

Compiling the MEX opencv Library in Matlab

Network Interfacing with AXIS Camera

Petros Taskas

11/14/2014

Executive Summary

Measuring the diameter of a steel coil at the Arcelor-Mittal steel plant is a necessity. Using an Axis Communications P1355-E Network Camera will generate a better profit for Arcelor-Mittal due to the more accurate diameter measurements using the MEX opencv Library in Matlab.

Objective

This application note will explain how the Axis Communications P1355-E Network Camera will produce more accurately product yield and higher profit for Arcelor-Mittal.

Introduction

Axis communications is a “Smart” camera provider with indoor and outdoor camera models. These cameras can be used for many different tasks including the measurement of the diameter of a steel coil. This application note will describe the pre-existing functions and accessibility of the Axis Communications P1355-E Network Camera by using Matlab and Simulink to program the camera to measure the diameter of the steel coil being wound at the Arcelor-Mittal steel plant. For this using the MEX and opencv library in Matlab will prove to be a huge help with its functionality.

Hardware

The Axis Communications P1355-E Network Camera is a full high definition 1080p 30fps camera that uses power over Ethernet (POE) to communicate and send data with pre-existing servers and programs to the host computer. POE means that is the source of the power to turn on the camera and it also sends and receives data through it. The 30fps the camera can run at is faster than we need it run at to accurately capture the diameter size. The fps will be lower to 5-10fps since the steel comes in a 5m/s.

The P1355-E Network Camera will be programmed using Matlab and Simulink. On Matlab we will be using MEX opencv library to more effectively detect the growing coil in the live video stream.



Figure 1 Axis Communications P1355-E Network Camera

Software

The MEX opencv library does not come pre-existing on Matlab, opencv will have to be downloaded with other programs such as visual studio in order to be used. MEX helps you use opencv functions in Matlab which are very helpful. Opencv functions are very efficient and very fast when compared to their implementations in Matlab.

So in order to you the basic functions what is created first is a detector which can detect face, steel coils, and many other features depending on which .xml file you wish to import. After this the program must go and read the .jpeg or .png that is being imported. Once this is complete, you must then convert the color of the image to gray scale so the program can easily distinguish the magnitude of gray in the image to set boundaries on what you wish to detect. Tweaking the parameters of 'ScaleFactor', 'MinSize', and ect. will help get the best results. Once this is all complete the image will be plotted and from there rectangles, circles, and or lines can be plotted to show the user what is being detected.

Figure 2 is a sample code for face detection. This code does exactly as stated above. It first imports the picture, converts it to gray scale, parameters are set for best detection, and finally the image is outputted with rectangles around the faces.

```

1 % Load a face detector and an image
2 detector = cv.CascadeClassifier('haarcascade_frontalface_alt.xml');
3 im = imread('family.png');
4 % Preprocess
5 gr = cv.cvtColor(im, 'RGB2GRAY');
6 gr = cv.equalizeHist(gr);
7 % Detect
8 boxes = detector.detect(gr, 'ScaleFactor', 1.3, ...
9                             'MinNeighbors', 2, ...
10                             'MinSize', [30, 30]);
11 % Draw results
12 imshow(im);
13 for i = 1:numel(boxes)
14     rectangle('Position', boxes{i}, ...
15               'EdgeColor', 'g', 'LineWidth', 2);
16 end

```

Figure 2 Face Detection [1]

Video Input Capture

The next set of steps will help us capture the video being uploaded to the server for statistical analysis of the coil. To assist in capturing the image, the `cv.VideoCapture()` function will be used. This function will connect to the camera and start capturing the video footage. Next each frame will be taken and “created” to be read.

The function takes the video frame by frame and sends it to the server and also sends it to the next set of functions to extract the frames and notice differences and detect the object or objects wished to be measured or tracked.

