AVPR - Lab - First Exercise

Carlos García

November 2016

Contents

Search about video processing tools

Before getting into the problem I wanted to make an small research about most common tools when working with image and video processing, this tools are:

- Matlab
- Octave
- OpenCV (using Java or C#)
- ImageJ
- Gimp
- And some others

The last two (ImageJ and Gimp) are specially good when working with images, but for the case of video processing, we should work with faster, stronger and more robust tools.

OpenCV is really fast and has a lot of options (a perfect choice if we are dealing with real time videos), but it is harder for beginners than Matlab or Octave, so I rather going with one of this two.

Octave, is similar to Matlab, only that Octave is open source and free, the problem is that Octave lacks of some of the new options in Matlab.

Matlab is maybe the most known tool for image and video processing, it is fast (not as fast as OpenCV), powerful and has an incredible huge library for this topics. The only problem with Matlab is that you must pay in order to use it, so I'll give it a try using it for free in one of the computers from the university and afterwards will think about buying an student license.

Tips about Matlab software

- Install Matlab:

As I'm working on linux, I'll shortly talk about how to install Matlab in Linux/Debian

Just download the installer file (.tar) from the MathWorks website, then use the unzip command to extract the files from the archive, and after extracting, execute the installer command:

cd Downloads/matlab_R2016b_glnxa64/
sudo ./install

Usually on linux systems Matlab have a problem with a library for processing videos, so we have to install it manually using this commands:

sudo apt-get install gstreamer0.10-*
wget http://ppa.launchpad.net/mc3man/gstffmpeg-keep/ubuntu/pool/main/g/gstreamer0.10-ffmpeg/gstreamer0.10-ffmpeg_0.10.13-5ubuntu1~vivid_amd64.deb
sudo dpkg -i gstreamer0.10-ffmpeg_0.10.13-5ubuntu1~vivid_amd64.deb

- Learn basics about Matlab

Mathworks has a great on-line learning site which is just to get started, but includes all the basics, is free and can be completed in just two hours. It can be found in this link:
https://www.mathworks.com/support/learn-with-matlab-tutorials.html?s_tid=acport_gs_sp_til

  - Some matlab commands
    - clear: removes all variables from workspace
    - clc: cleans up the command window
    - etc
  - Some matlab constants
    - pi: 3.1415...
    - etc
  - Some matlab functions
    - abs: absolute value
    - sin: sin function for a given value
    - sqrt: square root
    - linspace: creates array of values from ini to end using an step parameter
    - ': transpose operator, change rows for columns
    - rand: creates matrix filled with random values
    - zeros: create matrix filled with zeros
    - save foo x: saves x var to a MAT-file named foo.mat
    - load foo: load all variables inside foo.mat file
    - max: return the max value from a vector
    - round: rounded average from vector
    - doc function_name: returns information about an specific function
    - hold on: keep last plots in the same grid
    - close all: closes all open figure windows
    - title: adds a title to a plot

        - ylabel: adds a label to y axis
        - xlabel: adds a label to x axis
        - plot: used to plot variables.
          This can also be acomplished by selecting variables in the workspace and then selecting a type of graph in the plot tab (main menu above)
        - fft: returns the Fourier transform, this gives information about the frequency content of the signal.
        - numel: returns the number of elements in an array
        - etc
    - Some matlab keywords
        - end: returns last position from array
        - etc

- Matlab image processing
    - Read image from file
        - imread('sample-image.bmp')
    - Show one image
        - showim
    - Show more than one image
        - imshopair(im1, im2, 'montage')
        - figure;
          subplot(2,2,1), imshow(rmat);
          title('Red plane');
          subplot(2,2,2), imshow(gmat);
          title('Green plane');
          subplot(2,2,3), imshow(bmat);
          title('Blue plane');
          subplot(2,2,4), imshow(I);
          title('Original plane');
    - Show image histogram
        - imhist
    - RGB to HSV image
        - rgb2hsv(ImageVar)
    - RGB to HSV
    - Morphological filters
        - Open

      se = strel('square', 25);
      Iopenned = imopen(Ifilled, se);
    - Fill image holes
        - imfill(bwImage, 'holes')
    - Binarize image
        - imbinarize(Im, T)
          where T can be an array

- Matlab video processing
  - Read video from file
    - VideoReader('sample-video.avi');
  - Get frame size from video
    - firstFrame = readFrame(V);
      frameSize = size(firstFrame(:,:,1));
  - Set current time for video reader
    - V.CurrentTime = 0;
  - Loop over video frames
    - while hasFrame(V)
      iFrame = readFrame(V);
      end
- Other Matlab tutorial
  - We can also use the basic tutorial uploaded to the Moodle:
    https://moodle.urv.cat/moodle/pluginfile.php/2391953/mod_resource/content/2/MatlabIntroduction.pdf
  - And for any other doubt we can refer to the Matlab official documentation:
    https://www.mathworks.com/help/matlab/

Get into the topic

Problem definition

Process the video "Robot-pushing 2 balls" to detect in each frame the position of the robot (center of mass). Show as a dots in an output image all the positions you have obtained.

