Project 2

Due: 10/25/23

Team Member Names: George Trupiano Andrew McGhee
Matthew Monroe

Instructions:

1. Fill in your name and attach this page as a cover sheet for your report.
2. This is a group project. Work with your group as assigned in the "ECE4510-F23 - Project Groups" document. List the contribution from each team member to this report, on the following page. Submit a single report for the group.
3. Attach all supporting material with the report.
4. Type / handwrite and sign the following honor pledge: "I pledge on my honor that I have not given or received any unauthorized assistance on this assignment."

Honor Pledge:

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment.

Signature

Matthew Monroe

Grading Information:

Task 1: MATLAB Image Analysis (40 Points)
a) Blob detection and boundary plotting (10 Points)
b) Find the different object types (20 Points)
c) Find the brightest washer in the image (10 Points)
Task 2: Edge Detection in OpenCV (30 Points)
Task 3: Keypoint Detection and Matching in OpenCV (30 Points)

Extra credit (25 Points)

Total **/ 100 Points**

Team member contributions to the report:

Team member 1 name: George Trupiano

Contribution: Evenly distributed

Team member 2 name: Andrew [illegible]

Contribution: Evenly distributed

Team member 3 name: Matthew Monroe

Contribution: Evenly distributed

Team member 4 name:

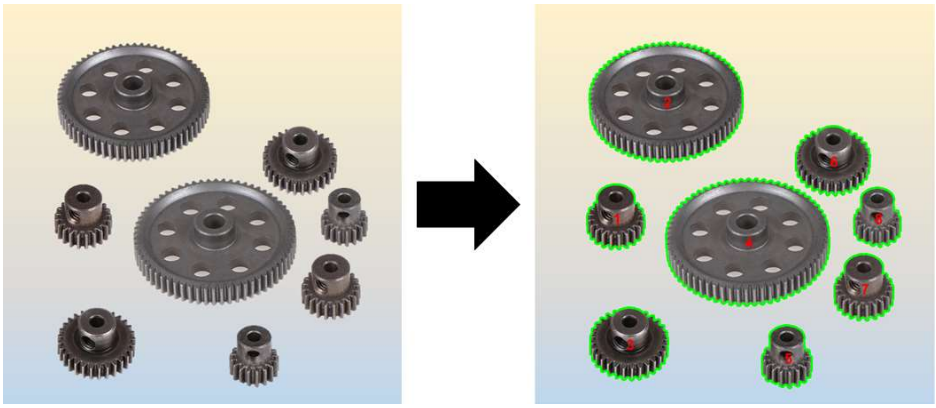
Contribution:

Task 1: MATLAB Image Analysis

a) Blob detection and boundary plotting

In the GearsAndWashers.tif image:

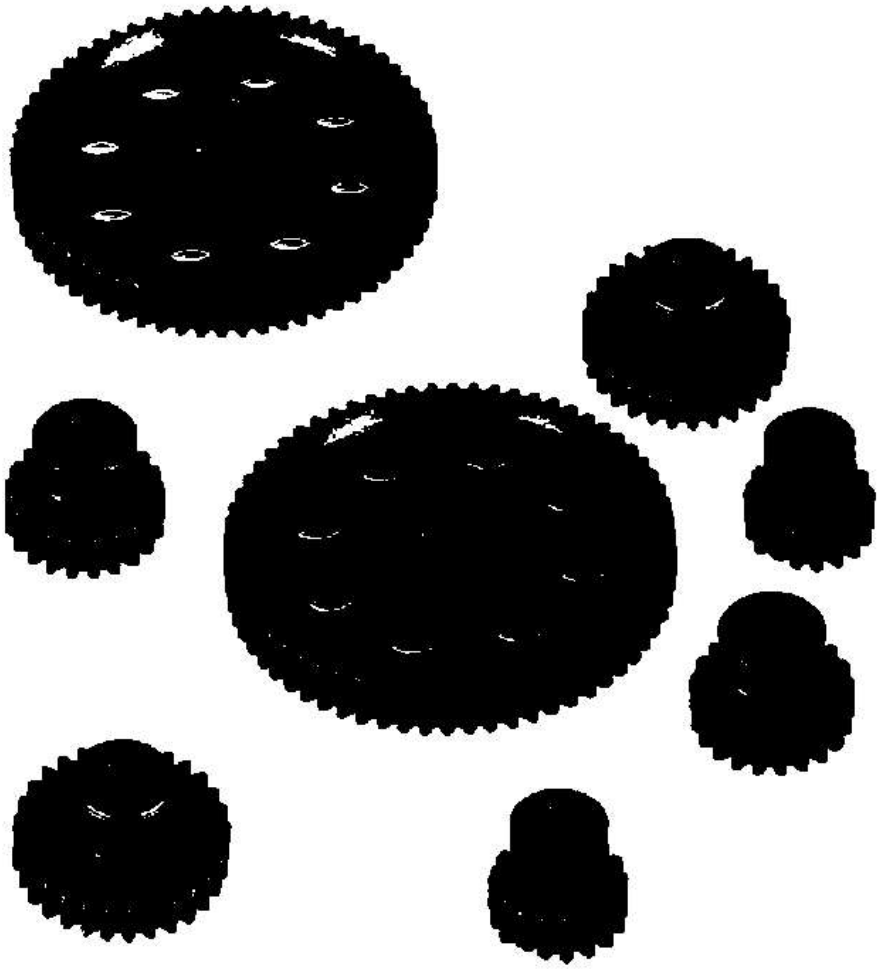
- Put a boundary on each of the object and a number label identifying them.
- Print the properties (mean intensity, area, perimeter, centroid, diameter) of each of the object.



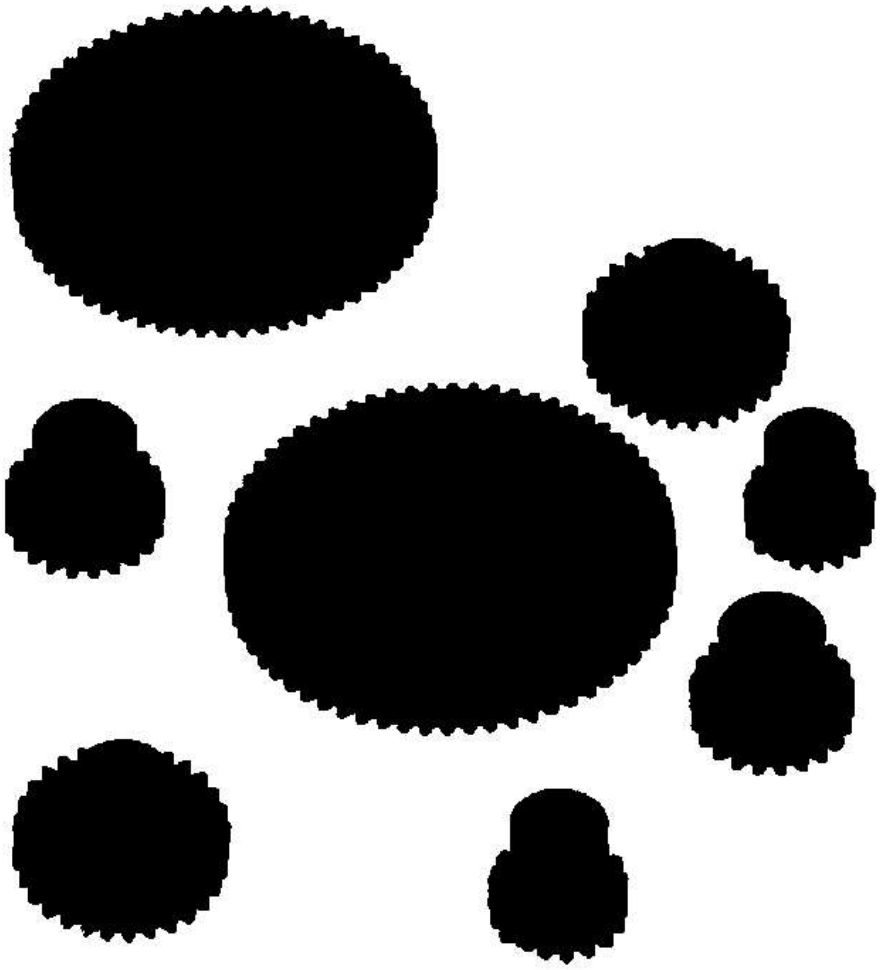
i. Convert the image to a clean binary image with no holes by thresholding and **`imfill`** function.

```
% Read and show the original image.
orig_gears = imread("Gears.tif");
imshow(orig_gears)
```

```
% Use a threshold value to convert from grayscale to a binary image and
% show the resulting image.
threshold = 0.8;
grey_gears = im2bw(orig_gears, threshold);
imshow(grey_gears);
```

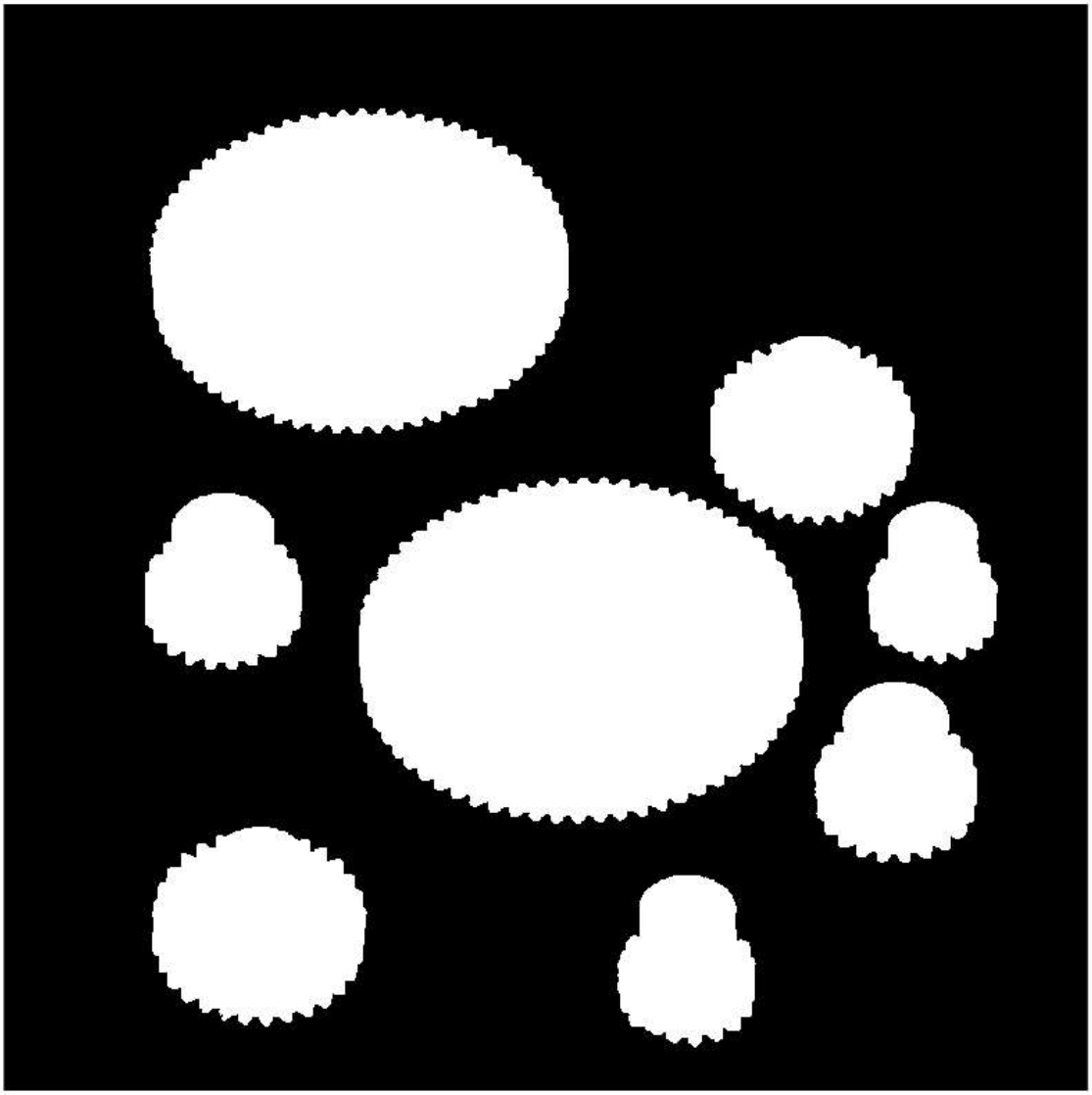


```
% Perform  a "hole fill" to get rid of holes in the gears and any background pixels
% inside the blobs and show the resulting image.
%gear_flipped = imcomplement(grey_gears);
gear_filled = imfill(~grey_gears, 'holes');
%gear_filled = imcomplement(gear_filled_flipped);
imshow(~gear_filled)
```

