UP2024896

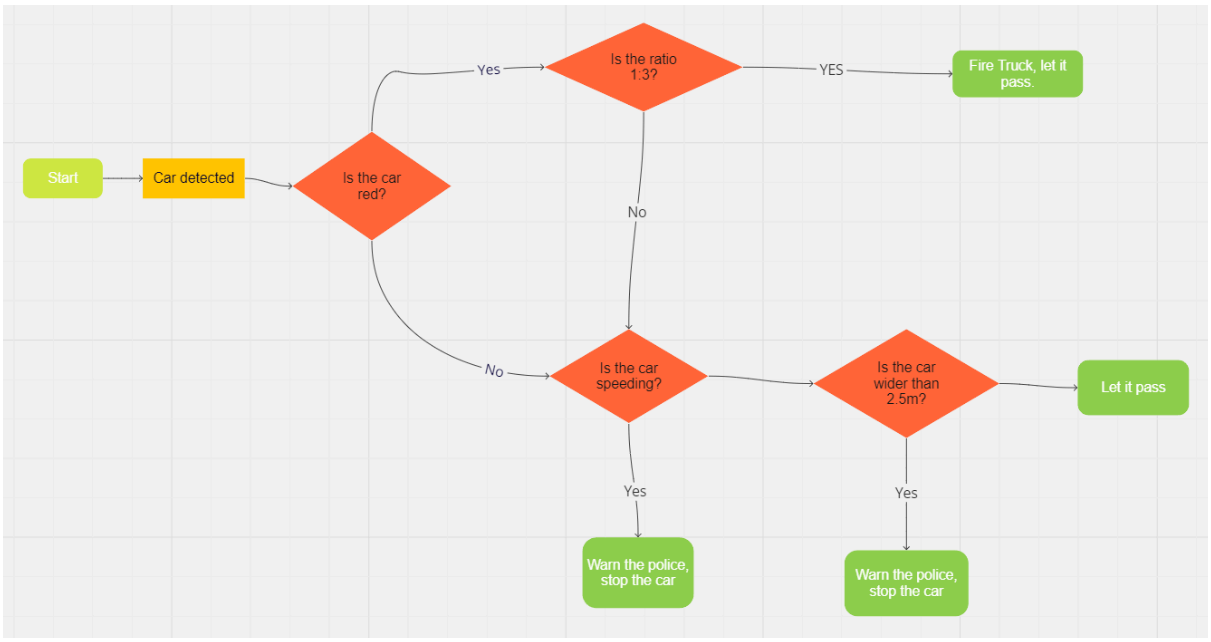
Report for task 1 Computer Vision and Graphics

