AMATH 582 Final Project

Kyle Schultz[*]

March 19, 2020

Abstract

In this project, we explored detecting features from a video. In particular, we used a video taken from inside of a 737 wing bay and detected fasteners. We employed a supervised learning technique where a person must label fasteners from selected frames of a video to construct a training set. We train a custom object detector in MatLab, and achieved up to an 83% accuracy.

1 Introduction and Overview

Currently, human quality engineers climb into the wing of aircraft in order to inspect the work done my mechanics and ensure compliance. One such quality check is ensuring that all required fasteners are installed. Our goal is to use a camera feed to autonomously perform these quality checks, thereby eliminating the need for humans to climb inside confined spaces. Here, we use supervised learning to train a cascade object detector to detect the location of fasteners in a camera feed. Although simply detecting the location of fasteners in a wing is a small component of the inspection process, we hope to continue this work to improve functionality.

Figure 1: Mechanic preparing to crawl inside of Boeing 737 fuel tank

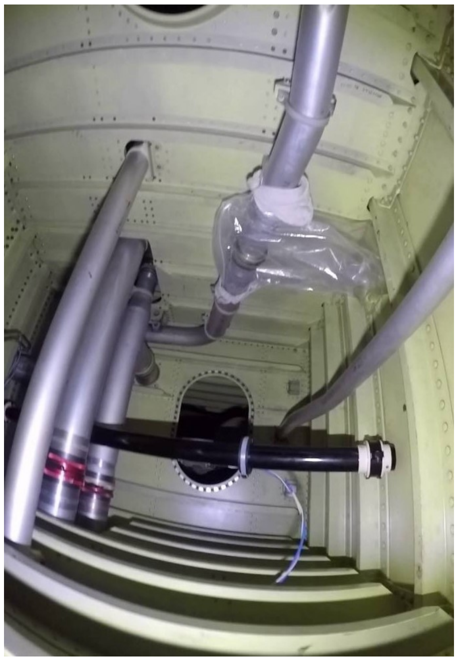
Figure 2: View of confined space inside of Boeing 737 fuel tank

[*]Code available on Github: https://github.com/kyleschultz0/

2 Theoretical Background

2.1 Singular Value Decomposition

Singular value decomposition (SVD) is an algorithm that expresses any matrix in terms of three matrices: U, S, V (equation 1).

$$A = USV^* \tag{1}$$

In equation 1, matrices U and V are unitary and S has real, positive values on its diagonal. When we are performed the SVD on our image data, we used a 'skinny' matrix: each column holds the pixel data of a different image. In this case, the columns of V are the dominant modes, U is how the data projects onto the modes in V, and S are the singular values. These give us a weighting of how important the dominant modes in V are. We exploit this idea that certain modes are dominant to determine if a fastener is present in a frame of the video. If we compute $U^T X$ where X is the matrix containing test images, we can extract the dominant modes in a photo. We exploit this to effectively determine if a fastener is in an image.

2.2 Supervised Learning

To detect the location of fasteners in our camera feed, we use a supervised learning architecture. This requires a labeled data set from which our algorithm will identify the most important features of fasteners. We construct our training set by manually labelling the location of fasteners in frames of video frame using a graphical user interface developed in MatLab (see Algorithm Implementation and Development). We then project our test set onto the principal components found by taking the SVD of the training set ($U^T X$). By splitting our data into test and training sets, we can determine how accurate our algorithm is at classifying new data. The training set has no prior knowledge of the test set from either the SVD.

3 Algorithm Implementation and Development

Our algorithm has two distinct parts functions: allow a use to easily label a training set, and detect fasteners based on training set. These sections are Training Set Creation and Object Detection.

3.1 Training Set Creation

To perform supervised learning, we must build a training set. This is achieved by allow the use to drag a rectangle over each faster in the frame using a graphical user interface (figure 3). When all the bolts in a frame have been selected, the use can then skip to the next frame. The algorithm to build a training set is described in algorithm 1.

