Junior Design Final Project Matlab Code Documentation

Date: 3/3/2020

Stephen More

Function Header

```
%*************************************************************************
%Defintion: % This takes 200 frames at 10fps from the c270 camera and
 saves the avi into
% diskLogger defined path into non volatile memory.
%Parameters: quality is a string input either "LD" or "HD" for 640x480
 and
%1280x720 resolution respectively. vidFileName is the name of the file
 the
%user wants the video saved into.
%Output: Saved uncompressed .avi file with desired quality at desired
%file name.
%*************************************************************************
```

Image Acquisition

```
function c270Acquire(vidFileName, quality)

    if(quality == "LD")
        vid = videoinput('winvideo', 2, 'I420_640x480');
    else
        vid = videoinput('winvideo', 2, 'I420_1280x720');
    end
    src = getselectedsource(vid); % src is not used however it is good
 for code flexibility and changes.
    vid.FramesPerTrigger = 200; %Frames per function call
    vid.ReturnedColorspace = 'rgb'; %Color space of .avi
    vid.LoggingMode = 'disk'; %Determines how the file is saved
    %writes video file to the filesAVI folder
    diskLogger = VideoWriter(['C:\Users\Stephen
 More\Documents\MATLAB\Jr Design FP\filesAVI\',
 sprintf('%s.avi',vidFileName)], 'Uncompressed AVI');
    vid.DiskLogger = diskLogger;
    %sets fps to 10
    diskLogger.FrameRate = 10;
    start(vid);

    stoppreview(vid);

end
```

Published with MATLAB® R2019b

Table of Contents

Function Header

```
%************************************************
%Defintion: This function calibrates a logitech c270 Camera
%Parameters: quality is a string input either "LD" or "HD" for 640x480
 and
%1280x720 resolution respectively.
%Output: pitch,height,camIntrinsics,calibrationResolution, imageSize,
%coinsI, all of these combined allow for a user to see which surface
 the
%camera is calibrated for, perform inverse perspective mapping, and
 map
%pixels to real world distances.
%************************************************
```

c270 Camera Calibration

```
function [birdsEye, camIntrinsics, coinsI] = c270Calibrate(quality,
 distAheadi, spaceToOneSidei, bottomOffseti)

    HD = '1280x720';
    LD = '640x480';
    if(quality == "LD")
        calibrationResolution = LD;
    else
        calibrationResolution = HD;
    end
    % Define images to process
    imageFileNames640x480 = {
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\GPUImages\640x480GPUCheckerboard.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_38_58_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_39_09_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_39_14_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_39_27_Pro.jpg',...
```

```
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_39_32_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_39_40_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_39_54_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_40_04_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_40_30_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\WIN_20200122_16_40_51_Pro.jpg',...
        };
    %Put images which are of objects on the calibrated surface
    coinImageFileNames640x480 = {
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\GPUImages\640x480GPUCoinsImage_1.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\GPUImages\640x480GPUCoinsImage_1.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins1.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins2.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins3.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins4.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins5.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins6.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins7.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins8.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins9.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins10.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\CaliWCoins11.jpg',...
        };
    imageFileNames1280x720 = {'C:\Users\Stephen More\Documents
\MATLAB\Jr Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_19_58_40_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_19_59_13_Pro.jpg',...
```

```
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_00_02_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_00_59_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_01_02_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_01_25_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_01_29_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_01_50_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_01_54_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_02_09_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_02_16_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_02_34_Pro.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr
 Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\WIN_20200123_20_03_02_Pro.jpg',...
        };
    coinImageFileNames1280x720 = { 'C:\Users\Stephen More\Documents
\MATLAB\Jr Design FP\c270VideosandImages\CameraCaliCheckerBoard
\1280x720\coins1.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins2.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins3.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins4.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins5.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins6.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins7.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins8.jpg',...
        'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\c270VideosandImages\CameraCaliCheckerBoard\1280x720\coins9.jpg',...
        };
```

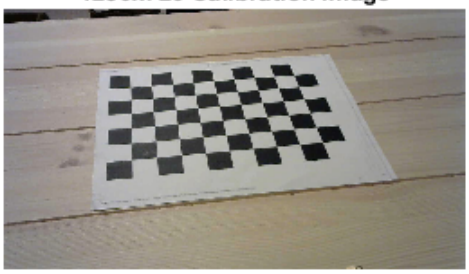
Compute Camera Intrinsic and Extrinsic Parameters

```
   % Display one of each calibration images to the user
   %640x480
   magnification = 25;
   I = imread(imageFileNames640x480{1});
   figure; imshow(I, 'InitialMagnification', magnification);
   title('640x480 Calibration Image');
   %1280x720
   IHD = imread(imageFileNames1280x720{3});
   figure; imshow(IHD, 'InitialMagnification', magnification);
   title('1280x720 Calibration Image');
   % Detect checkerboards in images
   if strcmp(calibrationResolution,LD)
       [imagePoints, boardSize, imagesUsed] =
detectCheckerboardPoints(imageFileNames640x480);
       imageFileNames640x480 = imageFileNames640x480(imagesUsed);

       % Read the first image to obtain image size
       originalImage = imread(imageFileNames640x480{1});
       [mrows, ncols, ~] = size(originalImage);
   else
       [imagePoints, boardSize, imagesUsed] =
detectCheckerboardPoints(imageFileNames640x480);
       imageFileNames1280x720 = imageFileNames1280x720(imagesUsed);

       % Read the first image to obtain image size
       originalImage = imread(imageFileNames1280x720{1});
       [mrows, ncols, ~] = size(originalImage);
   end


   % Generate world coordinates of the corners of the squares
   squareSize = 24;  % in units of 'millimeters'
   worldPoints = generateCheckerboardPoints(boardSize, squareSize);

   % Calibrate the camera, Images used and estimation errors are not
used
   % but are useful to have for flexibility in code.
   [cameraParams, imagesUsed, estimationErrors] =
estimateCameraParameters(imagePoints, worldPoints, ...
       'EstimateSkew', false, 'EstimateTangentialDistortion',
false, ...
       'NumRadialDistortionCoefficients',
2, 'WorldUnits', 'millimeters', ...
       'InitialIntrinsicMatrix', [], 'InitialRadialDistortion',
[], ...
       'ImageSize', [mrows, ncols]);

   %however is useful if you want to manually input intrinsic vals)
```

```
   camIntrinsics =
cameraIntrinsics(cameraParams.Intrinsics.FocalLength,cameraParams.Intrinsics.Prin
   % Now make sure the extrinsic Qualities are correct
   %Load an image with a checkerboard placed on ground for
calibration
   %SET I TO THE IMAGE YOU NEED TO CALCULATE CAMERA EXTRINSICS FOR!
   if strcmp(calibrationResolution,LD)
       I = imread(imageFileNames640x480{1});
       coinsI = imread(coinImageFileNames640x480{1});
   else
       I = imread(imageFileNames1280x720{3});
       coinsI = imread(coinImageFileNames1280x720{3});
   end
   %This function returns the image points on cornerse of
checkerboard
   [imagePoints,boardSize] = detectCheckerboardPoints(I);

   squareSize = 0.024; % Square size in meters
   %This function generates world distances from image points
   worldPoints = generateCheckerboardPoints(boardSize,squareSize);

   %set patternOriginHeight to 0 if checkerboard is on ground
   patternOriginHeight = 0;
   %I am using monocamera so I use the function to estimate its
   %parameters.
   [pitch,~,~,height] =
estimateMonoCameraParameters(camIntrinsics,imagePoints,worldPoints,patternOriginH
```