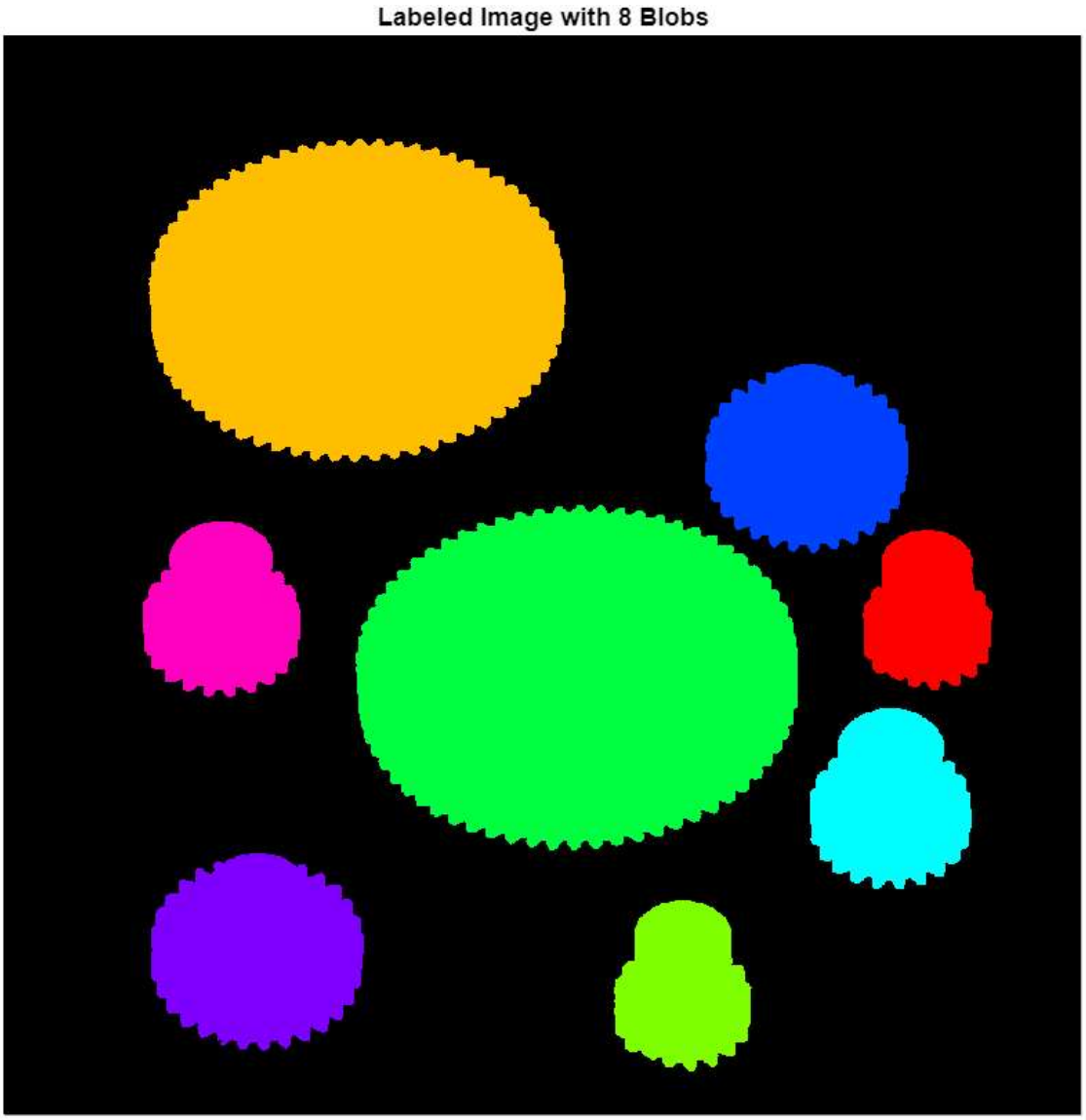


ii. Use the **bwlabel** command to generate a labeled image showing each of the blobs in the image.

```
% Label blobs in the image and show the resulting image.
[labeledImage, numBlobs] = bwlabel(gear_filled);

imshow(labeledImage)
```



```
% Show labels in pseudo random colors.

colormap = rand(numBlobs, 3);
NumGears = label2rgb(labeledImage, 'hsv', 'k', 'shuffle');
imshow(NumGears)
title(['Labeled Image with ', num2str(numBlobs), ' Blobs']);

xlim([1 1025])
ylim([1 1027])
```



iii. Use the **regionprops** command to get the nbwer of blobs found and the properties of each of the blobs.

```
% Get the number and properties of blobs found.

blobregions = regionprops(labeledImage, grey_gears, 'Area', 'Perimeter', 'Centroid', 'EquivDiameter', 'MeanIntensity')
```

blobregions = 8×1 struct

Fields	Area	Centroid	EquivDiameter	Perimeter	MeanIntensity
1	18909	[207.8482,548.8489]	155.1634	578.8780	0.0025
2	95632	[336.8806,252.5023]	348.9448	1.3772e+03	0.0242
3	29222	[241.7648,871.5010]	192.8901	762.6740	0.0059
4	108382	[545.6927,610.4708]	371.4785	1.4649e+03	0.0083
5	16007	[646.0342,905.8617]	142.7611	525.6770	7.4967e-04
6	26491	[763.9205,402.7607]	183.6556	722.2660	0.0031
7	19869	[843.3704,729.0863]	159.0534	595.3860	0.0011
8	14159	[878.1643,549.2070]	134.2676	489.5860	1.4125e-04

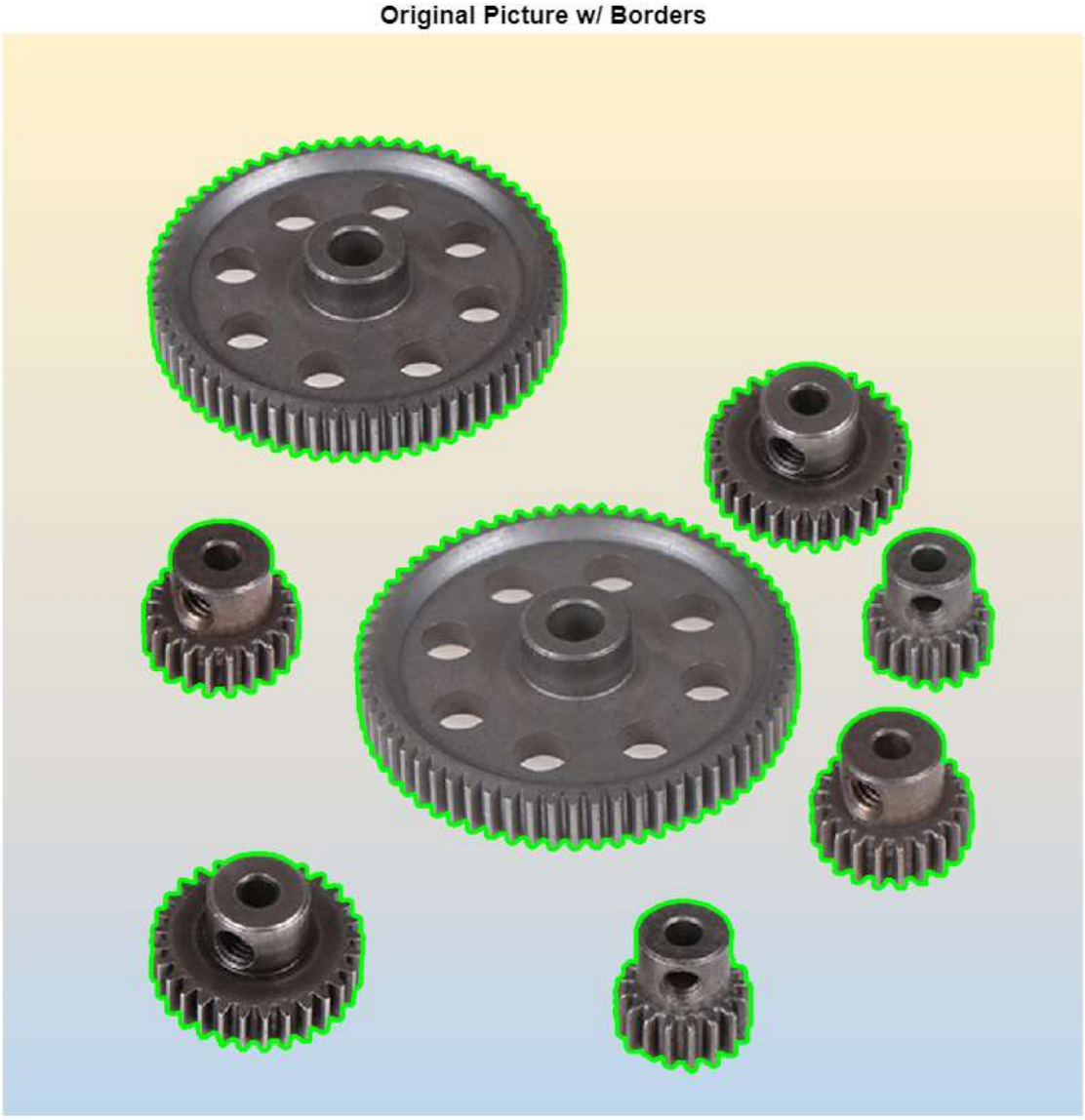
iv. Use the **bwboundaries** command to plot the boundary of each of the blobs found on the original grayscale image to highlight the objects.

```
% bwboundaries returns a cell array where each cell contains the
% row/column coordinates for an object in the image.

gearBoundries = bwboundaries(labeledImage)
```

gearBoundries = 8×1 cell

	1
1	545×2 double
2	1282×2 double
3	716×2 double
4	1364×2 double
5	490×2 double
6	678×2 double
7	556×2 double
8	457×2 double

```
% Plot the borders of all the objects on the original grayscale image
% using the coordinates returned by bwboundaries.

figure;
imshow(orig_gears);
title('Original Picture w/ Borders');

hold on;

for k = 1:length(gearBoundries)
    boundary = gearBoundries{k};
    plot(boundary(:, 2), boundary(:, 1), 'g', 'LineWidth', 2);
end

hold off;
```