Figure 3 is the sample code that just takes the frames and sends them off to the next process.

```

1 clear all;
2 % Connect to a camera
3 camera = cv.VideoCapture(1); % try -1, 0, 1, 2, 3
4 pause(2);
5 for i = 1:500
6     % Capture and show frame
7     frame = camera.read;
8     imshow(frame);
9     pause(0.1);
10 end

```

Figure 3 Camera Input [2]

Feature Extraction and Matching

This feature takes the image, runs it through the code, and then looks at the image to see if any features that were sought after were found in the image. The first part of the code is to read the image that is being sent from the camera. Next the `cv.FeatureDetector()` function has a parameter that can be changed to help with detection of different shapes and tolerances within the image. After this is set `cv.DescriptorExtractor()` is used with the same parameter as in the detector so once the parameter has been detected the extractor can pin point and extract the part of the image that is of interest.

The way the functions work together is they take and extract the computers of each key point. The key points are first detected and then the descriptors are used to extract the key points of the image from the detector. Then the program goes through and matches the features within the image.

The final step is to plot the points of interest on the image and show the user where and what needs to be looked at. The image below contains sample code used to detect certain features within an image and plot them by circling them in the image itself.

```
1 clear all;
2 close all;
3
4 a = imread('book.pgm');
5 b = imread('scene.pgm');
6 % basmati, book, box, scene
7 detector = cv.FeatureDetector('SIFT');
8 extractor = cv.DescriptorExtractor('SIFT');
9
10 tic
11 keypoints1 = detector.detect(a);
12 descriptors1 = extractor.compute(a, keypoints1);
13 keypoints2 = detector.detect(b);
14 descriptors2 = extractor.compute(b, keypoints2);
15 toc
16
17 index_pairs = matchFeatures(descriptors1,descriptors2);
18
19 matchpoints1 = [];
20 matchpoints2 = [];
21 for i = index_pairs(:,1)'
22     matchpoints1 = [matchpoints1 keypoints1(i).pt'];
23 end
24 for i = index_pairs(:,2)'
25     matchpoints2 = [matchpoints2 keypoints2(i).pt'];
26 end
27
28 figure;
29 subplot(121);
30 imshow(a);
31 hold on; scatter(matchpoints1(1,:),matchpoints1(2,:), 'yo', 'LineWidth', 2);
32 subplot(122);
33 imshow(b);
34 hold on; scatter(matchpoints2(1,:),matchpoints2(2,:), 'yo', 'LineWidth', 2);
```

Figure 4 Feature Extraction and Matching [3]

Recommendations

The code that has been shown throughout is only a model for what will be used to help with extracting features from the live video footage. The MEX opencv libraries and functions will have parameters changes to deal with darker area with a light colored steel coil being wound. The extraction portion can be used over the server to show the coil while it is being wound and after it is stopped and being moved out of the way to make sure nothing has malfunctioned.

Conclusion

This application note demonstrates how to use the MEX opencv libraries in Matlab. Using these techniques the users will be able to change parameters on other cameras to detect workers helmets and count how many times they get up to check and physically measure the coils. Figure 1 is the camera that will be used throughout the Arcelor-Mittal steel plant and can and will be used for many different tasks by using MEX opencv libraries in Matlab.

References

[1] Penmetsa. Surva. "Psurva1994/mexopencv-tutorial." *GitHub*. 9 Julv 2014. Web. 14 Nov. 2014. <[https://github.com/nsurya1994/mexopencv-tutorial/blob/master/Tutorial 2 - Face Detection/face_detection.m](https://github.com/nsurya1994/mexopencv-tutorial/blob/master/Tutorial%20-%20Face%20Detection/face_detection.m)>.

[2] Penmetsa. Surva. "Psurva1994/mexonencv-tutorial." *GitHub*. 9 Julv 2014. Web. 14 Nov. 2014. <[https://github.com/psurya1994/mexopencv-tutorial/blob/master/Tutorial 3 - Camera Input/camera_input.m](https://github.com/psurya1994/mexopencv-tutorial/blob/master/Tutorial%20-%20Camera%20Input/camera_input.m)>.

[3] Penmetsa. Surva. "Psurva1994/mexonencv-tutorial." *GitHub*. 9 Julv 2014. Web. 14 Nov. 2014. <[https://github.com/nsurya1994/mexonencv-tutorial/blob/master/Tutorial 4 - Feature Extraction and Matching/feature_extraction.m](https://github.com/nsurya1994/mexonencv-tutorial/blob/master/Tutorial%20-%20Feature%20Extraction%20and%20Matching/feature_extraction.m)>.