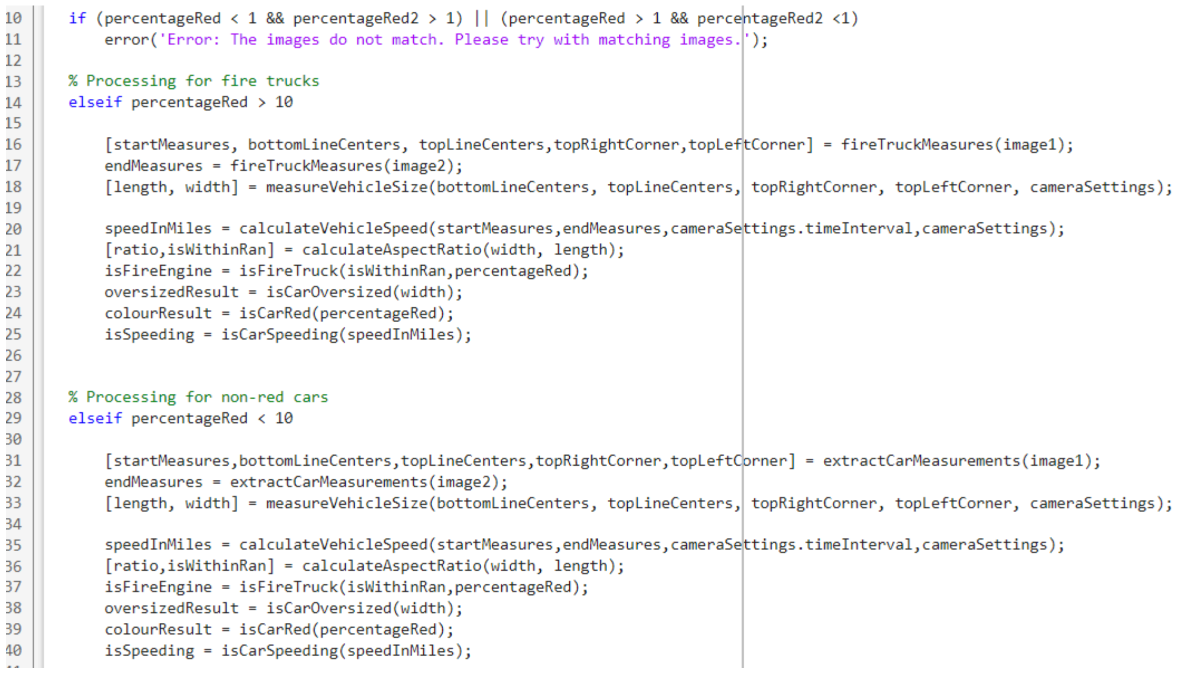
Introduction

This report outlines the development of a speeding camera simulator, which, using image processing tools, can detect the car and measure its dimensions, including checking the type of vehicle.

This attachment shows the simplified logic that was used in the program.

Program logic





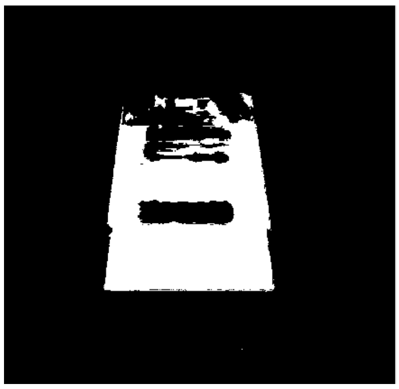
```
if (percentageRed < 1 && percentageRed2 > 1) || (percentageRed > 1 && percentageRed2 <1)
    error('Error: The images do not match. Please try with matching images.');

% Processing for fire trucks
elseif percentageRed > 10

    [startMeasures, bottomLineCenters, topLineCenters,topRightCorner,topLeftCorner] = fireTruckMeasures(image1);
    endMeasures = fireTruckMeasures(image2);
    [length, width] = measureVehicleSize(bottomLineCenters, topLineCenters, topRightCorner, topLeftCorner, cameraSettings);

    speedInMiles = calculateVehicleSpeed(startMeasures,endMeasures,cameraSettings.timeInterval,cameraSettings);
    [ratio,isWithinRan] = calculateAspectRatio(width, length);
    isFireEngine = isFireTruck(isWithinRan,percentageRed);
    oversizedResult = isCarOversized(width);
    colourResult = isCarRed(percentageRed);
    isSpeeding = isCarSpeeding(speedInMiles);


% Processing for non-red cars
elseif percentageRed < 10

    [startMeasures,bottomLineCenters,topLineCenters,topRightCorner,topLeftCorner] = extractCarMeasurements(image1);
    endMeasures = extractCarMeasurements(image2);
    [length, width] = measureVehicleSize(bottomLineCenters, topLineCenters, topRightCorner, topLeftCorner, cameraSettings);

    speedInMiles = calculateVehicleSpeed(startMeasures,endMeasures,cameraSettings.timeInterval,cameraSettings);
    [ratio,isWithinRan] = calculateAspectRatio(width, length);
    isFireEngine = isFireTruck(isWithinRan,percentageRed);
    oversizedResult = isCarOversized(width);
    colourResult = isCarRed(percentageRed);
    isSpeeding = isCarSpeeding(speedInMiles);
```

Objective 1 - Binary Images

To complete this section, I have used two different approaches.

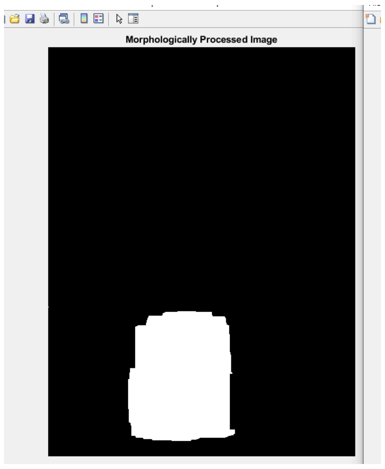
Fire truck

I have designed a function using the RGB image for the HSV colour space. It will then create a binary mask that will fall for a predefined red colour range that will satisfy the application requirements to detect the red object. The challenge was finding a suitable range, as the object had many shades of red. Another crucial step was in filtering out the insignificant regions of red. I ensured the detection of the most significant region by setting a minimum area threshold.

