Hochschule Rosenheim
University of Applied Sciences

Faculty of Electrical Engineering and Information Technology

Master's Project

Development of an algorithm for transparent, dark and empty bottles detection

Submitted on: 24th May 2017

Supervisor:

Prof. Dr.-Ing. Michael Wagner

Students:

Aurangzaib Ahmed Siddiqui (Project Lead)

Umair Ashraf Butt

Rizwan Nasir

Acknowledgement

First and foremost, we would like to thank Hochschule Rosenheim for a placement in the Master's Program, Electrical Engineering and Information Technology. Furthermore, we would like to express our gratitude on behalf of the team to our supervisor Professor Michael Wagner. With the assistance of him and dedicated involvement in every step throughout the process, this project has accomplished the desired results.

Finally, we would like to thank Mr. Thomas Hensel of Krones AG for kindly attending the project demonstration and for providing the remarks.

Dated: 24th May 2017

Contents

Chapter 1: Introduction

The main objective of the project is to detect positions and orientations of bottles that are moving over the conveyor belt. And to detect the disturbances on the conveyor belt, that is, if a bottle has fallen down on the conveyor belt, then the system should detect it in order to avoid any delay in the flow of the running system.

A conveyor belt carries a bulk of the bottles. A situation occurs where several bottles are not in the correct position and orientation, that is, they can be dropped down, damaged etc. These bottles need to be properly detected and picked by the robot from the conveyor belt before they reach the end of conveyor belt.

1.1 Filled Vertical Bottles:

These are the bottles which are not to be picked by the robot. The caps of the bottles need to be detected and the region around the caps within the bottle diameter should be marked as region of no interest (no ROI) in order to avoid false positive detections.

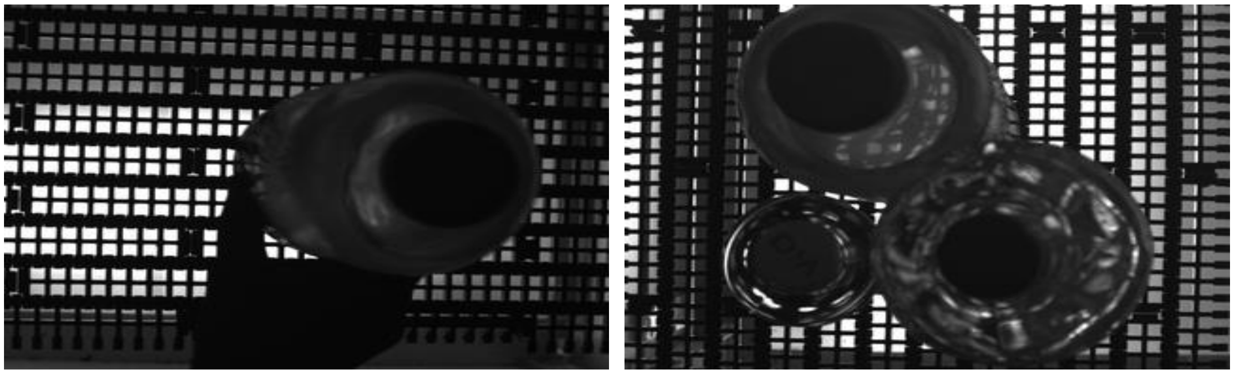


Figure 1.1: Filled Vertical Bottles

1.2 Filled Horizontal Bottles:

These bottles need to be picked by the robot and to be removed from the conveyor belt. The bottles could be placed in any position or orientation. The algorithm detects the laying (horizontal) bottles in a frame and provides the coordinates to the robot in order to remove the bottles. The challenge is to recognize the patterns in the bottle while at the same time ignoring the patterns of the conveyor belt.

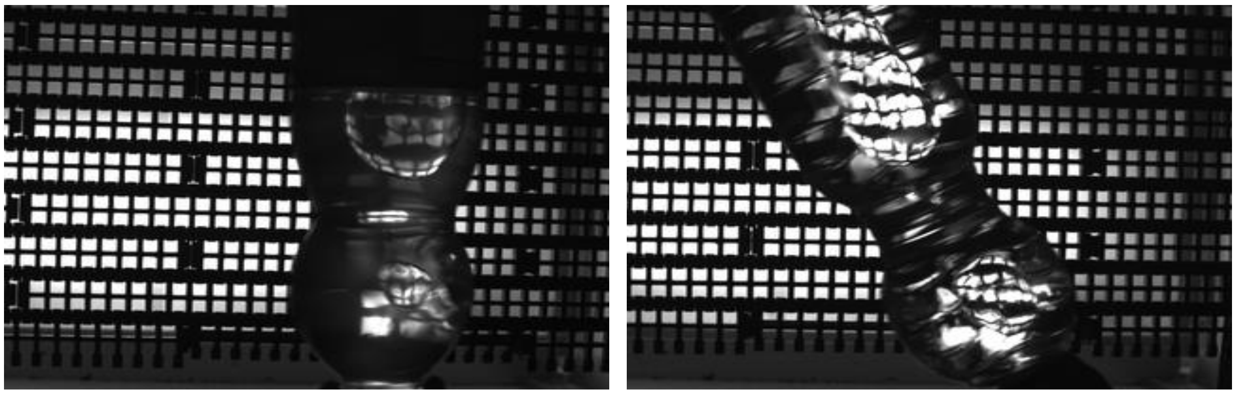


Figure 1.2: Filled Horizontal Bottles

1.3 Dark Bottles:

The light does not pass through the dark bottles. These bottles are composed of glass or darker materials. The ROI contains very few features compared to the background (the conveyor belt). The Morphological Erosion and Dilation are applied to handle the scenario with high efficiency.

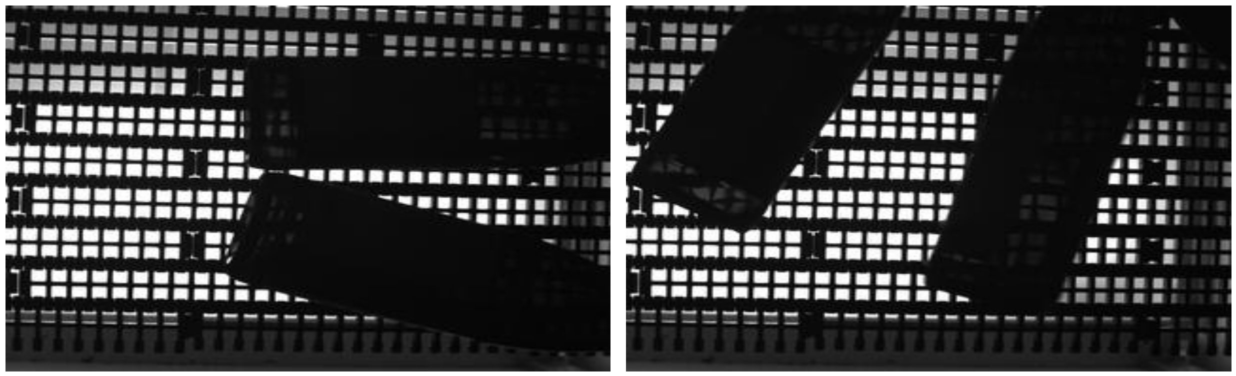


Figure 1.3: Dark Bottles

1.4 Empty Bottles:

These bottles contain no fluid and are usually manufactured with a transparent material. The detection and the tracking of this kind of bottles turned out to be the most difficult scenario. They are indistinguishable from the conveyor belt over which they are placed. A combination of Polarizer and Analyzer is used to extract the features.