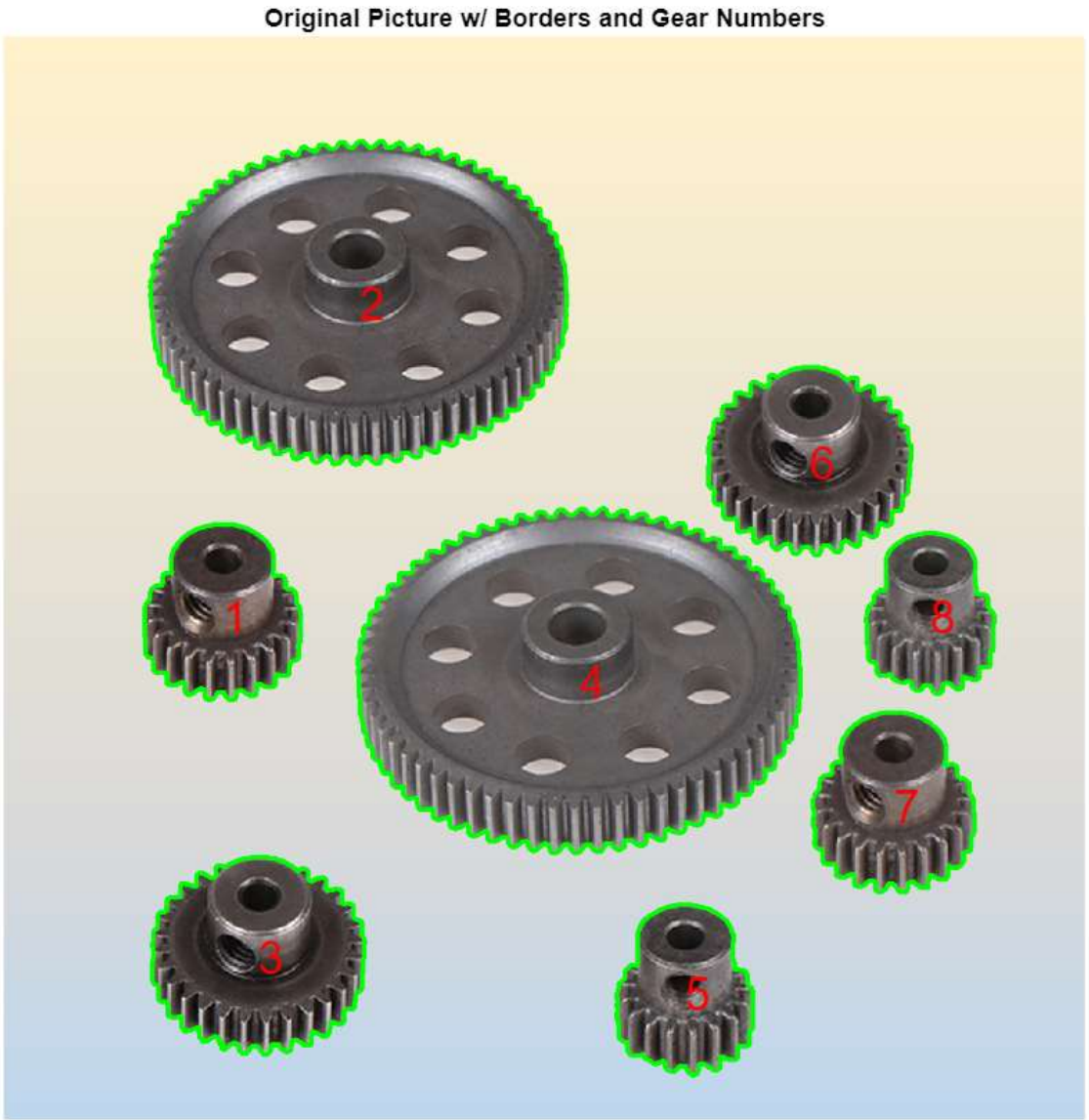
v. Print the blob properties (mean intensity, area, perimeter, centroid, diameter) of each blob found and plot blob number labels on the objects in the outlined grayscale image.

```
% Printing out the properties of each blob

%creating the table to print values to
properties = cell(numel(blobregions), 6);
for i = 1:numel(blobregions)
    properties{i, 1} = i;
    properties{i, 2} = (blobregions(i).Area);
    properties{i, 3} = (blobregions(i).Perimeter);
    properties{i, 4} = blobregions(i).Centroid(1); blobregions(i).Centroid(2);
    properties{i, 5} = (blobregions(i).EquivDiameter);
    properties{i, 6} = blobregions(i).MeanIntensity;
    % Calculating Mean Intensity
end

propTable = cell2table(properties, 'VariableNames', {'Blob Num', 'Area', 'Perimeter', 'Centroid', 'Diameter', 'Mean Intensity'});
disp(propTable)
```

Blob Num	Area	Perimeter	Centroid	Diameter	Mean Intensity
1	18909	578.88	207.85	155.16	0.0025385
2	95632	1377.2	336.88	348.94	0.024166

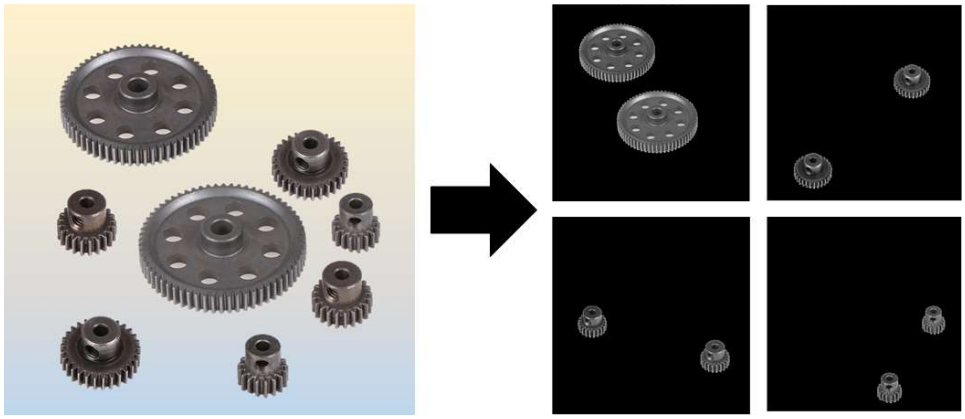
3	29222	762.67	241.76	192.89	0.0058518
4	1.0838e+05	1464.9	545.69	371.48	0.0083409
5	16007	525.68	646.03	142.76	0.00074967
6	26491	722.27	763.92	183.66	0.0030576
7	19869	595.39	843.37	159.05	0.0010569
8	14159	489.59	878.16	134.27	0.00014125

```
% Put the blob number labels on the boundaries grayscale image.
figure;
imshow(orig_gears);
title('Original Picture w/ Borders and Gear Numbers');

hold on;

for k = 1:length(gearBoundries)
    boundary = gearBoundries{k};
    plot(boundary(:, 2), boundary(:, 1), 'g', 'LineWidth', 2);
end

hold on
for i = 1:numel(blobregions)
    text(blobregions(i).Centroid(1), blobregions(i).Centroid(2), num2str(i), 'Color', 'Red', 'FontSize', 20);
end
hold off
```



b) Find the different object types in the image

Identify the different object types (large gears, medium-large gears, medium-small gears, and small gears) in the image using the blob properties obtained in part (a) and extract them in separate images.



i. Check blob properties from part (a) and identify which criteria to use to separate the four different types of the objects.

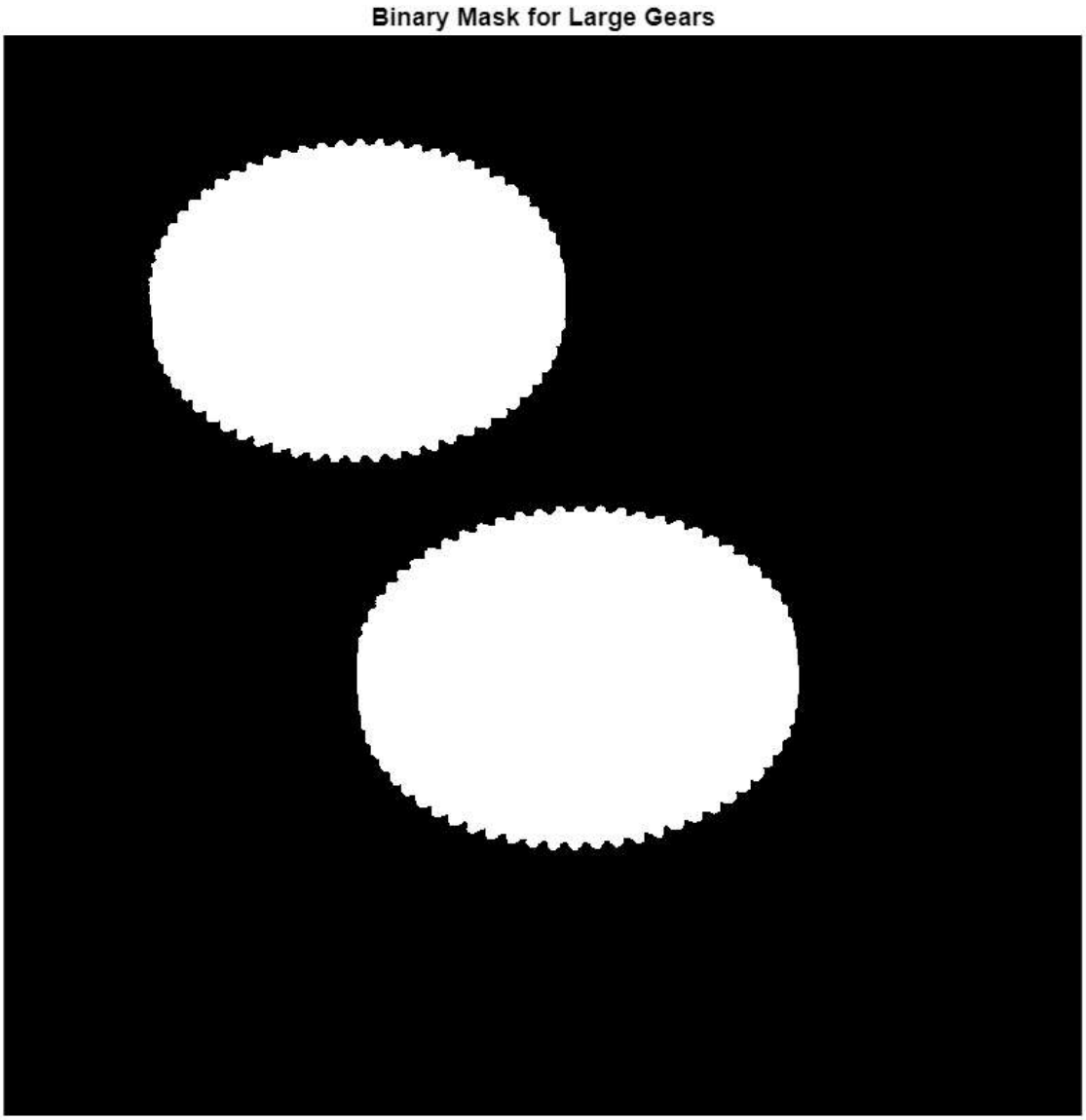
```
% Check criteria of blob properties from part(a)(v) to select the best one to
% separate object types

%Can I just assume to use Area seeing as the image we're using isn't
%changing for now?
```

ii. Use the **ismember** function to extract the selected blobs in the labeled image that was generated in part (a) using the **bwlabel** command. Create a binary mask from the labeled image and filter each type of the object for display in the original grayscale image.

```
% Extract large gears by selecting the blobs in the labeled image that meet
% the criteria and show the binary image mask with separated gears.
blobArea = regionprops(labeledImage, 'Area');

minAreaBig = 95000;
maxAreaBig = 110000;

ValidBlobsBig = [blobArea.Area] >= minAreaBig & [blobArea.Area] <= maxAreaBig;
validLabelsForBig = find(ValidBlobsBig)
```

```
validLabelsForBig = 1×2
     2     4
```

```

binaryMaskBig = ismember(labeledImage, validLabelsForBig);

figure;
imshow(binaryMaskBig);
title('Binary Mask for Large Gears');
```

