EW309: Computer Vision

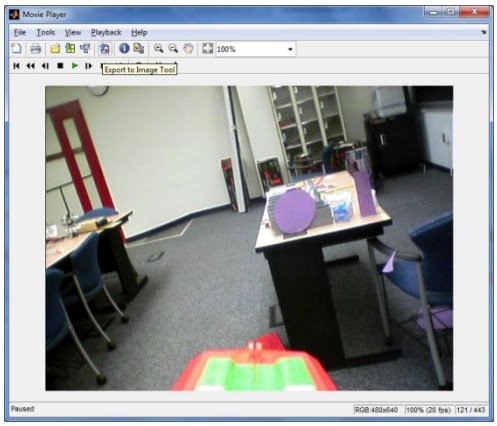
Image Processing

Step 1: Import the Video

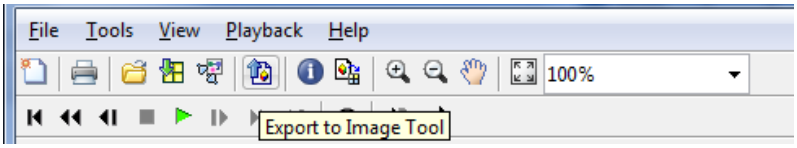
From within MATLAB's file browser pane, find the video file you created and double click it to import it. Select the first radio button (*Create variables matching preview*) and check the toggle next to the variable name (to *Import* it). In the workspace it will create a variable whose name matches the file name. The variable is actually a "structure" with the frames of the video. For example mine was called "PurpleTargets" and it has 443 frames. Tips:

- If you import too many videos you can run out of RAM. Try typing: `clear all` if you get a `Not enough memory available` error.
- You can do this from a command line or inside code via `importdata('PurpleTargets.avi');`

Step 2: View the video and select a sample frame.



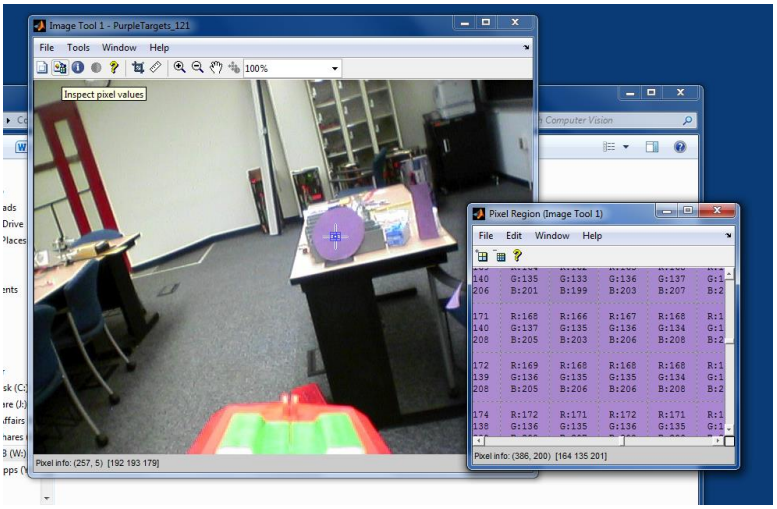
- Type implay(PurpleTargets);
- You may need to adjust the size of the window to see the entire scene
- Pause when you reach a frame with the object in it.
- Choose *Export to Image Tool* from the window toolbar.



Now within image tool we will do a few things: estimate color, estimate size, determine a way to "mask" or exclude the gun barrel from the image.

Step 3: Estimate the average color of the object.

- Click the Inspect Pixel values button. Make the inspection window small enough that MATLAB displays the numerical values for each pixel.

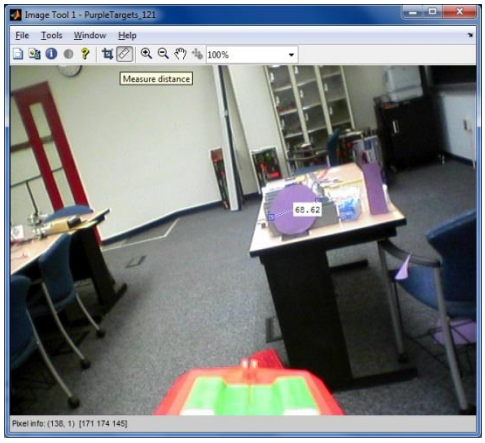


- Browse different regions of the object and get a feel for the approximate intensity values for the three channels and their range of variation. Record your estimate of the average.
  R:_____, G:_____, B:_____