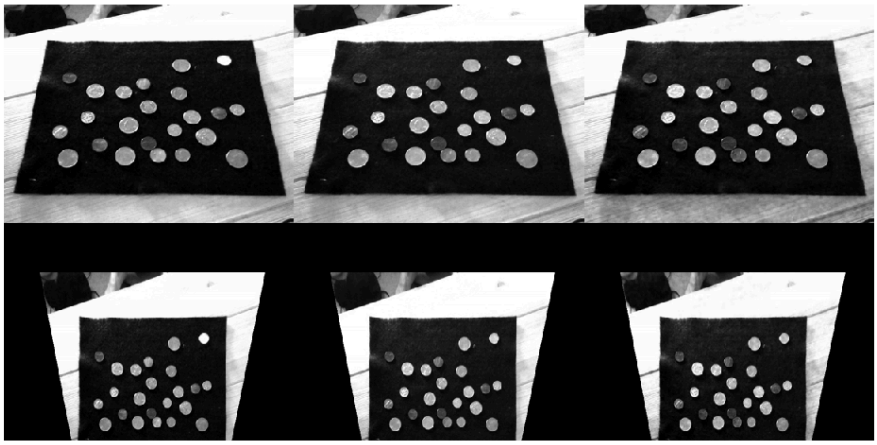
640x480 Calibration Image

1280x720 Calibration Image



Compute Birds Eye View Image and display to User to Verify Calibration

```
    %This allows the last 3 parameters to be optional inputs
    if (~exist('distAheadi','var') || ~exist('distAhead','var') ||
~exist('distanceAheadi','var'))
        if strcmp(calibrationResolution,LD)
            distAhead = .6;%1
            spaceToOneSide = .3;
            bottomOffset = .2;
        else
            distAhead = 0.4; %.4
            spaceToOneSide = 0.5;%0.5;
            bottomOffset = 0;
        end
    else
        distAhead = distAheadi;
        spaceToOneSide = spaceToOneSidei;
        bottomOffset = bottomOffseti;
    end
    %Sets output view for inverse perspective mapping in meters
    outView = [bottomOffset,distAhead,-spaceToOneSide,spaceToOneSide];
    %Sets output view for inverse perspective mapping in pixels
    outImageSize = cameraParams.Intrinsics.ImageSize;

    sensor = monoCamera(camIntrinsics,height,'Pitch',pitch);
    birdsEye = birdsEyeView(sensor,outView,outImageSize);
    %Outputs a montage of RGB Birds Eye View images to the user to
check
    %for accuracy
    BEV = transformImage(birdsEye,coinsI);
    multi = cat(3,coinsI,BEV);
    figure;
    montage(multi);
    return;
```



Published with MATLAB® R2019b

Table of Contents

Script Header

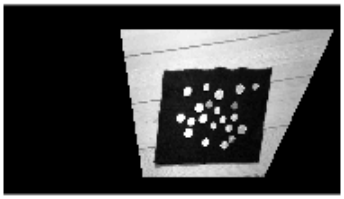
```
%*****************************************
%Defintion: This script takes in the c270 calibration data and uses it
 to
%detect all of the coins in an image as well as deploying various
 image
%processing methods.
%PreCondtions: This script cannot be executed before the
 c270Calibration
%function.
%Outputs: XY coordinates in meters of coin centers, number of each
 coin,
%and total value in USD.
%*****************************************
```

Image Pre Processing

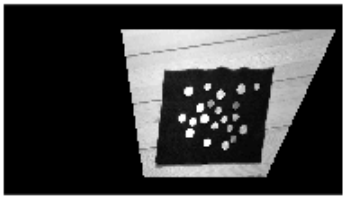
```
    BEV = transformImage(birdsEye,coinsI);
    TopLeftPixelX = 381;%612;
    TopLeftPixelY = 0;%277;
    cropWidth = size(BEV,2)-TopLeftPixelX;%396;
    cropHeight = 720;%325;
    LDf = camIntrinsics.ImageSize == [480 640];
    %Perform Cropping
    if(LDf)
        BEVc = imcrop(BEV,[138 16 454 446]);
    else
        %BEVc = imcrop(BEV,[455 180 768 478]);
        BEVc = imcrop(BEV,[TopLeftPixelX TopLeftPixelY cropWidth
 cropHeight]);
        %BEVc = imcrop(BEV,[450 0 820 720]);
    end
    %Color space analysis
    Im = BEV;

    rmat = Im(:,:,1); % matrix of R pixel values 0-255
    gmat = Im(:,:,2); % matrix of G pixel values 0-255
    bmat = Im(:,:,3); % matrix of B pixel values 0-255

    %Plot RGB matricies
    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(Im);
title('Original Image');

threshR = 0.3;
threshG = 0.3;
threshB = 0.3;
%i1 = im2bw(rmat,threshR);
%i2 = im2bw(gmat,threshG);
%i3 = im2bw(bmat,threshB);
i1 = imbinarize(rmat,threshR);
i2 = imbinarize(gmat,threshG);
i3 = imbinarize(bmat,threshB);

Isum = (i1&i2&i3); % sums up all binary images

%Plot of color data
figure;
subplot(2,2,1), imshow(i1);
title('Red Plane');
subplot(2,2,2), imshow(i1);
title('Green Plane');
subplot(2,2,3), imshow(i1);
title('Blue Plane');
subplot(2,2,4), imshow(i1);
title('Sum of all Planes');
```