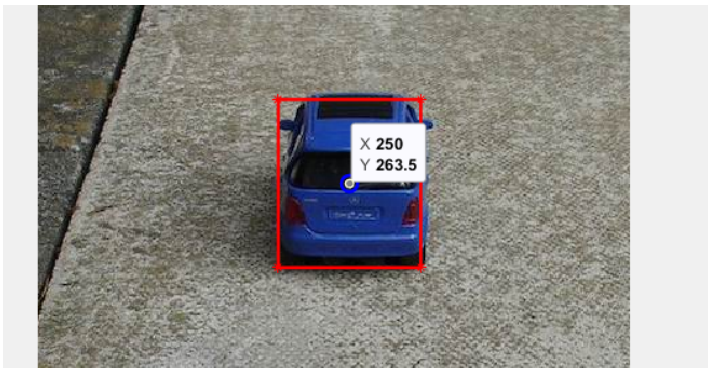


Normal Car

I have converted the RGB image to a grayscale image in this approach. While having that image, I applied the Gaussian filter to remove the noise and soften the edges to enhance the definition of the vehicle. Then, the smoothened image was converted into a binary image. A critical step was to invert the image. In the next step, I applied morphological operations such as closing (dilation followed by erosion).



During the development of the program, I implemented a function that helped me go through the verification process. It used visual clues for the corners and middle points for the images, which was very helpful at the later stage.



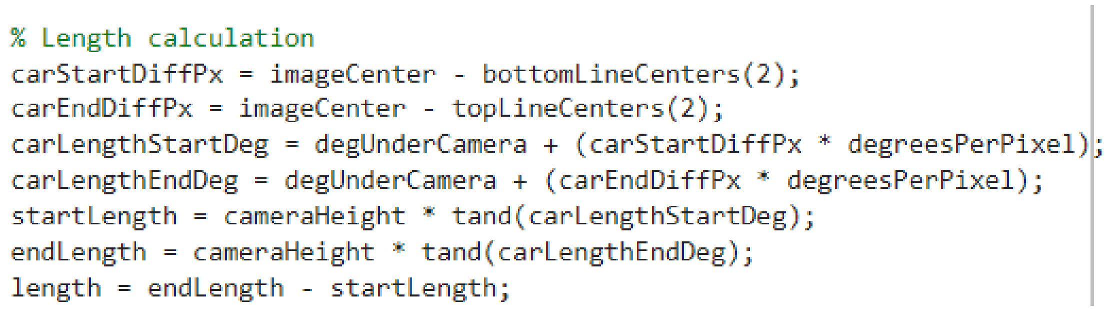
Objective 2 - Calculating the dimensions of the vehicle

To complete this objective, I have created the functions ‘**extractCarMeasurements**’ and ‘**fireTruckMeasures**’. I have used ‘**regionprops**’ in both functions to extract the properties. For getting the object's centre, it was useful to use ‘**Centroid**’.

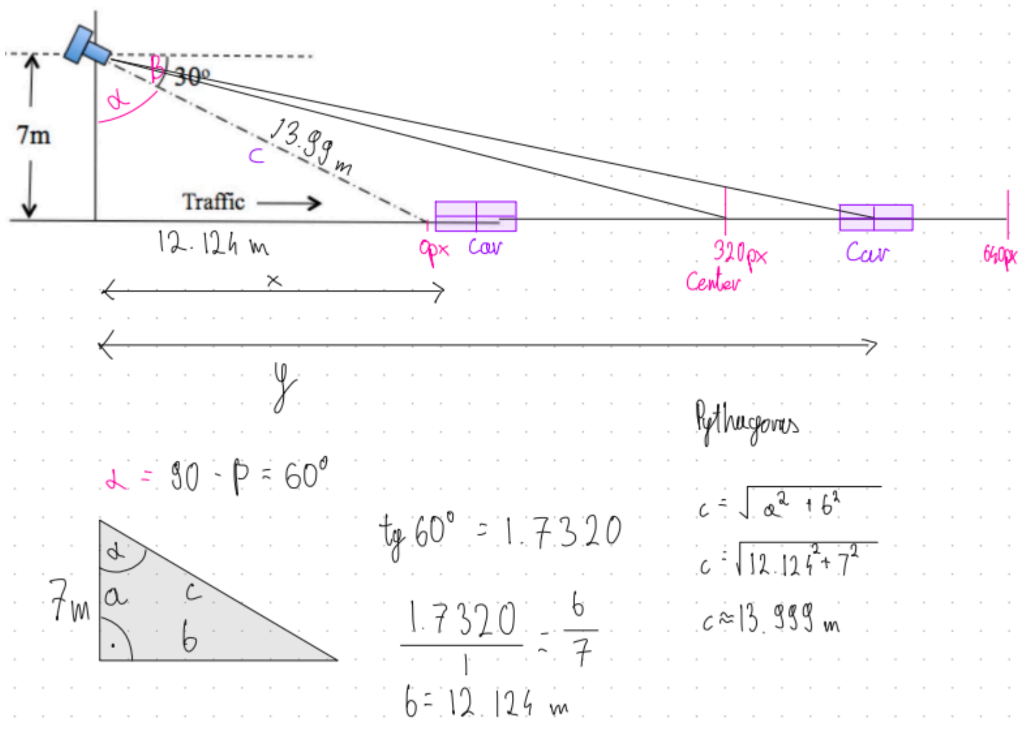
While having all the points selected on the image, I could proceed to the other part of the program to do the triangular calculations to check the distance from the camera to the car. To do such calculations, I have used the calculations below.