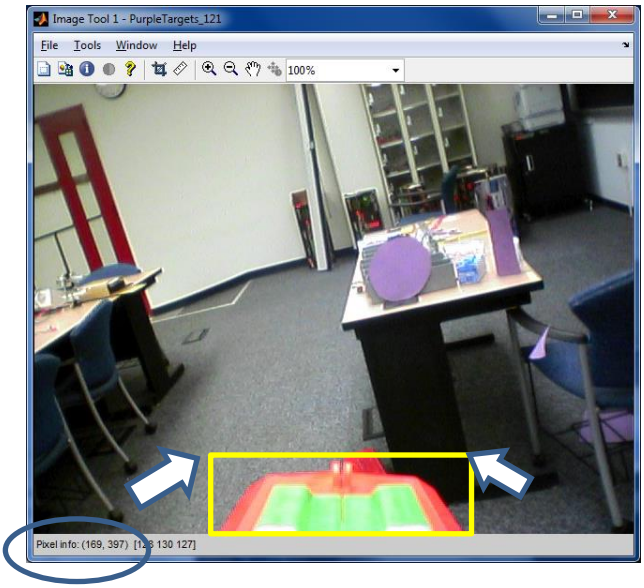
- Inspect the different regions of the background (ex: white walls, blue carpet, etc.) and record their approximate values.
  Background 1 R:_____, G:_____, B:_____
  Background 2 R:_____, G:_____, B:_____, ….

Step 4: Estimate the size of the object in pixels.

Use the ruler icon to roughly measure the diameter of your object in pixels. Compute the area (in pixels). This will be a very rough estimate since the area changes with viewing distance.



Step 5: Masking the Barrel. We don't want to confuse the barrel with the orange targets. The gun shouldn't move relative to the camera, so we can just figure out a certain region of the image to ignore. Hover your mouse in the upper left and upper right corners of the rectangular region you want to ignore. Note the coordinates (first number is column, second number is row). Note that I added the annotations – you won't see a yellow box.



Step 5: Export your sample frame as an image.

- Within the imtool window click File/Export To Workspace…Give this particular frame (image) a name -- I chose TestImage. Do not use "image" because it is the name of a built in command.

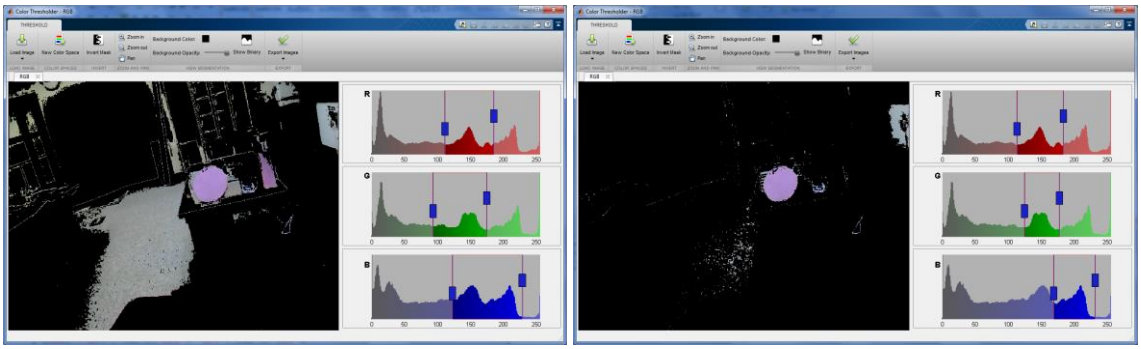
Step 6: Color thresholds

Now that we have the approximate size and color we need to come up with more robust thresholds.

- Type `colorThresholder(TestImage)` or whatever you named your sample image.
- Click the left column titled RGB. You are now shown 3 histograms for the R, G and B channels.
- Recall your approximate values of RGB. But the natural object is made up of a range of colors. Therefore we should define its color as a range of RGB values. Experiment with moving the sliders.

Notes:

- The statistically correct place to put the thresholds is at a valley between two modes (peaks in the histogram).
- It is impossible to get it "perfect". Don't over-tune the thresholds! Each frame will be slightly different anyway.
- Every pixel of your color should still be present in the image (including ones that are not part of your object -- the rug and the target look similar in my sample). All we are doing here is picking out purple *pixels*. Later we will use size and shape to help reject other pixel groups.



Note the high and low values of on each channel. You can just eye ball them from the x axis of the histograms, since they are approximate anyway. You can also choose *Export Images* and then *Export Function*… this will generate a MATLAB function that automatically includes the thresholds you set (we won't actually run that code though).

Rlo = ____, Rhi = ____, Glo = ____, Ghi = ____, Blo=___, Bhi=____

Step 7: Start our function and script file

We will make new programs. A “main” script file that loops through frames of the video; and a function that processes each frame.

