Robot Tracking

Luca Scimeca and Naohiro Kakimura

March 10, 2016

Contents

1 Introduction **1**

2 Methods **1**
2.1 Background formation 1
2.2 Background subtraction 2
2.3 Object location and Object recognition 3
2.4 Direction prediction 6

3 Results **8**
3.1 Background creation and background subtraction 8
3.2 Object location and object recognition 10
3.3 Direction prediction 11

4 Discussion **11**

5 Code **13**

1 Introduction

Object recognition is the task of identifying objects in an image or a sequence of them. In this report a particular approach to object recognition is discussed. The approach is structured around the location identification and property abstraction of three moving *robots* of different colors (red, green and blue) in varying backgrounds. The approach makes use of MATLAB to perform a pipeline of image processing tasks roughly divided into 5 categories, namely: background formation, background subtraction, robot location, object recognition, and direction prediction.

2 Methods

To build an algorithm capable of recognizing the three robots in any environment, three different data sets are used. Each of the sets comprises several hundreds *jpg* images corresponding to the respective frames of a video, recording the motion of the robots in various directions. The videos are recorded from a fixed camera placed above the field at slightly varying distances from the objects. Moreover, a different background is used in each of the three data set.
Initially, each loaded frame in MATLAB is represented by a 640x480x3 multidimensional vector, which corresponds to the Red, Green and Blue 640x480 matrices forming the picture. To locate the robots it is useful to first retrieve a binary representation of the objects in the image, i.e. for each frame, to compute a 640x480 binary matrix whose elements equal 1 if they are part of the object and 0 otherwise. Several different image processing techniques can be used to apply such a transformation, for the purpose of this experiment, background subtraction is performed.
The idea behind background subtraction is to use a background picture, taken at priory and containing none of the objects to recognize, and subtract away its values from the actual pictures to process; as the values representing the objects are meant to be different to those of the background (so that recognition is at all possible), the absolute value of the subtraction should be highest for those pixels which are part of the object, and lowest otherwise [1].
Other techniques were considered for the experiments but eventually ruled out. The simplest way of realizing a binary representation of the objects in an image is perhaps thresholding. The most basic thresholding technique is applied to gray scale images, where a simple value ranging from 0 to 255 describes each pixel. The idea behind thresholding lies in the darkness of objects which, if assumed darker or lighter than the background, can be easily separated by filtering to 1 only those pixels whose value is higher or lower than a threshold θ respectively. Thresholding is not ideal if the objects are not clearly separable from a gray scale prospective and as shown in Figure 1, it is hard to distinguish at least two curves representing each either the background or the object darkness distribution in the image.

2.1 Background formation

To perform background subtraction, it is necessary to first obtain a background image for each data set. Since the stream of images all contain the objects to recognize, a background must instead be computed. One property that is assumed to be true for the experiment is that the objects move to different positions as the stream of frames progresses. At this point, it is possible to pick a frame every fixed number of pixels and be confident, if enough samples

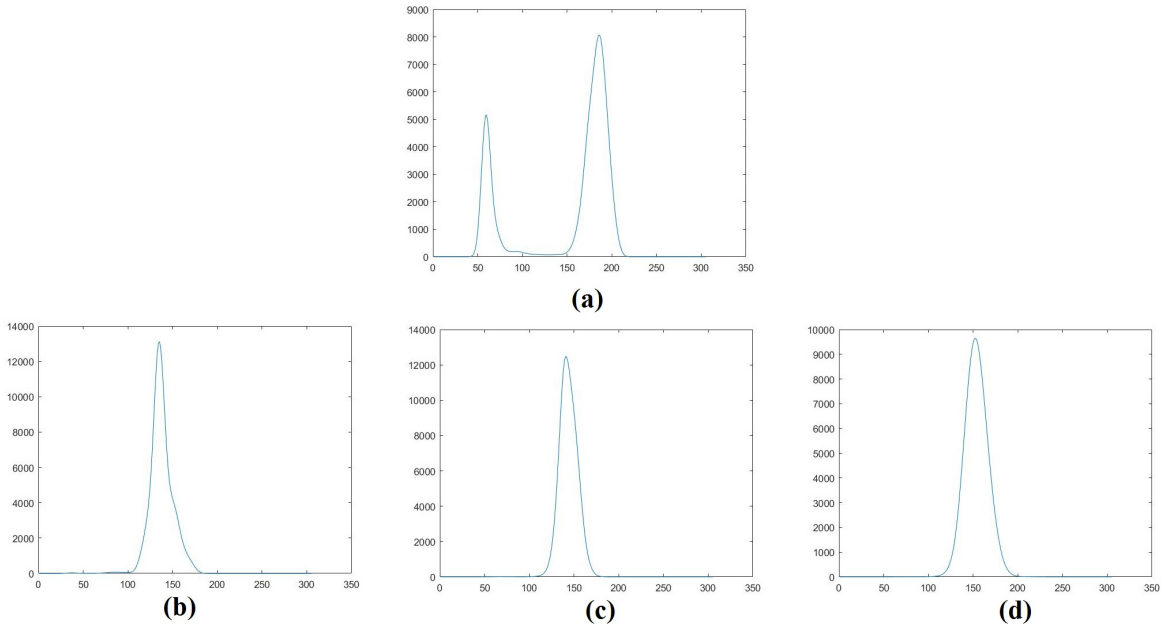


Figure 1: *Smoothed* histograms of the pixel darkness values of three random gray scale images from data set 1 (b), 2 (c) and 3 (d) against an ideal histogram for thresholding (a). The histograms show how the darkness of the objects and the background are not reliably separable (i.e. they do not present two easily separable curves).

are retrieved, that a pixel will have its *background value* at least most of the times. Initially then, 15 images for each data set are picked at increments of 10. The images are split into their Red (R), Green (G) and Blue (B) matrix component, appended into three temporal matrices and, only after, merged into one containing for each element, the median of the values of that elements in the chosen frames.
From an empirical observation of the data it was observed how the robots were initially stationary, i.e. some pixels in the image were initially occupied by the same objects for several frames in different data sets. The background image computation previously described relies on the assumption that objects are not stationary and can be affected by such a behavior as the median pixel values, in some locations, can be picked to be those showing when an object is present. To compute a more reliable background, the first 50 frames for each data set are thus skipped (see Figure 2).