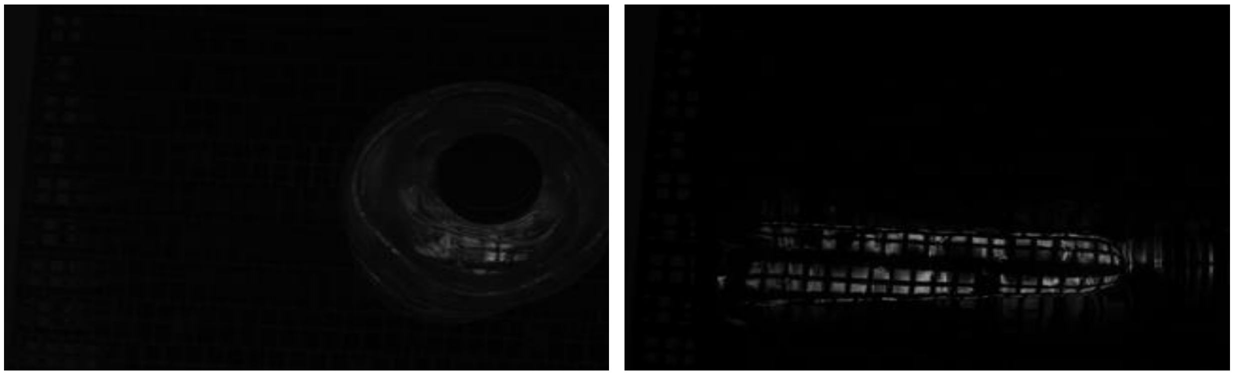


Figure 1.4: Empty Bottles

CHAPTER 2:

Model of the Algorithms

In this chapter, the applied theory and the inner working of the algorithms is explained.

The model and the flow of the algorithm is as follows:

1. Processing of the images
2. Hough Circle Transform:
3. Hough Probabilistic Line Transform:
4. Binary Large Objects (BLOBS) Detection
5. Dark Bottles Detection using Morphological Opening
6. Polarizer and Analyzer for Empty Transparent Bottles

Appendix A is provided for the commentary on the various image processing techniques which are studied and taken into consideration but are not the part of the final algorithm.

2.1 Preprocessing:

The purpose of preprocessing is to reduce the dimensions of an image and to remove the noise. It comprises of two major algorithms.

2.1.1 Cropping:

The aim is to reduce the field of view and thus to reduce the amount of information to be processed by a feature detection algorithm in a single frame.

The edges need to be removed from the frame as they contain no information and contribute in the increase of the noise for the algorithms.

2.1.2 Noise Removal Using Gaussian Blur Filter:

A Gaussian Blur Filter (Low Pass) is used to remove the noise from the image frames. It is represented by a Gaussian Exponential Function:

$$G(x,y) = \frac{1}{2\pi\sigma^2} * e^{-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)}$$

where:

σ^2	: Variance of Gaussian distribution
x	: Distance from origin to X axis
y	: Distance from origin to Y axis

Noise removal is performed by convolution of Gaussian kernel with a pixel and its surroundings. That is, Gaussian kernel is multiplied by a pixel and its eight neighbors. The size of the kernel determines the amount of noise removed and the normalized result is the value of the center pixel of the image. [R1]

A 3x3 Gaussian Blur kernel is:

$$1\,/\,16\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

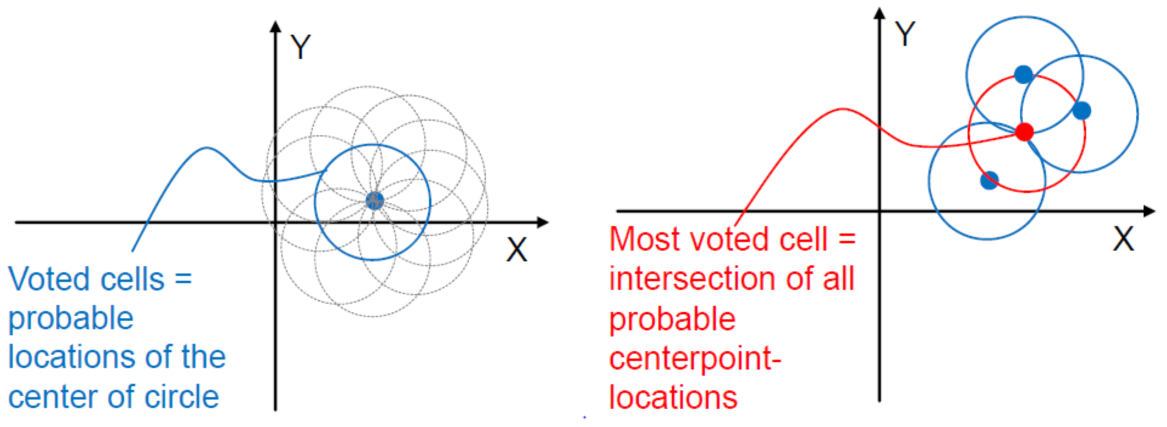
where 1/16 is the normalizer.

2.2 Hough Circle Transform:

The purpose of using this technique is to detect the vertical bottles. Basically, caps of the bottles can be treated as circles and if circles are detected in an image using Hough Circle Transform, it means that there are vertical bottles in the image under observation.

2.2.1 Explanation:

A range of diameters is given in which a circle is to be detected. The XY plane is regarded as a 2D-accumulator for the voting of possible centers of the circles. Each point which has been extracted from an input image votes for the probable center points of all circles which pass through this point. Several points vote for several potential circle centers.


Y
X
Voted cells = probable locations of the center of circle
Y
X
Most voted cell = intersection of all probable centerpoint-locations

i) Probable location of circles ii) Intersection of circle center points

Figure 2.1: Hough Circle Transform [R2]

Figure (i) is showing the probable locations of the centers of circles which is used for voting in a 2D accumulator. Figure (ii) shows the intersection point of most of the centers of circles which is a result of voting. The point with the most number of intersections is considered as a center point of circle.

2.2.2 Results:

1. This algorithm works very well for the detection of vertical bottles in an image under the observation.
2. By using Hough Circles Transform with the combination of dark BLOBS detection, we can detect vertical bottles with high accuracy.

2.3 Hough Probabilistic Line Transform:

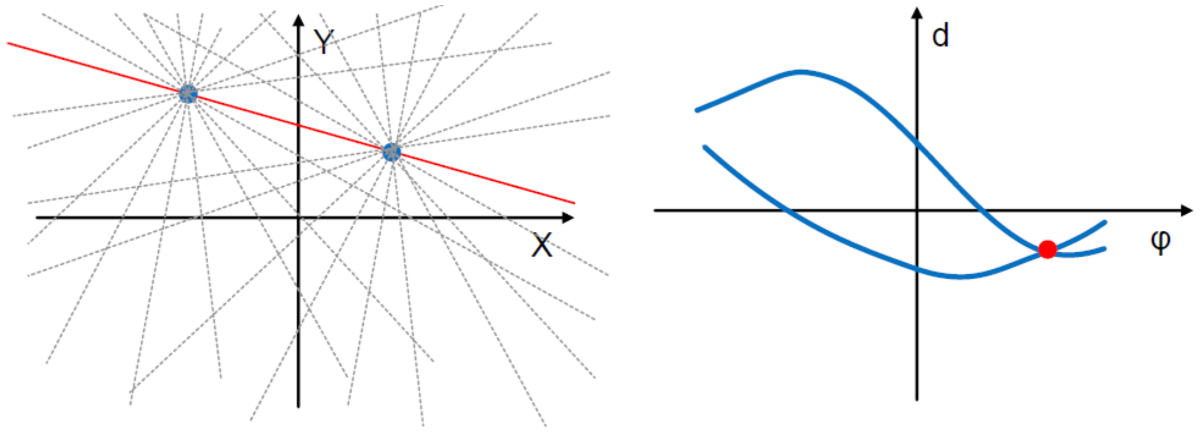
It identifies the edges in an image where edges are referred as the points of sudden changes in a Grayscale value.

The technique is used to detect the presence of bottles and their coordinates on a moving conveyor belt with the help of a LASER triangulation. The edges of a LASER line are computed to detect the bottles.

2.3.1 Explanation:

A straight line in XY domain is represented by a point in dθ domain. Similarly, a point in XY domain is represented by a curve in dθ domain. The collinear points in XY domain are represented by an intersection of curves in dθ domain.

