to link together detections of a single robot in consecutive frames. Since the images provided in the datasets are RGB, the linking algorithm has been developed to run on each of the three colour channels respectively.

The algorithm is shown in *Figure 1.* Initially, the image on the k^{th} iteration will have been split into three colour channels of red, green and blue respectively. Each colour channel is represented as a separate image and each image has been clipped to a dimension of $120x120$ in the detection algorithm. This enables a more accurate bounding box to be calculated for each robot as there will be less image noise in the clipped image.

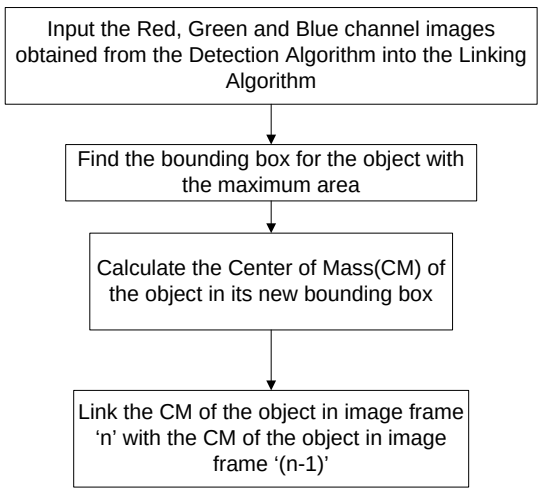

Figure 1: The algorithm developed to link detections of the robot in consecutive frames

2.1.1 Finding the Bounding Box

The bounding box of each robot is calculated using the function *calcBoundingBox*. A flow diagram of the function can be seen in *Figure 2*. This function calculates the bounding box for each robot as well as the box's corresponding centroid in each of the three colour channels respectively.

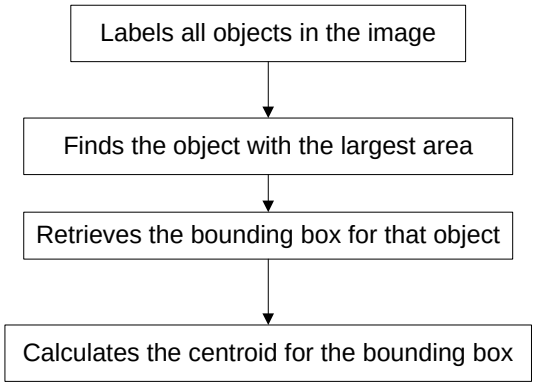

Figure 2: The bounding box calculation for each robot

An excerpt from the *calcBoundingBox* function is shown below.

```
%Excerpts from the bounding box calculation

%Label the image. The labelled image is labelX
[labelX, numXBlobs] = mybwlabel(TImgX);

%Calculate the area for each object found in 'labelX'
allBlobAreas = [];
for i=1:numXBlobs
    [rows,cols,vals] = find(labelX==i);
    blobSize = sum(vals);
    allBlobAreas = [allBlobAreas blobSize];

end

%Find the object with the maximum area in the image
[maxBlobArea, index] = max(allBlobAreas);

if HaveToolbox==0
%Without using the image toolkit, extract the bounding box
[rows,cols,vals] = find(labelX==index);
x1 = min(cols);
x2 = max(cols);
y1 = min(rows);
y2 = max(rows);
end
```

Each image is labelled using the *mybwlabel* function as seen in the code excerpt. This will identify all of the objects in the image and assign a set value to each pixel associated with a specific object.

The object with the largest area is then identified. A bounding box is subsequently calculated for the object with the largest area (I.e. the largest number of labelled pixels). This is achieved using the *find* Matlab function which identifies the largest object and stores each pixel belonging to the object with its respective row and column position. This is then used to determine the bounding box of the object.

The object will then be surrounded by its corresponding bounding box. The centroid of this bounding box is then calculated. This algorithm is performed on each of the three clipped images corresponding to the three respective colour channels.

2.1.2 Calculate the Center of Mass

Once a bounding box for each robot has been determined, the center of mass of the robot needs to be calculated in relation to this new bounding box. This is achieved by utilising the *calcBoundingBoxCM* function.

