EW309 Computer Vision

Validating your code

Goal: Refine your target identification algorithm until it correctly classifies at least 96% of the test frames. Note that both false positives (areas marked as the target that are not really it) and false negatives (target present but not identified) count as incorrect classifications.

Tip: Keep in mind that if your target is partially "cropped" out of the out of the frame it will no longer image as a circle. We won't count that as an error.

Why? With a 2 second settling time and frame rate of 15 frames per second, 96% accuracy permits no more than one mis-identified target frame prior to settle (i.e. 1-1/(2*15) X 100%).

Validating With Test Videos

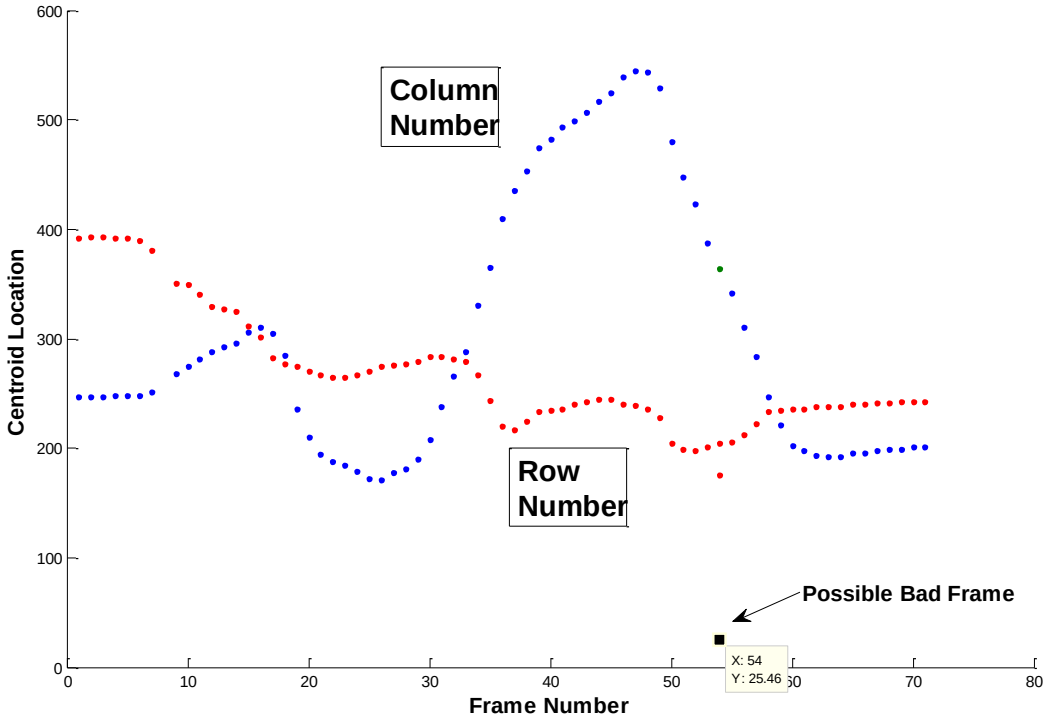
Ideally, we would score the result of each frame of the test videos. We would closely inspect the bad frames to determine how to adjust our parameters--is the problem with the color or the shape of the size parameters?

Because there are so many frames even in a short video, you have to come up with a way to do this efficiently. Looking at our main program, there is a collection of tips and ideas below.

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

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

- You can add information in the title of the plot. For example, the frame number can be made using title(num2str(i)) .
- When the "attention" is on the command window you can use CNTRL-C to abort the program if you don't want to have to sit through the entire video. Docking the figure window can help especially if you are printing things to the screen too.
- Say you ran the code and you noticed the algorithm has trouble with a particular set of frames in the middle of the video file. You can modify the limits of the for loop (e.g. for i = 200:250) to play that only that frame set without having to re-watch the whole video. (you can also skip frames if you want a "fast forward" effect e.g. for i = 1:5:200)
- You can add a pause inside the loop to slow it down to give you time to inspect the images. `pause()` pauses until you hit any key, while `pause(dt)` would pause for dt seconds.

- You can also put the pause inside an `if` statement so it only stops the sequence if there is a "problem" (no targets present or multiple targets present or the centroid jumps by more than X pixels. The size function may be useful to check this.).
- Instead of watching the frames, another approach would be to make a plot of the target location, or the number of targets across the video sequence. You would look for abrupt jumps in these values and go back to inspect that frame.



Going Live

With just w few changes we can experiment with a live video stream.

Copy the function **`initCamera`** to your EW309 Deliverables folder.

