

## Motion Segmentation

24<sup>th</sup> September 2019






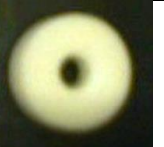
### 1.0 :: Introduction



*Figure 1 Q-Key Chain traversing facility on a conveyor belt*

The premise for this lab is that you have been hired by a widget manufacturer to upgrade their widget identification software for their new flexible and shared assembly line. In order to manage manufacturing costs, the company has implemented a new assembly line which allows them to cut costs by moving all their widgets across the facility using one common and shared conveyor belt (see Figure 1). To complete this facility upgrade, you have been hired to create the software to identify the widgets traversing the assembly line. The company plans on using this software to guide redirection of widgets to downstream assembly lines.

Per the software specification, the company has requested that the developed software correctly identifies the six widgets illustrated in Table 1. The specification does not call for 100% correct labelling of widgets, but the widgets should be predominantly labeled correctly, and classification should be withheld for frames with inconsistent or otherwise unreliable contours.

| Nut   | Peg   | Pipe  | Prong   | Q-Key Chain  | Washer  |
|---|---|---|---|--|---|
|  |  |  |  |  |  |

*Table 1 Widgets*

The purpose of this lab is to introduce the concepts of background segmentation and contour extraction. To correctly identify parts, you will need to analyze the morphology of widget contours.

## 2.0 :: Pre-Lab [1 Mark]

**Required Files:** belt\_bg.wmv and belt\_fg.wmv

Your task for the pre-lab component is to implement motion segmentation to differentiate between the background conveyor belt and the traversing foreground widgets. To achieve this, the factory has provided you with a video of the background conveyor belt (belt\_bg.wmv) and a video of the conveyor belt in operations with foreground widgets (belt\_fg.wmv). To differentiate between the foreground and background components of the scene, you will need to do some analysis of the background video to remove noise and variation. This can be achieved using two different approaches which are described below as *Method 1* and *Method 2* for Task#2, but ultimately both should provide clean outlines of the widgets as shown in Table 2.







| Nut   | Peg   | Pipe  | Prong  | Q-Key Chain   | Washer  |
|---|---|---|--|---|---|
|  |  |  |  |  |  |
|   |   |   |  |   |   |

Table 2 Background Subtracted Widgets

### Hints for the tasks:

- For an OpenCV example of cv::VideoCapture and cv::BackgroundSubtractor see the following link:
  - [https://docs.opencv.org/4.1.1/d1/dc5/tutorial\\_background\\_subtraction.html](https://docs.opencv.org/4.1.1/d1/dc5/tutorial_background_subtraction.html)

## Task #1 – Loading and Playing Video

Your first task is to load and play the video files `belt_bg.wmv` and `belt_fg.wmv` using `cv::VideoCapture`. Use `cv::imshow` to display the video.

*Need Help?*

- For `cv::VideoCapture` see [https://docs.opencv.org/4.1.1/d8/dfe/classcv\\_1\\_1VideoCapture.html](https://docs.opencv.org/4.1.1/d8/dfe/classcv_1_1VideoCapture.html)
- For an example, see [https://docs.opencv.org/4.1.1/d1/dc5/tutorial\\_background\\_subtraction.html](https://docs.opencv.org/4.1.1/d1/dc5/tutorial_background_subtraction.html)

## Task #2 - Background Subtraction: Method 1

Simple subtraction works well for static images but fails to account for noise and variation in the video feed. To account for these inconsistencies, you will implement the pseudocode in Figure 2 to differentiate between foreground and background pixels. The pseudocode subtracts the mean from each frame  $F_i$  and compares the result to a threshold multiple of standard deviations.

```
if ((| Fi[r,c] - Mean[r,c] |) > (thresh * Stdv[r,c])){  
    // Fi[r,c] is a foreground pixel ;  
}  
else {  
    // Fi[r,c] is a background pixel ;  
}
```

*Figure 2 Background Subtraction*

For this algorithm to work on the foreground video (`belt_fg.wmv`), you will have to calculate the mean and standard deviation of each pixel of the conveyor belt using the background video (`belt_bg.wmv`). Calculated for each pixel across  $N$  images (where  $N \geq 30$ ), the mean and standard deviation are defined in the following equations:

$$Mean[r, c] = \frac{1}{N} \left( \sum_{i=0}^{i=N} F_i[r, c] \right)$$

$$Stdv[r, c] = \sqrt{\frac{1}{N} \sum_{i=0}^{i=N} (F_i[r, c] - Mean[r, c])^2}$$

As a last step to improve the quality of your background subtractor, implement a blur to smoothen the input frames from the foreground and background video streams. OpenCV has many functions implemented to perform this operation.