The algorithm initially calculates the area of the object within the bounding box using *Equation 1.* This value is used to determine the center of mass of the object within the bounding box.

$$A = \Sigma_r \Sigma_c P_{rc} \tag{1}$$

The center of mass of the object is then determined using *Equation 2.* $(\widehat{r}, \widehat{c})$ are the row and column coordinates of the center of mass of the object respectively.

$$(\widehat{r}, \widehat{c}) = (\frac{1}{A}\Sigma_r \Sigma_c r P_{rc}, \frac{1}{A}\Sigma_r \Sigma_c c P_{rc}) \tag{2}$$

2.1.3 Linking and Plotting

The final step in the linking algorithm is to link the object in image frame n with the object in the previous image frame $(n-1)$. A criteria needs to be developed in order to define when this 'linking' of objects should occur.

The criteria has been defined as follows. It has been previously mentioned that each image has been separated into its R,G and B channels. The object with the largest area of pixels in each channel is chosen to represent the robot. For example, the object with the largest area of red pixels in the red colour channel, represents the red robot. Based on this assumption, the largest objects detected in adjacent image frames in the same colour channel are connected. This is a safe assumption as the largest concentration of red, green and blue pixels are usually associated with the red, green and blue robots respectively.

The center of mass of the red, green and blue robots are stored in 'linker' arrays as shown below. The links are plotted on the estimated background image at the end of the program.

```
%***********************************************
%To link tracks on the estimated background image
XLinkerR = [XLinkerR trueCMXR];
YLinkerR = [YLinkerR trueCMYR];
XLinkerG = [XLinkerG trueCMXG];
YLinkerG = [YLinkerG trueCMYG];
XLinkerB = [XLinkerB trueCMXB];
YLinkerB = [YLinkerB trueCMYB];
```

2.2 Identifying Robot Orientation

Once the positions of the robots have been identified, the next step is to determine the direction in which the robot is currently travelling. In order to achieve this objective, an orientation algorithm

has been developed.

This algorithm ultimately connects the center of mass of the robot to the centroid of the bounding box surrounding the robot as seen in *Figure 3*. Since the center of mass of the robot is found near the robot's base, connecting this coordinate to the centroid will generally result in an accurate vector describing the robot's orientation.

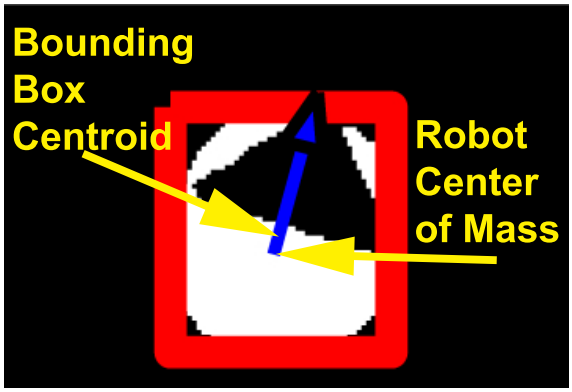

Figure 3: Connecting the center of mass of the robot to the bounding box centroid to determine the robot's orientation

The algorithm initially receives the center of mass coordinates of the robot as well as the bounding box centroid as shown in the code below. Using these values, the algorithm then defines vectors *DR, DG* and *DB* for each of the three colour channels. By normalising these vectors using *Equation 3*, unit vectors are obtained and are then scaled to create the arrows pointing in the direction of the robots current orientation. These arrows are then plotted on the current image frame using a third-party function called *plot_arrow*.

$$D_x = \frac{D_x}{||D_x||} \tag{3}$$

```
%Define unit vectors in the direction of the bounding
%box centroid for each of the rgb channels respectively
CenterMassR = [CenterMassXR CenterMassYR];
DR = (CentroidR - CenterMassR);

CenterMassG = [CenterMassXG CenterMassYG];
DG = (CentroidG - CenterMassG);

CenterMassB = [CenterMassXB CenterMassYB];
DB = (CentroidB - CenterMassB);

%Calculate the unit vectors for each channel
DR = DR/norm(DR);
DG = DG/norm(DG);
DB = DB/norm(DB);

%*****************************************************
%Excerpt of code for plotting the arrows connecting the
%centroids and center of mass of each robot
plot_arrow(trueCMXR,trueCMYR, trueCentroidBBXR+30*DR(1),trueCentroidBBYR+30*DR(2)...
,'linewidth',2,'headwidth',0.25,'headheight',0.33,'color',LineColRArrow,...
'facecolor',LineColRArrow);
hold on
```

```
plot_arrow(trueCMXG,trueCMYG, trueCentroidBBXG+30*DG(1),trueCentroidBBYG+30*DG(2)...
,'linewidth',2,'headwidth',0.25,'headheight',0.33,'color',LineColGArrow,...
'facecolor',LineColGArrow);
hold on
plot_arrow(trueCMXB,trueCMYB, trueCentroidBBXB+30*DB(1),trueCentroidBBYB+30*DB(2)...
,'linewidth',2,'headwidth',0.25,'headheight',0.33,'color',LineColBArrow,...
'facecolor',LineColBArrow);
```