Therefore, a 2D accumulator grid is created and each cell is incremented by 1 when a curve passes through it. At the end, the cell with the highest value represents the intersection point in dθ domain using voting algorithm.



i) A Point in X-Y Plane ii) Curve in dθ domain

Figure 2.2: Hough Line Transform [R3]

*Figure shows the mapping of two collinear points from image space (x, y) on the left to Hough space (d, φ) on the right. Note that each colored point in image space corresponds to a sinusoidal curve of the same color in Hough space, where each point on the curve represents a line through the point in image space at a different angle. All curves intersect at one point in the φ-d domain. This point is called the **Line Representing Point**.*

LASER line scatters after striking with the bottles on the conveyor belt into several lines. By applying the Hough Probabilistic Line Transform, it is possible to get two distinct points of each line. In this way, we can get the coordinates of the bottles in spatial domain using

Hough Transform. If LASER line is scattered, it means that there are some bottles which are either vertical or horizontal.

2.3.2 Results:

1. With the help of this algorithm, it is possible to detect the orientation and the position of only the vertical and the transparent horizontal bottles.
2. It is not possible to detect an empty transparent plastic bottle.
3. It is not possible to detect a dark bottle because of the reason that dark bottles absorb LASER light and no reflections are observed.

2.4 Binary Large Objects (BLOBS) Detection:

BLOB is an acronym for Binary Large Objects. They represent a group of connected pixels in a binary image. The term 'Large' is often used to indicate that the objects of a certain size are of interest and smaller objects are considered as a noise.

The idea behind using this technique is to detect blobs in an image. A conveyor belt with some specific pattern is used along with back light source to create blobs in image. Each confined object in an image can be treated as a blob. Different blobs criteria's can be applied to detect horizontal and vertical bottles.

2.4.1 Explanation of the algorithm:

The purpose of blobs extraction is to isolate the desired blobs from a given image. The connectivity is defined as if two pixels are connected to their neighboring pixels. [R4]

The algorithms used for extractions are:

1. Recursive Grassfire Algorithm
2. Sequential Grassfire Algorithm

Blobs can be classified as follows:

Color Criterion — This criterion compares the intensity of the binary image at the center of blob. Dark blobs are detected by setting color parameter to 0. Bright blobs are detected by setting color parameter to 255.

$$0 \leq Color \leq 255$$

Area Criterion — Desired blobs with specific area can be obtained by setting a range of area. In that case only those blobs are detected that are in defined area range. *[R5]*

$$A = \pi r^2$$

Circularity Criterion — Blobs can be classified on the basis of shapes. Blobs can be circular, rectangular, square or of other shapes.

$$Circularity = \frac{4\pi Area}{perimeter^2}$$

Inertia Criterion — It determines that a blob either will be circle, ellipse or parabola. By setting inertia parameter it is possible to get the blob with desired inertia specifications.

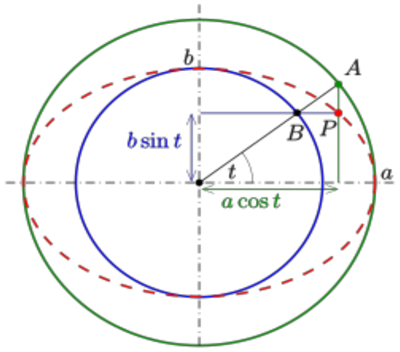


Figure 2.3: Inertia of a Circle [R6]

Convexity Criterion — Convexity determines how much percentage of noise tolerance can be acceptable in the extraction of blobs with selected shape criteria. By setting convexity to 1 means there will be zero noise tolerance in the extraction of the blobs with specified shape. By setting convexity to 0.9 means 10 percent noise in the extraction of blobs with desired shape criteria is acceptable.

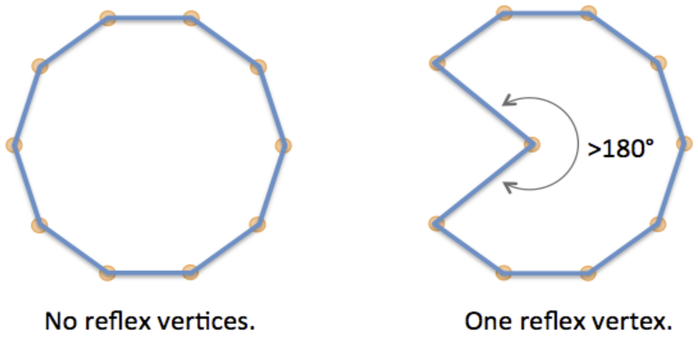


Figure 2.5: Convexity of a Circle [R7]

2.4.2 Extraction of the Blobs:

In this step, the blobs are filtered from the result of all the available blobs. It is usually done by setting the criterion described above. In our working, the algorithm selects the optimal criteria for area, convexity, inertia and color.

Extraction of blobs based on area is the most important parameter for our algorithm. Initially, the system is trained by calculating an average blob area of all the available blobs according to the pattern of the conveyor belt in the absence of any bottle. This average area parameter is then stored in the working algorithm. When the light passes through the transparent bottles, the size of blobs in a bottle is larger than the trained blob size.

Now, a range for the area criteria is defined which is larger than the initial trained average blob area. Any blob within the selected area range can be detected and all others are considered as a noise. In this case, the bright blobs are extracted.

2.4.3 Results:

1. By using Blobs Extraction, we can detect vertical bottles. Caps of bottles are dark and can also be detected by dark blobs criteria.
2. Dark bottles can be detected by using Blobs Detection with Morphological Opening.
3. Empty bottles can be detected by using Blob Detection with the combination of Polarizer and Analyzer.
4. The Blob Detection technique works with high accuracy. All the detections are done in real time. It is a very fast algorithm for detection.

2.5 Dark Bottles and Morphological Opening:

First of all, morphological opening is applied on the captured images to eliminate the pattern of conveyor belt from the images. This is done in order to eliminate all the blobs from the image background, that is, segmentation.

Now, for the detection of dark bottles the criteria of extracting the blobs is based mostly on the color in combination with other criteria as well. In this case, we are extracting dark blobs and not the bright blobs. Each isolated object in image which is dark can be considered as a blob which in our case is a dark bottle. Then by setting area range we can extract dark blobs to detect dark bottles.

2.6 Empty Bottles with Polarizer and Analyzer:

The transparent empty horizontal bottles are the most difficult case to detect. For this, the bright blobs extraction in combination with a polarizer and an analyzer is used. The analyzer installed on the camera lens captures the light of only a specific wavelength and all other wavelengths are discarded.

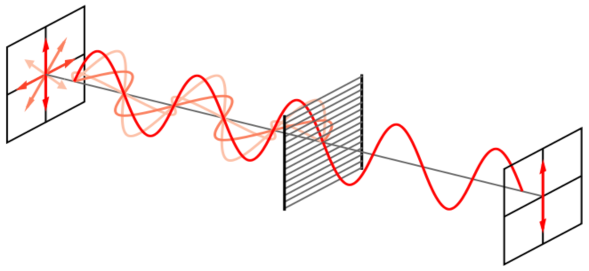


Figure 2.6: Empty Bottles with Polarizer and Analyzer [R8]

When back light passes through empty transparent bottles, light of specific wavelength is emitted which is captured by analyzer. Afterwards the blobs are detected by using area range with convexity, color and inertia.

Chapter 3:

Development Environment:

3.1 C++ Programming Language:

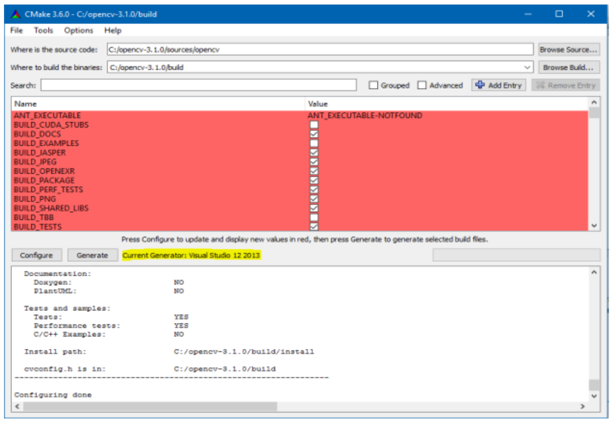
With the consideration of real time requirements and the future scope for running the algorithms in the resource constrained Embedded Environment, the implementation is done in C++ 11 standard with the usage of new features not available in the prior C++ standards, namely, Range based Iterators, Auto Specifiers etc.