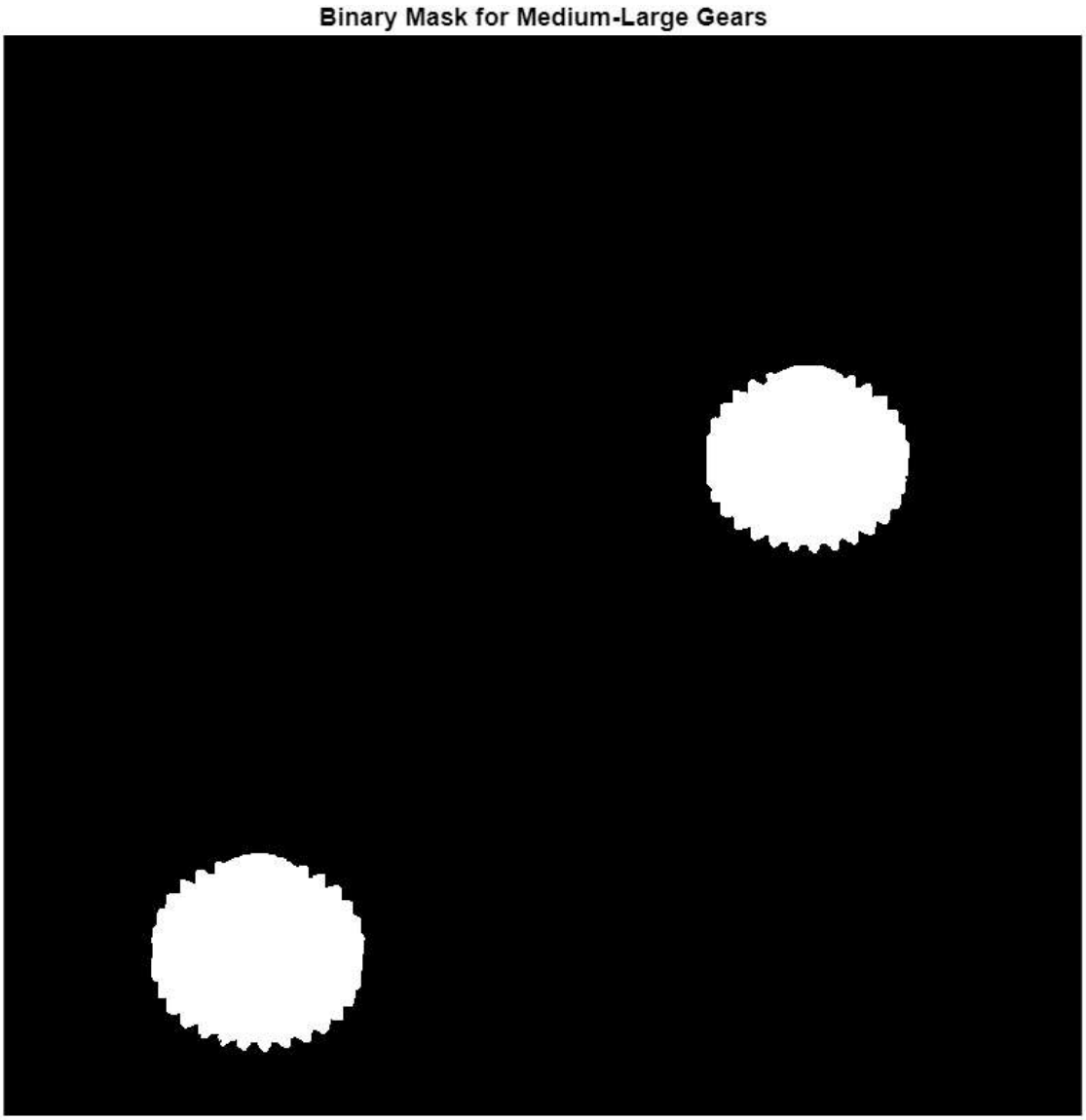



```
% Use the binary image mask to filter out the large gears from the grayscale image
% and show the original image with only the separated gears.

% Big Gears
gearImageBig = orig_gears .* uint8(binaryMaskBig);

figure;
imshow(gearImageBig)
title('Isolated Large Gears');
```

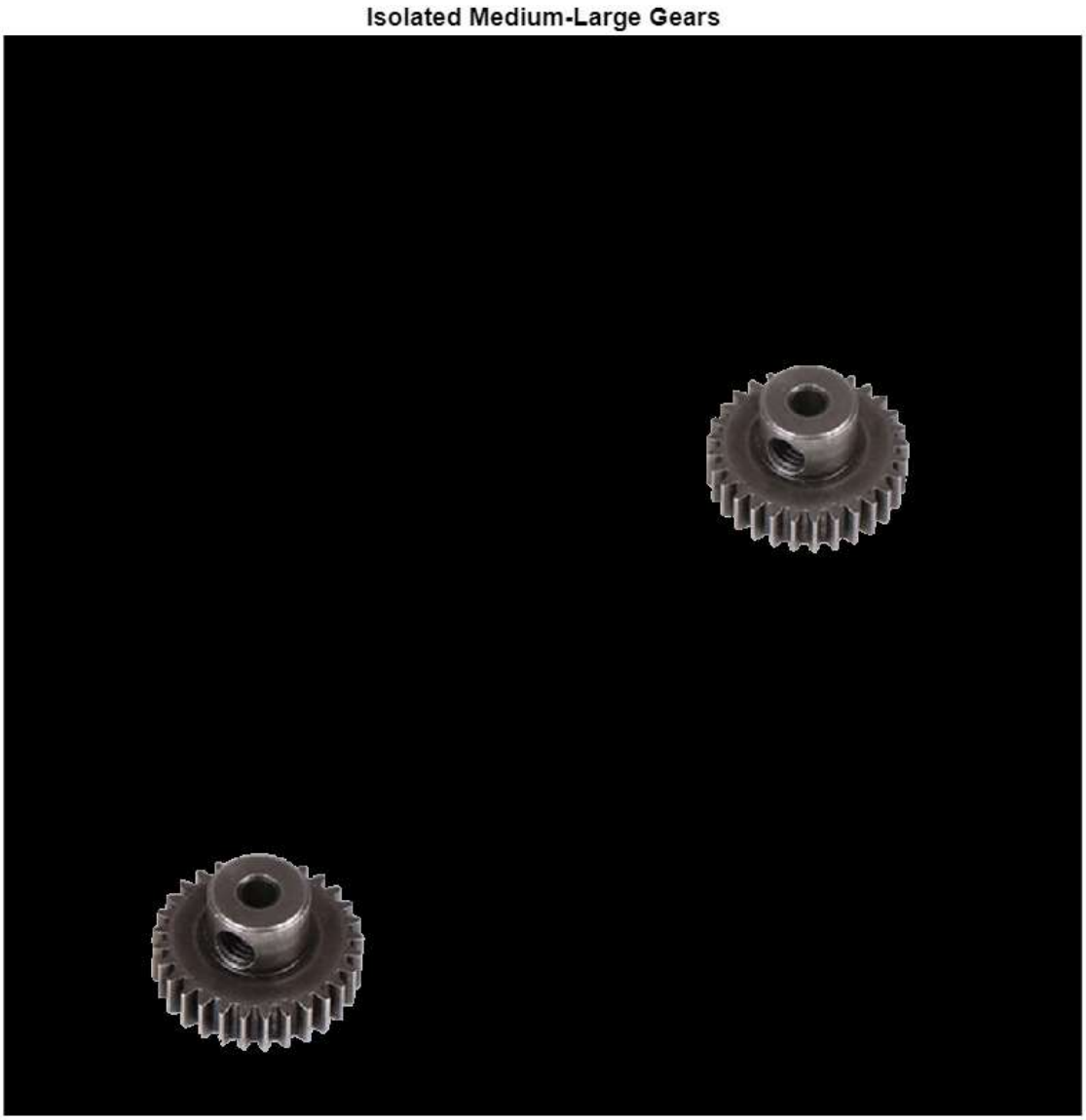

```
% Extract medium-large gears by selecting the blobs in the labeled image that meet
% the criteria and show the binary image mask with separated gears.

%Med-Big Gears

minAreaMedBig = 26000;
maxAreaMedBig = 30000;
ValidBlobsMedBig = [blobArea.Area] >= minAreaMedBig & [blobArea.Area] <= maxAreaMedBig;
validLabelsForMedBig = find(ValidBlobsMedBig)
```

```
validLabelsForMedBig = 1×2
     3     6
```

```
binaryMaskMedBig = ismember(labeledImage, validLabelsForMedBig);

figure;
imshow(binaryMaskMedBig);
title('Binary Mask for Medium-Large Gears');
```



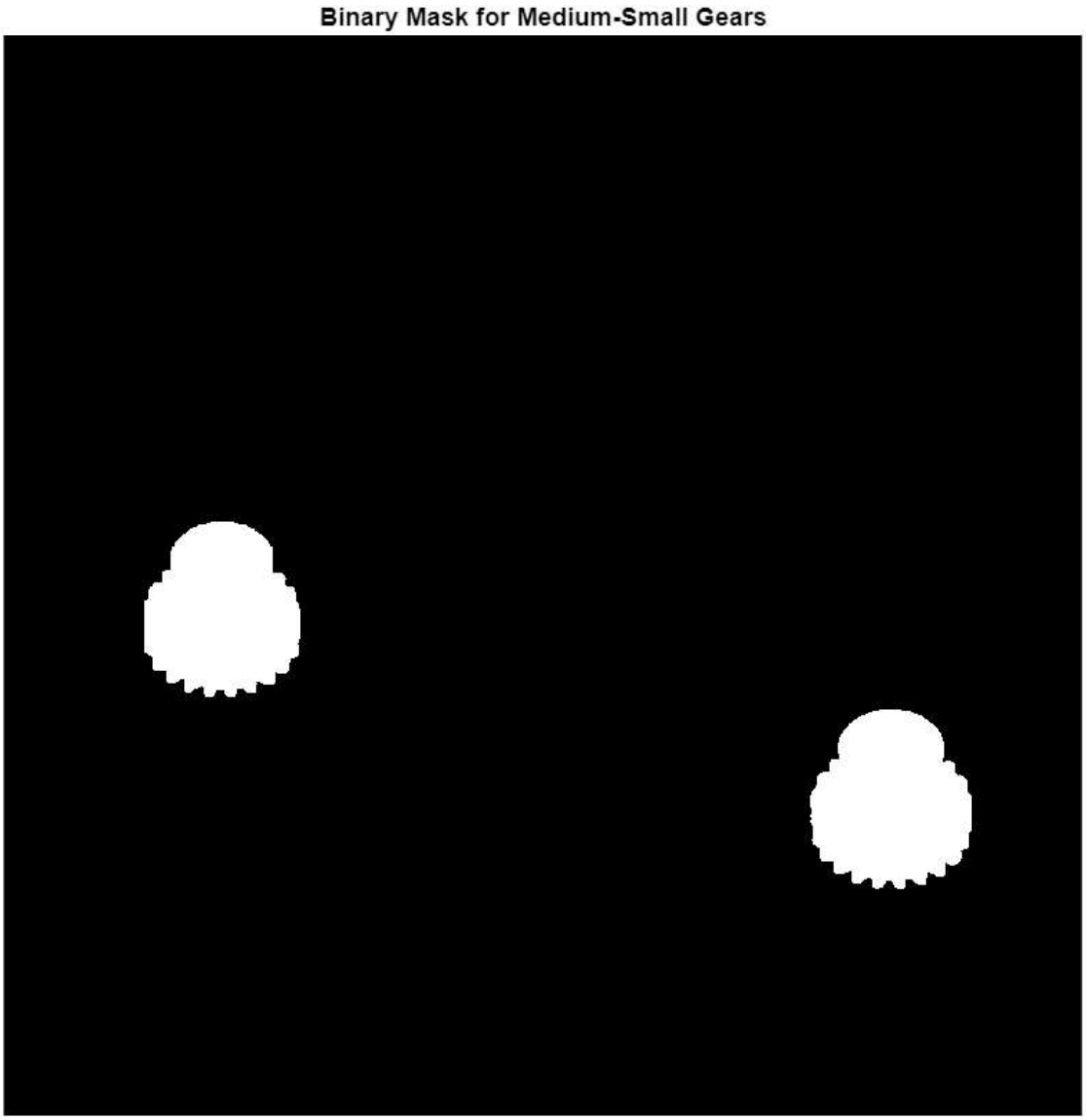

```
% Use the binary image mask to filter out the medium-large gears from the grayscale image
% and show the original image with only the separated gears.

gearImageMedBig = orig_gears .* uint8(binaryMaskMedBig);

figure;
imshow(gearImageMedBig)
title('Isolated Medium-Large Gears');
```