2.3 Background Estimation

Estimation of the background image required the implementation of a variety of different image processing techniques. This is because the datasets provided presented a number of challenges. One such challenge was that of removing robots that were stationary for a significant portion of a frame sequence. This prevented the simple implementation of a median filter to calculate the background image. This is due to the fact that objects that are stationary for a large subset of the frames will be incorporated into the median and subsequently into the background image.

Therefore, in order to solve this problem, an algorithm has been developed to calculate the background image, regardless of stationary robots being in large subsets of the frame sequence.

The background estimation algorithm consists of three main functional blocks. These blocks are *Channel Processing*, *Region Erasing and Filling* and *Median Filtering* respectively. The *Channel Processing* block receives an image frame and processes the image. This includes blurring, normalising, separating the image into its three channels, subtracting the blue channel from the green channel and renormalising the image. This creates an image that is ready for *Region Erasing and Filling*. In order to implement this functional block, the image channels for the current frame are input into a function called *eraseRegion*. A code snippet of the function is shown below.

```
function [Out] = eraseRegion(Img, C, Size)
    Out = Img;
    %Find the pixels on the vertices of the bounding box
    %surrounding the robot. top—t, bottom—b, left—l, right—r
    t = max(1, C(1) — Size/2);
    b = min(size(Img, 1), C(1) + Size/2);
    l = max(1, C(2) — Size/2);
    r = min(size(Img, 2), C(2) + Size/2);
    %Find the average of the four pixels at the vertices of the
    %bounding box
    Avg = Img(t, l, :) + Img(t, r, :) + Img(b, l, :) + Img(b, r, :);
    Avg = Avg / 4;
    %Replace the image segment (I.e. the robot) with the average
    %of the four pixels
    for y = t:b
        for x = l:r
            Out(y, x, :) = Avg;
        end
    end
```

This function calculates the average value of the pixels found at the corner vertices of the bounding boxes surrounding each robot as shown in *Figure 4*. The corner pixels are averaged across all three colour channels and are stored in the variable *Avg*. All of the pixels in the region containing the robot are then filled with this new average value to produce the image shown in *Figure 4*. This is applied to each of the robots in the image.

The *Region Erasing and Filling* functional block is implemented on the three image frames that are used for background estimation. The reason three image frames have been chosen is because this is the minimum number of frames required to effectively use a median filter which is utilised

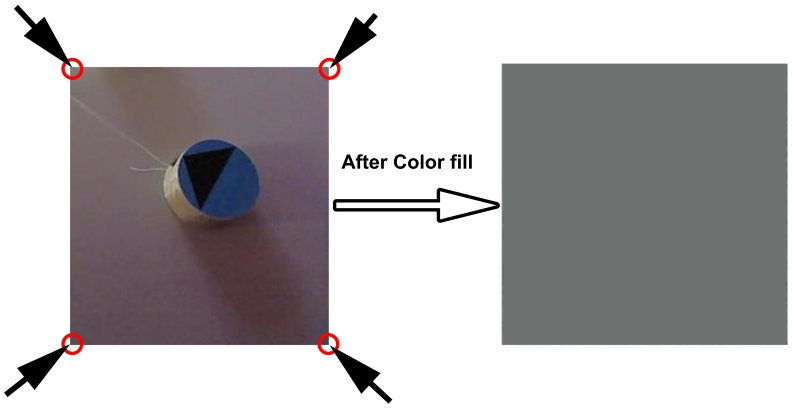

Figure 4: The corner pixel intensities are averaged and are then used to fill the region of the image containing the robot

in the *Median Filtering* functional block. In addition to this, using a small number of frames to compute the median minimises the algorithm's processing time. The median of the three images is subsequently computed to produce an estimation of the background image.