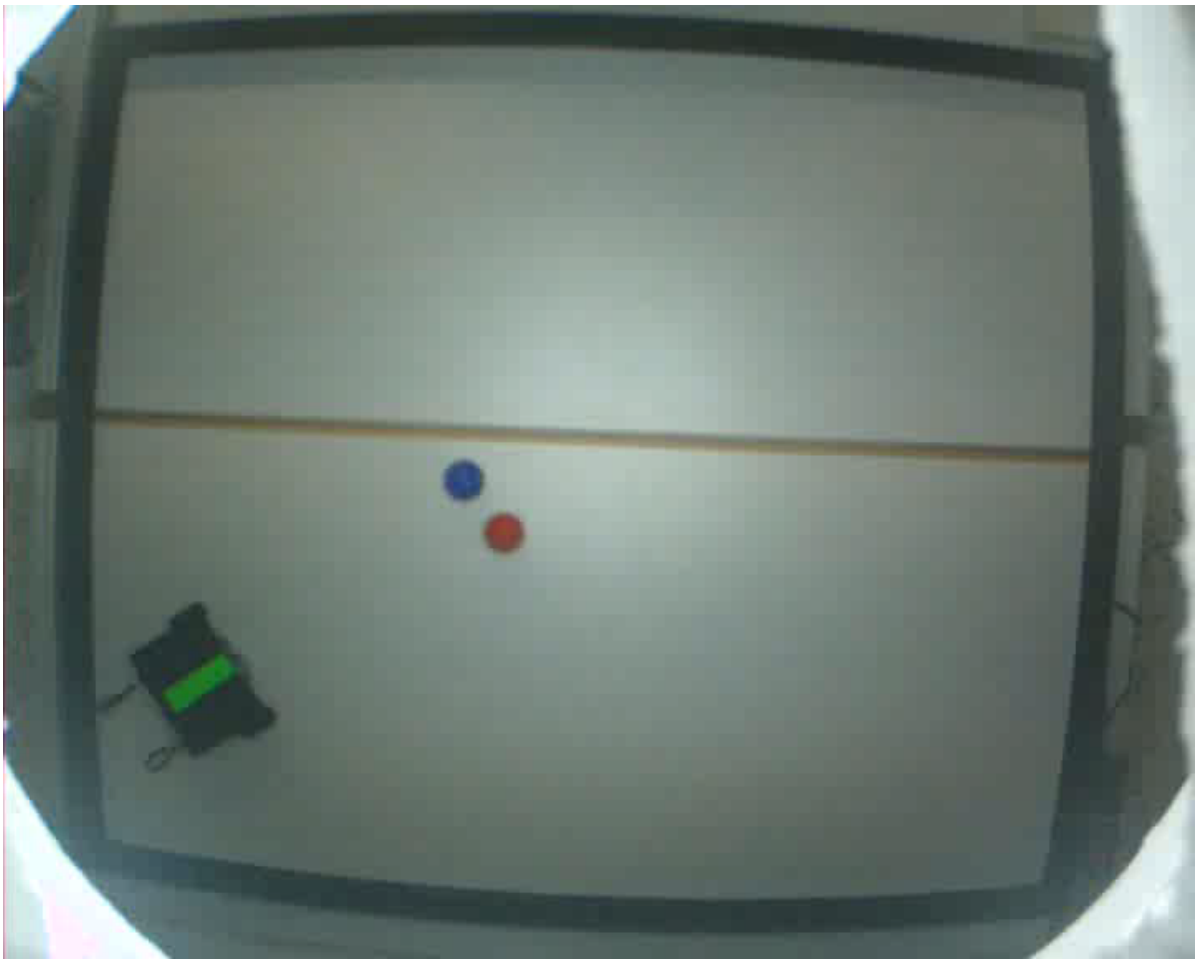
General approach

Before beginning we must first understand the difficult parts in a single image processing and once we finish this process for one image, we try to replicate it for every frame in the video.

Start working with a single image

- First ideas

Once I looked the image for first frame, I knew I had to work processing either color or shapes, as we have used color in laboratory sessions, I chose this as my first option.



First frame from video

As we learned in last sessions of laboratory, one good idea when working with colors is separating the images by channels and start working with those channels which gives you better or more useful information.

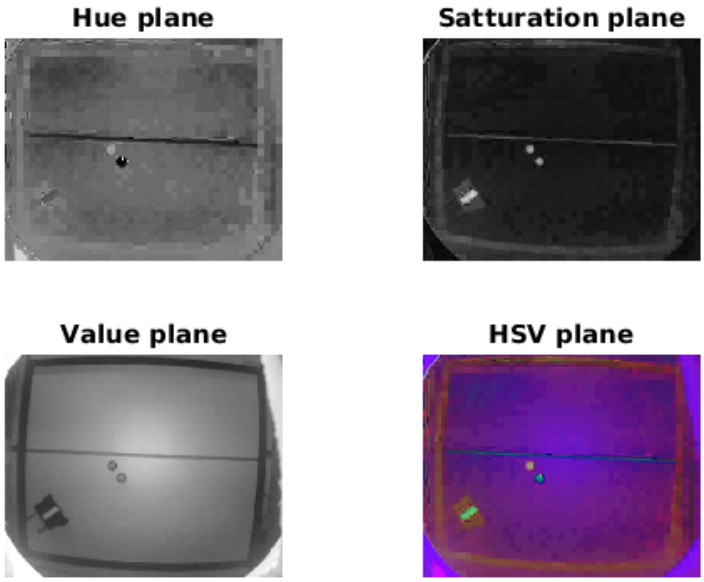
- Try separating image for channels

First channel separation I tried was RGB and the result was this one:

RGB channels from first frame

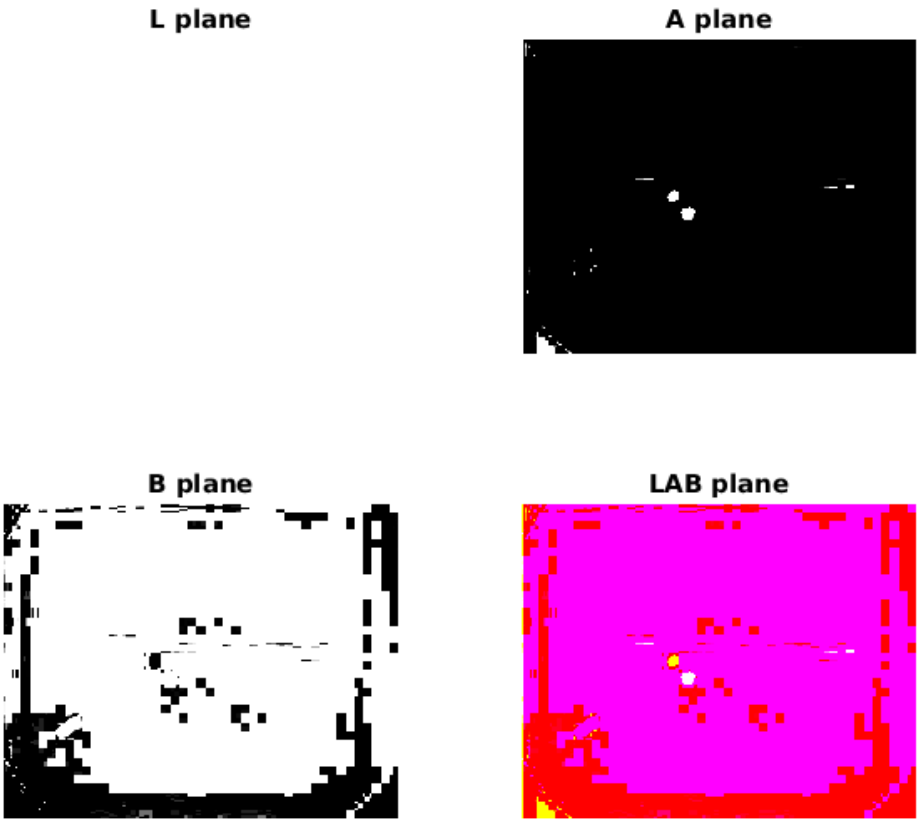
RGB channels seem to show a good separated information (maybe needs a background removing), at least for the most important color in my case, which is the green square over the robot.
It is important to note that the center of mass for the green square is the same as for the robot.

Any way I tried separating image in HSV and LAB channels just to check if the information in those channels were better than RGB.