Algorithm 1: Algorithm for labeled training set

```
Load video data
Initialize push button
Initialize numberbolts = 0
for $i = 1 : NumberFrames$ do
    Display frame $i$
    while Button not pressed do
        Get coordinates of used defined rectangle
        numberbolts = numberbolts + 1
    end while
end for
```

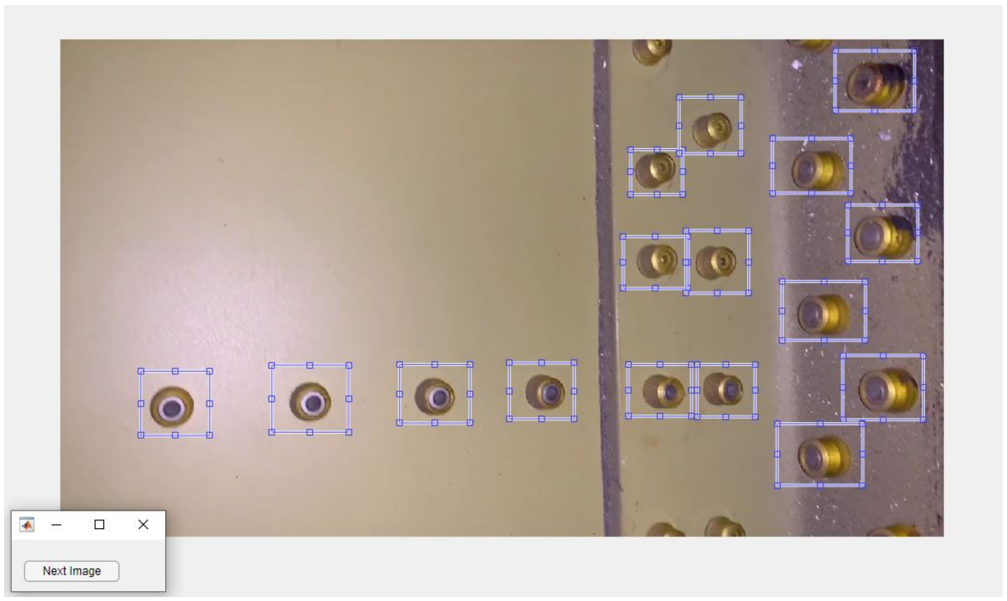

Figure 3: Graphical user interface (GUI) used to label fastener locations from video feed

3.2 Detecting Fastener Location

To detect the location of fasteners, we first compute the SVD of our training set to determine the best features to extract in order to determine the presence of a bolt. We then run a window over our image to crop it in to many different photos, and project each of these photos onto the SVD modes found. This gives us the representation of a small piece of a frame in our video in terms of the features of the bolts. We found using the first 4 SVD modes to give us the best detection characteristics. Concretely, we computed $U(:,1:4)' * window$ which gives a vector of how window projected onto our SVD modes. We take the 2-norm of this vector which we use as a 'score' of how similar the modes of the window are to the training set. If the 'score' reaches a threshold, we classify it as a bolt and store its location. This process is described in algorithm 2.

Algorithm 2: Algorithm to detect locations of fasteners

```
for $i = 1 : NumberFrames$ do
  Load frame $i$
  BoltsDetected = 0
  for $j = 0 : NumberWindows_x - 1$ do
    Set location of window to $j$*shifty in y direction
    for $k = 0 : NumberWindows_y - 1$ do
      Create window of size (windowx, windowy)
      Shift window by $k$*shiftx in x direction
      Project windowed image onto SVD modes, $U^T X$
      Norm = norm($U^T X$)
      if norm >threshold then
        Store position of window
      end if
    end for
  end for
end for
```

4 Computational Results

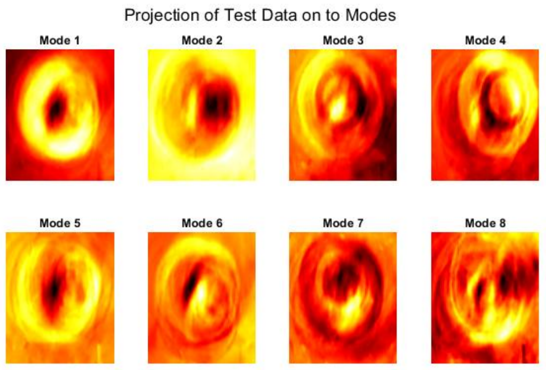

Figure 4: Projection of test data on to modes computed using SVD. We can see that there are various circular features, and these modes represent the different types of fasteners as well as different orientations.