The reason to opt for C++ is that OpenCV library is implemented natively in C++, so using C++ provides raw performance without having to use the wrappers or any bridge APIs (for example `opencv-python`).

3.2 OpenCV Computer Vision Library:

OpenCV 3.2 is used for the implementation of the project. The current release of the library is compiled directly from the source code due to the reason that the repositories for macOS (`homebrew`) and Ubuntu (`apt`) usually contains the old versions of the library.

For Windows, CMake GUI tool can be used with Visual Studio 2013 as Generator Target.


CMake 3.6.0 - C:/opencv-3.1.0/build

Where is the source code: C:/opencv-3.1.0/sources/opencv

Where to build the binaries: C:/opencv-3.1.0/build

Name	Value
ANT_EXECUTABLE	ANT_EXECUTABLE-NOTFOUND
BUILD_CUDA_STUBS	☐
BUILD_DOCS	✓
BUILD_EXAMPLES	☐
BUILD_JASPER	✓
BUILD_JPEG	✓
BUILD_OPENEXR	✓
BUILD_PACKAGE	✓
BUILD_PERF_TESTS	✓
BUILD_PNG	✓
BUILD_SHARED_LIBS	✓
BUILD_TBB	☐
BUILD_TESTS	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 12 2013

```
  Documentation:
    Doxygen:                    NO
    PlantUML:                   NO

  Tests and samples:
    Tests:                      YES
    Performance tests:          YES
    C/C++ Examples:             NO

  Install path:                 C:/opencv-3.1.0/build/install

  cvconfig.h is in:             C:/opencv-3.1.0/build
-----------------------------------------------------------------

Configuring done
```

Figure 3.1: Compiling OpenCV in Windows 10

On macOS and Linux, OpenCV can be compiled using CMake command line tool:

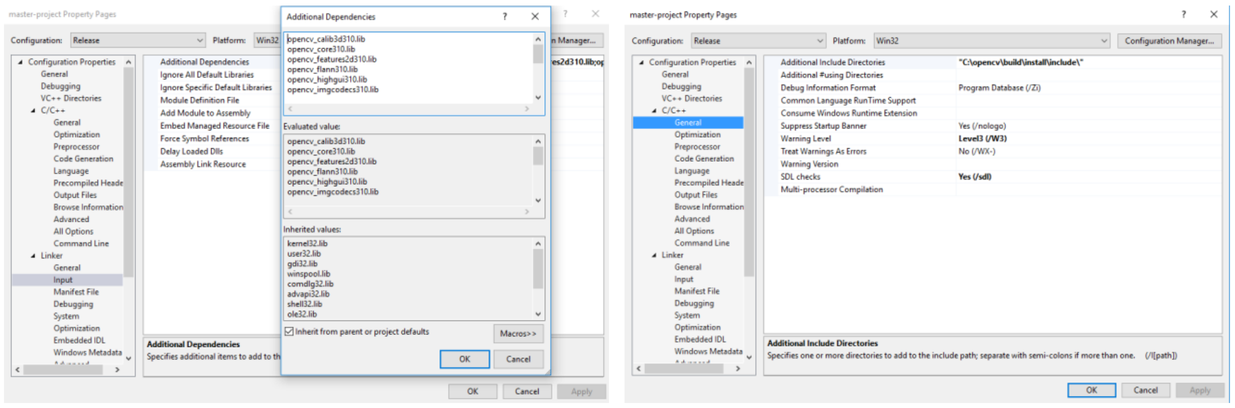
```
wget https://github.com/opencv/opencv/archive/3.2.0.zip
# after unzipping and cd into the directory
cd build
cmake -DWITH_QT=ON -DWITH_OPENGL=ON -DFORCE_VTK=ON -DWITH_TBB=ON -
DWITH_GDAL=ON -DWITH_XINE=ON -DBUILD_EXAMPLES=OFF ..
make -j4 && sudo make install && sudo ldconfig
```

Listing 3.1: Compiling OpenCV from source

3.3 Linking and CMakeLists:

To compile the project correctly, OpenCV needs to be linked so that the compiler can find the references to the library during compilation.

For Visual Studio, project properties need to be configured to set the `include` and `lib` paths of OpenCV.



Configuration: Release   Platform: Win32

Additional Dependencies
Ignore All Default Libraries
Ignore Specific Default Libraries
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Delay Loaded Dlls
Assembly Link Resource

Additional Dependencies

```
opencv_calib3d310.lib
opencv_core310.lib
opencv_features2d310.lib
opencv_flann310.lib
opencv_highgui310.lib
opencv_imgcodecs310.lib
```

Evaluated value:

```
opencv_calib3d310.lib
opencv_core310.lib
opencv_features2d310.lib
opencv_flann310.lib
opencv_highgui310.lib
opencv_imgcodecs310.lib
```

Inherited values:

```
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
advapi32.lib
shell32.lib
ole32.lib
```

☑ Inherit from parent or project defaults

Additional Dependencies
Specifies additional items to add to th

Additional Include Directories	"C:\opencv\build\install\include\"
Additional #using Directories	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Warning Version	
SDL checks	Yes (/sdl)
Multi-processor Compilation	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

Figure 3.2: Linking OpenCV in Visual Studio 2015 [R9]

On macOS and Linux, to compile project with `LLVM Clang Compiler` or with `GCC Compiler`, CmakeLists.txt can be authored as follows:

```
cmake_minimum_required(VERSION 3.7)
project(opencv_run)

set(OpenCV_DIR "/usr/local/Cellar/opencv3/3.2.0/include/")
set(CMAKE_CXX_STANDARD 11)

set(SOURCE_FILES main.cpp)
add_executable(opencv_run ${SOURCE_FILES})

find_package(OpenCV REQUIRED calib3d core highgui imgproc features2d)
include_directories(${OpenCV_INCLUDE_DIRS})

set(SOURCE_FILES main.cpp)
set(OpenCV_LIBS opencv_calib3d opencv_features2d opencv_core
opencv_imgproc opencv_highgui opencv_imgcodecs)
target_link_libraries(opencv_run ${OpenCV_LIBS})
```

Listing 3.2: Linking OpenCV in macOS and Linux

3.4 IcoNET:

IcoNET is a software built by Professor Wagner and his colleague in Hochschule Rosenheim. The tool is primarily used to acquire the images from the camera using an automated script.

The script captures hundreds of frames over the period of time in order to get the information about the movements and the orientations effects of the objects on the performance of the algorithm.

```
GFLOAT( #CamCommand )
GFLOAT( #CamCommandValue )
GFLOAT( #NImages )
GIMAGE( #CamImage )
GSTRING( #ImageNumberStr )
GSTRING( #ImageBaseName )
GSTRING( #ImageFileName )
GFLOAT( #Stop )
GFLOAT( #Counter )
SETFLOAT( #Stop; 0 )
SETFLOAT( #Counter; 0 )
SETFLOAT( #CamCommand; 5 )            // set exposure time
SETFLOAT( #CamCommandValue; 20000 )
EXECUTEBOX( 1 )
SETFLOAT( #CamCommand; 0 )            // start capturing mode
SETFLOAT( #CamCommandValue; 0 )
EXECUTEBOX( 1 )
SETSTRING( #ImageBaseName; Images\\Img_ )
WHILE( #Stop; ==; 0 ) {
  FORMATFLOAT( #ImageNumberStr; #Counter; 0 )
  SETSTRING( #ImageFileName; #ImageBaseName )
  STRCAT( #ImageFileName; #ImageNumberStr )
  STRCAT( #ImageFileName; .bmp )
  SETFLOAT( #CamCommand; 1 )          // Trigger
  EXECUTEBOX( 1 ) // send trigger
  SETFLOAT( #CamCommand; 3 )          // # of images in buffer
  SETFLOAT( #NImages; 0 )
  WHILE( #NImages; ==; 0 ) {
     EXECUTEBOX( 1 )
     GUIUPDATE()}
  SETFLOAT( #CamCommand; 4 )          // load image from buffer
  EXECUTEBOX( 1 )
  PROTOSAVEIMG( #ImageFileName; #CamImage )
  IF( #Counter; >; 2000 ) {
     SETFLOAT( #Stop; 1 )}
  ADD( #Counter; #Counter; 1 )
  GUIUPDATE()
}
SETFLOAT( #CamCommand; 2 )            // stop capturing mode
SETFLOAT( #CamCommandValue; 0 )
EXECUTEBOX( 1 )
```

Listing 3.3: IcoNET

Chapter 4: Development of the Algorithms:

This chapter provides the implementation details and the results of the algorithms. Each step is accompanied by the source code and the effects on the images.