HSV channels from first frame

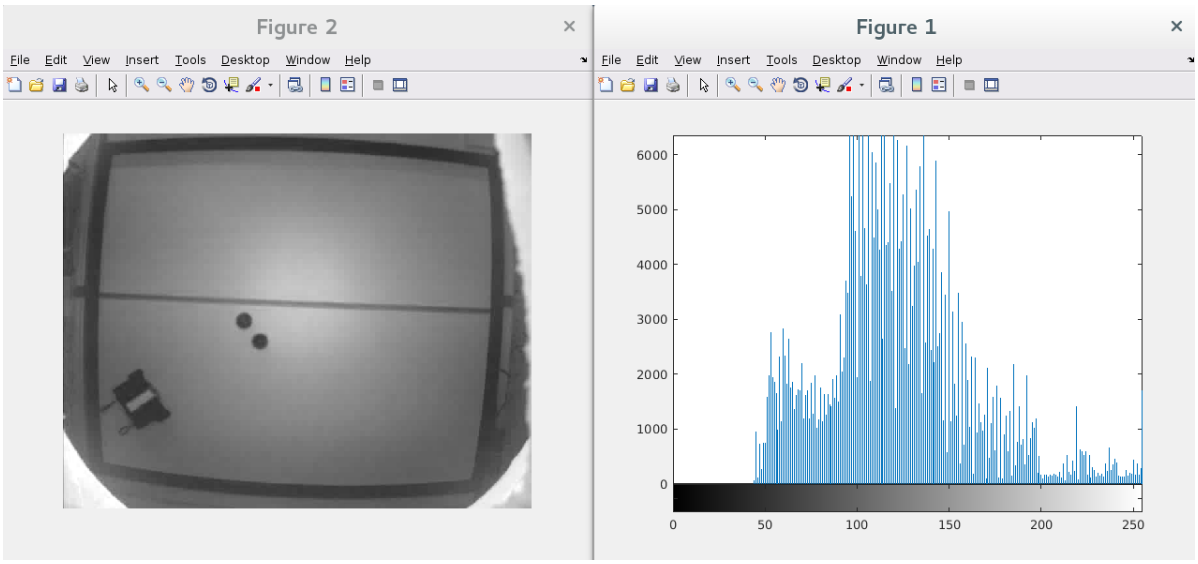
As LAB space is considered a color space, I decided giving it a try too.



LAB channels from first frame

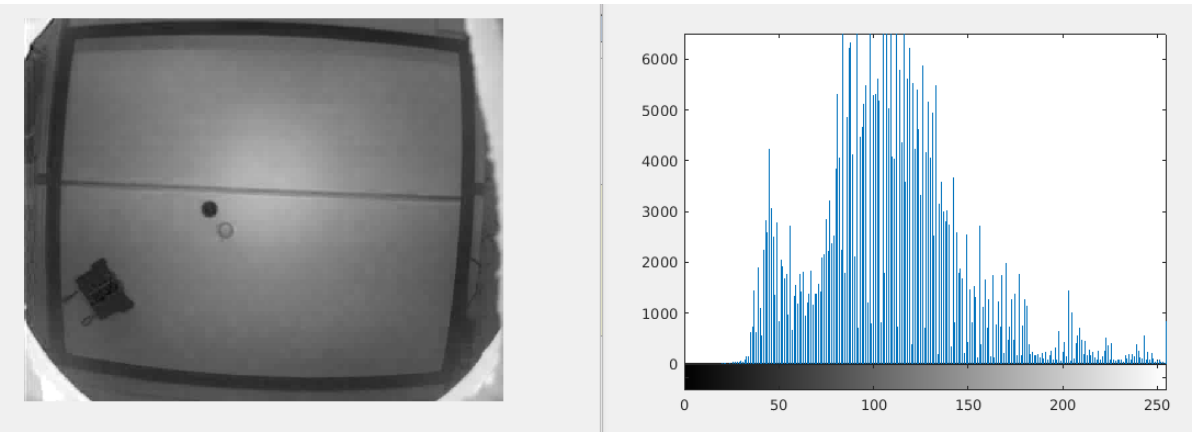
This is maybe the worst result from different spaces, so I don't think I'll use this channels.

- Now check histograms for different channels



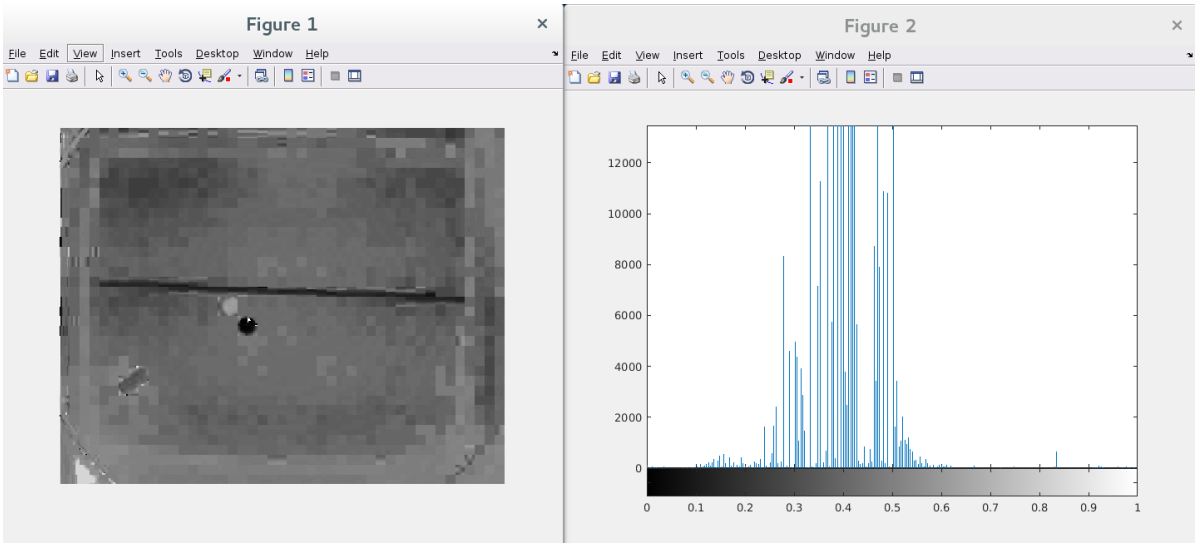
Histogram for RGB green channel

This channel has good information but is difficult to differentiate between white background and green color.