Make a new file in the MATLAB Editor. Call it ProcessTarget.m This file will be a function so the first line needs the key word *function,* the outputs (the location of the target), the name of the function (which matches the file name) and a list of inputs (a single image).

```
function [TargetLocation] = ProcessTarget(im)
TargetLocation = []; % initialized to empty

% high and low thresholds for each color channel
Rlo = 100;
Rhi = 255;
Glo = 50;
Ghi = 255;
Blo = 0;
Bhi = 255;

% Pixels must meet all (AND) of these criteria
imBin = im(:,:,1) >= Rlo & im(:,:,1) <= Rhi & ...
 im(:,:,2) >= Glo & im(:,:,2) <= Ghi & ...
 im(:,:,3) >= Blo & im(:,:,3) <= Bhi;

% Black out the gun barrel - note that the order is horizontal, vertical
imBin(400:480, 160:490) = 0;

% display results
subplot(1,3,1) % 1 row and 3 columns of images, 1st image
imshow(im)
title('Original Image')
subplot(1,3,2) % 2nd image…
imshow(imBin)
title('Purple Pixels')
shg % Pop window to foreground
```

In a separate file called “TestImageProcessing.m”:

```
% Main script file to test computer vision system
importdata('PurpleTargets.avi'); %import the video. Looks in current folder
NumFrames = length(PurpleTargets);

for i = 1:NumFrames
      im = PurpleTargets(i).cdata; %Gets the color (RGB) data
      [TargetLocation] = ProcessTarget(im);
end
```

Step 8: Size, Shape or Centroid

Now we are ready to add to `ProcessTarget` to reject other regions based on their size or shape, and find the centroid of the final target.

First let's eliminate any small speckles that may be the correct color but are far below the minimum target size. This will greatly speed up the next step. Add this line to Process target, right after you apply your color thresholds to compute imBin.

```
imBinMinArea = bwareaopen(imBin, MinArea);
```

Now we will use a function called `regionprops` to get more properties. Type `doc regionprops`. Ignore the syntax for the moment and scroll down to the definition of the shape properties. You will need area, centroid and eccentricity. Read their definitions. Explore other properties you may want to add later.

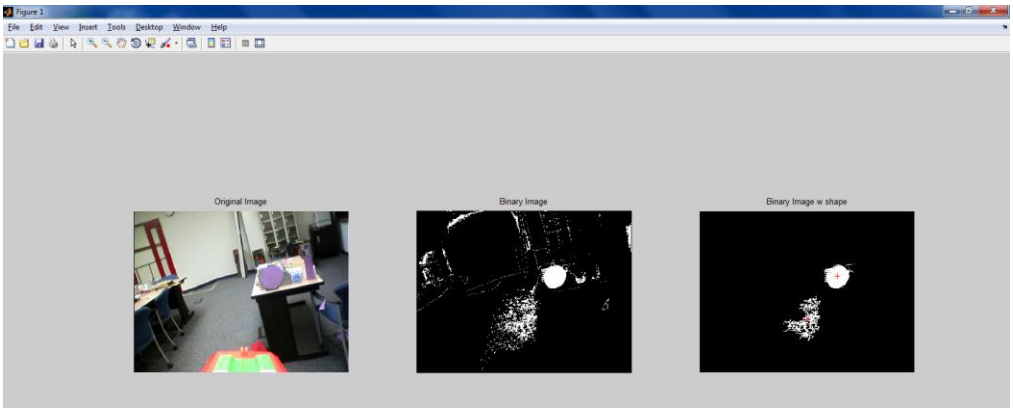
- We will use some code from the "Tips" section of the documentation to select regions that match a certain criteria. Add something like this to the `ProcessTarget` function.

```
% Black out the gun barrel - note that the order is horizontal, vertical
imBin(400:480, 160:490) = 0;
imBinMinArea = bwareaopen(imBin, MinArea);
cc = bwconncomp(imBinMinArea);
stats = regionprops(cc, 'Area', 'Centroid', 'Eccentricity');
idx = find([stats.Area]<MaxArea & [stats.Eccentricity]<MaxEccentricity );
imBinShapeSize = ismember(labelmatrix(cc), idx);
Areas = cat(1,stats(idx).Area); %put the properties of the good regions in an array
CX_CY = cat(1,stats(idx).Centroid);
% display results
…
```

- You will have to set values for MaxArea and MaxEccentricity based on your tests. It's best to set really loose thresholds. Remember, if an object fails even one of these tests it will be rejected.
- You can add as many conditions as you like to that line to check other properties. Eccentricity is useful for finding circular objects. Make sure you understand its range. Solidity is also useful.

Now let's add some display information to this code.

```
% display results
subplot(1,3,1) % 1 row and 3 columns of images, 1st image
imshow(im)
title('Original Image')
subplot(1,3,2)
imshow(imBin)
title('Purple Pixels')
subplot(1,3,3)
imshow(imBinShapeSize) % purple pixels who ALSO meet the size criteria
title('Purple Pixels with right shape or size')
if ~isempty(CX_CY) % plot centroids if there are any
 hold on
 plot(CX_CY(:,1), CX_CY(:,2), 'r+')
 hold off
end
shg % Pop window to foreground
```



Step 9: Target Selection

At this point you have a list of regions that are within the range of acceptable color AND shape AND size criteria. Your function needs to return a single 1X2 array with the X and Y pixel coordinates of the target.

- It is possible this list may be empty in certain frames. Consider what you will return for the `TargetLocation` output in this case.
- It is also possible that multiple regions will fit all these criteria. How will you select a single centroid (a row of CX_CY) that you think is the best estimate of the `TargetLocation`?