Length calculations

In the image provided are my calculations to get the correct information for determining the car's position.

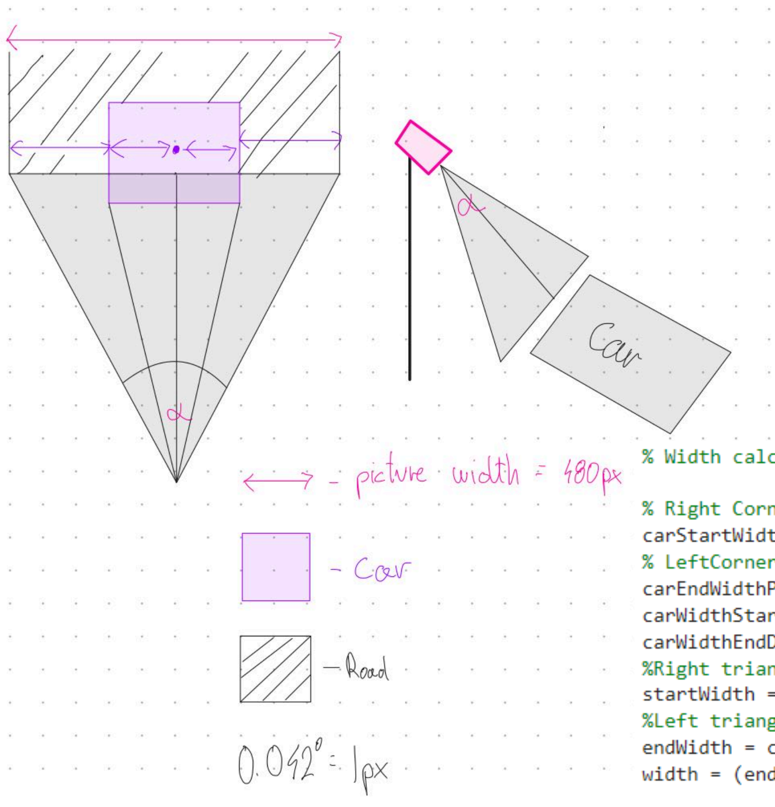


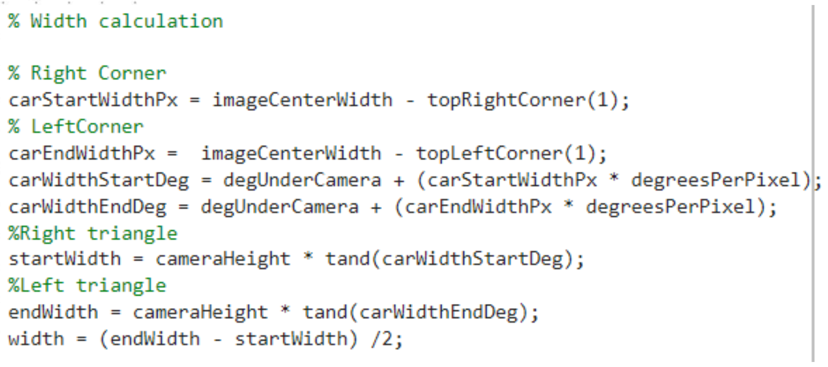
```
% Length calculation
carStartDiffPx = imageCenter - bottomLineCenters(2);
carEndDiffPx = imageCenter - topLineCenters(2);
carLengthStartDeg = degUnderCamera + (carStartDiffPx * degreesPerPixel);
carLengthEndDeg = degUnderCamera + (carEndDiffPx * degreesPerPixel);
startLength = cameraHeight * tand(carLengthStartDeg);
endLength = cameraHeight * tand(carLengthEndDeg);
length = endLength - startLength;
```