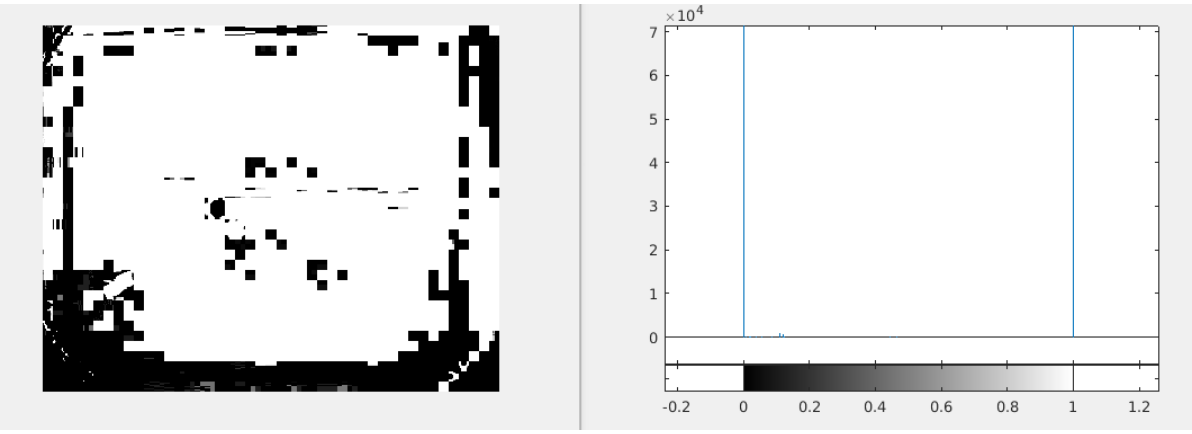
Histogram for RGB red channel

This red channel seems like a good channel for detecting white background.



Histogram for HSV hue channel

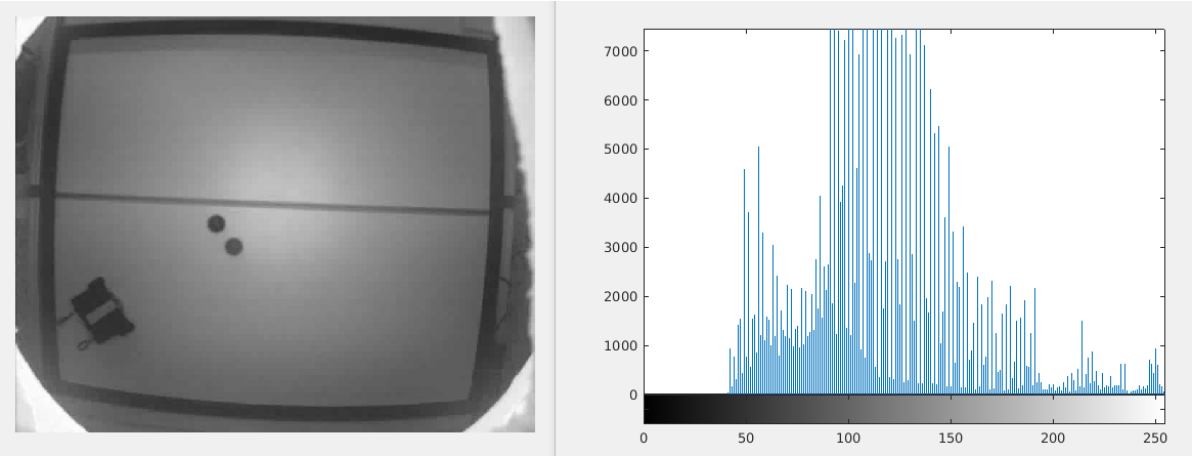
The problem with this channel, is that almost all values are very similar, so is difficult to know which value is what object.



Histogram for LAB b channel

It is almost a binarized image, so it is completely useless for what we mean to do next.

One more important histogram to check is for the gray levels image

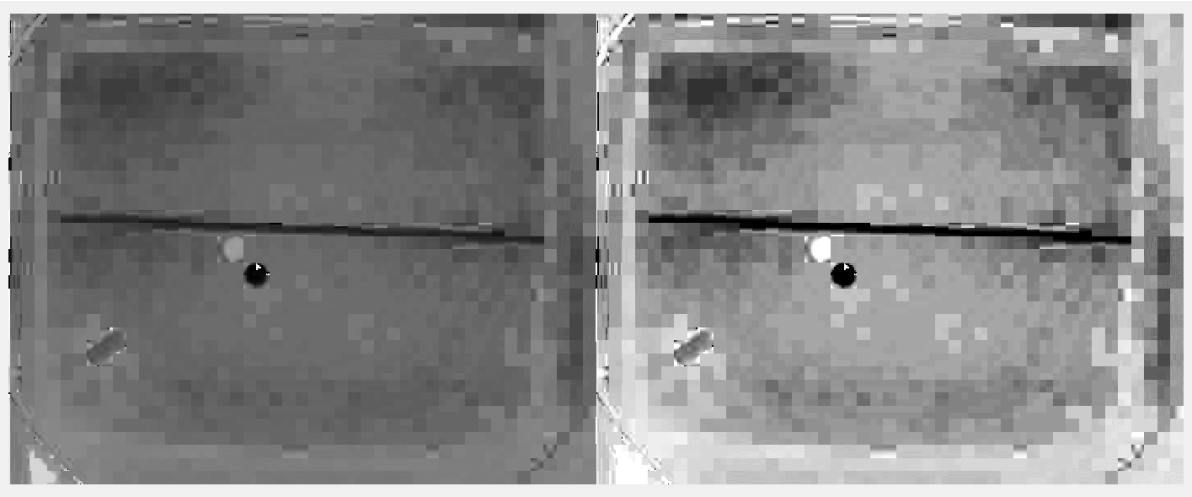


Histogram for gray levels image

This histogram has good information, but as the green channel, it may be difficult to differentiate if a pixel comes from the green square or the white background.

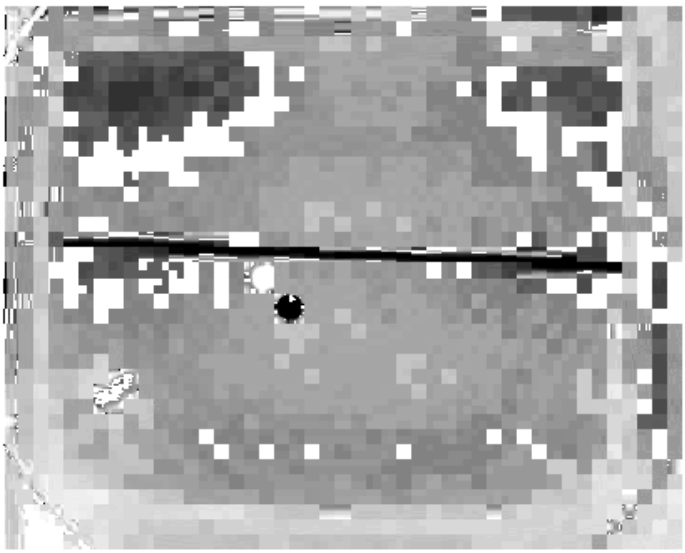
- One more chance for HSV space

As we saw before, the HSV hue channel appeared to have few information, anyway I chose giving it one more change by stretching the histogram.