The algorithms are comprised of the following:

1. Fetch Images from Directory
2. Reduce Field of View
3. Reduce Image Density
4. Morphological Opening
5. Parameters for the Blobs
6. Blob Detection
7. Hough Probabilistic Line Transform
8. Hough Circle Transform

4.1 Fetch Images from Directory:

The implementation is capable of applying the algorithms on a continuous stream in the real time speed by fetching the images via IP address. This speeds up the development iteration, several hundred images of different real world scenarios are fetched and dumped in a directory and the program is executed on these images.

Future improvement can be to integrate Vimba SDK to fetch the stream directly from the camera through Ethernet port instead of capturing the images over the air using IP address.

```
void fetchImagesFromFolder(vector<Mat> &data, const string path) {
      vector<String> _fn_;
      glob(path, _fn_, true);              // recursive
      for (size_t k = 0; k < _fn_.size(); ++k) {
            cv::Mat im = cv::imread(_fn_[k]);
            if (im.empty()) continue;      // only proceed if empty
                  data.push_back(im);
      }
}
```

Listing 4.1: Fetch Images from Directory

Figure 4.2: Fetch Images from Directory

4.2 Reduce Field of View:

The visibility cone of the camera is very wide. Dimensions of the frames are reduced by cropping the rectangular surface out of each frame [R10].

```
void BottleDetection::getROI(Mat &referenceImage, const int x,
     const int y, const int width, const int height) {
          Rect Rec(x, y,                           // x, y coordinates
                 width,                            // width
                 height                            // height
          );
          referenceImage = referenceImage(Rec);// ROI
}
```

Listing 4.2: Get Region of Interest

4.3 Reduce Image Density:

Gaussian Blur filter is applied on the frames, followed by thresholding. The combination of the filtering and the thresholding is a very important process and several combinations of the parameters are applied while training and testing the algorithm on a range of real world scenarios. The parameter tuning is not shown here for conciseness. [R11]

```
Mat reduceDensity(const Mat algorithmImage,    // image for operation
         const unsigned minThreshValue,        // minimum threshold
         const unsigned method,                // binary, inverse
         const unsigned filterKernelSize) {    // kernel size
            Mat resultImage;
            algorithmImage.copyTo(resultImage);
            Mat median;
            medianBlur(resultImage, median,    // source, destination
                       filterKernelSize);      // aperture, odd & >1
            threshold(median, resultImage, minThreshValue, 255);
            return resultImage;
}
```

Listing 4.3: Reduce Image Density

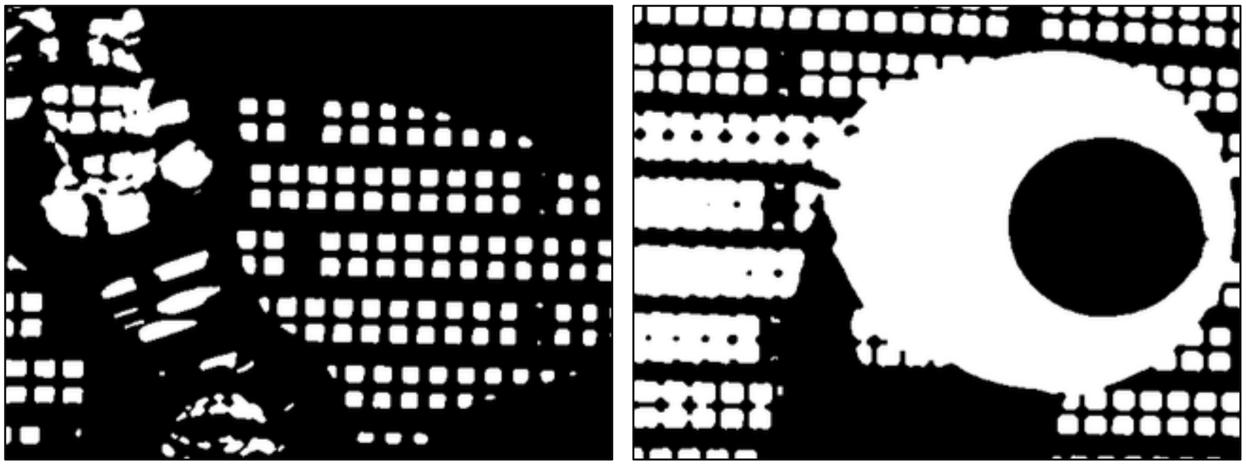


Figure 4.3: Reduce Image Density

After this, range of gray scale is reduced from (0..255) to (0..1), we have a binary image.

4.4 Morphological Opening:

In the case of the dark bottles, additional layer of preprocessing is applied. Opening is the combination of Morphological Erosion and Dilation. A structuring element of 5 pixels (in both dimensions) is applied on each frame. This technique yields the results with a lot of information density removed and helps to avoid applying complex mathematical checks on the results for false detection removal.

```
// Create a structuring element (SE)
Mat _element_ = getStructuringElement(
   MORPH_RECT,
   Size(1 * dark_flag.morphSize + 1,
        1 * dark_flag.morphSize + 1),
   Point(dark_flag.morphSize,
          dark_flag.morphSize);
   );
for (unsigned loop = 0; loop < 5; loop++) {
    morphologyEx(detectionImage,                 // source
                 detectionImage,                 // destination
                 MORPH_OPEN,                     // morphology
                 _element_,                      // structuring element
                 Point(-1, -1), 5);              // center
}
```

Listing 4.4: Morphological Opening

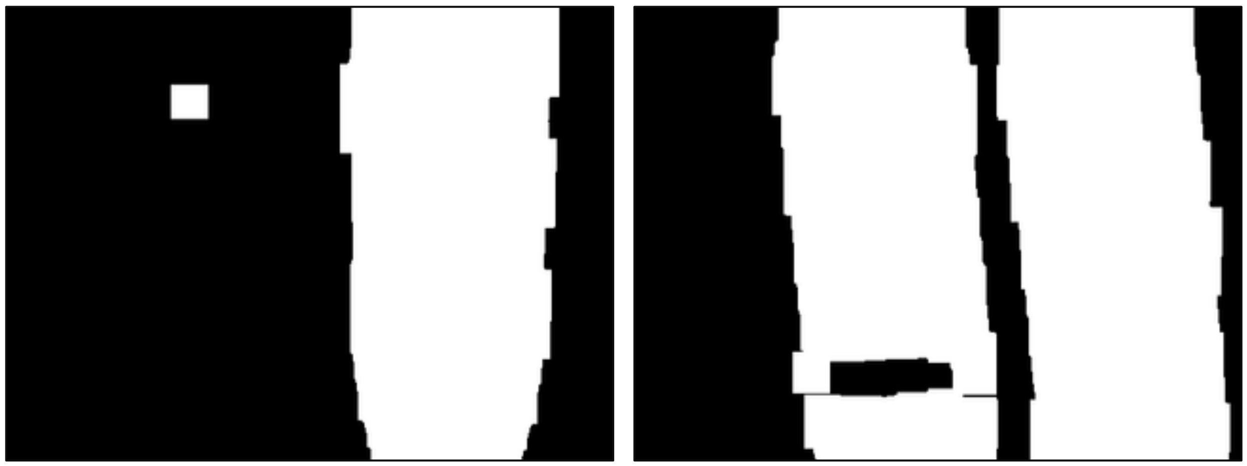


Figure 4.4: Morphological Opening

After Morphological Opening, the information density of an image is significantly reduced allowing for the efficient detection of the Region of Interest (ROI).