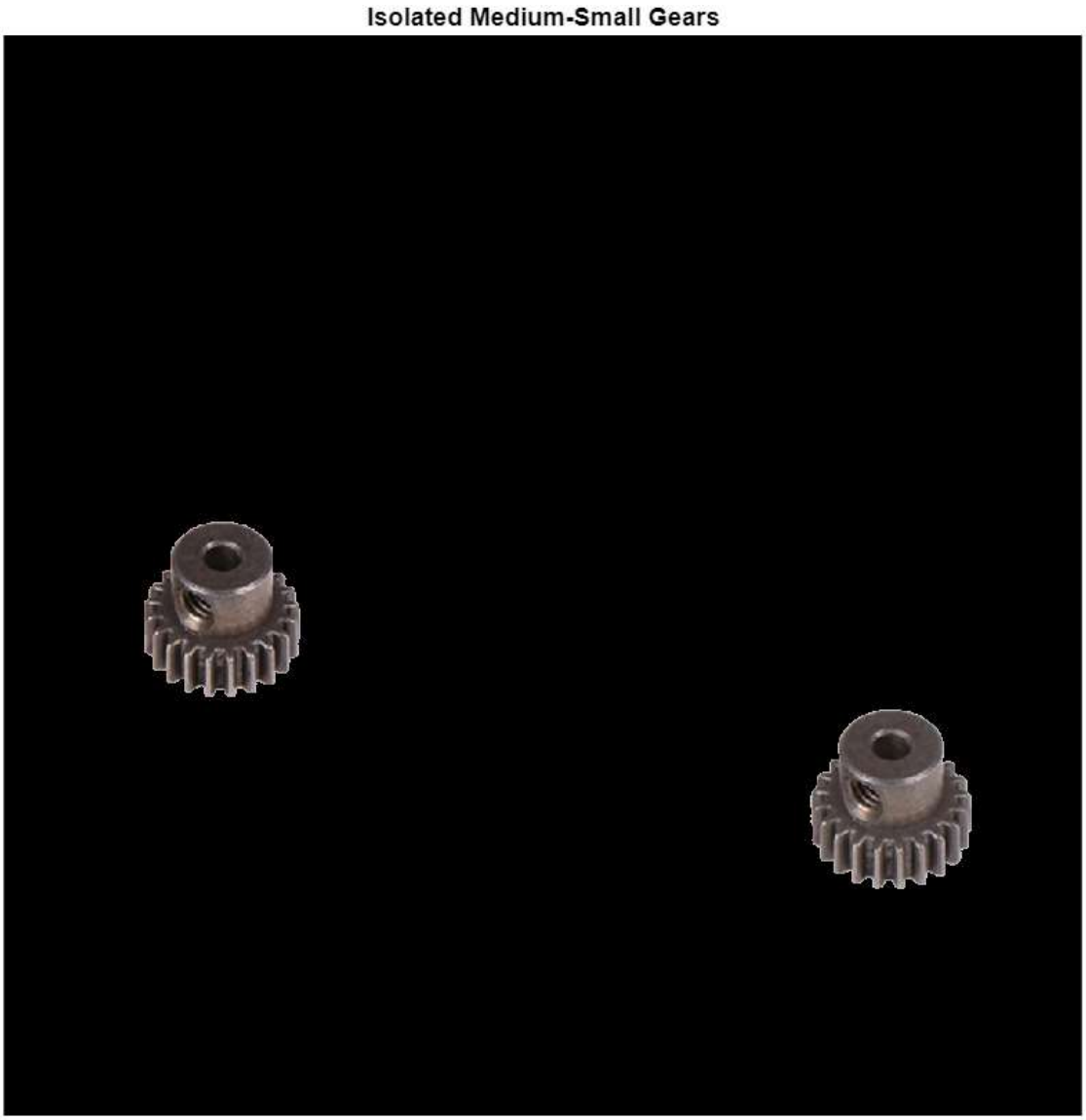

```
% Extract medium-small gears by selecting the blobs in the labeled image that meet
% the criteria and show the binary image mask with separated gears.

% Med-Small Gears

minAreaMedSmall = 18500;
maxAreaMedSmall = 20000;
ValidBlobsMedSmall = [blobArea.Area] >= minAreaMedSmall & [blobArea.Area] <= maxAreaMedSmall;
validLabelsForMedSmall = find(ValidBlobsMedSmall)
```

```
validLabelsForMedSmall = 1×2
     1     7
```

```
binaryMaskMedSmall = ismember(labeledImage, validLabelsForMedSmall);

figure;
imshow(binaryMaskMedSmall);
title('Binary Mask for Medium-Small Gears');
```



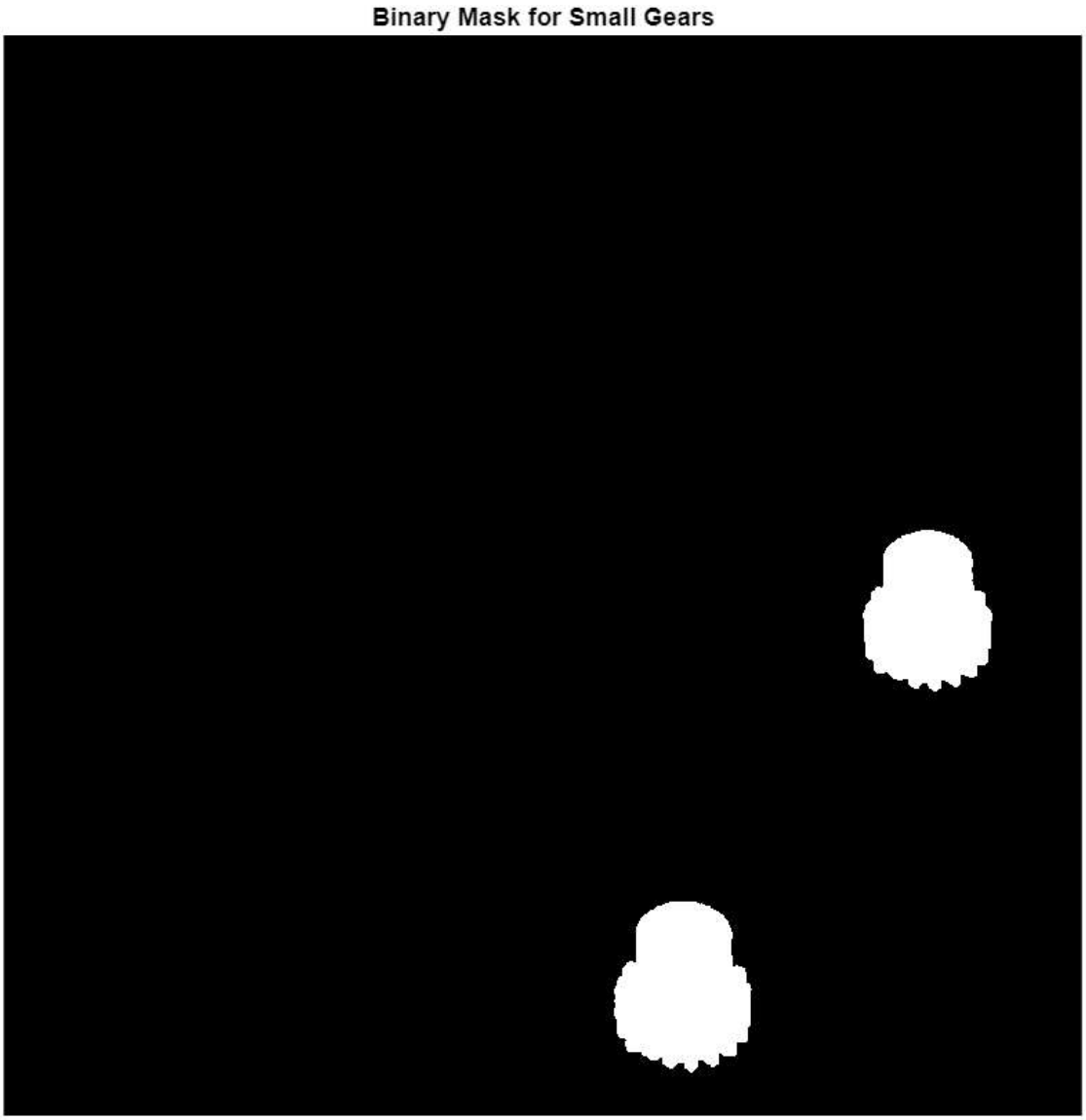

```
% Use the binary image mask to filter out the medium-small gears from the grayscale image
% and show the original image with only the separated gears.

gearImageMedSmall = orig_gears .* uint8(binaryMaskMedSmall);

figure;
imshow(gearImageMedSmall)
title('Isolated Medium-Small Gears');
```



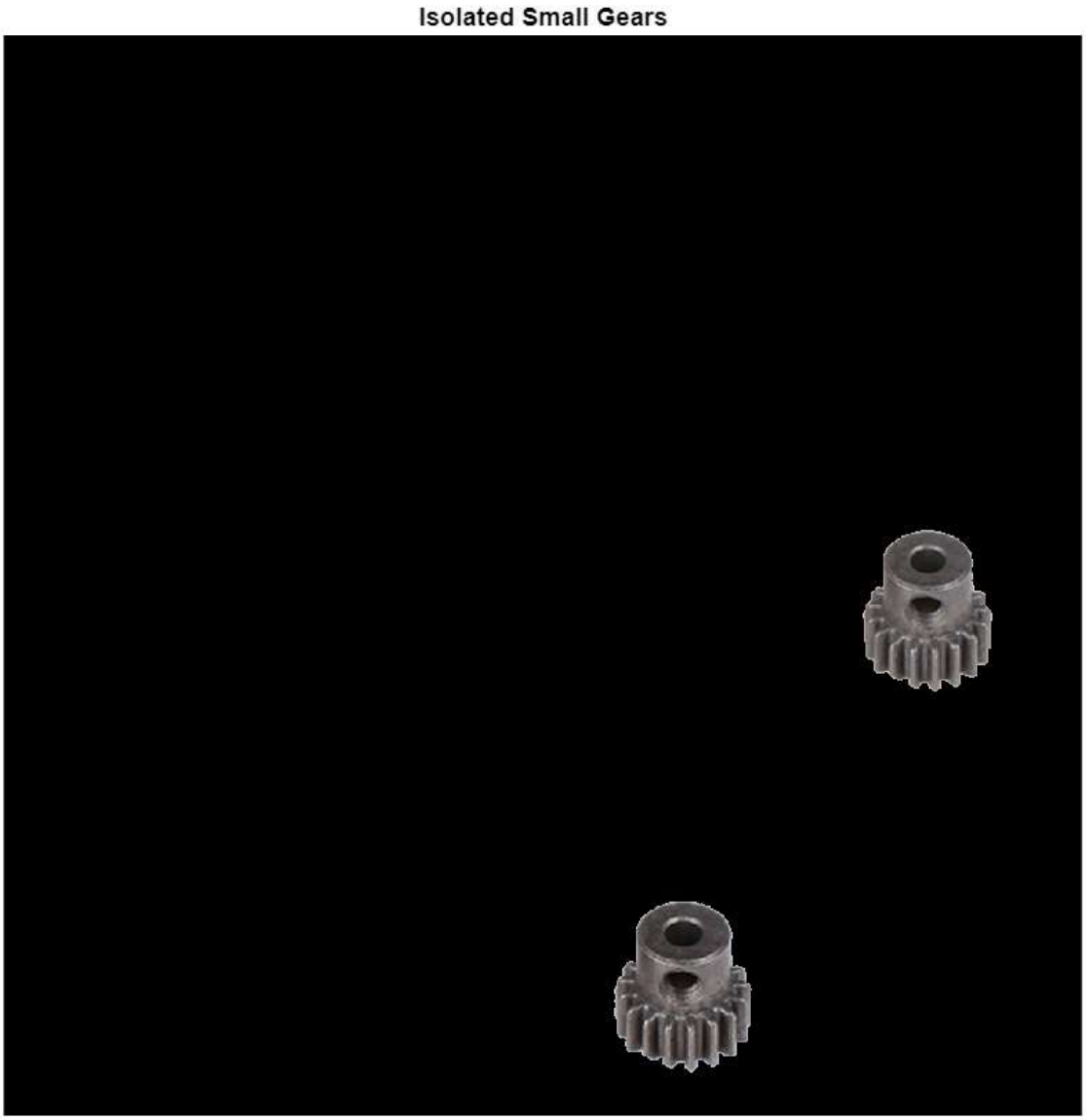

```
% Extract small gears by selecting the blobs in the labeled image that meet
% the criteria and show the binary image mask with separated gears.

% Small Gears

minAreaSmall = 14000;
maxAreaSmall = 16500;
ValidBlobsSmall = [blobArea.Area] >= minAreaSmall & [blobArea.Area] <= maxAreaSmall;
validLabelsForSmall = find(ValidBlobsSmall)
```

```
validLabelsForSmall = 1×2
     5     8
```

```
binaryMaskSmall = ismember(labeledImage, validLabelsForSmall);

figure;
imshow(binaryMaskSmall);
title('Binary Mask for Small Gears');
```




```
% Use the binary image mask to filter out the small gears from the grayscale image
% and show the original image with only the separated gears.

gearImageSmall = orig_gears .* uint8(binaryMaskSmall);

figure;
imshow(gearImageSmall)
title('Isolated Small Gears');
```



c) Find the brightest medium-large gear in the image

Identify the brightest medium-large gear in the image using the blob intensity and other properties obtained in part (a) and extract it in a separate image.