3 Results

Test data

3.1 Robot Detection

Table 1: Results obtained from trying to detect whether or not a robot is in an image frame

	Dataset 1			Dataset 8			Dataset 10		
Type	Red	Green	Blue	Red	Green	Blue	Red	Green	Blue
Correct Detection	73	91	95	95	95	95	95	95	95
Missed Detection	0	0	0	0	0	0	0	0	0
Incorrect Detection	22	4	0	0	0	0	0	0	0

An example of an incorrect detection of a robot due to the robot leaving the image frame.

An example of an incorrect detection of a robot due to a region of highly saturated pixels that have a similar colour to that of the robot.

3.2 Linking Robot Tracks

As detailed in *Section 3.2*, a linking algorithm for the robots has been developed. This algorithm was tested on a variety of datasets and the results are tabulated in *Table 2*. This table details the number of *Correct tracks, Incorrect Tracks* and *Broken Tracks*. A *Broken Track* has been defined as a connection between two tracks whereby the latter track is an incorrect detection of a robot.

The linking algorithm performed well on dataset 8 and dataset 10. However, dataset 1 had a large number of incorrect and broken tracks, especially on the red channel. These errors occurred as a result of the red robot leaving the image frame. As the robot leaves the frame, the algorithm looks for a set of red pixels that correspond to a red robot. Since the robot is not present, the algorithm identifies the next largest area of red pixels as being the red robot. This is incorrect

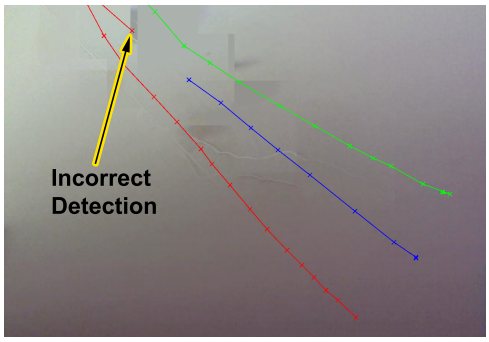

Figure 5: An incorrect detection of the red robot. This track has been linked to the track of the blue robot creating an incorrect track.

Figure 6: The incorrect detection of the red robot as shown in the red channel binary image.

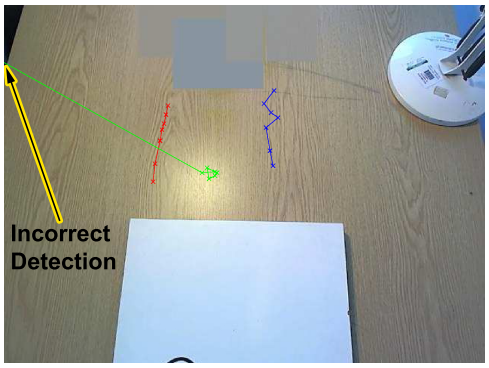

Figure 7: An incorrect detection of the green robot. This results in a broken track

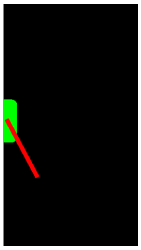

Figure 8: The incorrect detection of the green robot due to a small region of highly saturated pixels.

and causes the results below.

An example of this is shown in *Figure 5* and *Figure 6*. In this example, the red robot has left the scene and a group of red pixels, as shown in *Figure 6*, have been incorrectly defined as being the red robot. This results in an incorrect linkage between two tracks.

Table 2: Results obtained from linking robot tracks on a variety of different datasets

	Dataset 1			Dataset 8			Dataset 10		
Type	Red	Green	Blue	Red	Green	Blue	Red	Green	Blue
Correct Tracks	73	91	95	95	95	95	95	95	95
Incorrect Tracks	10	2	0	0	0	0	0	0	0
Broken Tracks	12	2	0	0	0	0	0	0	0

Another situation occurs whereby the robot tracks are incorrectly linked. As seen in *Figure 7*, the green robot track has been incorrectly linked to the left hand side of the image frame.