Width calculations

For the width, I have followed similar principles. The following image helped me understand the entire task of calculating the width. I have followed the approach to count the distances to both corners of the car from the middle of the picture, assuming that the **width/2** will return to the middle of the screen (x-axis).





```
% Width calculation

% Right Corner
carStartWidthPx = imageCenterWidth - topRightCorner(1);
% LeftCorner
carEndWidthPx =  imageCenterWidth - topLeftCorner(1);
carWidthStartDeg = degUnderCamera + (carStartWidthPx * degreesPerPixel);
carWidthEndDeg = degUnderCamera + (carEndWidthPx * degreesPerPixel);
%Right triangle
startWidth = cameraHeight * tand(carWidthStartDeg);
%Left triangle
endWidth = cameraHeight * tand(carWidthEndDeg);
width = (endWidth - startWidth) /2;
```

Objective 3 - Calculate the vehicle speed

The last step was calculating the vehicle speed involving two images. To complete this objective, I had first to calculate the difference in pixels and then convert the position in pixels to degrees (1px = 0.042deg). Then, the trigonometric tangent function was combined with camera height to translate that into the distance. The vehicle speed was calculated with the distance and the time interval of 0.1s. It then included a conversion to miles per hour from meters per second.

Objective 4 Implementation of side functions

At this stage, I could create the application's main loop with other functions built on top of the already implemented functions. The functions are:

- **isCarOversized(),** returns 'Y' or 'N'
- **isCarRed()**, returns 'Y' or 'N'
- **checkRedDominance()** returns a numeric value
- **isCarSpeeding()** returns 'Y' or 'N'
- **isFireTruck()** returns 'Y' or 'N'

While having all these self-explanatory functions, I could combine them to complete the final task: output them to the user in the console.

Future Improvements

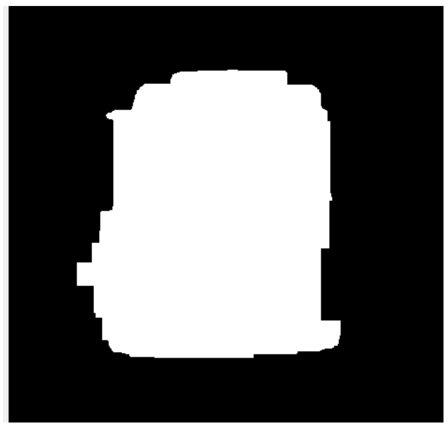
- The application works well only on the given colours, although it should be able to detect all the colours and their shades.
- The loop for getting the measurements can be a separate function to make it even more modular than it is at the moment.
- It should detect all vehicles (e.g. lorries, motorcycles, ambulances, tricycles).

Assumptions:

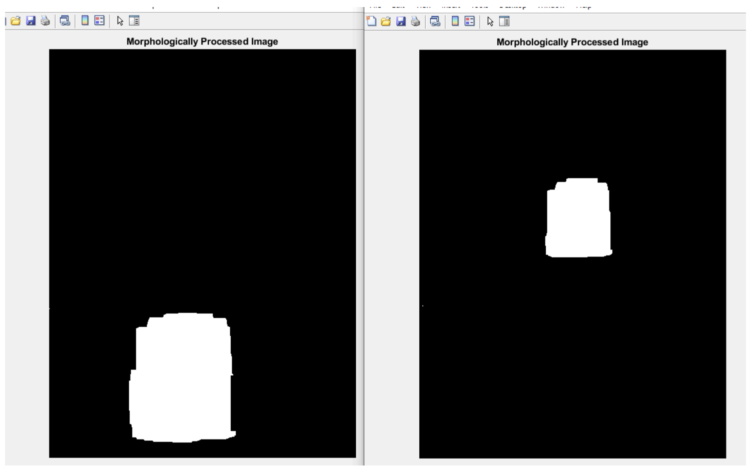
- Pictures are made in 0.1s intervals.