2.2 Background subtraction

Once the background for the dataset is obtained, the images are progressively loaded and processed as they are shown through the *imshow(image)* MATLAB function.
The binarization of the picture frame is achieved in two steps:

1. **Background subtraction**: Initially, the background image is subtracted from the frame to be processed and simple thresholding on the absolute values of the obtained matrix is used to locate those parts in the image occupied by the objects. The matrix resulting from the subtraction, as expected, presents low RGB values where the object is

(a) (b)

Figure 2: Image (a) shows the background image created by using 20 frames at 10 increments from the third data set. Image (b) shows the same computation on 15 frames loaded at increments of 10 after the first 50 frames. Image (a) if heavily affected by the red and green object being stationary for several consecutive frames.

not present, and higher values for the pixels contained within those. A simple threshold on the obtained matrices filters the background from the objects by setting to 1 only those values (in the difference matrix) high enough to be most likely part of the object, and zero the remaining.
The pixel values in both the current frame and the background image range from 0 to 255 in all dimensions. The threshold for the binarization of the subtracted matrix is picked to be 20, as almost all pixel values, in positions where the object is present in the matrix, show to be less than the threshold (see Figure 3). Figure 4 shows a cumulative frequency of the absolute difference of the pixels in the RGB channels of each robot in a random image.

2. **Noise removal**: After the subtraction, variations in the background of the frame with respect to the computed background can make the binary image retrieved present small *blobs* scattered around the objects. To remove the blobs, all clusters in the binary image with areas smaller than 300 pixels are set to 0. As the objects in all datasets present areas covering at least 600 pixels, the threshold reliably removes background noise. Afterwards, morphological closing is applied through the *imclose()* MATLAB function, which returns a close to perfect match between the clusters and the objects' shapes.

2.3 Object location and Object recognition

To link each object in the current frame to be processed to the previous, two information can be very useful: the color of the object, and its position. To extract these two properties, some preprocessing is necessary.
Once a binary image is obtained by background subtraction, the MATLAB function

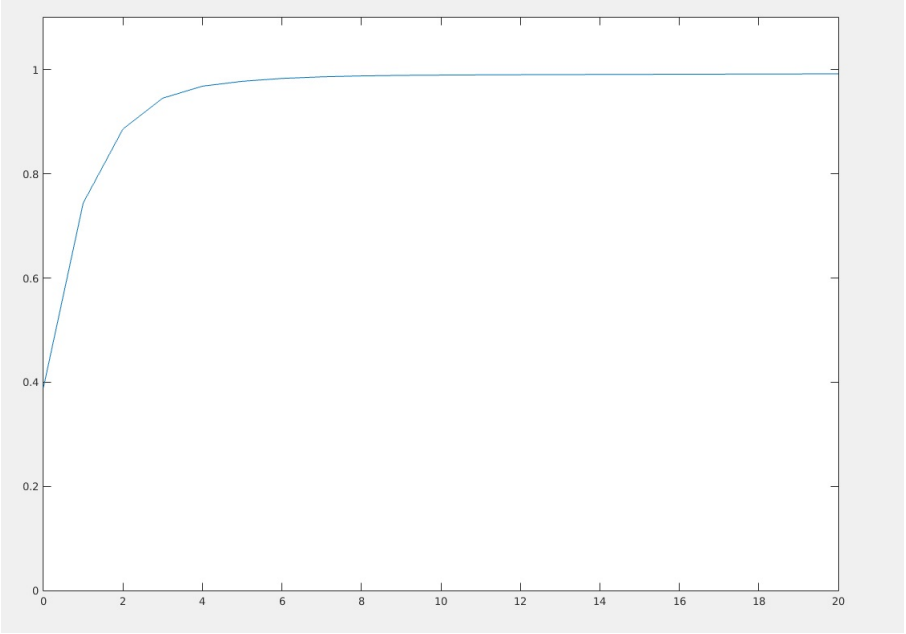


Figure 3: The figure shows shows a cumulative frequency of the absolute difference of the pixels in the red channel.

regionprops(BW, properties) can be used to obtain information on the shape and position of the binary objects within it. The information includes the *centroids* of the object clusters and the length of the *longest axis* within each of them. This step of the algorithm can be divided in 5 tasks:

- **Preprocessing**: Initially, it is useful to normalize the RGB values of the image to obtain what is known as the RG chromaticity. The normalized pixel values will range from 0 to 1 (as opposed to the initial 0-255), and always sum to 1 across the three dimensions (RGB). The RG chromaticity remains fairly constant from one frame to the other, and has very little dependencies to the lighting conditions, two very desirable properties in an image. Normalization for a pixel value can be achieved by $R = \frac{R}{R+G+B}, G = \frac{G}{R+G+B}, B = \frac{B}{R+G+B}$ where R, G and B will be the updated component matrices of the new normalized RGB image.
  Once the current frame is normalized, it is useful to *filter* it in such a way that the RGB pixel values are zeros everywhere but where the objects are located. This, allows for any processing done on the objects not to be affected by the neighboring background pixels. Filtering can be achieved by doing a point wise product between the binary image (extended to three dimensions) and the original frame.

- **Cropping**: The preprocessed image has now all the characteristics needed to be used for object properties retrieval. The major axis lengths previously extracted are here used to create a square frame around each object and the *imcrop(img, frame)* MATLAB function is used to crop out each robot in the frame.

- **Template creation**: As each object is cropped out from the frame, a red, green and blue target templates, equal in size to the cropped objects in the image, are created.