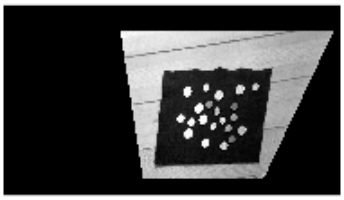
Red Plane



Green Plane



Blue Plane



Original Image

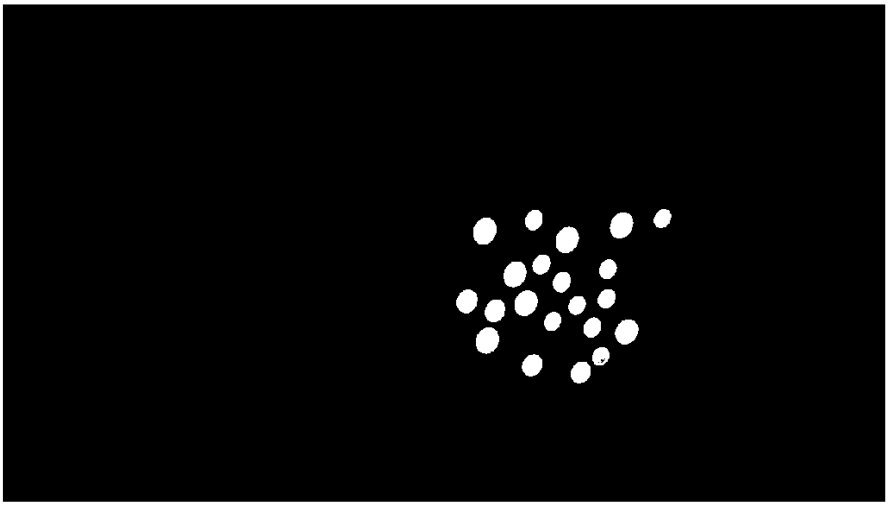
Red Plane

Green Plane

Blue Plane

Sum of all Planes

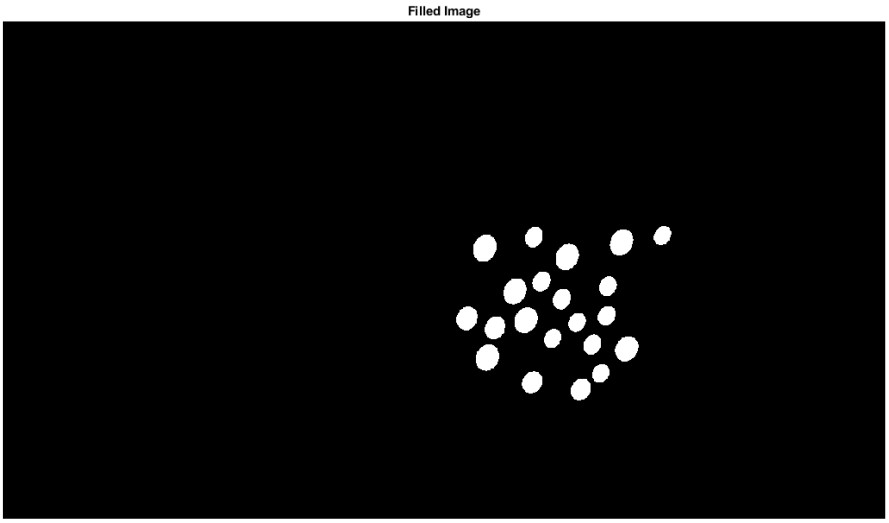
```
%figure
%imshow(Isum);
BW = Isum;
CC = bwconncomp(Isum);
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
BW(CC.PixelIdxList{idx}) = 0;
figure
imshow(BW);
%hold on;
```



Feature analysis

```
Ifilled = imfill(BW, 'holes');
%figure,
imshow(Ifilled);
title('Filled Image');


% Extract features
Iregion = regionprops(Ifilled, 'Centroid');
[labeled] = bwlabel(Ifilled,4);
stats =
regionprops(labeled,'Eccentricity', 'Area', 'BoundingBox', 'Centroid', 'MajorAxis
areas = [stats.Area];
eccentricities = [stats.Eccentricity];

%se = strel('disk', 6);
%Iopenned = imopen(Ifilled,se);
%imshowpair(Iopenned, Ifilled);
%figure;
%imshow(Iopenned);
```

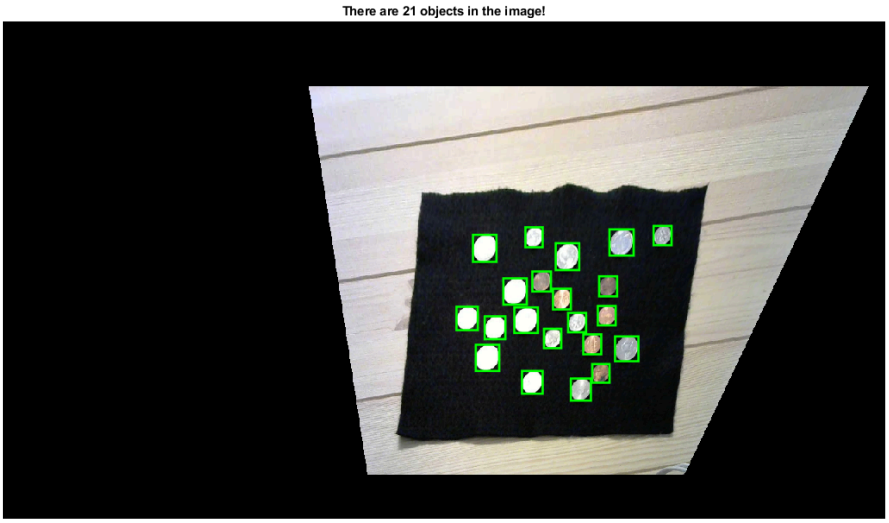
```
   %closeBW = imclose(Iopenned,se);
   %figure, imshow(closeBW)

   idxOfObjects = find(eccentricities);
   statsDefects = stats(idxOfObjects);
```




```
   %imArr = zeros(size(BEVc));
   %figure; imshow(BEVc);
   figure; imshow(BEV);
   hold on;

   rectangleValues = zeros(length(idxOfObjects),4);
   threshSizeInPixelsHi = 50;
   threshSizeInPixelsLo = 20;
   %Filter height and width values by the threshold height/width in
pixels
       for idx = 1 : length(idxOfObjects)
           if( ((statsDefects(idx).BoundingBox(3) <
threshSizeInPixelsHi) && (statsDefects(idx).BoundingBox(4) <
threshSizeInPixelsHi)) && (statsDefects(idx).BoundingBox(3) >
threshSizeInPixelsLo) && (statsDefects(idx).BoundingBox(4) >
threshSizeInPixelsLo) )
               rectangleValues(idx,:) =
statsDefects(idx).BoundingBox;
           else
               %Assign the value of -1 to all rows that need to be
deleted
               rectangleValues(idx,:) = -1;
           end
       end
       %Create a logical array
       idx = (rectangleValues == -1);
```

```
        %use "find" to find where logical index is 1
        [delRows, ~] = find(idx);
        %Delete Rows
        rectangleValues(delRows,:) = [];

    numObjects = length(rectangleValues);
    meanRGB = zeros(numObjects,4);
    %DO THIS LATER ITS WAY FASTER TO PREALLOCATE STRUCTS
    %coinStruct = struct('meanRGB', cell(numObjects, 4),
'boundingBox', cell(numObjects, 10));
    %Crop boxes around coins
    for idx = 1 : numObjects

            h =
rectangle('Position',statsDefects(idx).BoundingBox,'LineWidth',2);
            set(h,'EdgeColor',[0 1 0]);
            hold on;
            %First method to calculate radii and diameter of coins

            imCurr = imcrop(BEV,rectangleValues(idx:idx,1:4));
            imCurrR = imCurr(:,:,1);
            imCurrG = imCurr(:,:,2);
            imCurrB = imCurr(:,:,3);
            meanR = mean(mean(imCurrR));
            meanG = mean(mean(imCurrG));
            meanB = mean(mean(imCurrB));
            meanRGB(idx,1) = meanR;
            meanRGB(idx,2) = meanG;
            meanRGB(idx,3) = meanG;
            if (meanRGB(idx,1) > meanRGB(idx,2) && meanRGB(idx,1) >
meanRGB(idx,3))
                meanRGB(idx,4) = 1;
            elseif (meanRGB(idx,2) > meanRGB(idx,1) && meanRGB(idx,2)
> meanRGB(idx,3))
                meanRGB(idx,4) = 2;
                else
                meanRGB(idx,4) = 3;
            end
            coinStruct(idx).meanRGB = meanRGB(idx,:);
            coinStruct(idx).boundingBox = rectangleValues(idx:idx,:);
            coinStruct(idx).centers.morph = stats(idx).Centroid;
            coinStruct(idx).diameters.morph =
mean([stats(idx).MajorAxisLength stats(idx).MinorAxisLength],2);
            coinStruct(idx).radii.morph =
coinStruct(idx).diameters.morph/2;
            coinStruct(idx).eccentricity = stats(idx).Eccentricity;
            coinStruct(idx).area = stats(idx).Area;
            coinStruct(idx).coinIdx = idx;
            %subplot(2,length(idxOfObjects)/2,idx), imshow(imCurr);
            hold on;
    end
    if idx > 10
    title(['There are ', num2str(numObjects), ' objects in the
image!']);
```

