Logo

Description automatically generated**MOD002643 Image Processing**

Coursework Brief, Trimester 1, 2022-23

**Tracking Objects in Motion**

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Marks Available** |
| **A**. Complete MATAB Onramp and Image Processing Onramp | This is an all-or-nothing mark awarded only if both training certificates are fully completed. You are required to supply *hyperlinks* to your certificates, not PDFs (see next page). Each certificate takes 2 hours to complete. | 10 |
| **B**. All quizzes completed with all marks 12 or higher. | This is an all-or-nothing mark awarded only if all 11 tutorial quizzes are completed with a mark of 12 or higher. Your lecturer will verify full completion using your Canvas G*rades* record. | 10 |
| **C**. Threshold static image to isolate Tag 1 and Tag 2. | Take a static photograph of you holding up your two tags standing against a plan background (e.g., blank white wall). Include your face in the image. Use multi-band thresholding (only) in RGB space to segment the two tags as neatly as possible, demonstrating your results with two binary images. | 10 |
| **D**. Use morphological and non-linear methods to clean Tag binary images | Attempt to convert false negatives to hits and false positives to correct rejections by experimenting with morphological operators/structuring elements and non-linear filters (e.g., median, majority). In particular, try to *fill in any* interior holes, marks or writing on your tags. | 10 |
| **E**. Annotate Tag boundaries on original image. | Use the non-linear range filter, or a linear edge detection filter, to trace the boundary around the two tags. Use this to annotate the original image with a border around each tag. Plot the borders in two different colours. | 10 |
| **F**. Calculate Tag centroids and distances | Determine the centroids of the two tags using the binary images produced up to this point. Display the centroid coordinates and the distances between the two tags in city block, chessboard and Euclidean distance. Annotate the image with centre markers and distance lines (e.g., using plot). | 10 |
| **G**. Adapt for real-time webcam use. | Install and use the webcam Support Package to collect snapshots images from your webcam in a loop in real-time. Refine and apply your best workflow, developed above, to enable the tag centroids and distances to be reported in real-time. Save a 1-minute video of you moving the tags. | 20 |
| **H**. Interpretation and light drawing. | Add the functionality to *draw* onto the image using the two tags. The drawing can be superimposed on the live video feed, and also saved as a separate image at the end of the session. Save a 1-minute video of you drawing an object like a house. For add value, interpret relative position for interface. | 20 |
| **TOTAL** | | **100** |

**NOTES**: (**1**) The pass mark for the coursework is 40%, so as a minimum complete A..F. Do not just complete A..D as it is unlikely that you will score full-marks for programming tasks as your answer is unlikely to be objectively perfect. (**2**) You must upload your work in this this template as a Word document or PDF. See Appendix A for more detailed presentation guidelines.

**A. Certificates (0 or 10 Marks)**

MATLAB Onramp Certificate Hyperlink:

https://matlabacademy.mathworks.com/progress/share/certificate.html?id=e4756a29-68c0-4ae5-b963-8476d8070398&https://matlabacademy.mathworks.com/progress/share/certificate.html?id=e4756a29-68c0-4ae5-b963-8476d8070398&

Image Processing Onramp Certificate Hyperlink:

https://matlabacademy.mathworks.com/progress/share/certificate.html?id=d3993731-38d2-4cda-8d4a-7ed5a8471501&

1. **Quizzes 1.. 11 (0 or 10 Marks)**

Tick boxes to confirm completion with a mark of 12 or more.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |

1. **Threshold static image to isolate Tag 1 and Tag 2 (0..10 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

%% C. Static image

% Thresholding RGB image to segment the tokens

% Result 2 binary images

clear all;

close all;

clc;

RGB = imread("TagImage.jpg");

% imtool(RGB); - was used to inspect pixel values

yellowTag = RGB(:,:,1)>195 & RGB(:,:,2)>170 & RGB(:,:,3)<115; % thresholding for the yellow tag

redTag = RGB(:,:,1)>150 & RGB(:,:,2)<65 & RGB(:,:,3)<65; % thresholding for the red tag

figure;

montage({RGB, yellowTag, redTag, yellowTag | redTag}, "Size", [2 2]);

% Displaying the initial picture, each threshold image and a combinations of both threshold tags

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input: RGB Tag-holding Image** | **Output 1: Tag 1 Binary Image** |
|  |  |
| **Output 2: Tag 2 Binary Image** | **Output 3: Combined (Union) Tag 1 and 2 Binary Image** |

Any Comments on Effectiveness of Process/Threshold Values Adopted (50 words max):

Thresholding this image was tricky, I had to change my clothes to not get picked up by the thresholding, and make sure the way I hols the tags won’t make my fingers look too red or yellow due to lighting. I have used the ‘imtool’ and Colour Thresholder App to check the pixel colour range.