Stretched out histogram on hue channel

Next I tried a threshold on stretched out hue channel using a two level threshold with some values. It was only to check how easy or not it would be to separate the green color in this channel.

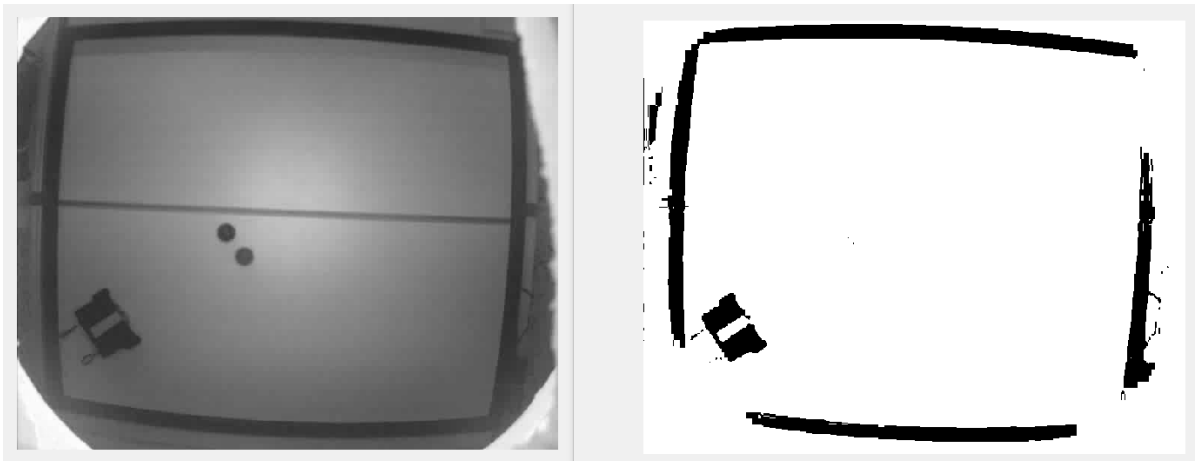


Apply a sample double threshold (t1 = 0.39; t2 = 0.51;) on hue channel

No good results for any of threshold values, it seems like HSV will not be as useful as RGB and that's why from now on I'll keep working only with RGB channels.

- Detecting background

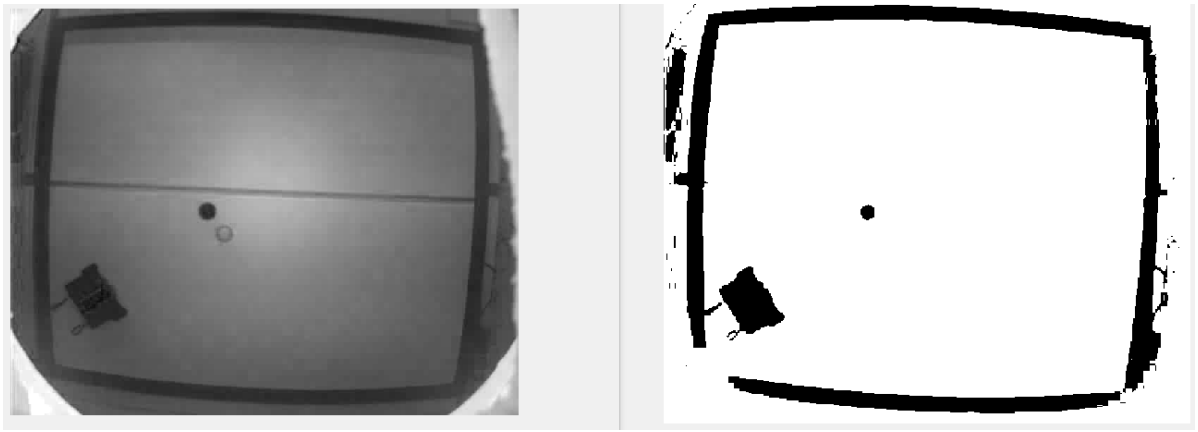
Before removing the background we must first detect it, to accomplish this I first tried using gray levels image and then tried with RGB red channel (it could also may worked on blue channel but not on green because green and white are similar in this channel).



Background detection on gray levels image, using imbinarize(0.25), this 0.25 was found by trying with several values.

This gray levels background detection is good, but confuses white background and green square over the robot.

Now trying with the RGB red channels

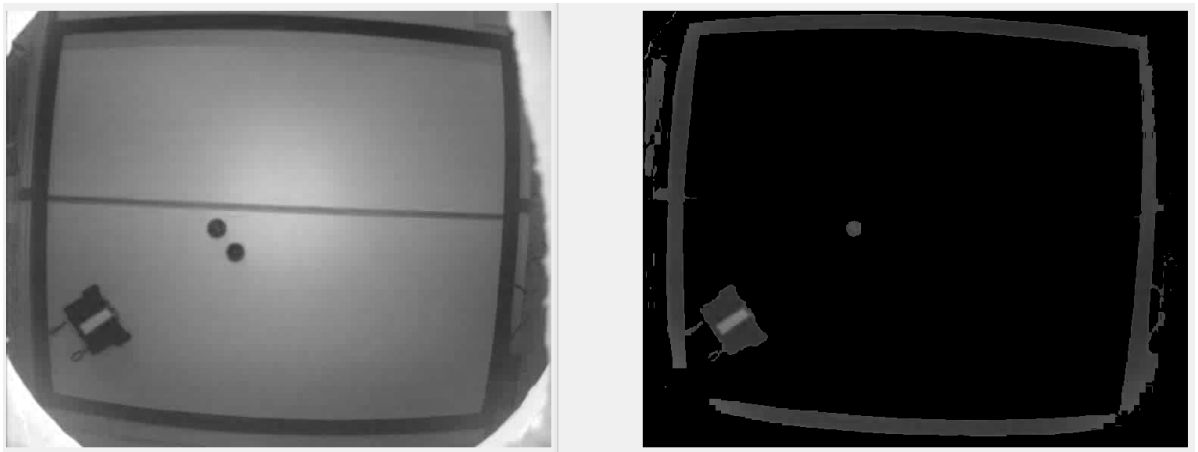


Background detection on red channel image, using imbinarize(0.25), this 0.25 was found by trying with several values.

This is a much better background detection because using this mask we will remove almost every thing except for the robot and the black frame of the image.