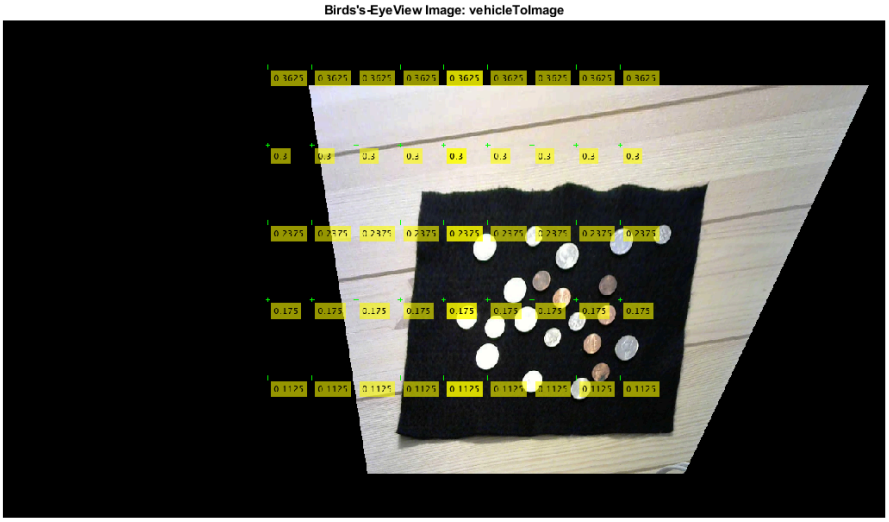
```
end
hold off;


%se = strel('disk', 8);
%Iopenned = imopen(Isum, se);
%imshow(Iopenned);
%title('Disk Morphologically Modified Image');

%coinsIgrey = rgb2gray(BEVc);
%BW_in = im2bw(coinsI);
%imshow(coinsI);
%figure;
%imshow(coinsIgrey);
%figure;
%IHC = histeq(coinsIgrey);
%imshow(IHC);
%figure;
%BW_in = im2bw(coinsIgrey);
%BW_in = imfill(BW_in,'holes');
%imshow(BW_in);
```



Pixel coordinates to real world distance

```
   %In the bird's-eye-view image, place a meter marker directly in
front of the sensor. Use the vehicleToImage function to specify the
location of the marker in vehicle coordinates. Display the marker on
the bird's-eye-view image.
   %The code below puts the desired marker points into a format that
can be
   %used by the insertMarker Function
   horizInterval = 0.05;
```

```
    invertCount = 0;
    NumPoints = 5;
    imagePointCellMat = zeros(2*NumPoints^2,2);
    distanceVal = zeros(1,2*(NumPoints^2));
    startingMarkerDistance = .05;
    markerDistanceInterval = 1/16;
    for h = 1:2
        for k = 1:NumPoints
            for i = 1:NumPoints
                if (h == 2 && invertCount == 0)
                    horizInterval = horizInterval * (-1);
                    invertCount = invertCount+1;
                end
                % The rows term of the VehicleToImage 2nd Parameter
 always
                % needs to equal the distanceVal to reflect accurate
                % measurements!
                tempPos = vehicleToImage(birdsEye,
[startingMarkerDistance+(i*markerDistanceInterval)
 horizInterval*(k-1)]);
                distanceVal(1,(i+NumPoints*(k-1)+(h-1)*NumPoints^2)) =
 startingMarkerDistance+(i*markerDistanceInterval);
                for j = 1:2
                    imagePointCellMat((i
+NumPoints*(k-1)+(h-1)*NumPoints^2),j) = tempPos(1,j);
                end
            end
        end
    end
    %imagePoint = vehicleToImage(birdsEye,[0.5 0]);
    %imagePoint2 = vehicleToImage(birdsEye,[1 0]);
    annotatedBEV = insertMarker(BEV, imagePointCellMat);
    %annotatedBEV = insertMarker(BEV, imagePoint2);
    annotatedBEV = insertText(annotatedBEV, imagePointCellMat + 5,
 distanceVal);
    %annotatedBEV = insertText(annotatedBEV, imagePoint2 + 5, '1
 meters');
    figure
    imshow(annotatedBEV);
    title('Birds''s-EyeView Image: vehicleToImage');
    % Cropped image Image to World Distance
    %OriginalPixelDistanceVals = croppedPixelDistanceValues +
 TopLeftPixelValues
```