4.5 Blob Parameters:

Before applying Blob Detection, the conditions and parameters are tuned for different real-world scenarios. A sample combination of the tuning is shown here for Blob Convexity, Inertia and Area criteria. [R13]

```
SimpleBlobDetector::Params _params_;
_params_.filterByArea = false;
_params_.filterByCircularity = false;
_params_.filterByConvexity = true;  // high convexity i.e. no breakage
_params_.minConvexity = 0.75;       // near to the circle
_params_.maxConvexity = 1.0;
_params_.filterByInertia = true;
_params_.minInertiaRatio = 0.55;     // elongated
_params_.maxInertiaRatio = 1;
// filter blob based on black colors
// threshold is applied in a way
// that it makes caps as black (0,0,0)
_params_.filterByColor = true;
_params_.blobColor = 0;
```

Listing 4.5: Blob Parameters

4.6 Blob Detection:

With the parameters tuned, OpenCV SimpleBlobDetector class is utilized. Detected Blobs are further reduced and filtered using boundary checks and other techniques, A good amount of consideration went into making sure that there should be no false detections. Finally coordinates of the detected blobs are drawn on the image frames as markers.

```
Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(_params_);
vector<KeyPoint> keypoints;                             // keypoints vector
detector -> detect(detectionImage, keypoints);          // blob detection
auto imageSize = detectionImage.size();
for (const auto &p : keypoints) {
bool lowerBoundary = p.size >= _flag_.minBlobArea;
     bool upperBoundary = p.size <= _flag_.maxBlobArea;
     if (lowerBoundary && upperBoundary) {
          const bool xBoundary =
          p.pt.x < imageSize.width - _flag_.boundaryThreshold &&
          p.pt.x > dark_flag.boundaryThreshold;
          const bool yBoundary =
          p.pt.y < imageSize.height - _flag_.boundaryThreshold &&
          p.pt.y > _flag_.boundaryThreshold;
          if (xBoundary && yBoundary) {
               cv::drawMarker(inputImage,                  // image
               Point(p.pt.x, p.pt.y), Scalar(255, 0, 0),// color
               MARKER_CROSS, _flag_.markerSize, flag_.markerThicknes
               );
          }
     }
}
```

Listing 4.6: Blob Detection

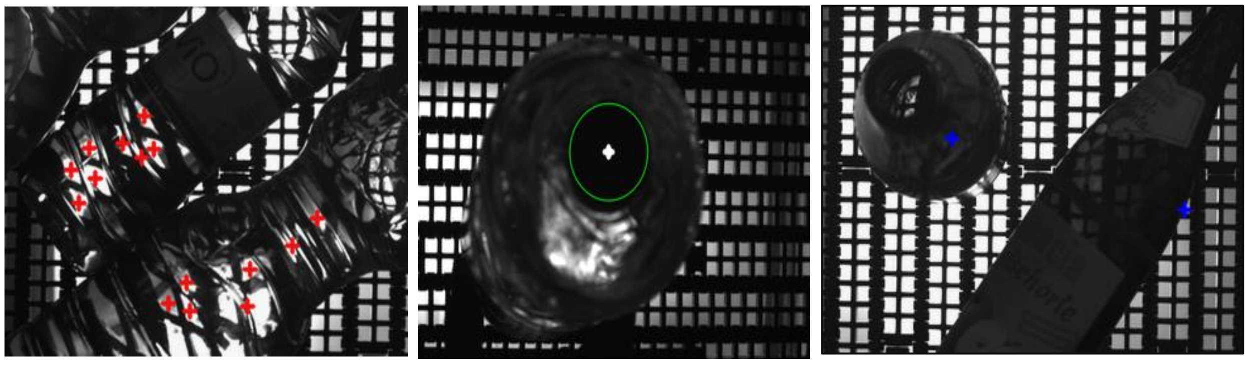


Figure 4.5: Blob Detection

4.6 Hough Probabilistic Line Transform:

This technique is not used in the final algorithm but is very helpful in the initial iterations of the algorithm. Hough Line Transform is applied on the frames and the lines are calculated using Linear Regression. [R14]

```
void BottleDetection::applyProbabilisticHoughTransform(Mat thresh){
     // to save lines from Hough transform
     Vector<Vec4i> is used for probabilistic Hough transform
     vector<Vec4i> lines;
     // it will generate the accumulator cell
     // 2xn matrix containing r->first row and theta->second row
     HoughLinesP(thresh,                          // source
                 lines, 1,                        // x, y, rho (0...2pi)
                 180 * (CV_PI / 180),             // 1-degree res.
                 10,                              // # of intersection
                 10,                              // # of pts. for line
                 10                               // max gap b/w pts
     );
     getROI(inputImage, roi.x, roi.y, roi.width, roi.height);
     // save a copy of inputImage in outputImage
     inputImage.copyTo(outputImage);
     vector<int> lineCoordinates;
     for (size_t i = 0; i < lines.size(); i++) {
          Vec4i singleLine = lines[i];
          line(outputImage,                        // destination
          Point(singleLine[0], singleLine[1]),     // start - x & y
          Point(singleLine[2], singleLine[3]),     // end - x & y
          Scalar(0, 0, 255),                       // color of line
          1, CV_AA );
     }
}
```

Listing 4.7: Hough Probabilistic Line Transform

4.7 Hough Circle Transform:

For Bottle Detection, Hough Circle Transform is applied in combination with Blob Detection to further separate the Region of Interest from the several moving and noise prone areas in an image frame. [R15]

Canny Edge Detector and Radius boundary conditions are tuned to reduce the number of mathematical calculations per frame.

```
void CapDetection::getCapsUsingHough() {
     vector<Vec3f> bottleCaps;                  // determine bottle caps
     // hough circle gives us [0]->x, [1]->y, [2]->radius
     HoughCircles(outputImage, bottleCaps,
                  CV_HOUGH_GRADIENT, 1,
                  outputImage.rows / 2,
                  20, 10,                       // canny parameters
                  minRadius, maxRadius
     );
     // draw the caps
     for (size_t i = 0; i < bottleCaps.size(); i++) {   // 2 caps
          Vec3i cap = bottleCaps[i];
          circle(inputImage,                   // image
                 Point(cap[0], cap[1]),        // x, y of circle
                 cap[2],                       // radius of the circle
                 Scalar(0, 255, 0),            // green color
                 3, 8, 0);
        // draw the center of bottle cap
        circle(inputImage,
                 Point(cap[0], cap[1]), 1,
                 Scalar (255, 255, 255),
                 3, 8, 0);
     }
}
```