### *Need Help?*

- To save some time, OpenCV has some handy functions to help you in implement these calculations. Some helpful functions include `cv::accumulate`, `cv::accumulateSquare` and `cv::accumulateProduct`.
  - [https://docs.opencv.org/4.1.1/d7/df3/group\\_imgproc\\_motion.html](https://docs.opencv.org/4.1.1/d7/df3/group_imgproc_motion.html)
- OpenCV has a library of image filtering functions you can use to blur your input frames. Many of these should be familiar from lecture.
  - [https://docs.opencv.org/4.1.1/d4/d86/group\\_imgproc\\_filter.html#ga8c45db9afe636703801b0b2e440fce37](https://docs.opencv.org/4.1.1/d4/d86/group_imgproc_filter.html#ga8c45db9afe636703801b0b2e440fce37)

### **Task #2 - Background Subtraction: Method 2**

As an alternative to Method 1, OpenCV provides some pre-built solutions for background extraction through `cv::BackgroundSubtractor`. This route can be more complicated but can provide better results.

### *Need Help?*

- For `cv::BackgroundSubtractor`, see:
  - [https://docs.opencv.org/4.1.1/d7/df6/classcv\\_1\\_1BackgroundSubtractor.html](https://docs.opencv.org/4.1.1/d7/df6/classcv_1_1BackgroundSubtractor.html)
- OpenCV has a library of image filtering functions you can use to blur your input frames. Many of these should be familiar from lecture.
  - [https://docs.opencv.org/4.1.1/d4/d86/group\\_imgproc\\_filter.html#ga8c45db9afe636703801b0b2e440fce37](https://docs.opencv.org/4.1.1/d4/d86/group_imgproc_filter.html#ga8c45db9afe636703801b0b2e440fce37)
- For an example, see:
  - [https://docs.opencv.org/4.1.1/d1/dc5/tutorial\\_background\\_subtraction.html](https://docs.opencv.org/4.1.1/d1/dc5/tutorial_background_subtraction.html)

### 3.0 :: Lab [6 Marks]

**Note!** The Lab is due in the off-week lab.

**Required Files:** belt\_bg.wmv and belt\_fg.wmv.

**Optional Files:** Widget Reference Files

- nut1.bmp, nut2.bmp, nut3.bmp
- peg1.bmp, peg2.bmp, peg3.bmp
- pipe1.bmp
- prong1.bmp, prong2.bmp, prong3.bmp, prong4.bmp
- q1.bmp, q2.bmp, q3.bmp
- washer1.bmp, washer2.bmp

The goal for the lab component is to implement an algorithm to classify the widgets dynamically as they traverse down the conveyor belt. This will be accomplished in two steps, starting with the analysis of the six widgets for distinguishing characteristics, and finally through the implementation of the classification algorithm.

#### **Task #1 – Widget Analysis:**

In order to perform classification on a widget its contour must first be extracted. This can be achieved by implementing `cv::findContours` to extract all the contours from a frame. To analyze the widgets and identify differentiating characteristics, you have two steps:

##### **Step 1 [Optional]: *Test characteristics from Sample Images***

As part of the resources supplied by the factory, 16 background subtracted images have been provided as samples to aid in your analysis. These samples were snipped from the output of a well calibrated background subtractor, where samples were only taken from frames with the widgets ideally positioned. This means that, for any given widget, these ‘ideal’ contours only exist for a few frames.

As these images are poor samples for creating a set of differentiating rules, it is suggested that you use these images to test your calculations of distinguishing characteristics (ex: arclength, area, etc.). When you are satisfied with their functionality, proceed to Step 2.

##### **Step 2: *Identify characteristics from foreground video***

For each of the contours extracted by `cv::findContours`, apply a set of calculations to extract distinguishing characteristics (ex: arclength, area, etc.). To create an effective rule set to distinguish widgets, you will need to account for variability in the extracted contours from the foreground video (belt\_fg.wmv).