Now we need to modify your main script file (see code below) to:

- initialize the camera (instead of importing the video file)
- set the number of frames to take (15X number of seconds or infinite)
- get a snapshot from the camera.

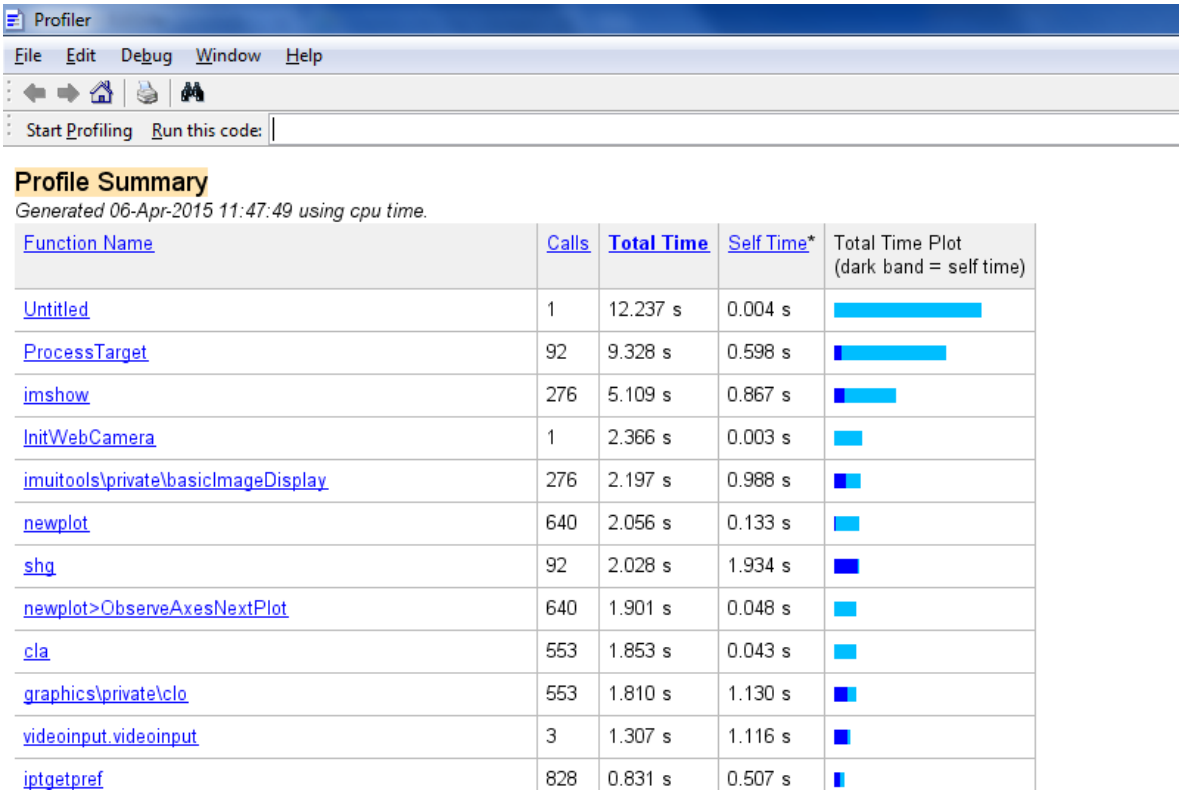
```
% Main script file to test computer vision system
%importdata('PurpleTargets.avi'); %import the video.
[cam,prv] = initCamera;
%NumFrames = length(PurpleTargets);
NumFrames = inf; % infinite or until you hit CNTRL-C

for i = 1:NumFrames
 % im = PurpleTargets(i); %Gets the color (RGB) data
 im = get(prv,'CData')
 [TargetLocation] = ProcessTarget(im);
end
```

Verify your code still finds the target.

Assessing Frame Rate

To see how fast the SCRIPT_TestStreamingCamera.m code updates, instead of clicking the play button titled “Run” on the toolbar of the editor window, click the button “Run and Time”.