Coin Recognition

```
%figure;
calibrationResoltuion = "HD";
imshow(BEV);
[centers,radii] = imfindcircles(Ifilled,[8 100]);
numCoins = size(radii,1);
%[dimeCenters, dimeRadii] = imfindcircles(BW_in, [50,64]);
%[pennyCenters, pennyRadii] = imfindcircles(BW_in, [65,70]);
%[nickleCenters, nickleRadii] = imfindcircles(BW_in, [70,79]);
%[quarterCenters, quarterRadii] = imfindcircles(BW_in, [80,85]);

dimeCircles = zeros(numCoins,3);
pennyCircles = zeros(numCoins,3);
nickleCircles = zeros(numCoins,3);
quarterCircles = zeros(numCoins,3);

dimeCenters = zeros(numCoins,2);
pennyCenters = zeros(numCoins,2);
dimeOrPennyCenters = zeros(numCoins,2);
nickleCenters = zeros(numCoins,2);
quarterCenters = zeros(numCoins,2);
dimeRadii = zeros(numCoins,1);
pennyRadii = zeros(numCoins,1);
dimeOrPennyRadii = zeros(numCoins,1);
nickleRadii = zeros(numCoins,1);
quarterRadii = zeros(numCoins,1);
dimeCount = 0;
pennyCount = 0;
dimeOrPennyCount = 0;
nickleCount = 0;
quarterCount = 0;
```

```
   %convert center x y pixel values from cropped to image to original
image
   %centersBEV = zeros(length(centers),2);
   %for i=1:length(centers)
   %    centersBEV(i,1) = (centers(i,1) + TopLeftPixelX);
   %    centersBEV(i,2) = (centers(i,1) + TopLeftPixelY);
   %end

   annotatedBEV = insertMarker(BEV, centers);
   %for i = 1:numCoins
   %    annotatedBEV = insertText(annotatedBEV,worldDistanceOnImage +
5, ("Xpos in m:" + num2str(WorldCoordinateMat(i,2)) + newline + "Ypos
in m:" + num2str(WorldCoordinateMat(i,1))));
   %end
   %figure;
   imshow(annotatedBEV);
   dimeRadiiThreshHD = 12.7;% 14.5 Commented Values used from edge
detection
   pennyRadiiThreshHD = 13.5;% 16.5
   nickleRadiiThreshHD = 15.5;% 18
   dimeOrPennyRadiiThreshHD = 13.5;

   dimmeOrPennyRadiiThreshLD = 11;
   nickleRadiiThreshLD = 12;
   %*********************************************************
   circles = [centers, radii];
   circles = sortrows(circles,1); %Now the circles are displayed Left
to Right (x = 0 to x = x1)

   WorldCoordinateMat =
imageToVehicle(birdsEye,circles(:,1:2)); %Real world distance values
   worldDistanceOnImage =
vehicleToImage(birdsEye,WorldCoordinateMat);
   for i = 1:numCoins
       coinStruct(i).circles.imfind = circles(i,:);
       coinStruct(i).centers.imfind = centers(i,:);
       coinStruct(i).radii.imfind = coinStruct(i).circles.imfind(3);
   end
   %*********************************************************
   %This code looks at all bounding boxes if needed
             %for j = 1:numCoins
                   %if((dimeOrPennyCenters(k,1) >=
(rectangleValues(j,1) + rectangleValues(j,3))) &&
(dimeOrPennyCenters(k,2) >= (rectangleValues(j,2)+
rectangleValues(j,4))) )
             %      if((dimeOrPennyCenters(k,1) <=
(coinStruct(j).boundingBox(1) + coinStruct(j).boundingBox(3)))
&& (dimeOrPennyCenters(k,2) <= (coinStruct(j).boundingBox(2) +
coinStruct(j).boundingBox(4))) )
             %          %The coin center values are inside of the
bounding box
             %              if( coinStruct(j).meanRGB(j,4) == 1) %&&
(coinStruct(j).meanRGB(1) > coinsStruct(j).meanRGB(3)) )
```

```
            %                 coinStruct(j).coinType = "Penny";
            %             else
            %                 coinStruct(j).coinType = "Dime";
            %             end
            %             break;
            %     end
            % end


   %*********************************************************
   % This section sorts the coins by radii size
   % Sometimes confuses dimes with pennies
   if strcmp(calibrationResolution,"LD")
       for k=1:numCoins
          if (coinStruct(k).radii.imfind < dimeOrPennyRadiiThreshLD)
               %dimeOrPennyCenters(k,:) =
coinStruct(k).centers.imfind;
               %dimeOrPennyRadii(k) = coinStruct(k).radii.imfind;
                if( coinStruct(k).meanRGB(4) == 1)
                   coinStruct(k).coinType = "Penny";
                   pennyCircles(k,:) = coinStruct(k).circles.imfind;
                   pennyCount = pennyCount + 1;
                else
                   coinStruct(k).coinType = "Dime";
                   dimeCircles(k,:) = coinStruct(k).circles.imfind;
                   dimeCount = dimeCount + 1;
                end
          elseif ((coinStruct(k).radii.imfind >
dimeOrPennyRadiiThreshLD) && (coinStruct(k).radii.imfind <
nickleRadiiThreshLD))
               coinStruct(k).coinType = "Nickel";
               nickleCircles(k,:) = coinStruct(k).circles.imfind;
               nickleCount = nickleCount+1;
          else
               coinStruct(k).coinType = "Quarter";
               quarterCircles(k,:) = coinStruct(k).circles.imfind;
               quarterCount = quarterCount+1;
          end
       end
   else
        for k=1:numCoins
          if (coinStruct(k).radii.imfind < dimeOrPennyRadiiThreshHD)
               %dimeOrPennyCenters(k,:) =
coinStruct(k).centers.imfind;
               %dimeOrPennyRadii(k) = coinStruct(k).radii.imfind;
                if( coinStruct(k).meanRGB(4) == 1)
                   coinStruct(k).coinType = "Penny";
                   pennyCircles(k,:) = coinStruct(k).circles.imfind;
                   pennyCount = pennyCount + 1;
                else
                   coinStruct(k).coinType = "Dime";
                   dimeCircles(k,:) = coinStruct(k).circles.imfind;
                   dimeCount = dimeCount + 1;
                end
```

```
          elseif ((coinStruct(k).radii.imfind >
dimeOrPennyRadiiThreshHD) && (coinStruct(k).radii.imfind <
nickleRadiiThreshHD))
               coinStruct(k).coinType = "Nickel";
               nickleCircles(k,:) = coinStruct(k).circles.imfind;
               nickleCount = nickleCount+1;
          else
               coinStruct(k).coinType = "Quarter";
               quarterCircles(k,:) = coinStruct(k).circles.imfind;
               quarterCount = quarterCount+1;
          end
        end
   end
   viscircles(dimeCircles(:,1:2),dimeCircles(:,3), 'EdgeColor','b');

viscircles(pennyCircles(:,1:2),pennyCircles(:,3), 'EdgeColor','r');

viscircles(nickleCircles(:,1:2),nickleCircles(:,3), 'EdgeColor','g');

viscircles(quarterCircles(:,1:2),quarterCircles(:,3), 'EdgeColor','c');

   totalUSD = 0.25*quarterCount + 0.05*nickleCount + 0.1*dimeCount +
0.01*pennyCount;
   coinMat = [totalUSD quarterCount dimeCount nickleCount
pennyCount];

   promptMessage = sprintf('Do you want to save the coin values into
csv files?');
       button = questdlg(promptMessage, 'Save coin
values?', 'Yes', 'No', 'Yes');
       if strcmp(button, 'Yes')
           csvwrite('coinYXCoordinates.csv',WorldCoordinateMat);
           csvwrite('coinVals.csv',coinMat);
       end
```