1. **Use morphological and non-linear methods to clean Tag binary images (0..10 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

%% D. Morphological Operators & Non-Linear Filters

% Filing holes and text using Dilation, Closing and Erosion

clear all;

close all;

clc;

RGB = imread("TagImage.jpg");

yellowTag = RGB(:,:,1)>195 & RGB(:,:,2)>170 & RGB(:,:,3)<115; % thresholding for the yellow tag

redTag = RGB(:,:,1)>150 & RGB(:,:,2)<65 & RGB(:,:,3)<65; % thresholding for the red tag

se = strel('diamond', 11);

yellowDilated = imdilate(yellowTag, se); % making object slightly bigger and filling partially

yellowClose = imclose(yellowDilated, se); % closing all the remaining gaps

yellow = imerode(yellowClose, se); % minimizing back to initial size

redDilated = imdilate(redTag, se); % making object slightly bigger and filling partially

redClose = imclose(redDilated, se); % closing all the remaining gaps

red = imerode(redClose, se); % minimizing back to initial size

figure;

montage({yellow, red}, "Size", [1 2]);

% Displaying each tag after manipulation and filing

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1: Tag 1 Binary Image from Task C** | **Input 2: Tag 2 Binary Image from Task C** |
|  |  |
| **Output 1: Updated Tag 1 Binary Image** | **Output 2: Updated Tag 2 Binary Image** |

Any Comments on Effectiveness of Process/Methods Adopted (50 words max):

I have decided to use a sequence of dilation, closing and erosion to achieve the result. I believe this is a rough-edged result and I would have liked to reach a smoother result, but I have not yet found the perfect method for this.

1. **Annotate Tag boundaries on original image (0..10 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

%% E. Token boundaries on original image

% - Tracing the tags boundaries and overlaying them on the original image

clear all;

close all;

clc;

RGB = imread("TagImage.jpg");

yellowTag = RGB(:,:,1)>195 & RGB(:,:,2)>170 & RGB(:,:,3)<115; % thresholding for the yellow tag

redTag = RGB(:,:,1)>150 & RGB(:,:,2)<65 & RGB(:,:,3)<65; % thresholding for the red tag

se = strel('diamond', 11);

yellowDilated = imdilate(yellowTag, se); % making object slightly bigger and filling partially

yellowClose = imclose(yellowDilated, se); % closing all the remaining gaps

yellow = imerode(yellowClose, se); % minimizing back to initial size

redDilated = imdilate(redTag, se); % making object slightly bigger and filling partially

redClose = imclose(redDilated, se); % closing all the remaining gaps

red = imerode(redClose, se); % minimizing back to initial size

yellowRangeFilt = rangefilt(yellow); % using range filter to identify the boundary of the tag

redRangeFilt = rangefilt(red); % using range filter to identify the boundary of the tag

RGBandY = imoverlay(RGB, yellowRangeFilt, "blue"); % overlaying the range filter in colour blue over the image

RGBboundaries = imoverlay(RGBandY, redRangeFilt, "green"); % overlaying the range filter in colour green over the already overlayed image

figure;

imshow(RGBboundaries);

% Displaying the image with the coloured boundaries

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1: RGB Tag-holding Image** | **Input 2: Tag 1 Binary Image from Task C or D** |
|  |  |
| **Input 3: Tag 2 Binary Image from Task C or D** | **Output 1: RGB Tag-holding Image with Tag Boundaries** |

Any Comments on Effectiveness of Process/Methods Adopted (50 words max):

The range filter is effective in finding the outer line of the object, but I had issues making it more visible as it is only a thin line. I have found that ‘entropyfilt’ would give me a thicker line but decided this is not the correct approach for this project.

1. **Tag Centroids and Distances (0..10 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

%% F. Tag centroids and distances

clear all;

close all;

clc;

RGB = imread("TagImage.jpg");

yellowTag = RGB(:,:,1)>195 & RGB(:,:,2)>170 & RGB(:,:,3)<115; % thresholding for the yellow tag

redTag = RGB(:,:,1)>150 & RGB(:,:,2)<65 & RGB(:,:,3)<65; % thresholding for the red tag

se = strel('diamond', 11);

yellowDilated = imdilate(yellowTag, se); % making object slightly bigger and filling partially

yellowClose = imclose(yellowDilated, se); % closing all the remaining gaps

yellow = imerode(yellowClose, se); % minimizing back to initial size

redDilated = imdilate(redTag, se); % making object slightly bigger and filling partially

redClose = imclose(redDilated, se); % closing all the remaining gaps

red = imerode(redClose, se); % minimizing back to initial size

yellowRangeFilt = rangefilt(yellow); % using range filter to identify the boundary of the tag

redRangeFilt = rangefilt(red); % using range filter to identify the boundary of the tag

RGBandY = imoverlay(RGB, yellowRangeFilt, "blue"); % overlaying the range filter in colour blue over the image