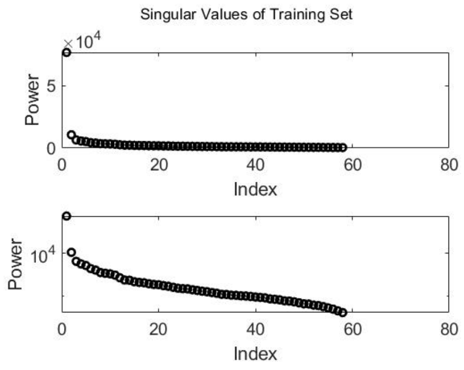

Figure 5: Singular values of training set

We first consider the results of the SVD alone. In figure 4, we plot the projection of our test set onto the SVD modes. The first mode primarily captures the features of the fastener from a top down view: two concentric circles. This makes sense since in the majority of the training set is taken from this top down view (see figure 5). The other modes appear to be captured the different orientation of fasteners as well as the different types of fasteners.

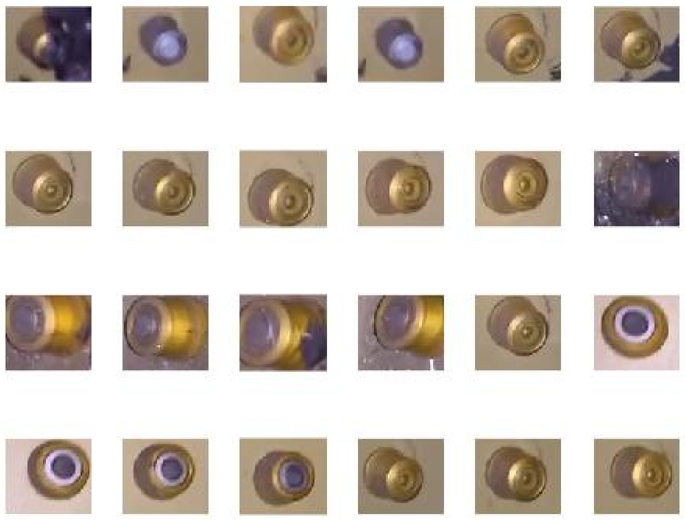

Figure 6: An abridged version of the training set used. 71 bolts used in full set.

In figure 7, we show the position detected fasteners overlaid on the frame. We can see that the algorithm generally does a good job of locating the fasteners. It misses 2 out of the 12 fasteners. One of the fasteners is missed because it is covered in sealant and thus its features are suppressed. The other fastener is likely

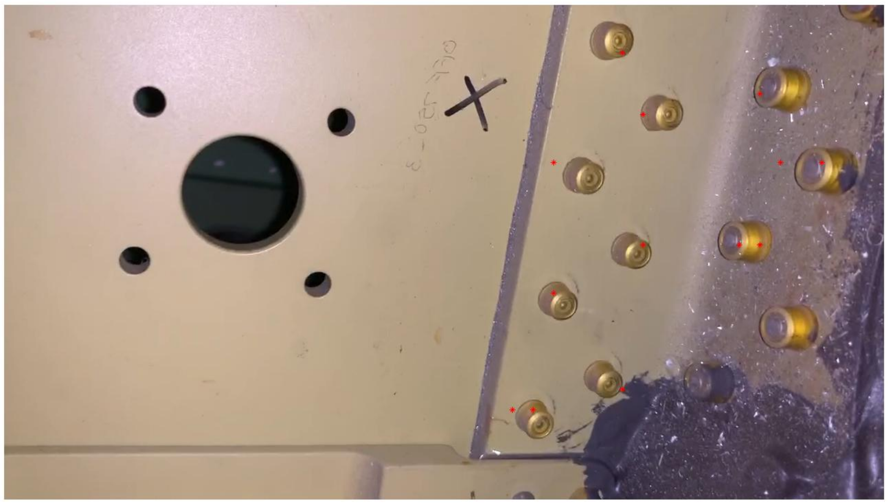

Figure 7: Results from a chosen frame, with red stars indicating the location of a fastener