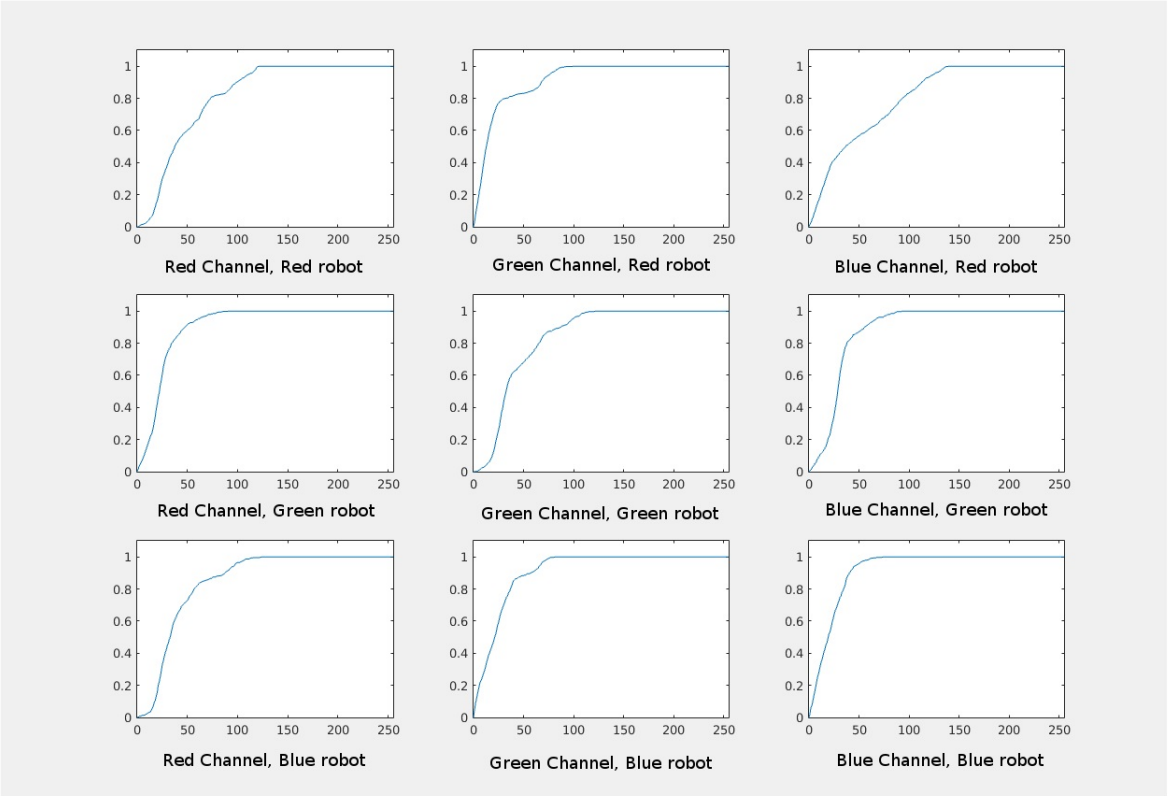


Figure 4: The figure shows the cumulative frequency of the absolute difference of the robot and the background in all combination of channels and robots

The templates are three images whose pixel values are zero everywhere but in the R, G and B component respectively, where a matrix of ones is instead placed. The target pictures represent the ideal (normalized) red, green and blue images to be matched against the obtained framed objects.

- **Template matching and recognition**: Each object needs now to be matched against the three templates so to assign it the color of its *closest* one. The Euclidean distance is here used as a measure of how close each cropped object is to each color.
  To improve object recognition, a second step *corrects* the assignments with the previous positions of each robots (red, green and blue). If the current position of an object is far too distant to its previous location, given the initial assumption, it is safe to assume the robot has not been recognized properly, and its location is overwritten with its previous values. The set distance after which an object position will be adjusted is proportional to skipping up to n frames (assuming the each object moves from frame to frame as initially described).

At each frame, the object positions of the red, green and blue robots are appended to a list of positions which allows the *path* so far followed to be traced (see Figure 12).

2.4 Direction prediction

Predicting the direction of the robots reliably is not by any means an easy task. The main idea behind the way the algorithm predicts the robot directions evolves around the observation that the arrows are shifted from the center, more importantly, shifted towards the direction the robot is facing. The direction prediction algorithm can be broke down to three fundamental tasks:

1. **Object windowing**: Much like object recognition, each object in a frame is windowed using its longest axis length. In fact, the code is built such that the windowed objects are used for both the task of recognition and direction prediction at once, increasing efficiency in the use of resources.

2. **Component isolation**: Once the object is windowed, two binary matrices must be build from it: one representing the blob or *colored disk* within the object and another representing the *arrow* within it. Note that the blob representing the colored disk may be different from the blob representing the object itself, given none of its sides should be present in the disk (see the *Discussion* section below).
   To isolate a disk such that no sides of the robots are visible, the color information within the windows are used. The cropped image, in fact, is first converted to *HSV* (Hue, Saturation, Value) and then split to its respective matrix components. The Hue Matrix holds the very important information of color, which can be used to isolate only the red, blue and green disks within the objects. Three different ranges are empirically found such that within each colored object only the right information is extracted. The Hue matrices are then thresholded through the ranges, and later post-processed to remove the noise within and outside the isolated region.
   Since a similar procedure on the Value component of the images results unreliable in isolating the arrows within the disks for different datasets, a slightly more complex procedure is needed. a better method to achieve automatic thresholding of the arrows is via *multi thresholding*. The *multithresh(Image, n)* MATLAB function uses the Otzu method to find up to n threshold levels in the image. The windowed objects are then thresholded through a specific level of threshold which removes all by the darkest region in the grayscale version of the image to be isolated. This is made more robust by removing the neighboring noise through the previously obtained binary disks (since the arrows are placed within them).

3. **Feature extraction**: Each objects' center can now be computed, much like previously, through the *regionprops(BW, properties)* MATLAB function, and a line can be traced from the centroid of the colored disk to the centroid of the arrow as the predicted direction of the object.

Figure 5 shows an overview of the processing steps to predict the direction of an object from the algorithm.