Published with MATLAB® R2019b

Table of Contents

Function Header

```
%************************************************************************
%Function: CaptureandMeasureAVI
%Parameters: aviFile - desired avi file with extension.
%numberOfFrames - desired number of frames to be processed/saved.
%aviPath - optional parameter needed if avi file is not in current
 dir.
%Outputs: Can save .jpg images of each frame of the avi file, as well
 as
%returning image data on the desired frames.
%Info:% This function extracts frames and get frame means from an avi
 movie
% and save individual frames to separate image files.
% Also computes the mean gray value of the color channels.
% I used a lot of open source code for this function written by "Image
% Anaylst"
% Updated by Stephen More for Matlab2019b
%************************************************************************
function CaptureandMeasureAVI(aviFile,numberOfFrames, aviPath, res)

    % HD = '1280x720';
    % LD = '640x480';
    fontSize = 14;
    % Change the current folder to the folder of this m-file.
    % (The line of code below is from Brett Shoelson of The
 Mathworks.)
    if(~isdeployed)
      mfile_name = mfilename('fullpath');
    [pathstr] = fileparts(mfile_name);
    cd(pathstr);
    end
    % Open the rhino.avi demo movie
    if(exist('aviPath','var'))
        folder = pwd;
    else
        %folder = aviPath;
        folder = 'C:\Users\Stephen More\Documents\MATLAB\Jr Design FP
\filesAVI';
    end
    movieFullFileName = fullfile(folder, aviFile);
```

```
   % Check to see that it exists.
   if ~exist(movieFullFileName, 'file')
     strErrorMessage = sprintf('File not found:\n%s\nYou can choose a
new one, cancel, or change to demo video', movieFullFileName);
     response = questdlg(strErrorMessage, 'File not found', 'OK -
choose a new movie.','Switch to rhino.avi demo video?', 'Cancel', 'OK
- choose a new movie.');
     if strcmpi(response, 'OK - choose a new movie.')
       [baseFileName, folderName, FilterIndex] = uigetfile('*.avi');
       if ~isequal(baseFileName, 0)
         movieFullFileName = fullfile(folderName, baseFileName);
       else
         return;
       end
     elseif strcmpi(response, 'Switch to rhino.avi demo video?')
       folder = fileparts(which('cameraman.tif'));
       movieFullFileName = fullfile(folder, 'rhinos.avi');
     else
       return;
     end
   end
```

TRY

```
   try

     %movieInfo = aviinfo(movieFullFileName)
     %mov = aviread(movieFullFileName);
     mov = VideoReader(movieFullFileName);
     % movie(mov);
     % Determine how many frames there are.
     if (numberOfFrames <= mov.NumFrames)
       %numberOfFrames = mov.NumFrames;
       numberOfFramesWritten = 0;
     else
         numberOfFrames = mov.NumFrames;
     end
     % Prepare a figure to show the images in the upper half of the
screen.
     figure;
     screenSize = get(0, 'ScreenSize');
     newWindowPosition = [1 screenSize(4)/2 - 70 screenSize(3)
screenSize(4)/2];
     set(gcf, 'Position', newWindowPosition); % Maximize figure.
```

User Input

Ask user if they want to write the individual frames out to disk.

```
        promptMessage = sprintf('Do you want to save the individual
frames out to individual disk files?');
        button = questdlg(promptMessage, 'Save individual
frames?', 'Yes', 'No', 'Yes');
        if strcmp(button, 'Yes')
            writeToDisk = true;
              % Extract out the various parts of the filename.
              [folder, baseFileName, extentions] =
fileparts(movieFullFileName);
              % Make up a special new output subfolder for all the
separate
              % movie frames that we're going to extract and save to
disk.
              % (Don't worry - windows can handle forward slashes in
the folder name.)
              folder = 'C:\Users\Stephen More\Documents\MATLAB\Jr
Design FP';   % Make it a subfolder of the folder where this m-file
lives.
              outputFolder = sprintf('%s/Movie Frames from %s',
folder, baseFileName);
              % Create the folder if it doesn't exist already.
              if ~exist(outputFolder, 'dir')
                  mkdir(outputFolder);
              end
          else
              writeToDisk = false;
        end
```

Gather Frame Data

Loop through the movie, writing all frames out. Each frame will be in a separate file with unique name.

```
        meanGrayLevels = zeros(numberOfFrames, 1);
        meanRedLevels = zeros(numberOfFrames, 1);
        meanGreenLevels = zeros(numberOfFrames, 1);
        meanBlueLevels = zeros(numberOfFrames, 1);
          for frame = 1 : numberOfFrames
          % Extract the frame from the movie structure.
```

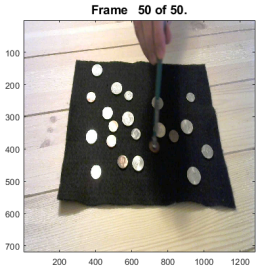
```
          %thisFrame = mov(frame).cdata;
          thisFrame = readFrame(mov);
              % Display it
              hImage = subplot(1,2,1);
              image(thisFrame);
              axis square;
              caption = sprintf('Frame %4d of %d.', frame,
numberOfFrames);
              title(caption, 'FontSize', fontSize);
              drawnow; % Force it to refresh the window.
        % Write the image array to the output file, if requested.
            if writeToDisk
                 % Construct an output image file name.
                outputBaseFileName = sprintf('Frame %4.4d.jpg',
frame);
                outputFullFileName = fullfile(outputFolder,
outputBaseFileName);
                  % Stamp the name and frame number onto the image.
                  % At this point it's just going into the overlay,
                  % not actually getting written into the pixel
values.
                  text(5, 15, outputBaseFileName, 'FontSize', 20);
                  % Extract the image with the text "burned into" it.
                  frameWithText = getframe(gca);
                  % frameWithText.cdata is the image with the text
                  % actually written into the pixel values.
                  % Write it out to disk.
                  if strcmp(res,'LD')
                    %imwrite(frameWithText.cdata, outputFullFileName,
'jpeg','Resolution',[640 480]);
                    imwrite(frameWithText.cdata,
outputFullFileName, 'jpg');
                  else
                    imwrite(frameWithText.cdata,
outputFullFileName, 'jpg');
                  end
            end
        % Calculate the mean gray level.
        grayImage = rgb2gray(thisFrame);
        meanGrayLevels(frame) = mean(grayImage(:));
        % Calculate the mean R, G, and B levels.
        meanRedLevels(frame) = mean(mean(thisFrame(:, :, 1)));
        meanGreenLevels(frame) = mean(mean(thisFrame(:, :, 2)));
        meanBlueLevels(frame) = mean(mean(thisFrame(:, :, 3)));
        % Plot the mean gray levels.
        hPlot = subplot(1,2,2);
            hold off;
        plot(meanGrayLevels, 'k-', 'LineWidth', 2);
            hold on;
            plot(meanRedLevels, 'r-');
            plot(meanGreenLevels, 'g-');
            plot(meanBlueLevels, 'b-');
              % Put title back because plot() erases the existing
title.
```