This opens a hyperlinked report:



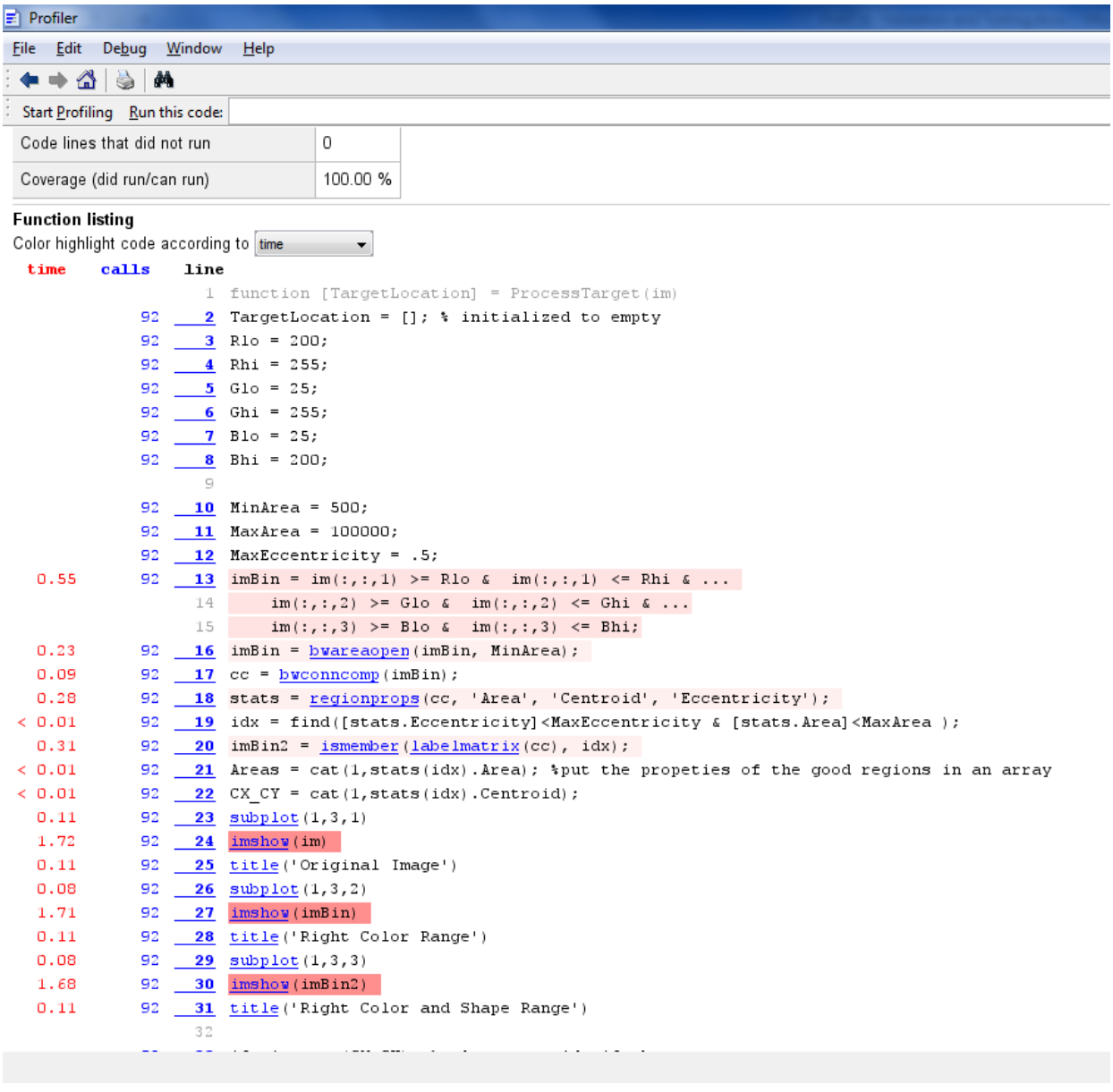
Profile Summary

Generated 06-Apr-2015 11:47:49 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Untitled	1	12.237 s	0.004 s	
ProcessTarget	92	9.328 s	0.598 s	
imshow	276	5.109 s	0.867 s	
InitWebCamera	1	2.366 s	0.003 s	
imuitools\private\basicImageDisplay	276	2.197 s	0.988 s	
newplot	640	2.056 s	0.133 s	
shg	92	2.028 s	1.934 s	
newplot>ObserveAxesNextPlot	640	1.901 s	0.048 s	
cla	553	1.853 s	0.043 s	
graphics\private\clo	553	1.810 s	1.130 s	
videoinput.videoinput	3	1.307 s	1.116 s	
iptgetpref	828	0.831 s	0.507 s	

We ignore the code that runs once (like initializing the camera) and focus on the Process Target code which runs at every frame. You can see that this code ran 92 times (92 frames) and took a total of 9.328 sec. You should use this to compute your frame rate. Keep in mind that the camera we are using only acquires images at 15 Hz, so that is the upper limit on frame rate.

To get more insight into what steps of the processing take the most time, click on ProcessTarget (hyperlinked in the above report). And scroll down. Here you see next to each line its execution time.



Consider making changes to improve your frame rate.