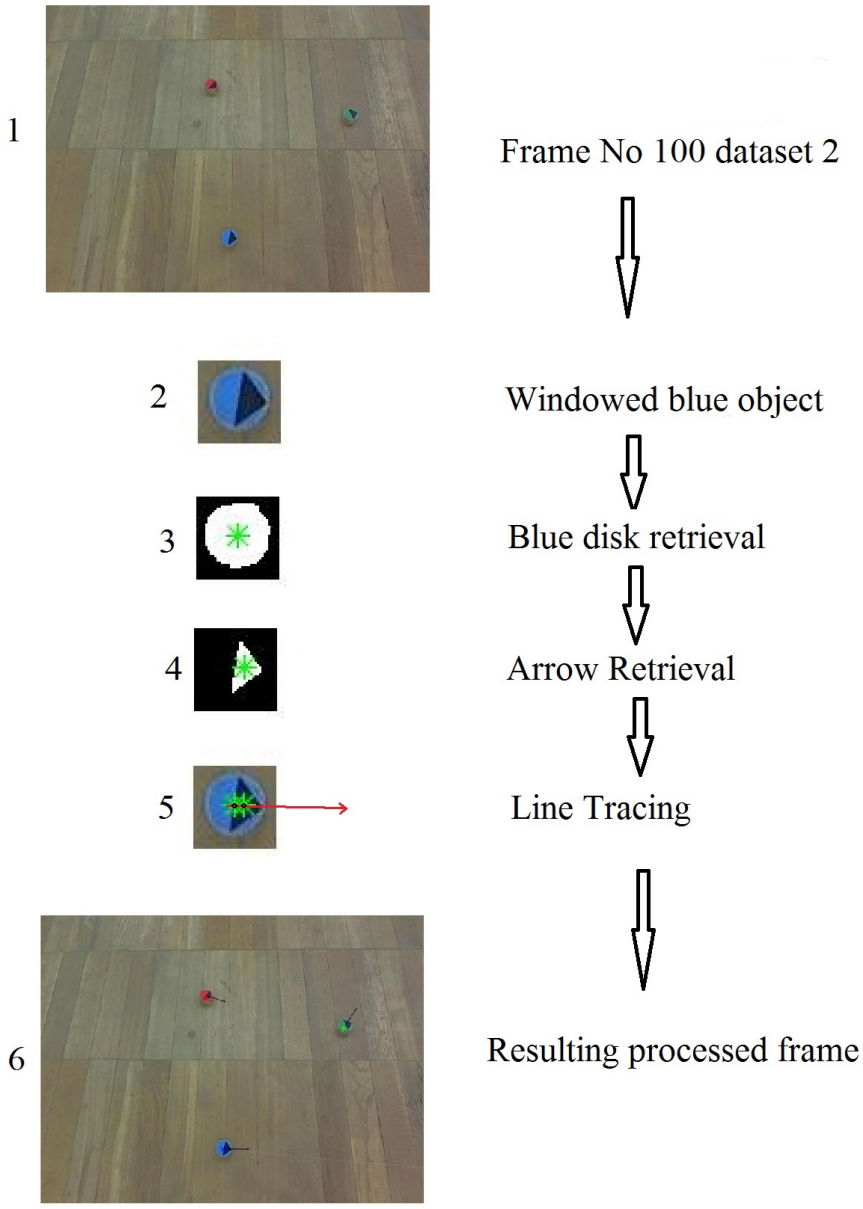


Figure 5: Process pipeline of an object whose direction is being predicted.

1. The frame is captured and loaded in MATLAB.
2. The object is windowed.
3. The blue disk is binarized and its center of mass is computed. For binarization, thresholding is applied to the Hue component of the HSV version of the framed object. The range for thresholding is chosen empirically, and results stable as it is independent of both Saturation and Light.
4. The arrow within the disk is binarized and its center of mass is computed.
5. The two centers are connected and used to predict the object's direction.
6. The process is extended to all objects in the current frame.

3 Results

3.1 Background creation and background subtraction

Each of the three data sets is used to test the algorithms separately but following the same steps. Initially, as the path to the images is specified, the images are loaded at steps of 10 after the first 50 frames, and the backgrounds are created using the median values for each of the pixels in the different images. Figure 6 shows the three backgrounds as they are created after the first step of the algorithm. The backgrounds result accurate as they present all the characteristics of the original pictures but no trace of the robots within those. The times for the background images to be created are reported in Table 1.

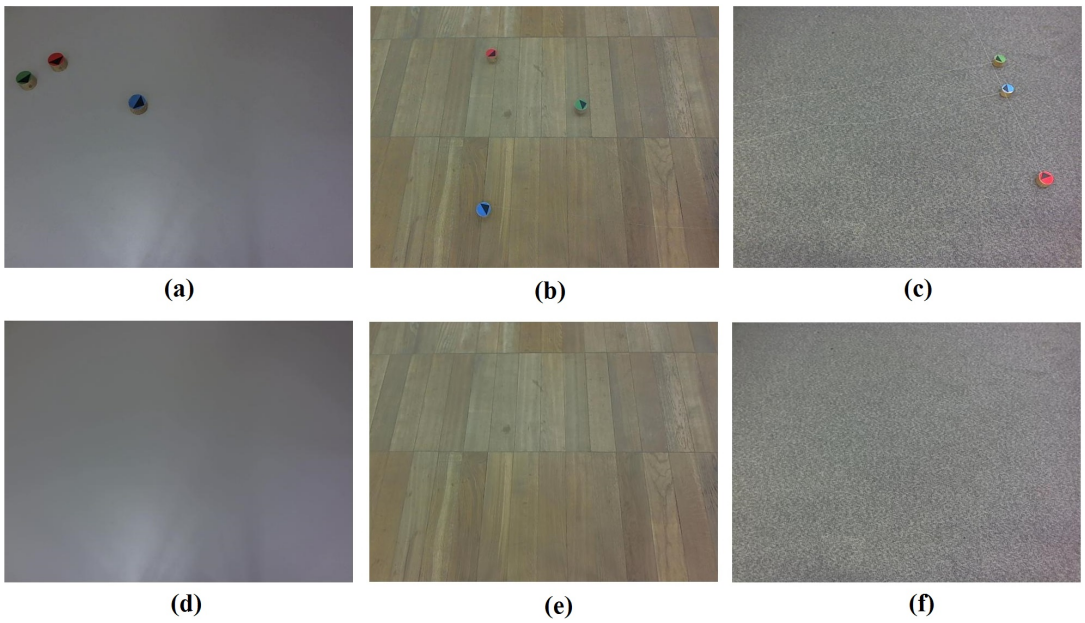


Figure 6: The figure shows the backgrounds created from the data sets one, two and three respectively as (d), (e) and (f). The backgrounds show no trace of robots ((a), (b), (c)) and are thus suited for background subtraction.

Table 1: Table showing the time (sec) elapsed for the creation of the backgrounds in dataset one, two and three.

	time elapsed for creation (sec)	No. of elements loaded for creation
Background dataset one	0.30542	15
Background dataset two	0.50298	15
Background dataset three	0.47082	20

Once the background is created all images in the data set are loaded one by one, and shown in sequence as they are processed. For each loaded frame, background subtraction allows for a binary image, showing the locations of the three objects, to be created. Background removal always shows big clusters where the robots are located, but noise is often present both within and outside the object areas.

Figure 7 shows how some noise is present in the binary version of a random frame after background subtraction. Figure 9 shows instead how the algorithm picks up the strings used for the robots' motion.
From the graph, graph about 40% of the pixels are less than 20, however to create a binary image we take logical or function and not all pixels have high RGB value differences. As a result only 10% of the robot is removed together with the background. 99% of the background for 10% of the robot is a good trade off. As previously described, noise removal is achieved through the *imclose()* MATLAB function. Figure 8 and Figure 10 show the binary images after being processed by the function.

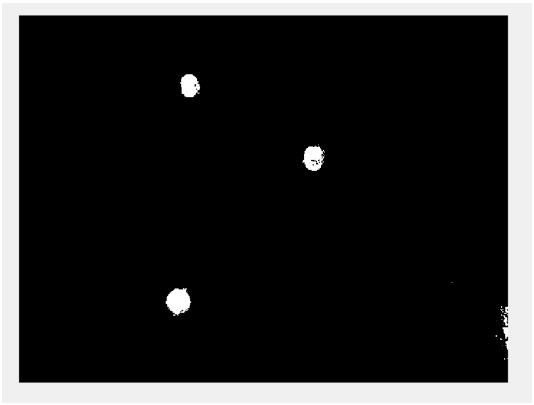


Figure 7: Binary image obtained after simple background subtraction in dataset two.