RGBboundaries = imoverlay(RGBandY, redRangeFilt, "green"); % overlaying the range filter in colour green over the already overlayed image

[iY, jY] = find(yellow); % searching for the (i,j) coordinates of the yellow tag

yY = mean2(iY); % mean of the length to find centroid

xY = mean2(jY); %mean if the hight to find centroid

[iR,jR] = find(red); % searching for the (i,j) coordinates of the red tag

yR = mean2(iR); % mean of the length to find centroid

xR = mean2(jR); %mean if the hight to find centroid

dCityBlock = abs(yY-yR)+abs(xY-xR); % calculating the City Block distance

dChessboard = max(abs(yY-yR),abs(xY-xR)); % calculating the Chessboard distance

dEuclidean = sqrt((yY-yR)^2+(xY-xR).^2); % calculating the Euclidean distance

chessx = (((xY+xR)/2) + xR)/2;

chessy = (((yY+yR)/2) + yY)/2;

dim = [0.1 0.8 0.8 0]; % dimensions of annotation box

imshow(RGBboundaries) % displaying image

% plotting visuals for the above calculations

hold on

plot([xY, xR],[yY, yY], '-mo', [xR, xR], [yR, yY], '-mo', 'LineWidth', 1); % city block line

plot([xY, chessx], [yY, chessy], '-go', [chessx, xR], [chessy, yR], '-go', 'Linewidth', 1); % chessboard line

plot([xY, xR], [yY, yR],'-c\*','Linewidth',1); % eucledian line

str = {['Red Tag Centroid x coordinates = ',num2str(xR)],['Red Tag Centroid y coordinates = ',num2str(yR)], ...

['Yellow Tag Centroid x coordinates = ',num2str(xY)],['Yellow Tag Centroid y coordinates = ',num2str(yY)], ...

['City Block Distance = ',num2str(dCityBlock)], ...

['Chessboard Distance = ',num2str(dChessboard)], ...

['Euclidean Distance = ',num2str(dEuclidean)]}; % string that will be displayed in the annotation box

annotation('textbox',dim,'String',str,'FitBoxToText','on','EdgeColor','black','Color','black'); % creating the annotation box

hold off

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1: RGB Tag-holding Image** | **Input 2: Tag 1 Binary Image from Task C or D** |
|  |  |
| **Input 3: Tag 2 Binary Image from Task C or D** | **Output 1: RGB Tag Image with Centroid/Distance Marks** |

Any Comments on Effectiveness of Process (50 words max):

I have calculated the centroids, displayed the centres, calculated the distances and displayed them. Displayed lines for the distances. Created an annotation box for all the data required. I could not find a proper way of displaying the chessboard line, but I have improvised for displaying purposes on line 37 and 38.

1. **Adapt for real-time webcam use (0..20 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

%% G. Adapt for webcam use

clear all;

close all;

clc;

camera = webcam(1); % start webcam 1

videoFile = VideoWriter('TaskG.mp4', 'MPEG-4'); % write video to file TaskG.mp4

videoFile.FrameRate = 3; % set frame rate

open(videoFile) % start video file

ax = gcf(); % settings constant variable for video writing capabilities at the bottom of the loop

frames = 310; %settings the frame number for the loop

dim = [0.3 0.9 0.9 0]; % dimensions of annotation box

for i = 1:frames

RGB = snapshot(camera); % opens figure with the live feed

yellowTag = RGB(:,:,1)>195 & RGB(:,:,2)>170 & RGB(:,:,3)<115; % thresholding for the yellow tag

redTag = RGB(:,:,1)>150 & RGB(:,:,2)<65 & RGB(:,:,3)<65; % thresholding for the red tag

se = strel('diamond', 11);

yellowDilated = imdilate(yellowTag, se); % making object slightly bigger and filling holes partially

yellowClose = imclose(yellowDilated, se); % closing all the remaining gaps

yellow = imerode(yellowClose, se); % minimizing back to initial size

redDilated = imdilate(redTag, se); % making object slightly bigger and filling holes partially

redClose = imclose(redDilated, se); % closing all the remaining gaps

red = imerode(redClose, se); % minimizing back to initial size

yellowRangeFilt = rangefilt(yellow); % using range filter to identify the boundary of the tag

redRangeFilt = rangefilt(red); % using range filter to identify the boundary of the tag

RGBandY = imoverlay(RGB, yellowRangeFilt, "blue"); % overlaying the range filter in colour blue over the image

RGBboundaries = imoverlay(RGBandY, redRangeFilt, "green"); % overlaying the range filter in colour green over the already overlayed image