i. Check blob properties from part (a) and identify which criteria to use to separate the brightest medium-large gear.

```
% Compare the area of the gears using the blob properties to separate
% medium-large gears and set up the criteria to find the brightest one.

minAreaMedBig = 26000; % From Part B. Min threshold used to determine MedBig Gears


% Extracts the labels of the gears from the blobs within the given region.
medBigMask = ismember(labeledImage, find([blobregions.Area] >= minAreaMedBig & [blobregions.Area] < minAreaBig));
medBigGearLabels = unique(labeledImage(medBigMask))
```

```
medBigGearLabels = 2×1
     3
     6
```

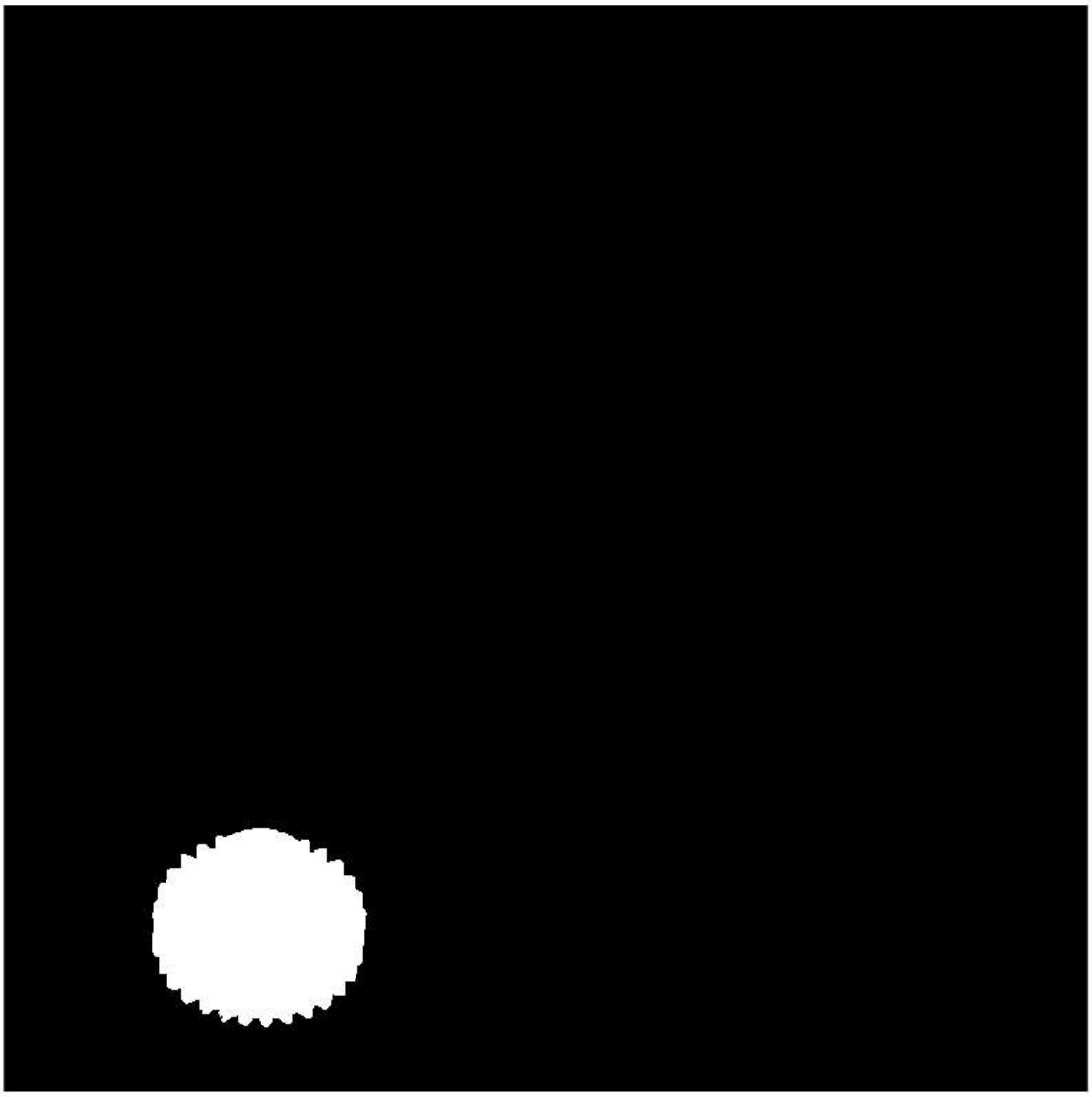
```
brightestMeanIntensity = 0;  % Initialize with a very small value
brightestMediumLargeGearLabel = 0;


for i = 1:numel(medBigGearLabels)
    gearLabel = medBigGearLabels(i);
    rowofLabel = propTable{:, 1} == gearLabel;
    meanIntensity = propTable{rowofLabel, 6};
```

```
    if max(meanIntensity) > brightestMeanIntensity
        brightestMediumLargeGearLabel = gearLabel;
        brightestMeanIntensity = max(meanIntensity);
    end
end
fprintf("Brightest Gear is %d\n", brightestMediumLargeGearLabel);
```

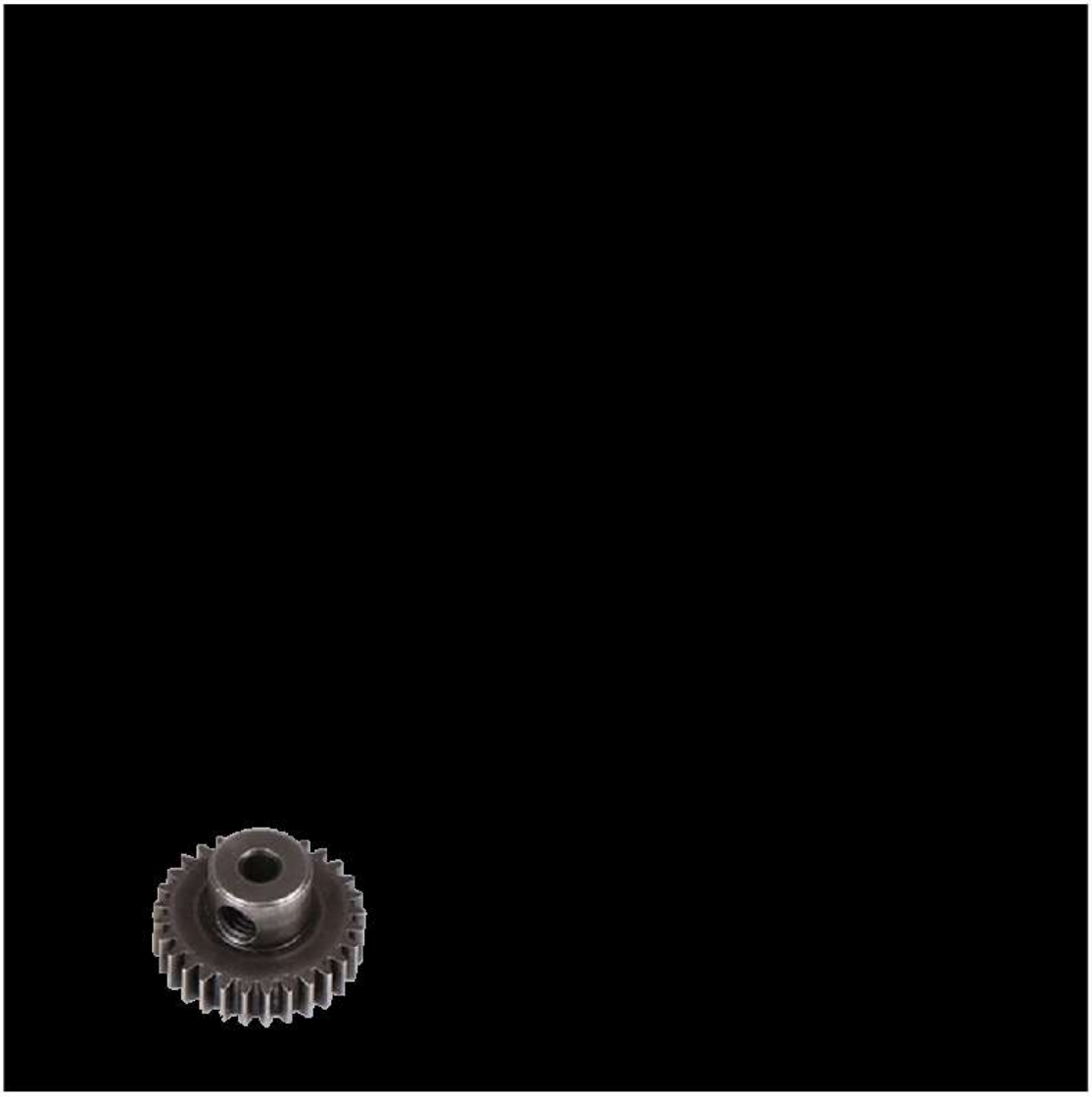
```
Brightest Gear is 3
```

ii. Use the `ismember` function to extract the identified blobs in the labeled image that was generated in part (a) using the `bwlabel` command. Create a binary mask from the labeled image and filter the brightest medium-large gear for display in the original grayscale image.

```
% Extract the brightest medium-large gear by selecting the blobs in the labeled image that
% meet the criteria and show the binary image mask with separated gear.

brightestMedBigGear = ismember(labeledImage, brightestMediumLargeGearLabel);

imshow(brightestMedBigGear);
```



```
% Use the binary image mask to filter out the brightest medium-large gear from the
% grayscale image and show the original image with only that gear.

imshow(brightestMediumLargeGearGray)
```