missed because of issues with windowing. If a window cuts the fastener so that the circular shapes are not fully represented, it might be missed. We also see that two markers are inserted on some of the bolts; they are 'double marked'. This also happens because of the windowing: if the window intersects the bolt in a way such that both windows have many features of the bolt, it will be double counted. Since the bolts are sufficiently far apart, this can be overcome by averaging the location if bolts are found in two adjacent widows. The accuracy of this test is

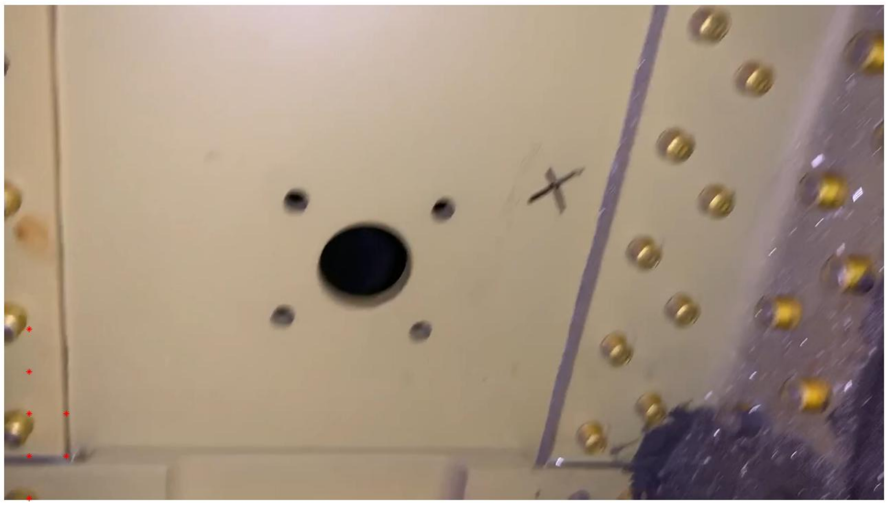

Figure 8: Results from another frame, with red stars indicating the location of a fastener

Figure 8 shows an example where the bolt recognition largely failed. There are two reasonable explanations for this compared to figure 7. First, there is a significant blur in the image caused by camera shake. This washes out the bolt features and makes it harder to recognize them. Second, the image taken from further away, and the window size is likely too large for this case. If our window is too large, we will capture features in our window other than the bolts and the threshold is unlikely to be met.

Test Case	Type 1 Error	Type 2 Error	Double Counted
1	17%	0%	27%
2	82%	63%	0%

Table 1: Accuracy of classification tests. These statistics are calculated by using human fastener recognition (i.e. the authors counted). Type one error calculated as bolts not detected divided by number of bolts in photo. Type two error is erroneous bolts divided by total bolts. Double counting refers to the same bolt being detected twice.

5 Summary and Conclusions

In this project, we used SVD to create a low rank representation of bolt features and build a bolt detection algorithm. We trained this algorithm using a labeled data set that was generated through a GUI developed in MatLab that allowed users to specify the location of a bolt. We then used a windowing method to find the dominant modes in cropped images pulled from video frames. This allowed us to compute the dominant modes in space and compare them to our training data. We achieved an 83% detection accuracy in ideal conditions. Accuracy diminished when there was camera shake of the video was taken too far away from the fasteners. In the future, we hope to apply this algorithm to different types of features (eg holes, other fastener types). We also hope to test algorithms to remove camera shake and the effect of using dynamic window sizes depending on zoom level.