Listing 4.8: Hough Circle Transform

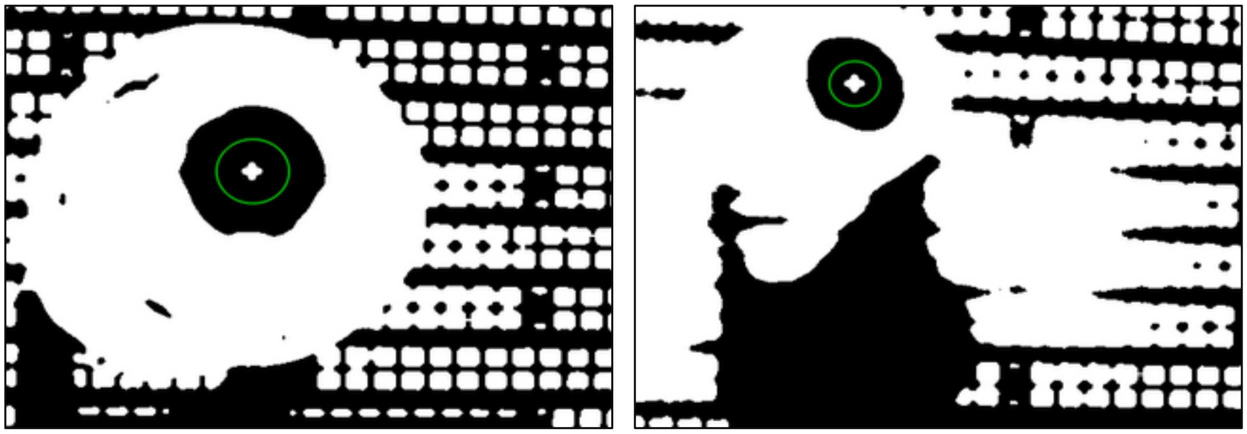


Figure 4.6: Hough Circle Transform

Chapter 5: Hardware Implementation

5.1 Monochrome Camera:

For the image and video capturing, Allied Manta G032B which is package around GigE Vision Camera is used. The camera provides high frame rates (80 fps) when used with Region of Interest (ROI). The camera uses power over Ethernet (POE) as a power source to capture images. [R16]

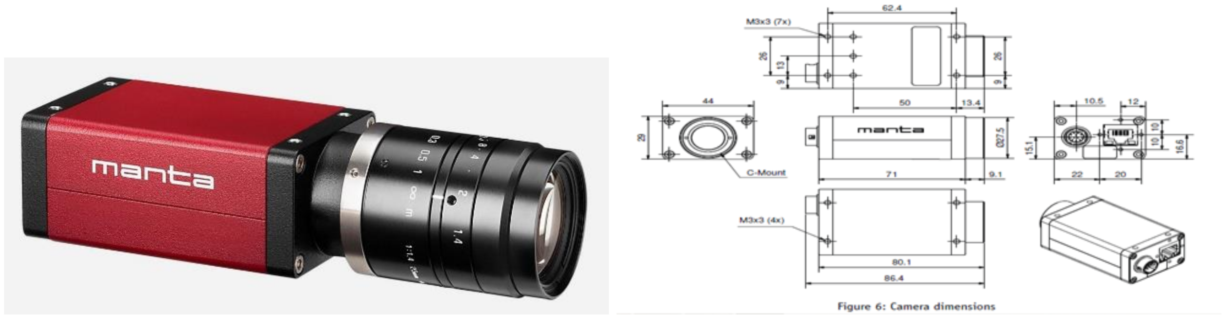


Figure 5.1: Manta G032B with Schematics [R17]

Following are the salient characteristics of the camera:

Features	**Measurements**
Sensor	Sony ICX424 (type 1/3)
Type	CCD Progressive
Resolution	656(h) 494(v)
Pixels	324,064 (0.3 MP)
Max Frame Rate	80.7 fps
Interface	GigE
Exposure Time	25.8 µs to 60 s (Auto/Manual)
Dimensions	86.4 x 44 x 29 mm
Supply	Power Over Ethernet (POE)

Table 5.1: Manta G032B

Vimba SDK from Allied Vision is used to configure the camera, to adjust the exposure time and to automate the image capturing mechanism.

5.2 LASER Source:

The LASER generator consists of a mechanical enclosure compose of different mechanisms and electrical devices, that together control the laser output and the trigger for the camera. The operation consists of reflecting a LASER line on the objective that needs to be identified.

The reflection of this LASER line is held by two mirrors located inside the enclosure. The position of this projected line must be controlled and known due to the principles of triangulation.

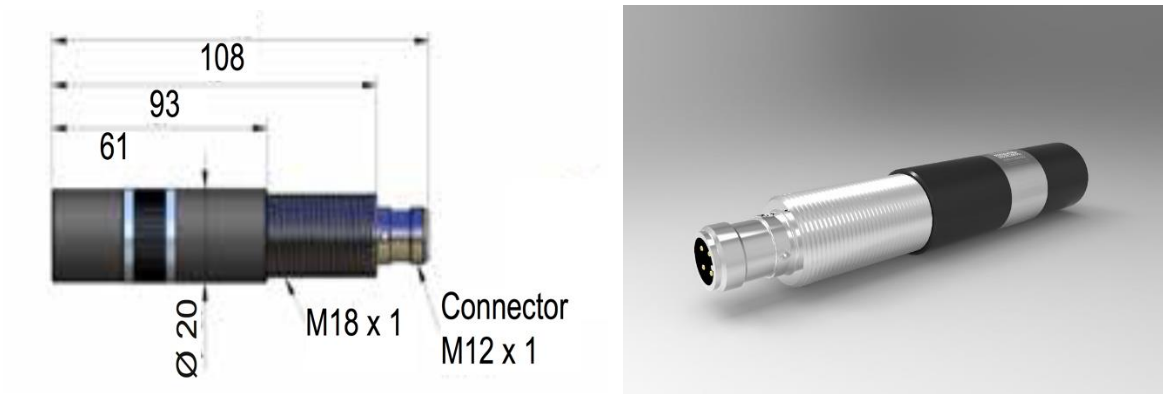


Figure 5.2: LASER with Dimensions [R18]

Following are the salient characteristics of the LASER Source:

Features	**Measurements**
Supply	20mW
Wavelength	635nm, 640nm, 643nm, 685nm, 785nm 808nm, 830nm, 850nm, 980nm
Point	Elliptical or circular
Focus range	100mm up to ∞
Connector	M12 plug, 4-pin

Table 5.2: LASER Specifications

Electrical Connection of LASER ZM-18B is shown below:

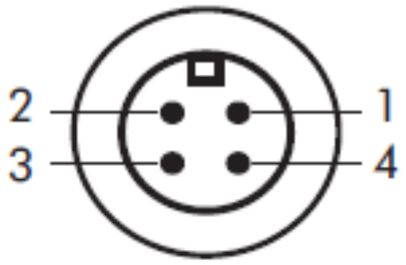


Figure 5.2: LASER Connection Pins

The table below describes the Connection Pins details:

Connectors	**Details**
Pin 1: Voltage Supply (+)	12 VDC
Pin 2: TTL Mode	Enable/Trigger
Pin 3: Voltage Supply (-)	-12 VDC
Pin 4: Analog Mode	Short to Pin 1

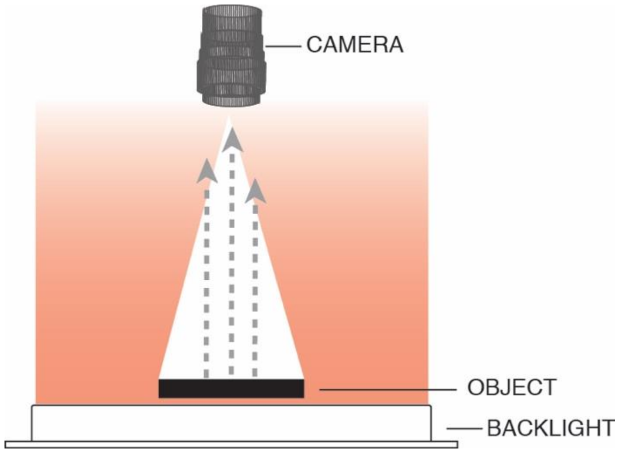
Table 5.3: LASER Connection Details

5.3 Backlight source and Conveyor Belt:

The motivation for using Backlight is as follows:

The Light coming from the source should strike conveyor chain belt first and then pass through the bottle in order to get the magnified patterns of the conveyor chain visible to the camera. In this case, the light passes through the bottles towards the camera.

If overhead light source were to be used, then the light first strikes with the bottle instead of the conveyor chain belt. Also, in this case light will not move towards camera.