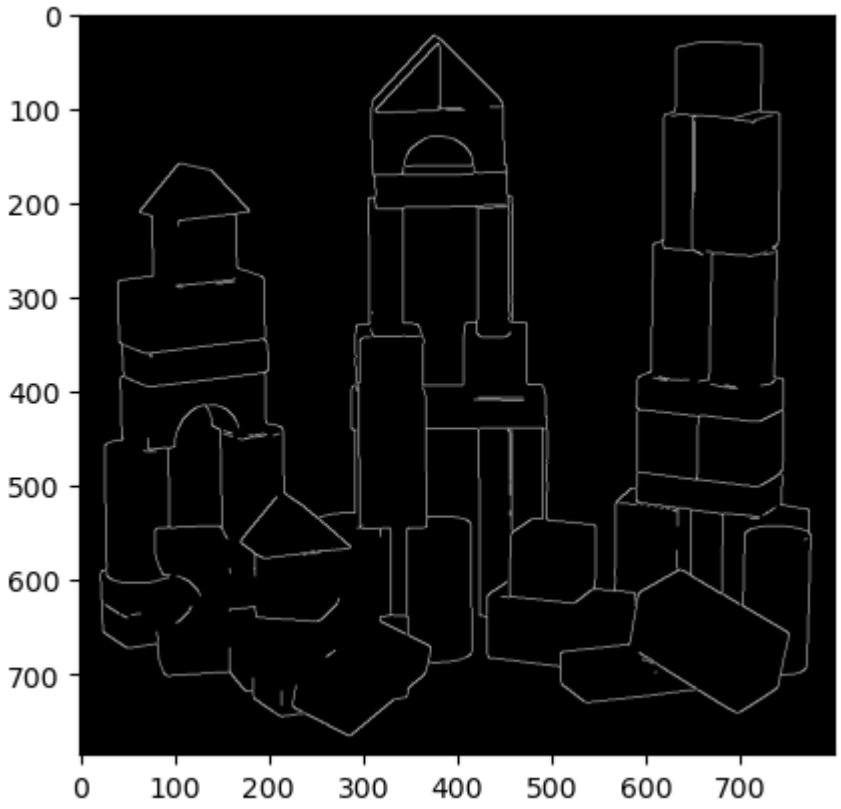


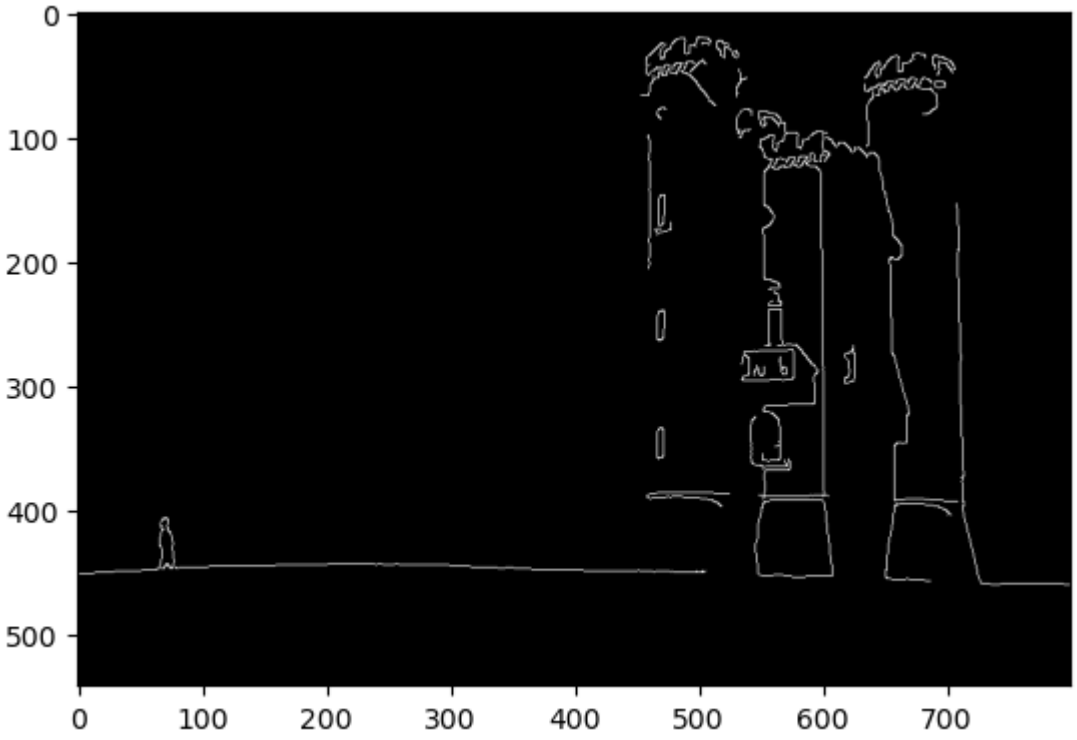
In [1]:

```
import cv2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

def null(x):
    pass

cv2.namedWindow('Castle')

cv2.createTrackbar("Canny T1: ", "Castle", 0, 255, null)
cv2.createTrackbar("Canny T2: ", "Castle", 0, 255, null)
cv2.createTrackbar("Gauss Kernal: ", "Castle", 0, 21, null)


cv2.namedWindow('Blocks')

cv2.createTrackbar("Canny T1: ", "Blocks", 0, 255, null)
cv2.createTrackbar("Canny T2: ", "Blocks", 0, 255, null)
cv2.createTrackbar("Gauss Kernal: ", "Blocks", 0, 21, null)


cv2.namedWindow('Parts')

cv2.createTrackbar("Canny T1: ", "Parts", 0, 255, null)
cv2.createTrackbar("Canny T2: ", "Parts", 0, 255, null)
cv2.createTrackbar("Gauss Kernal: ", "Parts", 0, 21, null)

picShow1 = False
picShow2 = False
picShow3 = False

while True:

# Castle Image
    castle = cv2.imread('castle_small.tif')

    castleCannyTh1 = cv2.getTrackbarPos('Canny T1: ', 'Castle')

    castleCannyTh2 = cv2.getTrackbarPos('Canny T2: ', 'Castle')

    castleGaussKernalSize = cv2.getTrackbarPos('Gauss Kernal: ' ,'Castle')

    # Kernal values that make it like instructions is (3,3)
    if castleGaussKernalSize % 2 == 1: # Kernal value has to be odd
         bluredCastle = cv2.GaussianBlur(castle,(castleGaussKernalSize,castleGaussKerr
    else:
         castleGaussKernalSize -= 1 # Else case means kernal value is an even number
         if castleGaussKernalSize <= 0: # Checks to make sure that it doesn't go out c
              bluredCastle = castle # If so apply no filter
         else:
              bluredCastle = cv2.GaussianBlur(castle,((castleGaussKernalSize),(castleG

    castleEdges = cv2.Canny(bluredCastle,castleCannyTh1,castleCannyTh2) # Since both t

    cv2.imshow('Castle', castleEdges)
    cv2.imwrite('Edges_Castle.jpg', castleEdges)
```

```
# Blocks Image

    blocks = cv2.imread('Blocks.jpg')

    blockCannyTh1 = cv2.getTrackbarPos('Canny T1: ', 'Blocks')

    blockCannyTh2 = cv2.getTrackbarPos('Canny T2: ', 'Blocks')

    blockGaussKernalSize = cv2.getTrackbarPos('Gauss Kernal: ' ,'Blocks')

    # Kernal values that make it like instructions is (3,3)
    if blockGaussKernalSize % 2 == 1: # Kernal value has to be odd
         bluredBlocks = cv2.GaussianBlur(blocks,(blockGaussKernalSize,blockGaussKernal
    else:
         blockGaussKernalSize -= 1 # Else case means kernal value is an even number
         if blockGaussKernalSize <= 0: # Checks to make sure that it doesn't go out of
              bluredBlocks = blocks # If so apply no filter
         else:
              bluredBlocks = cv2.GaussianBlur(blocks,((blockGaussKernalSize),(blockGau

    blockEdges = cv2.Canny(bluredBlocks,blockCannyTh1,blockCannyTh2) # Since both thre

    cv2.imshow('Blocks', blockEdges)

    cv2.imwrite('Edges_Blocks.jpg', blockEdges)



# Parts Image

    parts = cv2.imread('Parts.jpg')

    partsCannyTh1 = cv2.getTrackbarPos('Canny T1: ', 'Parts')

    partsCannyTh2 = cv2.getTrackbarPos('Canny T2: ', 'Parts')

    partsGaussKernalSize = cv2.getTrackbarPos('Gauss Kernal: ' ,'Parts')

    # Kernal values that make it like instructions is (3,3)
    if partsGaussKernalSize % 2 == 1: # Kernal value has to be odd
         bluredParts = cv2.GaussianBlur(parts,(partsGaussKernalSize,partsGaussKernalSi
    else:
         partsGaussKernalSize -= 1 # Else case means kernal value is an even number
         if partsGaussKernalSize <= 0: # Checks to make sure that it doesn't go out of
              bluredParts = parts # If so apply no filter
         else:
              bluredParts = cv2.GaussianBlur(parts,((partsGaussKernalSize),(partsGauss

    partEdges = cv2.Canny(bluredParts,partsCannyTh1,partsCannyTh2) # Since both thresh

    cv2.imshow('Parts', partEdges)

    cv2.imwrite('Edges_Parts.jpg', partEdges)

    key = cv2.waitKey(1)
    if key == 27: # exit on ESC (27 is ASCII for ESC)
          cv2.destroyAllWindows()
          break
```

```
blockDisplayImg = cv2.cvtColor(blockEdges,cv2.COLOR_BGR2RGB)
plt.imshow(blockDisplayImg)
plt.show()

castleDisplayImg = cv2.cvtColor(castleEdges,cv2.COLOR_BGR2RGB)
plt.imshow(castleDisplayImg)
plt.show()

partsDisplayImg = cv2.cvtColor(partEdges,cv2.COLOR_BGR2RGB)
plt.imshow(partsDisplayImg)
plt.show()

chartView = cv2.imread('ChartSize.png')
chartViewImg = cv2.cvtColor(chartView,cv2.COLOR_BGR2RGB)
plt.imshow(chartViewImg)
plt.show()
```