Binary Images

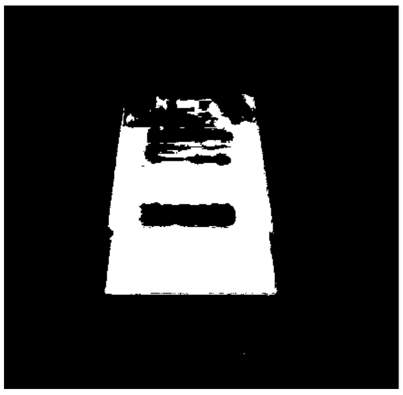
Binary Image of the oversized car



Binary images of the normal-sized car


Morphologically Processed Image
Morphologically Processed Image

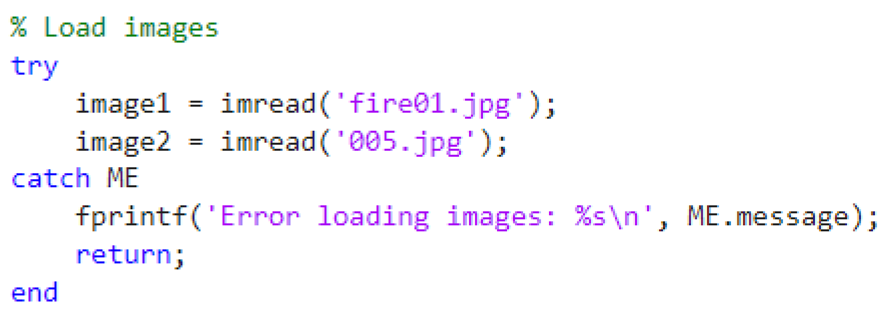
Binary Image of a fire truck



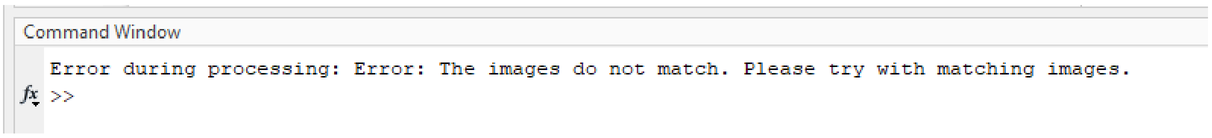
Testing

All tests below are returning the expected positive outcome.

TEST 1:
Test for mismatched pictures.

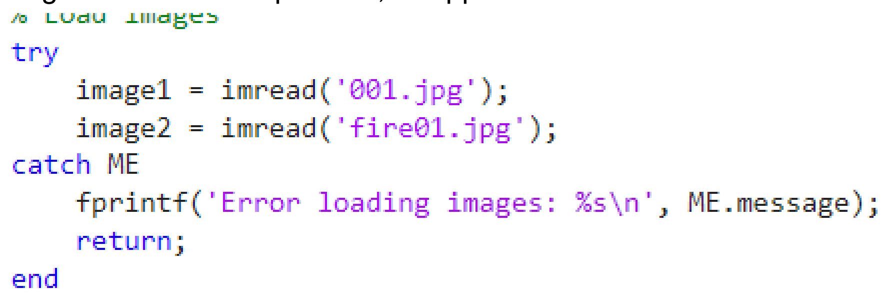


```
% Load images
try
    image1 = imread('fire01.jpg');
    image2 = imread('005.jpg');
catch ME
    fprintf('Error loading images: %s\n', ME.message);
    return;
end
```

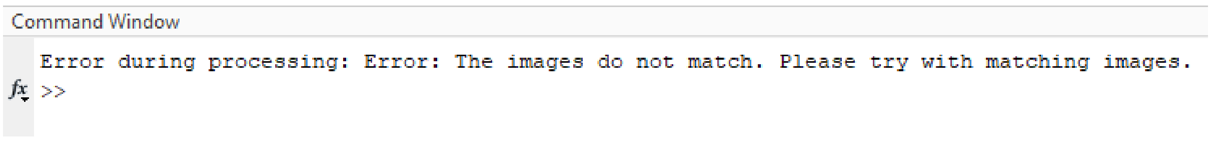


```
Command Window
  Error during processing: Error: The images do not match. Please try with matching images.
fx >>
```

TEST 2:
Testing for mismatched pictures, swapped order.