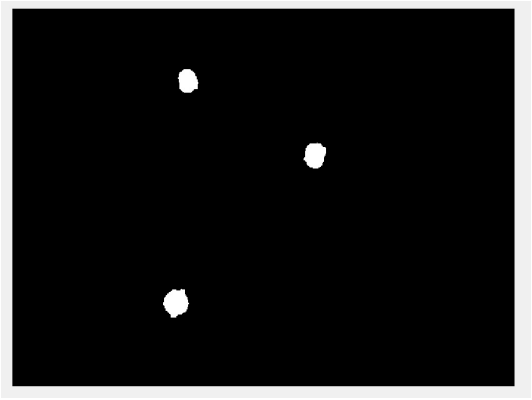


Figure 8: Binary image obtained after noise removal in the image in Figure 7

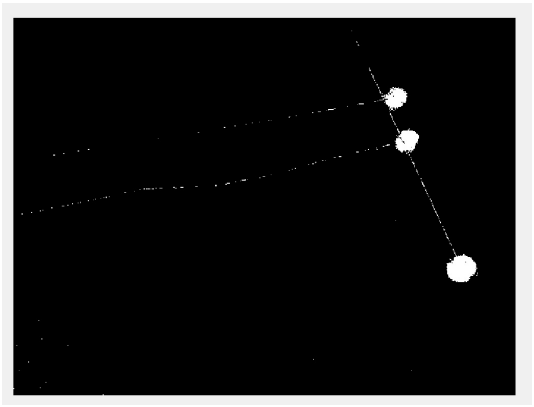


Figure 9: Binary image obtained after simple background subtraction in dataset three.

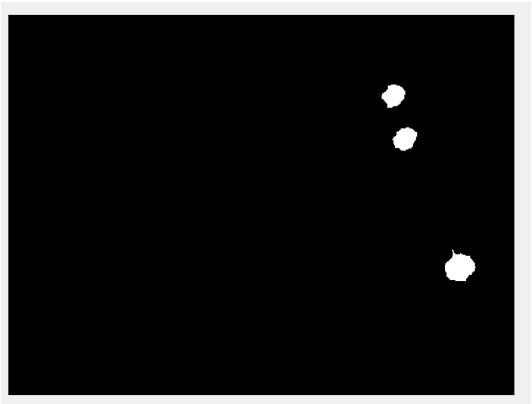


Figure 10: Binary image obtained after noise removal in the image in Figure 9

As the area specified for noise detection is of 300 pixels, the algorithm fails to detect objects whose area covers less than the threshold. Although no objects in the dataset present such size, binarization fails to detect objects moving out of the camera's reach as these become smaller and smaller and eventually fade (see Figure 11). Moreover, objects who's RGB values are similar to those of the background may also be removed and morphological closing may

cluster together objects too close to each other, mistaking two robots as one (although no frame in any dataset shows this behavior).

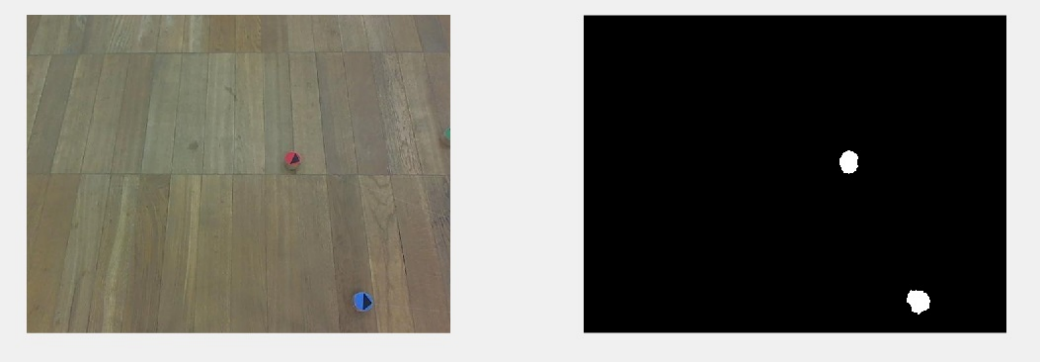
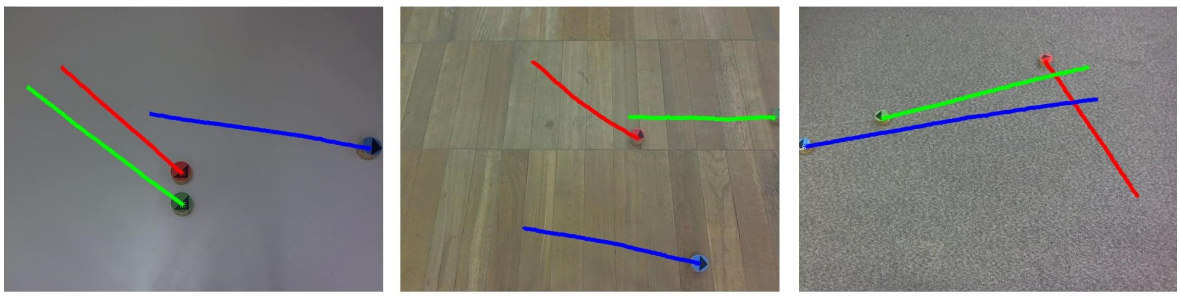


Figure 11: The figure shows how the algorithm fails to detect the green object on the right side of the camera field. The object's area results to be less than 300, and is therefor removed by the *imclose()* function as noise.

3.2 Object location and object recognition

As previously explained, the algorithm bases recognition (i.e. which colored robot is which) on *closeness* to a template color image created artificially. After assigning colors, a second step adjusts the assignments based on the previous positions of the robots. Like shown in Figure 12, the paths traced by the algorithm in all three datasets result in homogeneous color lines. Since the counts for color readjustments based on position correspond to 0 for all datasets (and thus the positions for each colored robot, in the three datasets, are always those assigned by the Euclidean distance classifier), is it safe to infer the algorithm does not miss classify any of the objects in the processed frames.



(a) (b) (c)

Figure 12: Images corresponding to the paths traced by the algorithm after 200 frames in data set one (a), data set two (b) and dataset three (c).