Appendix A MATLAB Functions

Below are some of the most critical functions in our MATLAB implementation. Included are details pertinent to how we used them.

- `[U, S, V] = svd(A, ’econ’)` returns the singular value decomposition of A, with economy returning only the relevant rows/columns to save storage and computation time
- `class = classify(test,training,group)` classifies each row of test using one of the labels in group.
- `objectRegion = getPosition(imrect)` returns the corner points of a user prescribed rectangle. This is used to specify the region for tracking.
- `pcolor(tslide,ks,spectrum)` creates pseudocolor plot using the values in matrix `spectrum`, in our case the Gabor transform. It is defined by the x and y coordinates `tslide` and `ks`.
- `Un(:,:,:)  = reshape(Undata(j,:),n,n,n` turns `Undata(j,:)` into a nxnxn array.
- $[y_i, x_i, z_i]$ `= ind2sub(sz, index)` converts a linear index, `index`, to the equivalent subscripts in an array of size `sz`.

Appendix B MATLAB Code

```
clc; clear all; close all;

%% Get user to identify bolts in training set

videoFileReader_train = vision.VideoFileReader('IMG_1208.mp4');
videoPlayer_train = vision.VideoPlayer();

num_samples = 16;

fig = uifigure;
btn = uibutton(fig,'state','Text','Next Image', 'Value', false);
```

```
num_bolt = zeros(1, num_samples);

for i = 1:num_samples
    for j = 1:80
        objectFrame1_1 = videoFileReader_train();
    end
    figure; imshow(objectFrame1_1);
    btn.Value = false;
    pause(5)
    while true
        drawnow()
        stop_state = get(btn, 'Value');
        if stop_state
            break
        end
        num_bolt(i) = num_bolt(i) + 1;
        objectRegion_test(:,num_bolt(i),i)=round(getPosition(imrect));
    end
end

%% Create training images of bolts from user input

v = VideoReader('IMG_1208.mp4');
bolt_train = zeros(720, 1280, 3, max(num_bolt), num_samples);

for i = 1:num_samples
    frames(:, :, :, i) = read(v, 80*i);
    for j = 1:num_bolt(i)
        object_Region = objectRegion_test(:,j,i);
        bolt_train(object_Region(2):object_Region(2)+object_Region(4),...
            object_Region(1):object_Region(1)+object_Region(3), :, j, i) = ...
            permute(frames(object_Region(1):object_Region(1)+object_Region(3),...
            720-object_Region(2)-object_Region(4):720-object_Region(2), :, i), [2 1 3]);
    end
end

total_bolts = 0;

for i = 1:num_samples
    for j = 1:num_bolt(i)
        object_Region = objectRegion_test(:,j,i);
        total_bolts = total_bolts + 1;
        bolt_images(:, :, :, total_bolts) = uint8(bolt_train(:, :, :, j, i));
        eval(sprintf('Bolt_%d = bolt_images(:, :, :, total_bolts);', total_bolts));
        eval(sprintf('Boltcropped_%d = Bolt_%d(object_Region(2):object_Region(2)+object_Region(4), obje
        figure; imshow(eval(sprintf('Boltcropped_%d', total_bolts)));
        pause(0.5)
    end
end

cd 'training'

for i = 1:total_bolts
    q1 = sprintf("Boltcropped_%d", i);
    q2 = sprintf("Boltcropped_%d.png", i);
    imwrite(eval(q1), q2);
end

%% Creating cropped images of same size

for i = 1:total_bolts
    name = sprintf("Boltcropped_%d", i);
    sz = size(eval(name));
    sizex(i) = sz(1); sizey(i) = sz(2);
end

minx = min(sizex); miny = min(sizey);