3.3 Robot Orientation

The orientation of the robots was tested on a variety of different datasets and the results for each dataset is presented in *Table 3*. It has been found that it is generally possible to determine the orientation of the robot. However, there are a number of scenarios whereby the orientation is incorrectly determined causing the arrowheads to point in incorrect directions.

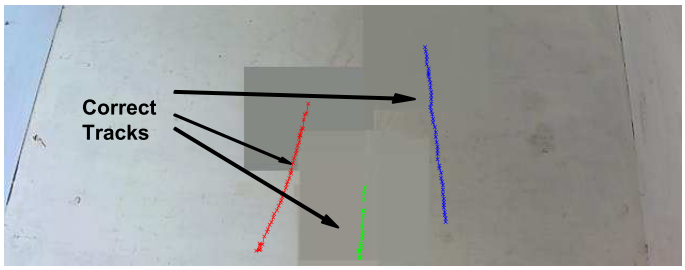

Figure 9: An example of correct robot tracks

Table 3: Results obtained from determining which direction the robot is facing

	Dataset 1			Dataset 8			Dataset 10		
Type	Red	Green	Blue	Red	Green	Blue	Red	Green	Blue
Missed Direction	6	6	2	0	0	0	0	2	4
Incorrect Direction	0	6	0	1	11	3	4	12	2
Correct Direction	89	83	93	94	84	92	91	81	89

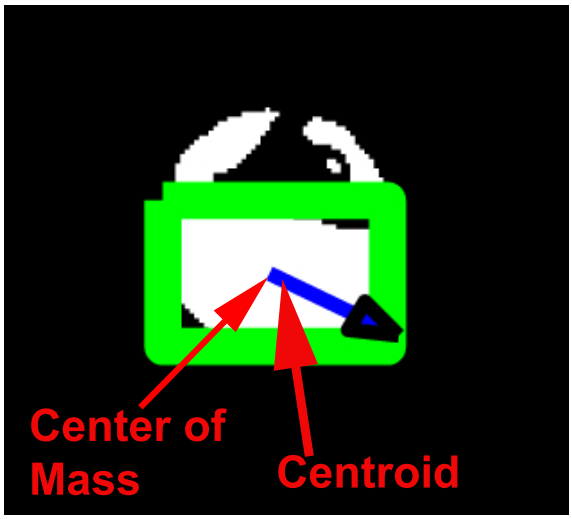

Figure 10: Examples of correct, incorrect and missed arrow orientations respectively

An example of incorrect and missed orientation in an image frame.

3.4 Background Images

An example of background images produced using the Background Estimation Algorithm detailed in *Section 3.4*.

4 Discussion

Assess the success of the program with regard to:

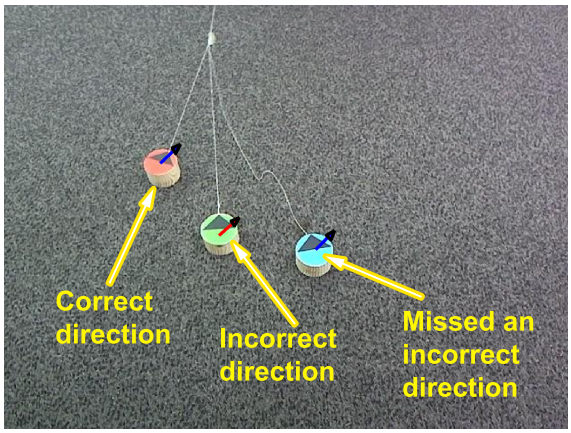

Figure 11: Examples of correct, incorrect and missed arrow orientations respectively

Figure 12: The estimated background image of the first dataset provided for image processing

Figure 13: The estimated background image of the second dataset provided for image processing

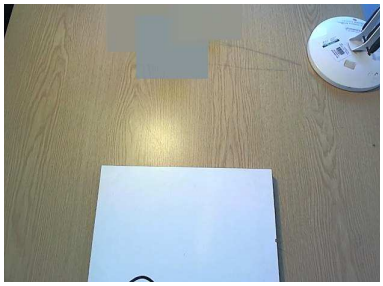

Figure 14: The estimated background image of a new test dataset

Figure 15: The estimated background image of a new test dataset

5 Code

6 Conclusion

APPENDIX A

A.1 Control Program