The misclassification rates are highly dependent on background subtraction, as the images are preprocessed and filtered through their previously obtained binary version. Filtering, allows for anything but the area where the objects are located to *participate* to the euclidean distance classification. Previous versions of the algorithm skipped filtering and applied a fixed window to each object. The objects, where thus taken with a small portion of the background

which would throw off the classification by a large margin in some of the datasets. The small background portion captured, in fact, would make each of the framed objects very distant from the target color template and misclassification were consistently higher. After filtering and applying a scaled window of the exact size of the object, to be fed to the Euclidean classifier, however, the objects become much closer to their true templates and thus achieve better performance overall (see Discussion section for more on Euclidean classification).

3.3 Direction prediction

Direction prediction results stable for most datasets. The reliability of the algorithm is directly proportional to the accuracy of the binary colored disk and arrow retrieval. When this fails, the predicted direction is thrown off by a large margin (see Figure 13).

4 Discussion

The algorithm used for the tasks of object location, object recognition and direction prediction gives overall accurate information on the position, type and direction of each robot in different frames. Background subtraction, although simple and effective, has many drawbacks. One of the main issues is that the images must be taken from a fixed position and the background must be unchanged for the algorithm to work at its best. This will therefor not be solid against datasets whose images are taken from varying positions. Moreover, care must be taken in choosing which representation of the pixel values is to be used for the subtraction as different representation may be less robust under different conditions (e.g. lighting). Lastly, for a background to be created successfully the assumption by which each object in the image moves from frame to frame must be true. If any of the objects were to be stationary for most of the frames in the dataset, they could not be detected as subtracting away the background would only return zero matrices (or close to zero) for those frames where the object is in the same location as the background.
Once the binary image is correctly classified, object location relies on two main information: the color of the robots, and their position relative to the previous frames. Color distinction was initially implemented by utilizing the Hue values. This approach, if accurate for two data sets, would miss classify over 20% of the frames in data set three. The Hue values of the red and blue objects in the third data set in fact, started showing similar values roughly half way thorough the frames, and even position adjustment would eventually fail to keep the paths of the robots consistent through the whole run of the algorithm.
Template matching through Euclidean distance achieves perfect recognition for the data in the three sets. However, the human perception of the distance amongst different colors is hardly Euclidean [2]. The objects are geometrically closest in distance with their corresponding color templates, which allows the algorithm to correctly attribute colors to objects; the distance, however, becomes less and less accurate as colors differ more and more from the templates. The algorithm then, is highly dependent on how accurately the binary image *cuts* out each of the robots in a frame as the robots must be closest to their templates for classification to be done correctly.
The method for direction prediction results more robust than previous versions relying on the shape of the triangles on the robots. For this method to reliably give direction information, it is essential that the binarized regions belong solely to the colored disk and the arrow within the objects. thresholding by color isolation is robust for all frames in the datasets. The

arrow's retrieval, however, is thrown off by areas in the image (e.g. shadows cast by the robots) which result in similar tones of black. The masking through the previously obtained binary disks is here essential for noise removal.

References

[1] Robert B. Fisher. Finding objects by background removal, January 2016.

[2] Sagarmay Deb. *Multimedia Systems and Content-Based Image Retrieval.* IDEA GROUP PUBLISHING, July 2003.