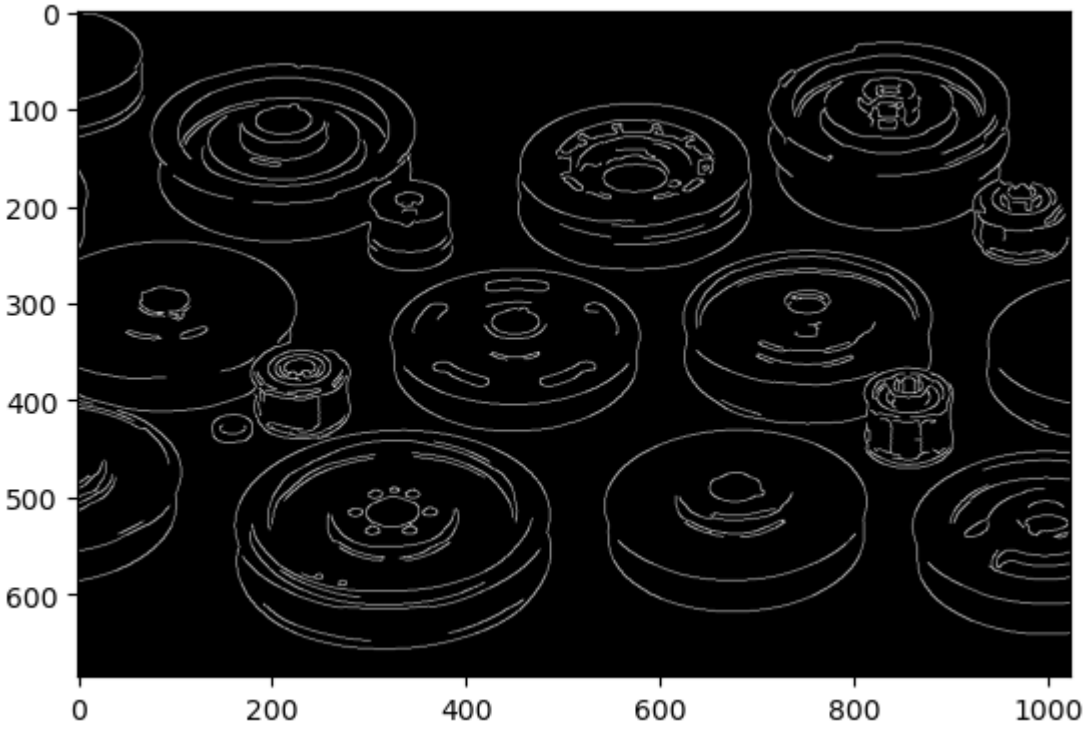
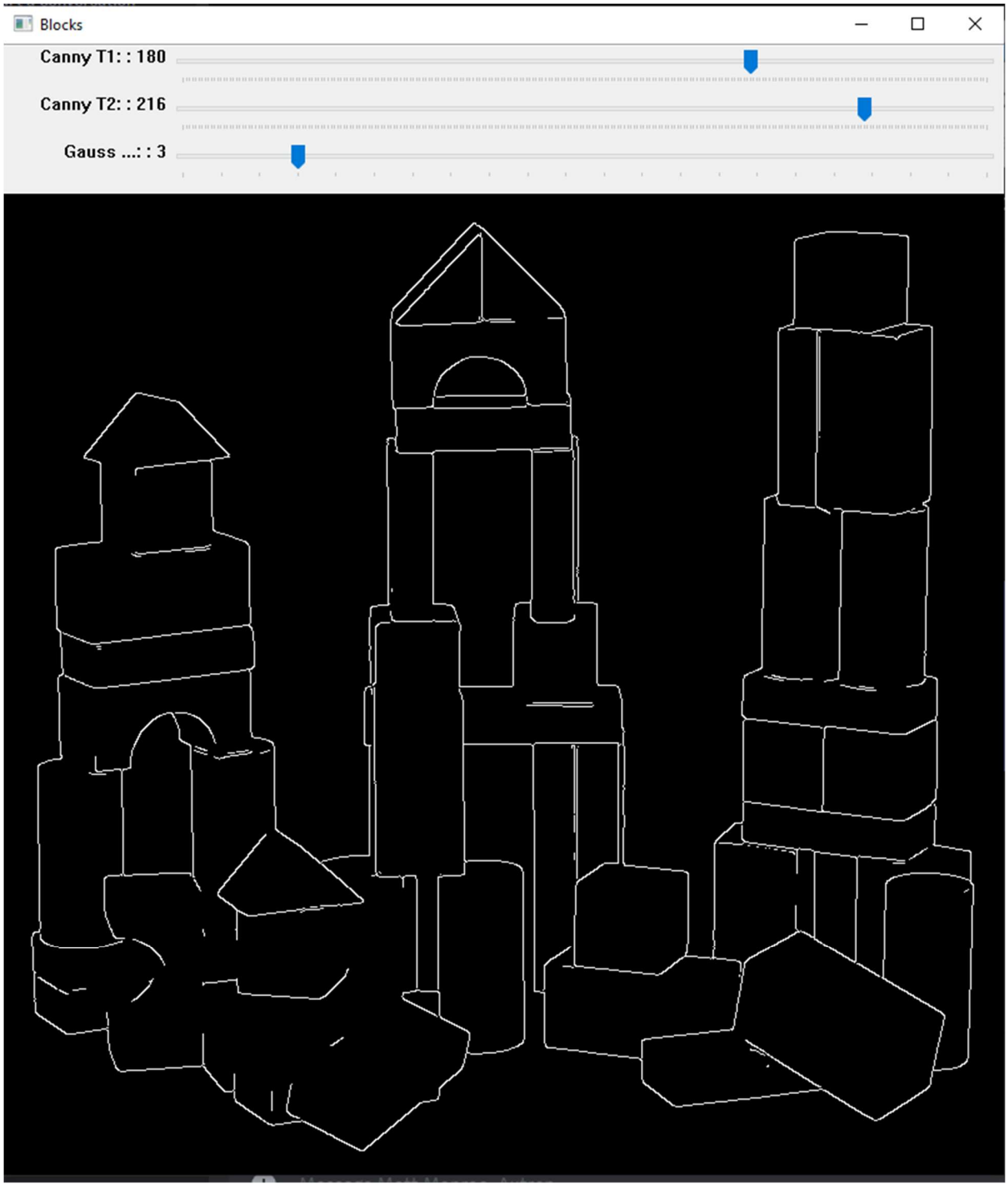
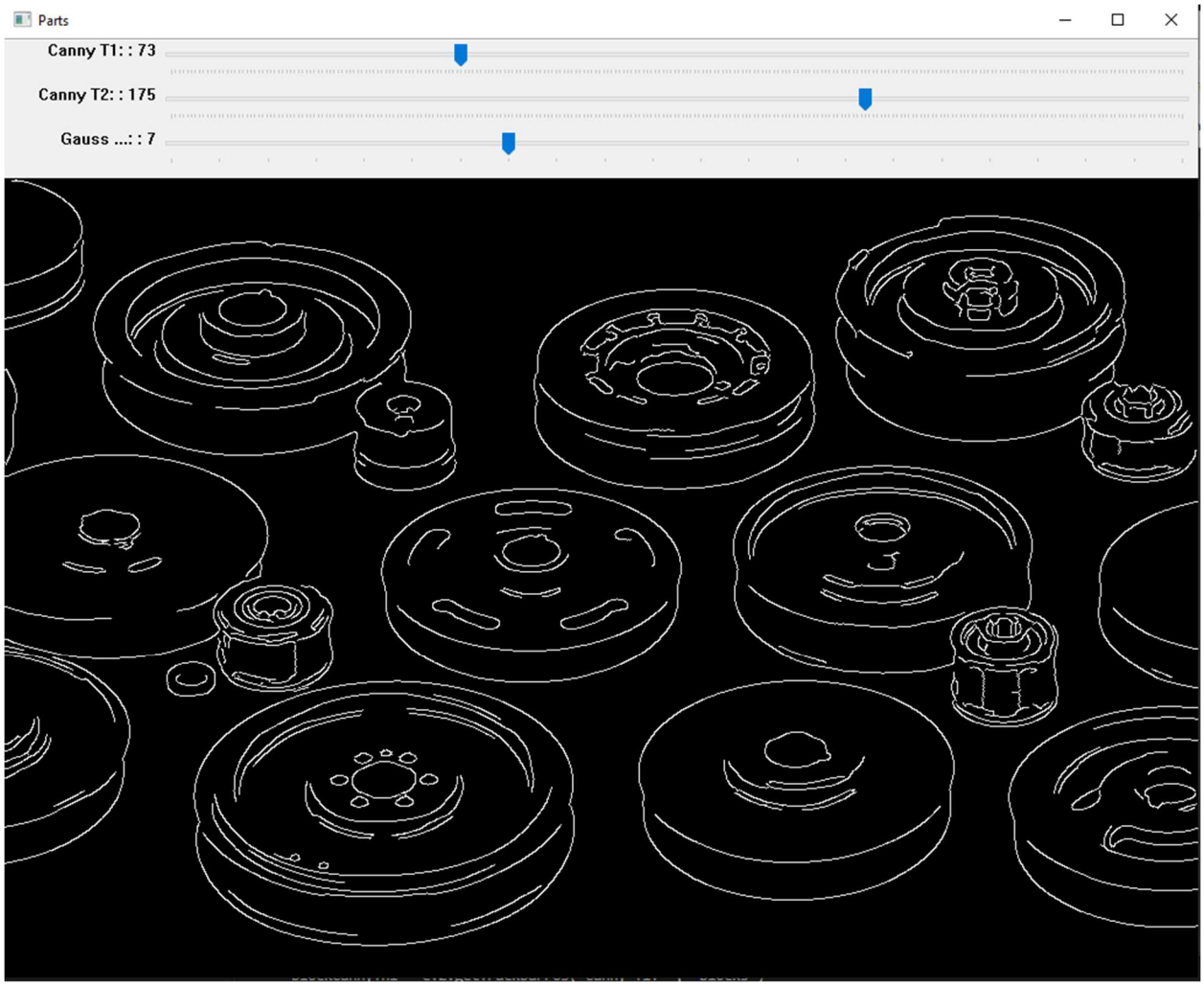


Image Name	Gaussian Radius	Canny Threshold 1	Canny Threshold 2
castle_small.tif	7	108	234
Blocks.jpg	3	180	216
Parts.jpg	7	73	175


Castle
Canny T1: : 108
Canny T2: : 234
Gauss ...: : 7


Blocks
Canny T1: : 180
Canny T2: : 216
Gauss ...: : 3

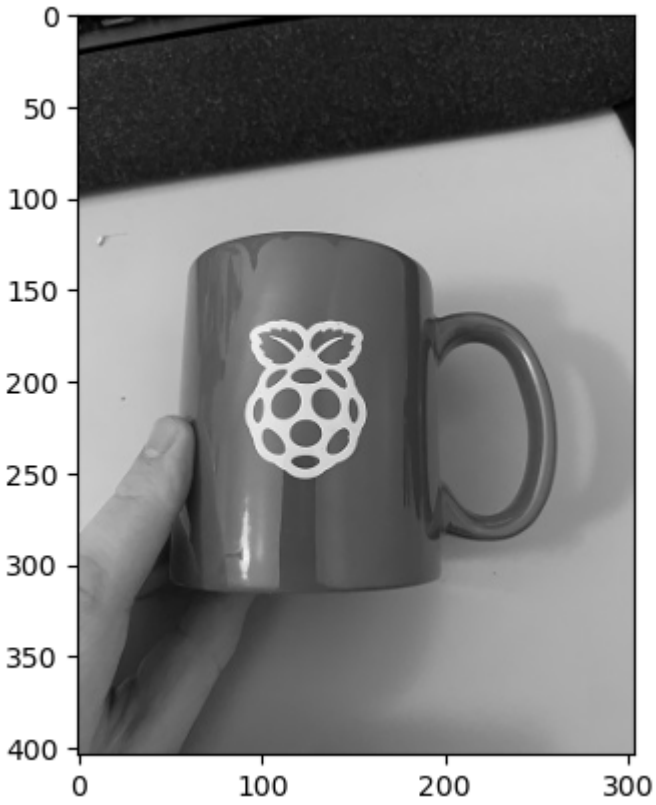

Parts
Canny T1: : 73
Canny T2: : 175
Gauss ...: : 7

Task 3

In [1]:
```
import numpy
import cv2
import matplotlib.pyplot as mpl
```

1) Opening the source image (and naming it image 1):

In [2]:
```
img1 = cv2.imread("RaspberryPiMug.jpg")
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
img1 = cv2.cvtColor(img1, cv2.COLOR_GRAY2RGB)

mpl.imshow(img1),mpl.show()
```



Out[2]:
```
(<matplotlib.image.AxesImage at 0x2ad2d7b3490>, None)
```

2) Detecting keypoints and descriptors on the source image (img1) using ORB and BFMatcher

In [3]:
```
#Start orb dectector and find keypoints and discriptors
orb = cv2.ORB_create()
keyp1, des1 = orb.detectAndCompute(img1, None)
```

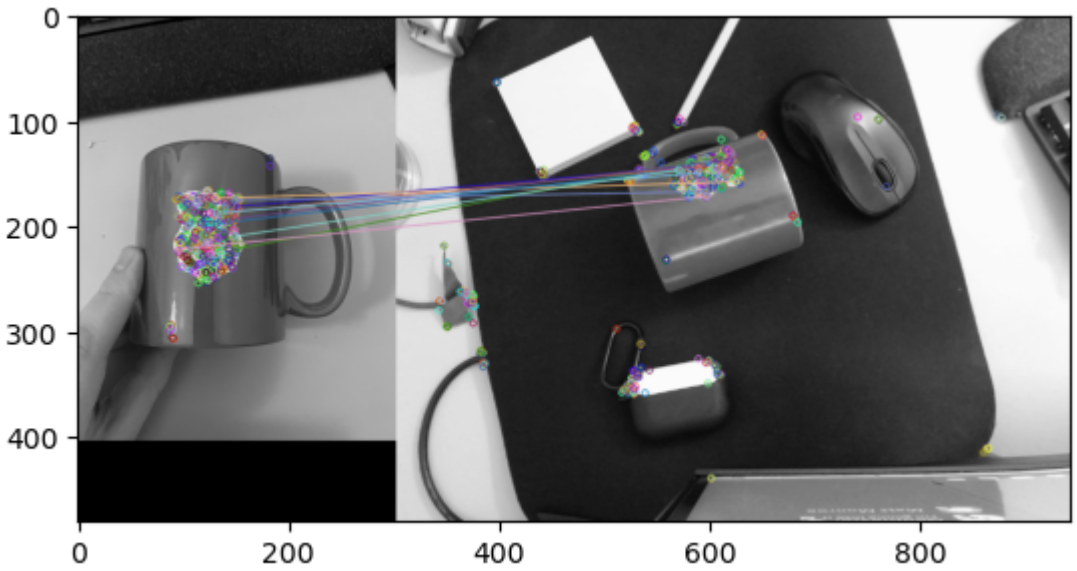
In [4]:
```
#Start BF matcher
bf = cv2.BFMatcher_create()
```

3) Opening a video capture window

In [5]:
```
cap = cv2.VideoCapture(0)
```

4) Process each frame of the video stream

In [6]:
```
# Loop that matches img1 to live video capture.
#ORB is used to find Keyp2 and des2 of the video frams
#BFMatcher is used to match des1 and des2, along with their distance
#The 10 best features are drawn
while True:
    ret, img2 = cap.read()
    img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    keyp2, des2 = orb.detectAndCompute(img2, None)
    matches = bf.match(des1, des2)

    img3 = cv2.drawMatches(img1, keyp1, img2, keyp2, matches[:10], outImg=None)
    cv2.imshow("Object Tracking", img3)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
mpl.imshow(img3),mpl.show()
```



Out[6]:
```
(<matplotlib.image.AxesImage at 0x2ad2d93eed0>, None)
```

Above is the result of our feature matching between the original image(left) and the last frame of the video (right).

8) Stop the video and save the last image

In [7]:
```
#Save the last image as "Last_Frame", and save the matched images as "Matched images"
cv2.imwrite("Last_Frame.jpg", img2) #saves last frame of video only. Not sure if this
cv2.imwrite("Matched_Images.jpg", img3)

cap.release()
cv2.destroyAllWindows()
```