- Removing background

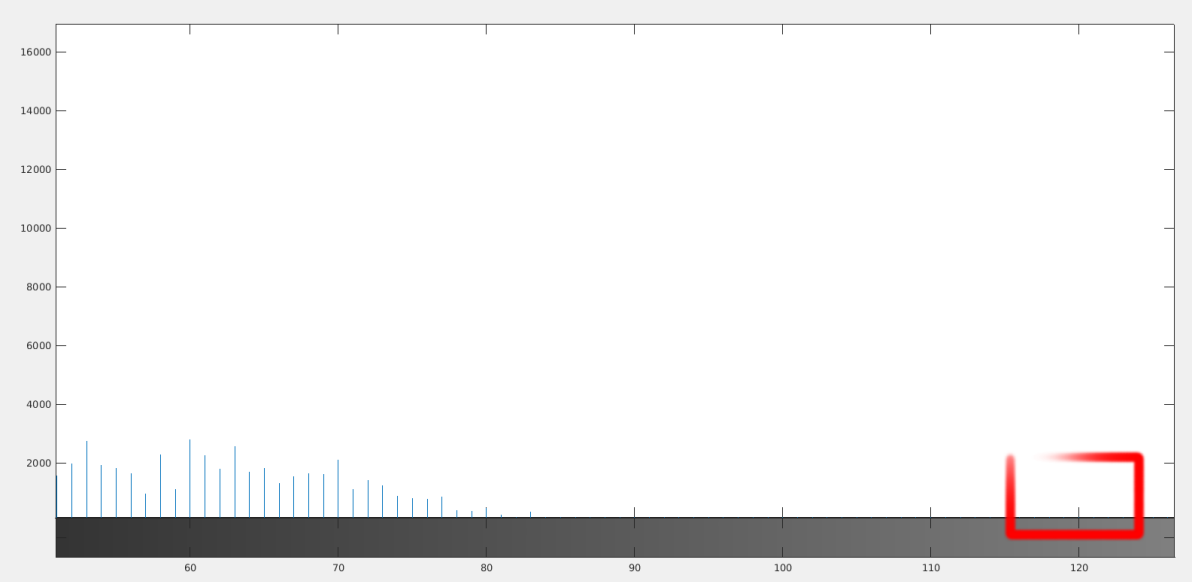
Now that I know how well the red channel worked with background detection, I will use this same resulting image as a mask for background removing in the green channel.



Green channel before and after background removing

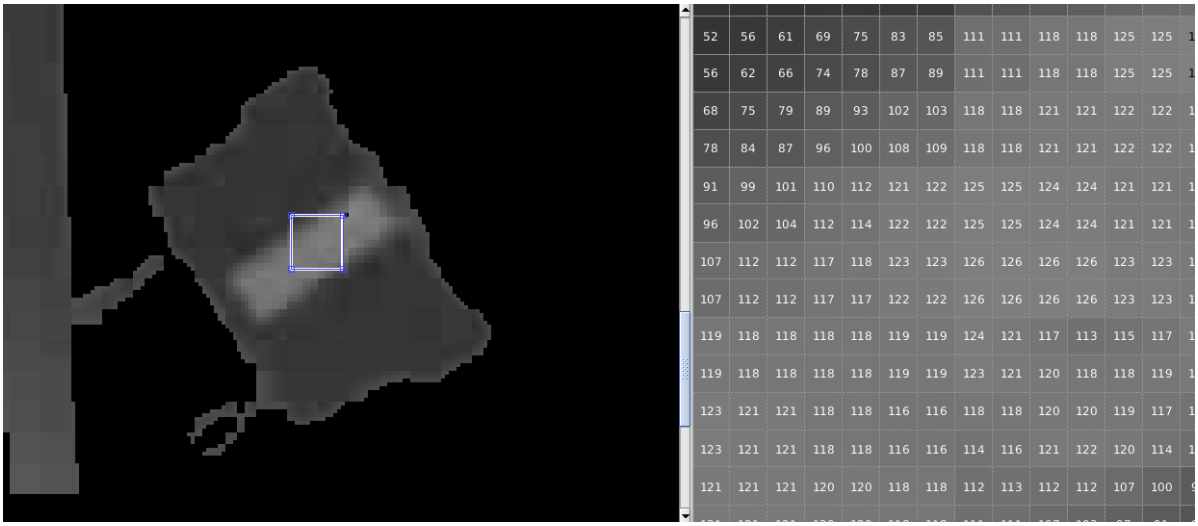
- Binarizing no-background image

After background remove, we can see how well the histogram shows a clear difference between green and other colors



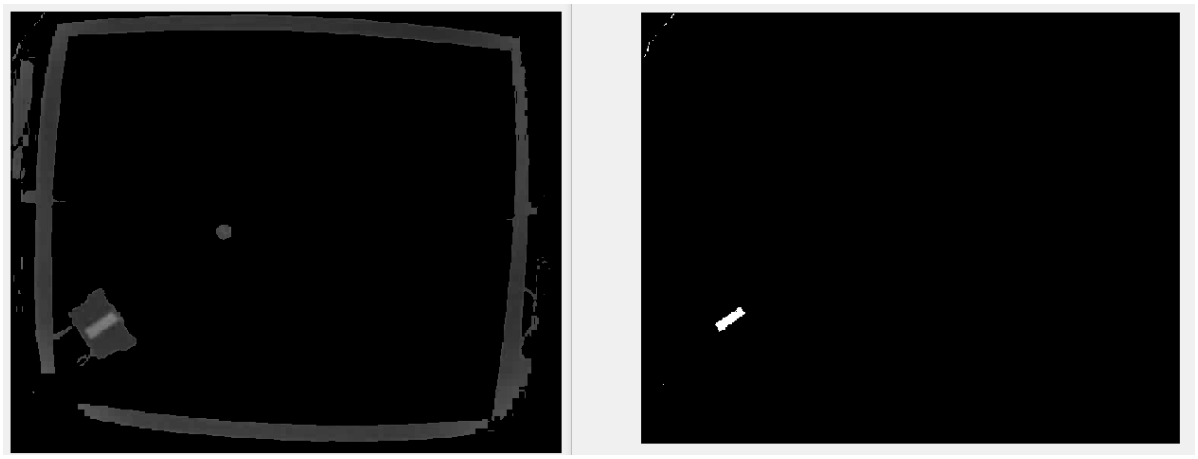
Histogram from no-background green channel. The red zone shows where the green pixels are.

To check the values on green zone we can use the imtool feature in matlab



Matlab imtool feature

As we know almost all pixels different from green are located below value 90, then we can work with a threshold close to this. As Matlab binarize tool works for values between 0 and 1, then $90/250 = 0.36$.