word count = 3893

5 Code

```
function  ivrProcessing( path, img_count, show )
%Function which traces a path of three (red, blue and green) moving objects.


%   This function taks a path to a dataset containg consecutive frames of
%   video feedback, loads a part to create a background image and uses the
%   background image to locate three objects in each frame. Each object is
%   recognized as red, blue or green and a path is traced connecting the
%   various positions the robots take through their motion. Moreover, a
%   direction for each of the object is shown. The direction is computed to
%   be the one which the robots are facing at any given time.
%
%    INPUTS:
%       path      - path to the folder containing the frames to be processed
%       img_count - number of frames to process in the folder
%       show      - set to 1 if the results are to be shown in a window


%---------------------creates background image-------------------------------

%   Create a background image for the dataset
    tic
    bkgrImg = background_image(path, 50, 200, 10, 0);
    toc
%   Background image created!!

%-------------------------------------------------------------------------

%-----------initialize variables for image processing ----------------------

    imgs = zeros(480,640,3,img_count,'uint8');
    thres = 20;
    blob_size = 300;

    % variables holding the paths of the robots throughout the frames
    red_centroids = [];
    green_centroids = [];
    blue_centroids = [];

    %belief positions (to be updated continuously thoughout the program)
    redPosition = [0,0];
    bluePosition = [0,0];
    greenPosition = [0,0];


%-------------------------------------------------------------------------%
%-------------------FRAME PROCESSING BEGINS-------------------------------%
%---------load one frame at each iteration of the for loop---------------%


    for j = 1:img_count
        current_frame = loadimage(path, j, 0);
```

```
%------------------------Do background subtraction-------------------------

        %subtract background grom image
        binaryImage = backgrSub(current_frame, bkgrImg, thres, 0);


%------------------------Do object recognition-----------------------------
%--------------------------------&-----------------------------------------
%-----------------------Direction prediction-----------------------------

%object recognition of objects in current frame with respect to the objects
%to the previous

        %Get the properties of the robots(centroids and blob's widths into
        %a struct)

        properties = regionprops(binaryImage,'Centroid', 'MajorAxisLength');

        %retrieve information on object identity, position and direction
        %from the frame.
        [robot, directions] = get_properties(properties, current_frame, binaryImage, ...
            redPosition, greenPosition, bluePosition);

        [~, id] = lastwarn;
        warning('off', id)

        % adds to arrays of centroids to build up current path
        if not(isnan(robot('red'))) & not(all(robot('red')==0))
            red_centroids = horzcat(red_centroids, robot('red'));
            redPosition = robot('red');
        end
        if not(isnan(robot('green'))) & not(all(robot('green')==0))
            green_centroids = horzcat(green_centroids, robot('green'));
            greenPosition = robot('green');
        end
        if not(isnan(robot('blue'))) & not(all(robot('blue')==0))
            blue_centroids = horzcat(blue_centroids, robot('blue'));
            bluePosition = robot('blue');
        end

%-------------------------------------------------------------------------

        %prints all results on the current image (frame) to be displayed
        if show > 0 || j == img_count
            figure(1)
            imshow(current_frame);
            hold on
            %plots the centroids of the robots
            plot(redPosition(1), redPosition(2), 'r*')
            plot(bluePosition(1), bluePosition(2), 'b*')
            plot(greenPosition(1), greenPosition(2), 'g*')

            %plots the paths of the robots
            if(not(isempty(red_centroids)))
                r = plot(red_centroids(1,:),red_centroids(2,:), '-g','LineWidth',4);
                set(r,'Color','red');
```

```
            end

            if(not(isempty(blue_centroids)))
                b = plot(blue_centroids(1,:),blue_centroids(2,:), '-g','LineWidth',4);
                set(b,'Color','blue');
            end
            if(not(isempty(green_centroids)))
                g = plot(green_centroids(1,:),green_centroids(2,:), '-g','LineWidth',4);
                set(g,'Color','green');
            end

            %plots predicted direction of robots
            [dims,~] = size(directions);
            for i = 1:dims
                p1 = directions(i,1:2);
                p2 = directions(i,3:4);
                diff = p2-p1;
                quiver(p1(1),p1(2),diff(1), diff(2),'AutoScaleFactor',10,'color',[0 0 0]);
            end

            hold off
            drawnow;
        end

    end

end


function img = loadimage(path, img_num, show)


%Function loading the image number 'img_num' from the path specified in
%'path'. The variable 'show' allows for the loaded image to be displayed
    img = importdata(sprintf('%s/%08d.jpg', path, img_num),'jpg');
    if show > 0
        figure(show);
        imagesc(img);
    end
end


function bk_img = background_image(path, from, to, inc, show)
%Function that computes a background image.


%   The background is created by loading all images from number 'from'
%   to number 'to' with increments 'inc' in folder 'path'. The show
%   variable if set to 1 shows the final background computed

    % load the needed images from the dataset
    imgs = loadimages(path, from, to, inc, 0);

    imgs_dim = size(imgs);
```

```
    % break the loaded images in their RGB components
    bk_r = imgs(:,:,1,:);
    bk_g = imgs(:,:,2,:);
    bk_b = imgs(:,:,3,:);

    bk_r = reshape(bk_r, imgs_dim(1),imgs_dim(2),imgs_dim(4));
    bk_g = reshape(bk_g, imgs_dim(1),imgs_dim(2),imgs_dim(4));
    bk_b = reshape(bk_b, imgs_dim(1),imgs_dim(2),imgs_dim(4));

    % computes median of all images for each pixel
    bk_r = median(bk_r, 3);
    bk_g = median(bk_g, 3);
    bk_b = median(bk_b, 3);

    %assembles for background creation
    bk_img = cat(3, bk_r,bk_g,bk_b);
    if show > 0
        figure(show);
        imagesc(bk_img);
    end
end



function bin_img = backgrSub(img, bk_img, thres, show)

%Function performin background subtraction



%   This function computes a binary image of the objects in a frame, given
%   the frame and a background of the same.
%
%   INPUTS:
%       img         - current frame to be processed
%       bk_img      - background image of the frame to binarize
%       thres       - threshold value to be used for masking
%       show        - variable set to 1 if a window with the resulting
%                     binary image is to be displayed
%
%   OUTPUT:
%       bin_img     - dictionary of robot colors (keys) and their respective
%                     positions
%

    % background subtract
    img_sub = imabsdiff(img, bk_img);
    bin_img = img_sub > thres;
    bin_img = bin_img(:,:,1) | bin_img(:,:,2) | bin_img(:,:,3);

    % remove noise from binary image
    bin_img = bwareaopen(bin_img, 300);
    se = strel('disk',5);
    bin_img = imclose(bin_img,se);


    if show > 0
```

```
        imshow(bin_img);
        drawnow;
    end

end


function [robots, directions] = get_properties( properties, image, mask, ...
    prevRed, prevGreen, prevBlue )
    %Function retrieving the basic object properties in a frame



    %   This function, computes the direction and position of three object: a
    %   red object, a blue object and a green object.
    %
    %   INPUTS:
    %       properties - struct of Centroids, MajorAxisLength and BoundingBox
    %       image      - current frame to be processed
    %       mask       - binary version of the frame with respect to the
    %                    objects in it
    %       prevRed    - previous position of the red object
    %       prevGreen  - previous position of the green object
    %       prevBlue   - previous position of the blue object
    %
    %   OUTPUT:
    %       robots     - dictionary of robot colors (keys) and their respective
    %                    positions
    %       directions - list of pairs of points giving the direction of each
    %                    object
    %


    % variable initialization for later use
    directions = [];
    centroids = cat(1,properties.Centroid);
    dims = cat(1,properties.MajorAxisLength);
    imgMask = double(cat(3,mask,mask,mask));


    %------------normalization----------------

    %normalizes RGB picture and filters out the objects
    image = double(image);

    Red = image(:,:,1);
    Green = image(:,:,2);
    Blue = image(:,:,3);
    base = Red + Green + Blue;

    NormalizedRed = Red./base;
    NormalizedGreen = Green./base;
    NormalizedBlue = Blue./base;

    newimg = cat(3,NormalizedRed,NormalizedGreen,NormalizedBlue);
    newimg = double(newimg).*imgMask;
    %newimg is normalized!!
```

```
%images for direction prediction
tmpimg = double(image).*imgMask;
BlackBackgroundImage = uint8(tmpimg);
WhiteBackroundImage = BlackBackgroundImage;
WhiteBackroundImage(~imgMask)=255;

%dictionary of recognized objects to be returned
robots = containers.Map;

%initializes dictionary for final positions
robots('red') = prevRed;
robots('green') = prevGreen;
robots('blue') = prevBlue;



%-----------Objects' processing-----------------------------------------------

%iterates through the objects one by one and process them
[propDims, ~] = size(centroids);

for i = 1:propDims

    x = int16(centroids(i,1));
    y = int16(centroids(i,2));

    HalfWindow = floor(dims(i)/2)+1;

    %specifies window to crop out the object from the image
    rect = [x-HalfWindow, y-HalfWindow, 2*HalfWindow, 2*HalfWindow];

    %crops
    crop = imcrop(newimg,rect);

    [xdim, ydim, ~] = size(crop);

    % creates template red, blue and green image
    red = double(cat(3,ones(xdim,ydim),zeros(xdim,ydim),zeros(xdim,ydim)));
    green = double(cat(3,zeros(xdim,ydim),ones(xdim,ydim),zeros(xdim,ydim)));
    blue = double(cat(3,zeros(xdim,ydim),zeros(xdim,ydim),ones(xdim,ydim)));


    %computes distance from cropped image to templates
    tmp = (crop-red).^2;
    redDist = sqrt(sum(tmp(:)));
    tmp = (crop-green).^2;
    greenDist = sqrt(sum(tmp(:)));
    tmp = (crop-blue).^2;
    blueDist = sqrt(sum(tmp(:)));

    %retrieves Hue values of the object to cut out only colored disk
    % (to be later used for direction prediction)
    framedObject = imcrop(WhiteBackroundImage,rect);
    framedObject = rgb2hsv(framedObject);
    framedObjectHue = framedObject(:,:,1);
```

```
    %assigns objects to their closest color match
    %and retrieves binary colored disks (for direction prediction later)
    if (redDist<greenDist&&redDist<blueDist)
        robots('red') = [x,y]';
        bwBlob = (framedObjectHue>.9 | (framedObjectHue>0 & framedObjectHue<.02));
    elseif (greenDist<redDist&&greenDist<blueDist)
        robots('green') = [x,y]';
        bwBlob = (framedObjectHue>.3&framedObjectHue<.48);
    else
        robots('blue') = [x,y]';
        bwBlob = (framedObjectHue>.51&framedObjectHue<.68);
    end

    bwBlob = imfill(bwBlob, 'holes');

    %prepares image frame for retrieving a binary arrow
    crop = imcrop(WhiteBackroundImage,rect);
    crop = rgb2gray(crop);
    crop = imadjust(crop);

    %isolates blobs and arrows in the objects through multithresholding
    threshArrow = multithresh(crop,8);
    seg_Arrow = imquantize(crop,threshArrow);

    %eliminates noisy background
    bwArrow = seg_Arrow==1;
    bwArrow = bwArrow.*bwBlob;
    bwArrow = imfill(bwArrow, 'holes');


    arrowProps = regionprops(bwArrow, 'Area', 'Centroid');
    blobProps = regionprops(bwBlob, 'Area', 'Centroid');

    %computes centeres of blobs and arrows to predict direction
    areas = cat(1,arrowProps.Area);
    centers = cat(1,arrowProps.Centroid);
    arrowProps = horzcat(areas, centers);
    areas = cat(1,blobProps.Area);
    centers = cat(1,blobProps.Centroid);
    blobProps = horzcat(areas, centers);
    if(not(isempty(arrowProps)) & not(isempty(blobProps)))
        arrowProps = sortrows(arrowProps, -1);
        blobProps = sortrows(blobProps, -1);
        arrowCenter = arrowProps(1,2:3);
        blobCenter = blobProps(1,2:3);
    end


    %computes points for direction arrows
    if(not(isempty(arrowProps)) & not(isempty(blobProps)))
        directions=vertcat(directions,[x+blobCenter(1)-HalfWindow,...
            y+blobCenter(2)-HalfWindow, ...
            x+arrowCenter(1)-HalfWindow, ...
            y+arrowCenter(2)-HalfWindow]);
    end
end
```

```
    %adjusts color assignments with previous positions
    if(not(all(prevBlue==0)) & pdist([double(robots('blue'));...
            double(prevBlue)],'euclidean')>35)
        robots('blue') = prevBlue;
    end
    if(not(all(prevGreen==0)) & pdist([double(robots('green'));...
            double(prevGreen)],'euclidean')>35)
        robots('green') = prevGreen;
    end
    if(not(all(prevRed==0)) & pdist([double(robots('red'));...
            double(prevRed)],'euclidean')>35)
        robots('red') = prevRed;
    end
end
```