[iY, jY] = find(yellow); % searching for the (i,j) coordinates of the yellow tag

yY = mean2(iY); % mean of the length to find centroid

xY = mean2(jY); %mean if the hight to find centroid

[iR,jR] = find(red); % searching for the (i,j) coordinates of the red tag

yR = mean2(iR); % mean of the length to find centroid

xR = mean2(jR); %mean if the hight to find centroid

dCityBlock = abs(yY-yR)+abs(xY-xR); % calculating the City Block distance

dChessboard = max(abs(yY-yR),abs(xY-xR)); % calculating the Chessboard distance

dEuclidean = sqrt((yY-yR)^2+(xY-xR).^2); % calculating the Euclidean distance

chessx = (((xY+xR)/2) + xR)/2;

chessy = (((yY+yR)/2) + yY)/2;

imshow(RGBboundaries) % displaying image

delete(findall(gcf,'type','annotation')); % delete old annotation if existent

% plotting visuals for the above calculations

hold on

plot([xY, xR],[yY, yY], '-mo', [xR, xR], [yR, yY], '-mo', 'LineWidth', 1); % city block line

plot([xY, chessx], [yY, chessy], '-go', [chessx, xR], [chessy, yR], '-go', 'Linewidth', 1); % chessboard line

plot([xY, xR], [yY, yR],'-c\*','Linewidth',1); % eucledian line

string = {['Red Tag Centroid x coordinates = ',num2str(xR)],['Red Tag Centroid y coordinates = ',num2str(yR)], ...

['Yellow Tag Centroid x coordinates = ',num2str(xY)],['Yellow Tag Centroid y coordinates = ',num2str(yY)], ...

['City Block Distance = ',num2str(dCityBlock)], ...

['Chessboard Distance = ',num2str(dChessboard)], ...

['Euclidean Distance = ',num2str(dEuclidean)]}; % string that will be displayed in the annotation box

annotation('textbox',dim,'String',string,'FitBoxToText','on','EdgeColor','black','Color','black'); % creating the annotation box

drawnow; % updates figure and processes any pending call-backs

frame = getframe(ax); % get the frame ready to be written to video file

writeVideo(videoFile, frame); % writing frame to video file

hold off

end

close(videoFile);

Input and Output Videos:

*Upload TaskG.mp4, a 1-minute (minimum) video showing the border-labelled, centroid/distance labelled video stream from the webcam.*

Any Comments on Effectiveness of Process/Design Decisions (50 words max):

Process works smoothly, there could be more refinement on the frame rate and frame count that would make the video smoother and overall feel live a live feed that isn’t processed frame by frame, and resolution would be my second concern.

1. **Interpretation and light drawing (0..20 marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

clear all;

close all;

clc;

Input and Output Videos:

*Upload TaskH.mp4, a 1-minute (minimum) video showing the border-labelled, centroid/distance labelled video stream from the webcam showing a live drawing.*

Any Comments on Effectiveness of Process/Design Decisions (50 words max):

**Appendix A:** PresentationGuidelines

1. To expedite marking, your work must be submitted using this template ONLY.
2. You are to submit ONE DOCX or PDF report, file along with .m files containing your code for each task. You will also upload two video files in mp4 format if you attempt Tasks G and Task H. You cannot upload a ZIP or RAR file since this would prevent your report being parsed by Turnitin, and would also mean that it cannot be annotated with feedback by the marker.
3. Call your code files **TaskC.m** up to **TaskH.m** so that they can be easily found. It is fine to have additional files containing functions too, if you break your code up in this way.
4. In your source code, variable names should make sense (i.e., avoid single letter variable names, except where these correspond to mathematical convention, like **L**, **M**, **N**, **i** and **j**, or where variable names correspond exactly to the variables used in equations from the course notes). Variable names should *never* be verbs. Where it is appropriate for variables to contain several words, use lower camelCaps (first letter lower case, first letter of each subsequent word in upper case).
5. Code should be well commented (in MATLAB, a comment starts with the % symbol). Where the end keyword is used to terminate a loop or if statement, place % end if, % end while, or % end for, as appropriate, to highlight what the end corresponds to. You can also use the Live Editor in MATLAB to document your code if you prefer.
6. Break large programs into functions, where appropriate, such that each function is informatively named, starting with a verb, and performs one small, well-defined task.
7. Avoid using literals in programs; either use arguments to control settings (e.g., inputs to functions), or collect settings from the user at run-time, or declare named constants (by convention, in UPPER CASE).
8. Code should be properly indented, and ordinarily one line should contain only one command. Before submitting your work, highlight all code and click “smart indent” to do this automatically. Code in your report should be presented in a fixed width font like Courier New to preserve indentation.
9. Ensure code is concise, efficient, and that unnecessary work is not done. Check that your program does not perform calculations or declare variables that are not subsequently used.
10. It is acceptable to use built-in functions provided in MATLAB or its official toolboxes, if these are adequately explained in the corresponding text. If you use MATLAB code from a third-party source, this source should be acknowledged, and the underpinning theory precisely explained.