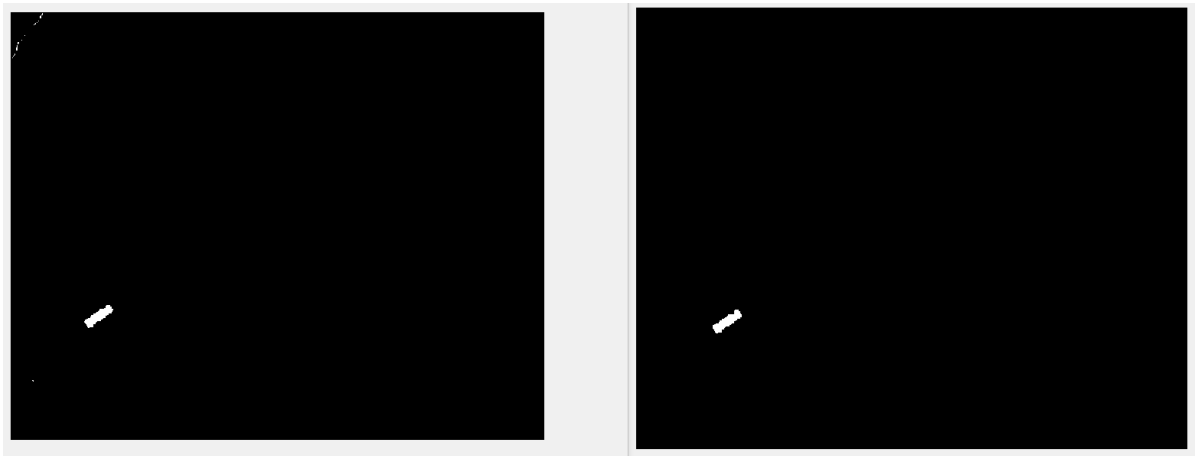
Binarizing from no-background image and using imbinarize(0.38)

Results are very good, but still remains some noise and holes which will be removed using morphological filters in next section.

- Applying morphological filters

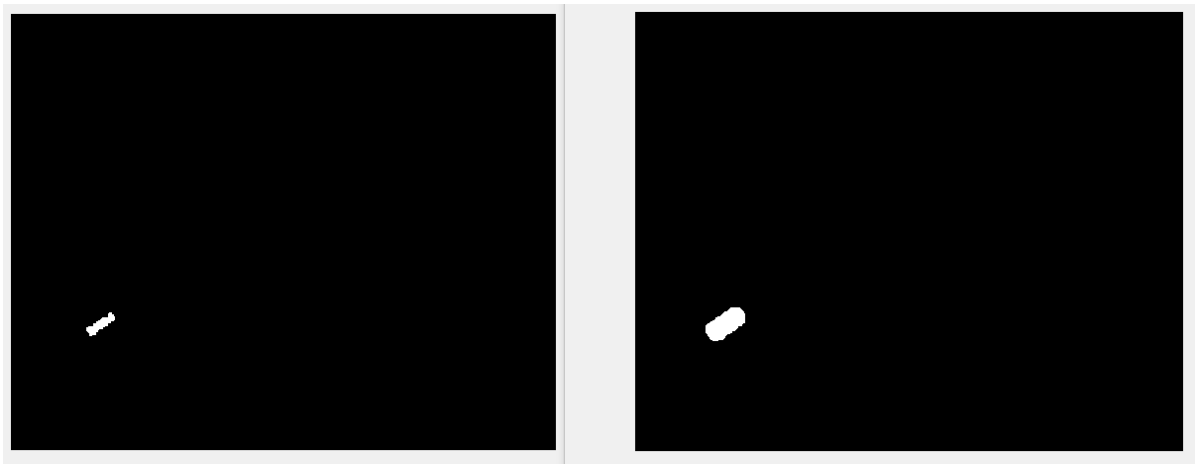
We usually use morphological filters to clean images, it means, removing all information unnecessary and fixing remaining errors. Usually it consists mainly on removing noise, filling holes, increase sizes of objects and other similar things.

First thing I applied was an open filter, so I can remove small noise pieces. Luckily, Matlab includes some useful tools to try different filters, shapes and values easily and that's how I chose mines.



Binarized image before and after open filter using shape='square' and size='3'

After applying the open filter, the noise removing is very well, but the area of interest (green square) seems kind of small, that's why I decided giving it a little more volume by using a dilate filter.

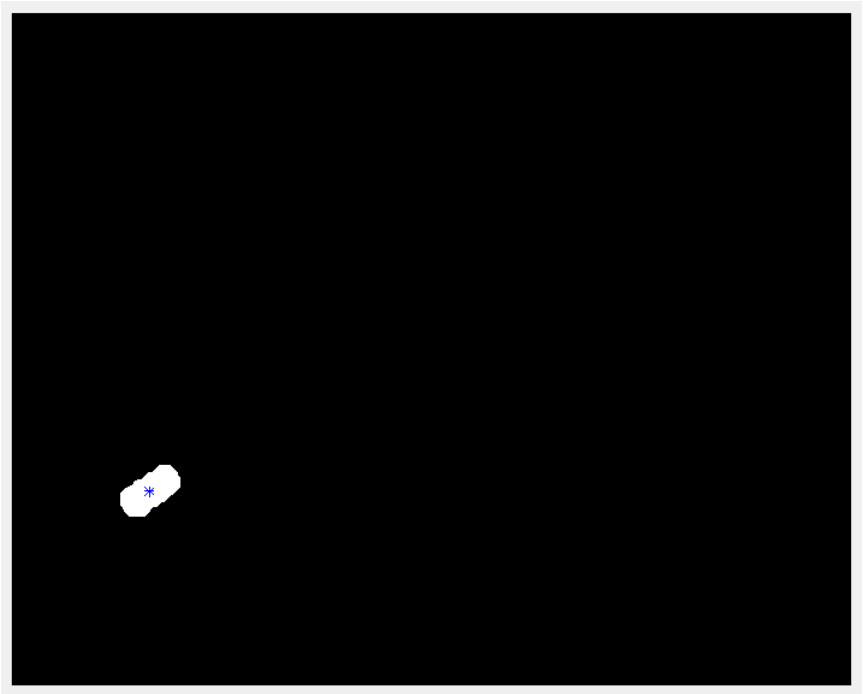


Binarized image before and after dilate filter using shape='disk' and size='8'

- Finding the center of mass

Matlab includes a lot of useful tools which can be used for image processing, one of this useful tools is called regionprops, it works with binarized images and is used to measure properties of image regions. This function will be used for finding centroids or center of mass of "objects" in my binarized image.

According to Matlab's documentation, we can calculate centroids for connected components in the image using regionprops as:
s = regionprops(BW,'centroid');



Centroid coordinates printed over binarized image

Now that I have the coordinates correctly calculated for one frame, I must loop over entire video and repeat the same process.