for i = 1:total_bolts
    name = sprintf("Boltcropped_%d", i);
    sz = size(eval(name));
    adjustx = sz(1) - minx;
    adjusty = sz(2) - miny;
    rx = rem(adjustx,2); qx = floor(adjustx/2);
    ry = rem(adjusty,2); qy = floor(adjusty/2);
    eval(sprintf('Boltadjust(:, :, :, i) = Boltcropped_%d(qx+rx+1:sz(1)-qx,qy+ry+1:sz(2)-qy , :);', i, 
end

%% SVD on adjusted bolts

close all

bolts_reshaped = reshape(Boltadjust, [minx*miny*3, total_bolts]);

[U, S, V] = svd(double(bolts_reshaped), 'econ');

figure;
subplot(2,1,1)
plot(diag(S),'ko','Linewidth',[2])
set(gca,'Fontsize',[14],'Xlim',[0 80])
subplot(2,1,2)
semilogy(diag(S),'ko','Linewidth',[2])
set(gca,'Fontsize',[14],'Xlim',[0 80])


figure;
for i = 1:total_bolts
    subplot(8,8,i)
    imshow(squeeze(uint8(Boltadjust(:, :, :, i))))
end


for j = 8
    figure;
    im_rank1 = U(:, 2:j)*S(2:j, 2:j)*V(:, 2:j)';
    im_rank1s = reshape(im_rank1, [minx, miny, 3, total_bolts]);
    for i = 1:total_bolts
        subplot(8,8,i)
        imshow(squeeze(uint8(im_rank1s(:, :, :, i))))
    end
end

figure;
for i = 1:8
    subplot(2, 4, i)
    Ut1 = reshape(U(:, i), minx, miny, 3);
    Ut2=Ut1(minx:-1:1,:);
    pcolor(Ut2), shading interp, colormap hot
    set(gca,'Xtick',[],'Ytick',[])
end

%% Loading test video

video_test = VideoReader('IMG_1210.mp4');
video_test = read(video_test);
video_test = permute(video_test, [2 1 3 4]);

%% Detecting bolts

clear bolts_positions x_detected y_detected
close all; clc

size_test = size(video_test);
sz_testx = size_test(1);
sz_testy = size_test(2);

numwindows_x = floor(sz_testx/(0.5*minx));
numwindows_y = floor(sz_testy/(0.5*miny));

window_x = minx;
window_y = miny;

shift_x = floor(sz_testx/numwindows_x);
shift_y = floor(sz_testy/numwindows_y);


numframes = 12;

threshold = 2*10^4;

for i = 1:numframes
    frame = video_test(:, :, :, 80*i);
    bolts_detected = 0;
    for j = 0:numwindows_y-2
        for k = 0:numwindows_x-2
            window = frame(k*shift_x+1:k*shift_x+window_x, j*shift_y+1:j*shift_y+window_y, :);
            window_reshape = double(reshape(window, [minx*miny*3, 1]));
            projection = U(:, 1:4)'*window_reshape;
            projection_score = norm(projection);
            if projection_score > threshold
                bolts_detected = bolts_detected+1;
                x_detected = k*shift_x+1 + (k*shift_x+window_x)/2;
                y_detected = j*shift_y+1 + (j*shift_y+window_y)/2;
                bolts_positions(bolts_detected, 1, i) = x_detected;
                bolts_positions(bolts_detected, 2, i) = y_detected;
            end
            bolt_classifier(j+1, k+1, i) = projection_score(1);
        end
    end
    if bolts_detected == 0
        bolts_positions(1, 1, i) = 0;
        bolts_positions(1, 2, i) = 0;
    end
    %     bolt_classifier = projection_score(:, :, i);
    %     [M, I] = max(bolt_classifier,[],'all','linear');
    %     bolt_classifier_norm = bolt_classifier/M;
    %     [row,col] = ind2sub([numwindows_x-1 numwindows_y-1],I);
    %     bolts_positions(1,1,i) = shift_y*(row+0.5);
    %     bolts_positions(1,2,i) = shift_x*(col+0.5);
    figure;
    imshow(frame)
    hold on
    scatter(bolts_positions(:, 1, i), bolts_positions(:, 2, i), 'r*')
end
```