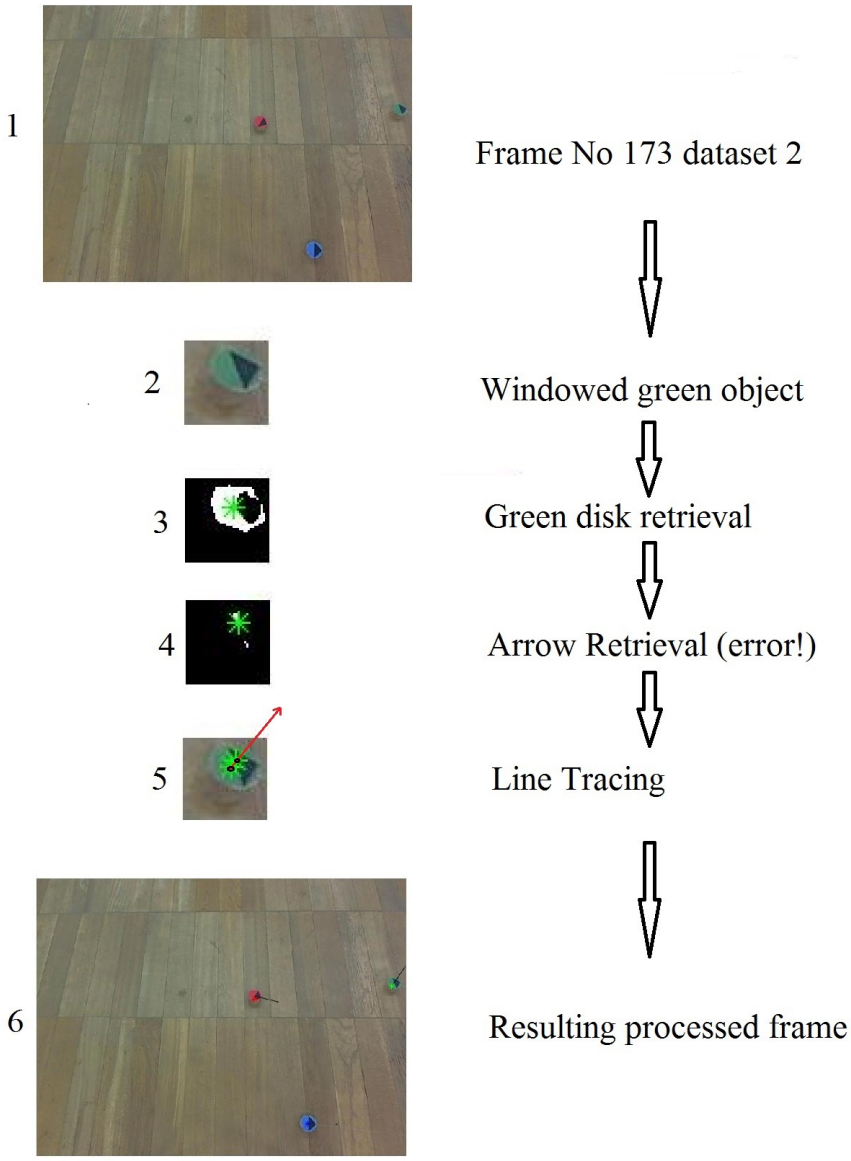


Figure 13: The Figure shows an error in the process of direction prediction for the green object in frame 173 of dataset two. The multi thresholding level picked by the algorithm fails to retrieve the arrow within it.