Loop over video frames

After finishing working with a single image, I tried simply replicate this code for the entire video, but an error occurred when the robot moves from the dark zone to the light zone. This error is related to threshold values during the binarize of the image, because when light focus over the robot, it change the pixel values for the green square and the surrounding areas.

Finally, after trying a couple of values, I found some parameters which worked well for all frames from start to end.

The changed values are:

- Threshold 0.25 when finding the background mask using the red channel.
  Inb_mask = imbinarize(redChan, 0.25);
- Threshold 0.40 when binarizing green-no-background image.
  green_bin = imbinarize(green_nb, 0.40);

Using this values the results were really good for every frame.

Matlab code

As Matlab allows us to work with functions, I decided to separate my code in different functions so it becomes more ordered, readable and easy to understand, this functions are:

PlotRobotPushing.m (main script)

```
%% Read image
% Remove all variables from workspace
clear;
% Clear command line window
clc;
% Read video from file
V = VideoReader('Robot-pushing 2 balls.avi');
% Get frames size
firstFrame = readFrame(V);
frameSize = size(firstFrame(:,:,1));

% Initialize vars
robotX = zeros(0,1);
robotY = zeros(0,1);
blackBack = zeros(frameSize); % Black backgrount to final plot

V.CurrentTime = 0;
while hasFrame(V)
iFrame = readFrame(V);
% Get binarize image specific just including the robot
iBin = BinarizeRobot(iFrame);
% Append new coordinates to arrays
[robotX(end + 1),robotY(end + 1)] = GetObjectCoordinates(iBin);
% Didactic printing
% pause(1/V.FrameRate);
% imshow(iBin) % In case we want to check the binarizing proccess
% Print current working time to follow proccess in Command Window
V.CurrentTime
end

% Print black background and robot coordinates as white dots
imshow(blackBack);
hold on
fig = plot(robotX,robotY, 'w.');
saveas(fig,'ouput-image','png')
hold off
```

BinarizeRobot.m

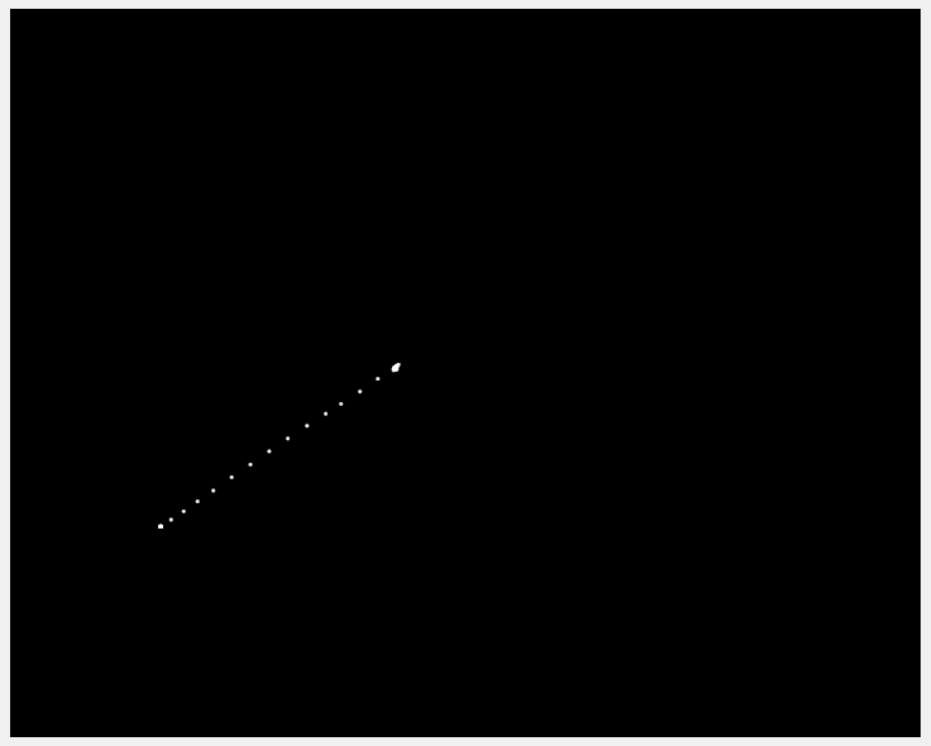
```
function iBin = BinarizeRobot( iOri )
%BINARIZEROBOT This function gets the original image for every fram and
%returns the binarized image with only one object on it
%------------------------------------------------------------------------
% Create gray level image
Igray = rgb2gray(iOri);
redChan = iOri(:,:,1);
greenChan = iOri(:,:,2);
% imshow(Igray);
% Remove white background to image
% Inb_mask = imbinarize(redChan, 0.35);
Inb_mask = imbinarize(redChan, 0.25);
Inb_mask_comp = imcomplement(Inb_mask);
green_nb = greenChan;
% Mask image so only remain main object
% green_nb(Inb_mask) = 0;
green_nb = uint8(Inb_mask_comp) .* green_nb;
% Binarize no-background image (working on green channel)
% green_bin = imbinarize(green_nb, 0.38);
green_bin = imbinarize(green_nb, 0.40);

% Morphological filter - Open
se = strel('square', 3);
green_bin = imopen(green_bin, se);
% Morphological filter - Dilate
se2 = strel('disk', 8);
green_bin = imdilate(green_bin, se2);
% Return result
iBin = green_bin;
end
```

GetObjectCoordinates.m

```
function [ x,y ] = GetObjectCoordinates( iBin )
%GETOBJECTCOORDINATES This function receives a binarized image and returns
%the center of mass coordinates from one object on it
%------------------------------------------------------------------------
% Finding object centroid
s = regionprops(iBin, 'Centroid');
centroids = cat(1, s.Centroid);
x = centroids(:,1);
y = centroids(:,2);
end
```

Final plot

Once the main script loops over the entire array of frames in the video, it creates a vector of coordinates which represent the position of the robot over time, then, I plot this vector as white points over a black background.



Output-image

Comments and Conclusions

After finishing this work, I realize how important it is to understand all of the concepts we have been working with at the laboratory, because even when Matlab is incredible powerful, it would be useless if we don't know when to use any of its features.
I am really happy I started learning Matlab, I am sure it will be very useful during the following part of this semester and the rest of the master.

I must say that this Matlab tools used in this work are not the most powerful or the newest ones, because there exists already some tools to segment by color or shapes, but the objective of this exercise is learning the image processing tools in general, how they work and when we should use them, and not only finding the best tools in Matlab.

Finally, I guess the best approach when working with image processing and computer vision in general, is to start by studying all the options we have and then choose the ones which better fit our needs, just like I did when separating the images for all channels in different spaces (RGB, HSV, LAB, Gray) before start doing any other thing.