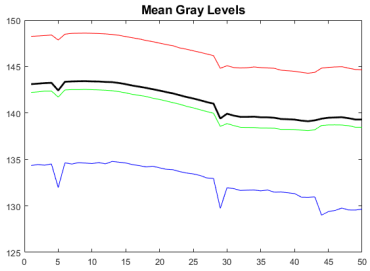
```
        title('Mean Gray Levels', 'FontSize', fontSize);
            if frame == 1
                xlabel('Frame Number');
                ylabel('Gray Level');
                % Get size data later for preallocation if we read
                % the movie back in from disk.
                [rows,columns,numberOfColorChannels] =
size(thisFrame);
            end
            % Update user with the progress.  Display in the command
window.
            if writeToDisk
            progressIndication = sprintf('Wrote frame %4d of %d.',
frame, numberOfFrames);
            else
            progressIndication = sprintf('Processed frame %4d of
%d.', frame, numberOfFrames);
            end
        disp(progressIndication);
        % Increment frame count (should eventually = numberOfFrames
        % unless an error happens).
        numberOfFramesWritten = numberOfFramesWritten + 1;
        end
        % Alert user that we're done.
        if writeToDisk
            finishedMessage = sprintf('Done!  It wrote %d frames to
folder\n"%s"', numberOfFramesWritten, outputFolder);
        else
            finishedMessage = sprintf('Done!  It processed %d frames
of\n"%s"', numberOfFramesWritten, movieFullFileName);
        end
        disp(finishedMessage); % Write to command window.
        uiwait(msgbox(finishedMessage)); % Also pop up a message
box.
        % Exit if they didn't write any individual frames out to
disk.
        if ~writeToDisk
            return;
        end
        % Ask user if they want to read the individual frames from
the disk,
        % that they just wrote out, back into a movie and display
it.
        promptMessage = sprintf('Do you want to recall the
individual frames\nback from disk into a movie?\n(This will take
several seconds.)');
        button = questdlg(promptMessage, 'Recall
Movie?', 'Yes', 'No', 'Yes');
        if strcmp(button, 'No')
            return;
        end
        % Read the frames back in, and convert them to a movie.
        % I don't know of any way to preallocate recalledMovie.
        for frame = 1 : numberOfFrames
```

```
            % Construct an output image file name.
            outputBaseFileName = sprintf('Frame %4.4d.png', frame);
            outputFullFileName = fullfile(outputFolder,
outputBaseFileName);
            % Read the image in from disk.
            thisFrame = imread(outputFullFileName);
            % Convert the image into a "movie frame" structure.
            recalledMovie(frame) = im2frame(thisFrame);
        end
        % Get rid of old image and plot.
        delete(hImage);
        delete(hPlot);
        % Create new axes for our movie.
        subPlot(1, 3, 2);
        axis off;  % Turn off axes numbers.
        title('Movie recalled from disk', 'FontSize', fontSize);
        % Play the movie in the axes.
        movie(recalledMovie);
        % Note: if you want to display graphics or text in the
overlay
        % as the movie plays back then you need to do it like I did
at first
        % (at the top of this file where you extract and imshow a
frame at a time.)
        msgbox('Done!');
```

```
Processed frame    1 of 50.
Processed frame    2 of 50.
Processed frame    3 of 50.
Processed frame    4 of 50.
Processed frame    5 of 50.
Processed frame    6 of 50.
Processed frame    7 of 50.
Processed frame    8 of 50.
Processed frame    9 of 50.
Processed frame   10 of 50.
Processed frame   11 of 50.
Processed frame   12 of 50.
Processed frame   13 of 50.
Processed frame   14 of 50.
Processed frame   15 of 50.
Processed frame   16 of 50.
Processed frame   17 of 50.
Processed frame   18 of 50.
Processed frame   19 of 50.
Processed frame   20 of 50.
Processed frame   21 of 50.
Processed frame   22 of 50.
Processed frame   23 of 50.
Processed frame   24 of 50.
Processed frame   25 of 50.
Processed frame   26 of 50.
Processed frame   27 of 50.
Processed frame   28 of 50.
```

```
Processed frame   29 of 50.
Processed frame   30 of 50.
Processed frame   31 of 50.
Processed frame   32 of 50.
Processed frame   33 of 50.
Processed frame   34 of 50.
Processed frame   35 of 50.
Processed frame   36 of 50.
Processed frame   37 of 50.
Processed frame   38 of 50.
Processed frame   39 of 50.
Processed frame   40 of 50.
Processed frame   41 of 50.
Processed frame   42 of 50.
Processed frame   43 of 50.
Processed frame   44 of 50.
Processed frame   45 of 50.
Processed frame   46 of 50.
Processed frame   47 of 50.
Processed frame   48 of 50.
Processed frame   49 of 50.
Processed frame   50 of 50.
Done!  It processed 50 frames of
"C:\Users\Stephen More\Documents\MATLAB\Jr Design FP\filesAVI
\c270Capture0003.avi"
```





Catch Exceptions

```
    catch ME
      % Some error happened if you get here.
      stError = lasterror;
      strErrorMessage = sprintf('Error extracting movie frames from:\n
\n%s\n\nError: %s\n\n)', movieFullFileName, stError.message);
      uiwait(msgbox(strErrorMessage));
    end
    return;

end
```

Published with MATLAB® R2019b

Table of Contents

Function Header

```
%#codegen
%*************************************************************************
%Function: objectDetection
%Parameters: None
%Outputs: Performs object detection on the jetson nano after codgen
%function is run.
function objectDetection()

%defines the hwobj as well as the camObj and dispObj, these only need
 to be
%done once.
hwobj = jetson;
camObj = camera(hwobj,"UVC Camera (046d:0825)",[640 480]);
dispObj = imageDisplay(hwobj);
%Define constants for the function as well as loading in compile time
%constants
threshR = 0.3;
%test = coder.load('test.mat','test');
%open the g code file which we will be writing commands to


%FIND A FUNCTION THAT IS EQUIVILANT TO EXIST THAT CAN BE GENERATED
%if ~exist('straight_line.g','file')
            %w+ enables read/write authority and makes a new file if
 one
            %doesnt exist.
%         fileID = fopen('gcodeTest.g','w+','UTF-8');
%else
%            fileID = fopen('gcodeTest.g','r+','n','UTF-8');
%end
```

Live Image Processing

```
%This loop controls the image processing of each frame.
for k = 1:500

    % Capture the image from the camera on hardware.
    img = snapshot(camObj);
```

```
   frameBEV = img;
   %frameBEV = transformImage(birdsEye,img);

   %Extract RGB matricies from frame
   frameR = frameBEV(:,:,1);
   frameG = frameBEV(:,:,2);
   frameB = frameBEV(:,:,3);

   %Convert RGB mats to BW images
   f1 = imbinarize(frameR,threshR);
   f2 = imbinarize(frameG,threshR);
   f3 = imbinarize(frameB,threshR);

   %Sum of all BW images
   frameSum = (f1&f2&f3);
   %Removes all patches of pixels less than 150 W
   frameSumCleaned = bwareaopen(frameSum, 150);

 %*********************************************************************
```