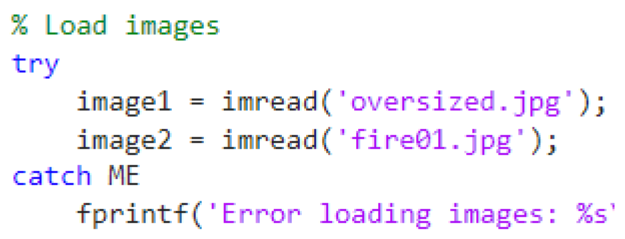


```
% Load images
try
    image1 = imread('001.jpg');
    image2 = imread('fire01.jpg');
catch ME
    fprintf('Error loading images: %s\n', ME.message);
    return;
end
```

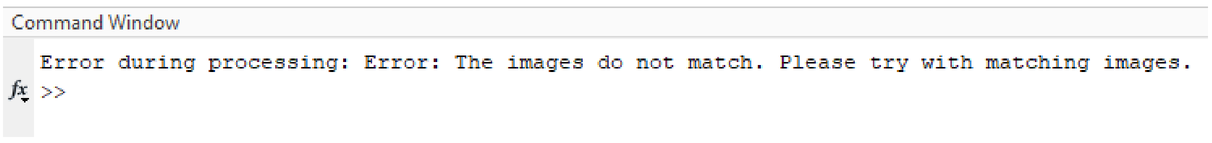


```
Command Window
  Error during processing: Error: The images do not match. Please try with matching images.
fx >>
```

TEST 3:
Test for mismatched pictures, part 3.



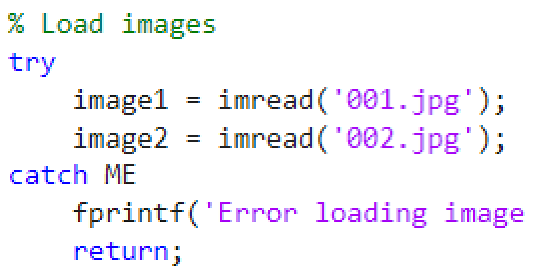
```
% Load images
try
    image1 = imread('oversized.jpg');
    image2 = imread('fire01.jpg');
catch ME
    fprintf('Error loading images: %s'
```



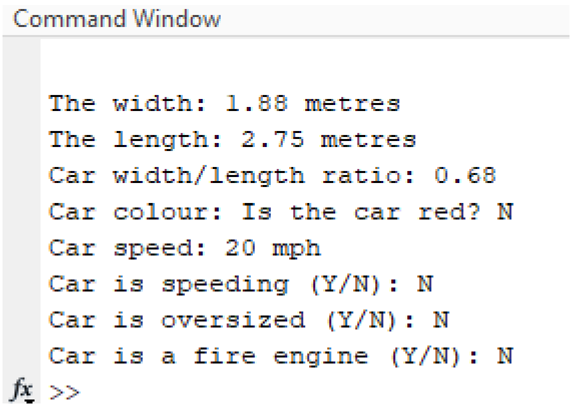
```
Command Window
  Error during processing: Error: The images do not match. Please try with matching images.
fx >>
```

UP2024896

TEST 4:
Using images “001” and “002”.



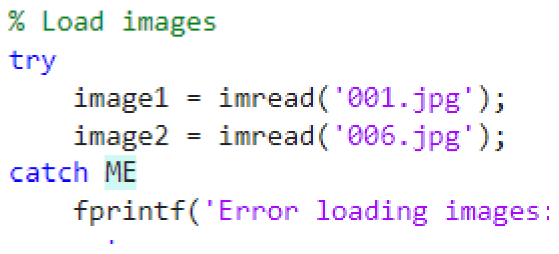
```
% Load images
try
    image1 = imread('001.jpg');
    image2 = imread('002.jpg');
catch ME
    fprintf('Error loading image
    return;
```



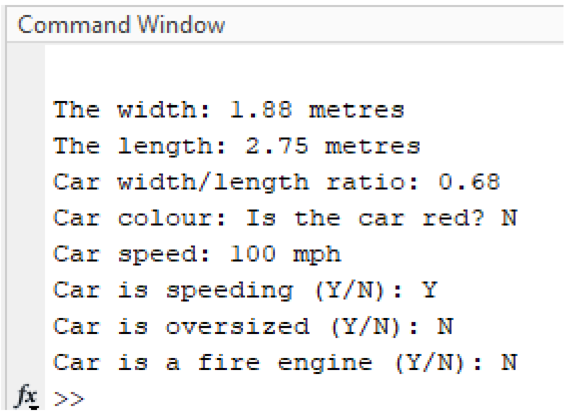
Command Window

```
The width: 1.88 metres
The length: 2.75 metres
Car width/length ratio: 0.68
Car colour: Is the car red? N
Car speed: 20 mph
Car is speeding (Y/N): N
Car is oversized (Y/N): N
Car is a fire engine (Y/N): N
>>
```

TEST 5:
Using images “001” and “006”.