When applying `cv::findContours` to the background subtracted video stream, you may find that OpenCV extracts a surprising number of contours from frames and extracted contours have quite a bit of

variability. Before moving on to create a classification rule set in Task #2, apply some basic filtering to extracted contours to remove the bulk of the incorrect and unwanted contours. This can be achieved by applying simple rules, such as discarding contours with an area below some threshold value. This filtering will improve your chances of correct widget classification in the next step.

### *Need Help?*

- OpenCV Structural Analysis and Shape Descriptors:
  - [https://docs.opencv.org/4.1.1/d3/dc0/group\\_imgproc\\_shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0](https://docs.opencv.org/4.1.1/d3/dc0/group_imgproc_shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0)
- A tutorial for OpenCV cv::findContours can be found here:
  - [https://docs.opencv.org/4.1.1/d5/daa/tutorial\\_js\\_contours\\_begin.html](https://docs.opencv.org/4.1.1/d5/daa/tutorial_js_contours_begin.html)
- Need some ideas or example of morphological characteristics? See:
  - [https://docs.opencv.org/4.1.1/da/dc1/tutorial\\_js\\_contour\\_properties.html](https://docs.opencv.org/4.1.1/da/dc1/tutorial_js_contour_properties.html)
  - [https://docs.opencv.org/4.1.1/dc/dcf/tutorial\\_js\\_contour\\_features.html](https://docs.opencv.org/4.1.1/dc/dcf/tutorial_js_contour_features.html)

### **Task #2 – Widget Classification:**

With a set of contours and a library of metrics for extracting widget characteristics, create a rule set which correctly classifies the widgets traversing the conveyor belt. This can be implemented as an if-else tree, switch-case, or however you see fit.

To improve classification accuracy, consider withholding predictions from your rule set until you have a consensus from N frames (as shown in the equation below). Using this method, a confidence score can be associated with each classification. To achieve a 75%+ accuracy in widget classification, report only classifications with a confidence greater than some threshold.

$$classification = \text{Max} \left( \frac{\sum_{i=0}^{i=N} \text{PredictionsForWidget}_x}{N} \right)$$

Example: A widget is going down the conveyor belt. Over the past N=10 frames your rule set has predicted 'washer' 6 times, and 'nut' 4 times. The classification is therefore 'washer' with 60% confidence, which is reported to the screen given it is greater than the threshold 50% confidence.

When you have completed this task to your satisfaction, hail a TA to get marked!

### Need Help?

- To determine how your rule set should work, it may be beneficial to plot, record or otherwise visualize the distinguishing characteristic metrics you calculate from contours. You do not need to do this in OpenCV (note the OpenCV graphing package is not included in your install), but using the `iostream (std::cout)` should do the trick.
- If your code is slow, consider restructuring your code such that metrics are only calculated when necessary. You shouldn't need all your metrics to determine if a contour should be discarded. Try to apply your set of metrics to the filtered and reduced set of good contours. This should reduce the computational load.

### 4.0 :: Mark Distribution

|             | Detailed Distribution |                           |                |    |
|-------------|-----------------------|---------------------------|----------------|----|
| Pre-Lab     | Complete<br>[ 1 Mark] | Incomplete<br>[0 Marks]   |                | /1 |
|             |                       |                           |                |    |
|             | Correctly Classified* | Sporadically Classified** | Not Classified |    |
| Nut         | [1 Marks]             | [0.5 Marks]               | [0 Marks]      | /1 |
| Peg         | [1 Marks]             | [0.5 Marks]               | [0 Marks]      | /1 |
| Pipe        | [1 Marks]             | [0.5 Marks]               | [0 Marks]      | /1 |
| Prong       | [1 Marks]             | [0.5 Marks]               | [0 Marks]      | /1 |
| Q-Key Chain | [1 Marks]             | [0.5 Marks]               | [0 Marks]      | /1 |
| Washer      | [1 Marks]             | [0.5 Marks]               | [0 Marks]      | /1 |
| Total:      |                       |                           |                | /7 |

\* Correctly classified = The object is predominantly classified correctly, with at least 70% of classifications being correctly labeled.

\*\* Sporadically classified = The object is occasionally identified correctly.