Perform Morphological Operations

```
   [labeled, frameNumObjects] = bwlabel(frameSumCleaned,4);
   %frameStats =
struct('Eccentricity',zeros(frameNumObjects,1),'Area',zeros(frameNumObjects,1),'B
'Centroid',zeros(frameNumObjects,1), 'MajorAxisLength',
zeros(frameNumObjects,1),'MinorAxisLength',zeros(frameNumObjects,1));
   frameStats =
regionprops(labeled,'Eccentricity', 'Area', 'BoundingBox', 'Centroid', 'MajorAxis
   %preallocate memory for object data arrays
   frameEccentricities = zeros(frameNumObjects,1);
   %coder.varsize bounds the variables
   coder.varsize('frameRectangleValues',[300 4]);
   coder.varsize('frameCenters', [300 2]);
   coder.varsize('radii',[300,1]);
   %coder.varsize('frameRect', [480 640 3]);
   for j = 1:frameNumObjects
       frameEccentricities(j,1) = [frameStats(j).Eccentricity];
   end
   %Finds the index of Non zero eccentricities
   idxOfObjects = find(frameEccentricities);
   frameNonZeroStats = frameStats(idxOfObjects);
   %*********************************************
```

Object Size Filtering

```
    frameRectangleValues = zeros(length(idxOfObjects),4);
    frameCenters = zeros(length(idxOfObjects),2);
    idxOfSizeBoundedObjects = zeros(length(idxOfObjects),1);
    frameQuarterBoxes = zeros(length(idxOfObjects),4);
    frameDimeBoxes = zeros(length(idxOfObjects),4);
       threshSizeInPixelsHi = 60;
       threshSizeInPixelsLo = 20;
```

```
       %Filter height and width values by the threshold height/width
in pixels
       for idx = 1 : length(idxOfObjects)
           if( ((frameNonZeroStats(idx).BoundingBox(3) <
threshSizeInPixelsHi) && (frameNonZeroStats(idx).BoundingBox(4) <
threshSizeInPixelsHi)) && (frameNonZeroStats(idx).BoundingBox(3) >
threshSizeInPixelsLo) && (frameNonZeroStats(idx).BoundingBox(4) >
threshSizeInPixelsLo) )
               frameRectangleValues(idx,:) =
frameNonZeroStats(idx).BoundingBox;
               frameCenters(idx,:) = frameNonZeroStats(idx).Centroid;
               idxOfSizeBoundedObjects(idx) = idx;
           else
               %Assign the value of -1 to all rows that need to be
deleted
               frameRectangleValues(idx,:) = -1;
               frameCenters(idx,:) = -1;
               idxOfSizeBoundedObjects(idx) = -1;
           end
       end
       %Create a logical array
       idx = (frameRectangleValues == -1);
       idx2 = (frameCenters == -1);
       idx3 = (idxOfSizeBoundedObjects == -1);
       %use "find" to find where logical index is 1
       [delRows, ~] = find(idx);
       [delRows2, ~] = find(idx2);
       [delRows3, ~] = find(idx3);
       %Delete Rows

       frameRectangleValues(delRows,:) = [];
       frameCenters(delRows2,:) = [];
       idxOfSizeBoundedObjects(delRows3) = [];

       numObjects = length(frameRectangleValues);
       frameBoundedStats =
frameNonZeroStats(idxOfSizeBoundedObjects);

 %**********************************************************************
   %frameRectangleValues and meanR go from left to right of the image
   meanR = zeros(numObjects,1);
   radii = zeros(numObjects,1);
   %centers = zeros(numObjects,2);
   %radii = zeros(numObjects);
   for idx = 1:numObjects
       %This section finds pennies by cropping
       frameCropped =
imcrop(frameBEV,frameRectangleValues(idx:idx,1:4));
       frameCroppedR = frameCropped(:,:,1);
       meanR(idx) = mean(mean(frameCroppedR));
       radii(idx) = (length(frameCropped)/2);

       %Write x y centers of the coins to straight_line.g
       %Need to perform pixel to real world distance calculation
```

```
    end


    %mask = zeros(size(img));
    %circleROI = roipoly
    %bwActiveContour = activecontour(img,

 %*********************************************************************
    frameSumCleaned = frameSumCleaned*255;
    %FrameSumCleaned is the binary white image with all objects of
size
    %less than 150 pixels removed
    frameSumCleaned = uint8(frameSumCleaned);
    %Inserts bounding boxes into the image
```

Map Shapes Onto Image

```
    frameRect = insertMarker(frameBEV, frameCenters, 'Color', [255 0
0]);
    %frameRect = insertShape(frameRect, 'Rectangle',
frameRectangleValues, 'Color', [0 255 0]);
    frameRect = insertShape(frameRect, 'FilledCircle', [frameCenters,
radii], 'Color', [0 0 255], 'Opacity', 0.7);
    %*****************************************************



    %*****************************************************
    %Display BW image.
    %image(dispObj,frameSumCleaned);

    %Display RGB image
    %image(dispObj,img);

    %Display BoundingBoxes on RGB image
    image(dispObj,frameRect);

end

end
```

Published with MATLAB® R2019b

Function Header

```
%*************************************************************************
%Function: sobelEdgeDetection.m
%Parameters: None
%Outputs: Performs edge detection on the jetson nano after codgen
%function is run.
function sobelEdgeDetection() %#codegen



hwobj = jetson;
camObj = camera(hwobj,"UVC Camera (046d:0825)",[640 480]);
dispObj = imageDisplay(hwobj);

% Sobel kernel
kern = [1 2 1; 0 0 0; -1 -2 -1];

% Main loop
for k = 1:500
    % Capture the image from the camera on hardware.
    img = snapshot(camObj);

    % Finding horizontal and vertical gradients.
    h = conv2(img(:,:,2),kern,'same');
    v = conv2(img(:,:,2),kern','same');

    % Finding magnitude of the gradients.
    e = sqrt(h.*h + v.*v);

    % Threshold the edges
    edgeImg = uint8((e > 100) * 240);

    % Display image.
    image(dispObj,edgeImg);
    % imwrite(edgeImg,'test.png');
end

end
```


Published with MATLAB® R2019b

Function Header

```
% Description: This is a test bench function to connect to, interact
 with, and generate C++/Cuda Code for Nvidia
% Jetson Nano Board
% Parameters: sudoFlag - if you want to sign in as root enter 1
% usbFlag - if you are sshing over usb c enter 1
% codegenFlag - if you want code to be generated enter 1
% matlabFileName - enter the name of the matlab file you wish to
 generate
% code for.
```

Test Bench

```
function GPUCoderTestBench(sudoFlag, usbFlag, codegenFlag,
 matlabFileName)
if (sudoFlag == 1)
    hwobj= jetson('10.0.0.207','root','root');
else
    if (usbFlag == 1)
        hwobj= jetson('192.168.55.1','nvidia','nvidia');
    else
        hwobj= jetson('10.0.0.207','nvidia','nvidia');
    end
end
%Generation of CUDA Code using GPU Coder --------
%Create the config object for generating .exe
cfg = coder.gpuConfig('exe');
%Assign Hardware to config
cfg.Hardware = coder.hardware('NVIDIA Jetson');
%Specify Build Directory
cfg.Hardware.BuildDir = '~/remoteBuildDir';
%Generate example C++ code
cfg.GenerateExampleMain = 'GenerateCodeAndCompile';
if (codegenFlag == 1)
    codegen('-config ',cfg, matlabFileName,'-report');
end
```

Published with MATLAB® R2019b