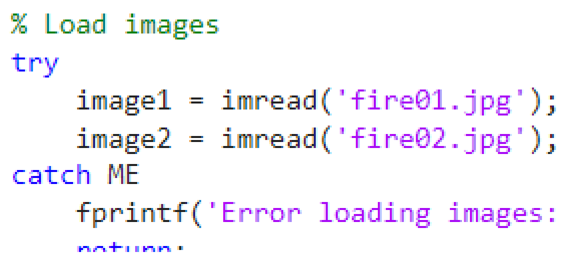
```
% Load images
try
    image1 = imread('001.jpg');
    image2 = imread('006.jpg');
catch ME
    fprintf('Error loading images:
```



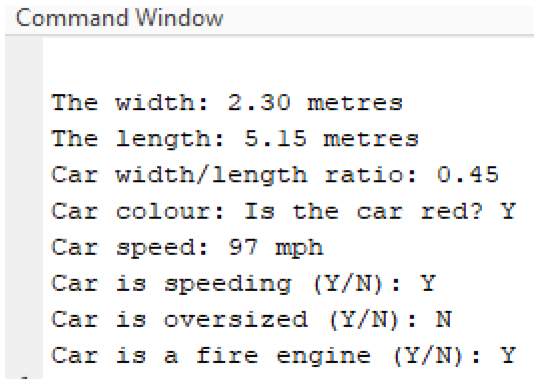
Command Window

```
The width: 1.88 metres
The length: 2.75 metres
Car width/length ratio: 0.68
Car colour: Is the car red? N
Car speed: 100 mph
Car is speeding (Y/N): Y
Car is oversized (Y/N): N
Car is a fire engine (Y/N): N
>>
```

TEST 6:
Testing the fire truck.



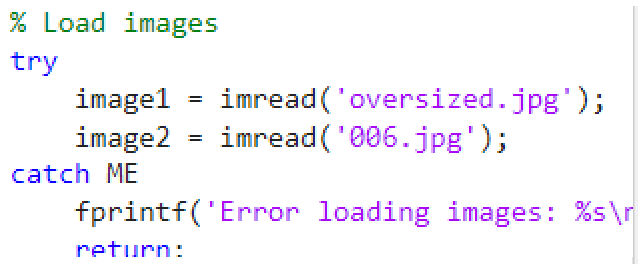
```
% Load images
try
    image1 = imread('fire01.jpg');
    image2 = imread('fire02.jpg');
catch ME
    fprintf('Error loading images:
    return;
```



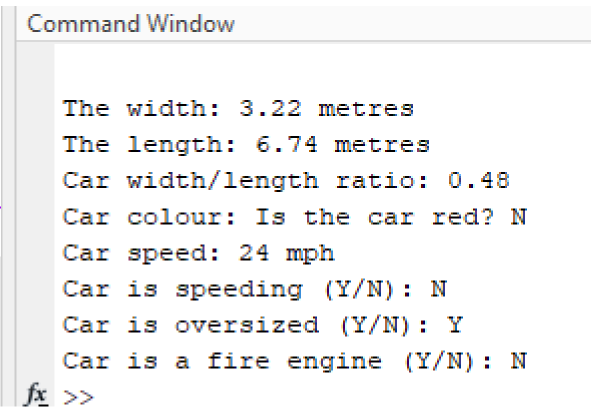
Command Window

```
The width: 2.30 metres
The length: 5.15 metres
Car width/length ratio: 0.45
Car colour: Is the car red? Y
Car speed: 97 mph
Car is speeding (Y/N): Y
Car is oversized (Y/N): N
Car is a fire engine (Y/N): Y
```

TEST 7:
Testing the oversized car with a standard vehicle.



```
% Load images
try
    image1 = imread('oversized.jpg');
    image2 = imread('006.jpg');
catch ME
    fprintf('Error loading images: %s\
    return;
```



Command Window

```
The width: 3.22 metres
The length: 6.74 metres
Car width/length ratio: 0.48
Car colour: Is the car red? N
Car speed: 24 mph
Car is speeding (Y/N): N
Car is oversized (Y/N): Y
Car is a fire engine (Y/N): N
fx >>
```