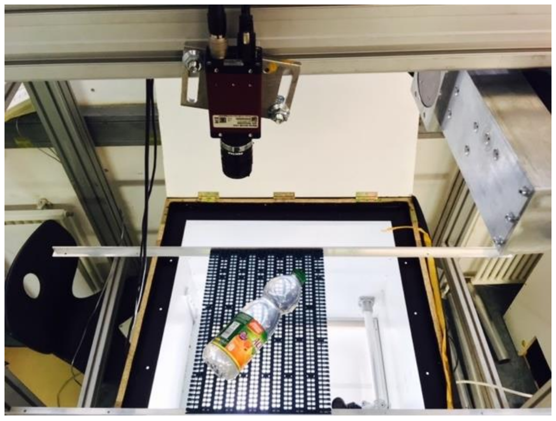


Figure 5.3: Backlight source with camera visible [R19]

In the scope of the project, a specific conveyor belt provided by Krones AG is used which has the perforated patterns to allow the light to be passed through it. Perforation is essential as the light from the backlight source towards the camera should not be blocked by the conveyor belt.

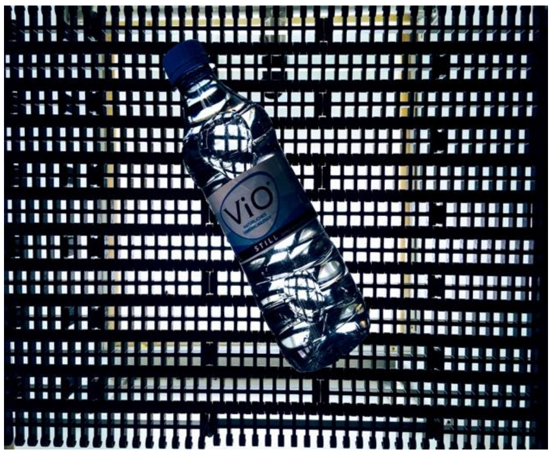


Figure 5.3: Chain Conveyor Belt Patterns as seen by the Camera

Appendix A:
Commentary on the various Algorithms

This Appendix is the commentary on the collection of the image processing techniques which were taken into the consideration and were studied and experimented with, but are not part of the final implementation. Each technique is accompanied by an explanation, goal for the use and results with shortcomings.

It is comprised as follows:

1. Hu's Moments
2. Discrete Fourier Transform
3. LASER Triangulation
4. Marker Based Labelling (Relative Maxima)
5. Template Matching

A.1 Hu’s Moments:

An image moment is a certain particular weighted average of the image pixels or intensities. The technique is used to analyze the movements of object contours in an image and provides 7 key points which are called Hu’s Seven Moments Invariants. The Hu’s Moments [R20] are defined as:

$$\begin{aligned}
hu[0] &= \eta_{20} + \eta_{02} \\
hu[1] &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
hu[2] &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
hu[3] &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
hu[4] &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
hu[5] &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
hu[6] &= (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}$$

where η_{ji} stands for $\texttt{Moments::nu}_{ji}$.

A.1.1 Goal for The Use:

The purpose of using the algorithm is to analyze the pattern of movement of objects at different positions. It can be done by calculating the key points for the movements of bottles of different position and orientations over a moving conveyor belt. The moments can be analyzed by computing Hu’s 7 Moments Invariants.

A.1.2 Results:

1. Hu’s Moments give high accuracy when there is only one object in an image whose moments have to be calculated.
2. In case of our project, there are multiple bottles in an image and that too are of different orientations and positions. This resulted in low accuracy and was not sufficient to achieve the results effectively.

A.2 Discrete Fourier Transform:

Discrete Fourier Transform (DFT) transforms an image from spatial domain to frequency domain. It is used to decompose an image into its sine and cosine components. It provides the frequency components of an object within an image.

A.2.1 Goal for The Use:

The purpose is to detect bottles' positions and orientations by computing their frequency components with the help of DFT. The idea is that frequency components of vertical or horizontal bottles should have some high peaks in their spectrum which can be helpful in the detection of bottles.

A.2.2 Explanation:

Images are captured using the combination of ambient light with sharp red LASER. The camera and the LASER source are installed above the conveyor belt at a certain height. The purpose of using LASER is to have a sharp light over the bottles which in turn produces a sharp spectrum in frequency domain.

DFT is applied [R21] on the captured images to convert them from spatial to frequency domain.

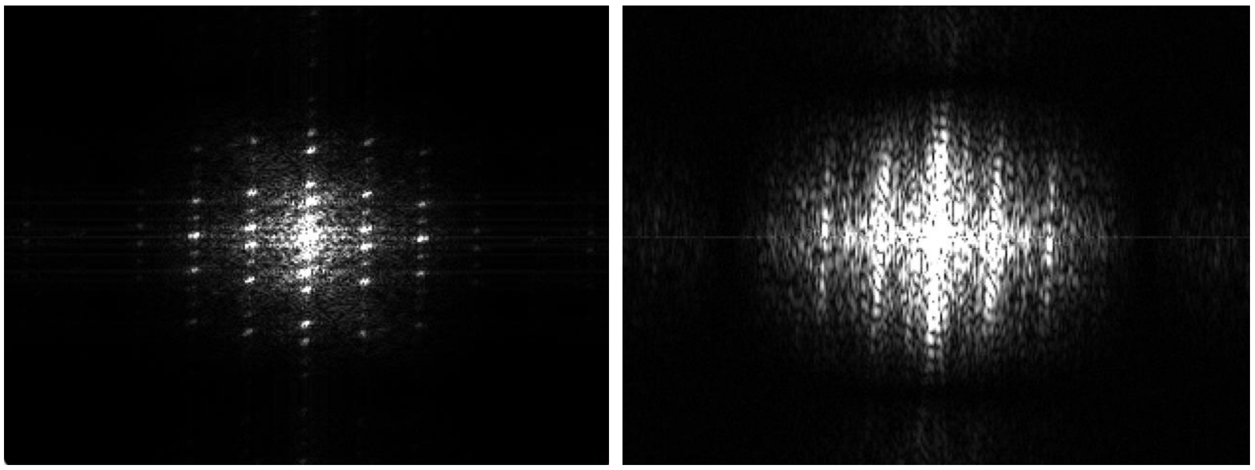


Figure A.1: Discrete Fourier Transform

A.2.3 Results:

1. Horizontal dark bottles can be detected by this technique with low accuracy.
2. Expectation is to get sharp point or few points in the frequency spectrum because of sharp laser line. But after transformation the resulted spectrum is normally quite irregular and difficult to interpret.
3. The spectrum varies significantly for same orientation of bottle at different positions. Because of the high randomness of frequency spectrum, it is not possible to achieve acceptable results from DFT.

A.3 LASER Triangulation:

In LASER triangulation, a camera determines the position/distance of the target by measuring the reflected light from the target surface.

A.3.1 Goal for The Use:

The purpose is to compute the height of vertical and horizontal bottles in an image and to detect whether a bottle has vertical or horizontal orientation in an image. This can be done by using sharp red laser light with the combination of camera and ambient light.

A.3.2 Explanation:

The camera and LASER source are installed above the conveyor belt. The distance between camera and conveyor belt is known. The angle between LASER and camera is 15 degrees to form a cone which will cover any horizontal bottle between two adjacent vertical bottles. Light strikes the cap or surface of bottle and reflects back to camera.

In the case of vertical bottles, distance between the cap/surface of bottle and camera is less than the distance between camera and the surface of the horizontal bottles.

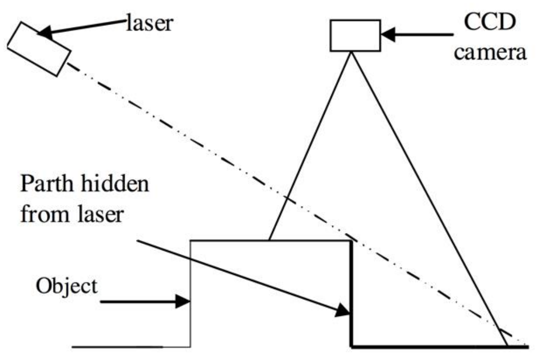


Figure A.2: Laser Triangulation [R22]

Figure shows a setup for triangulation with LASER and Camera. In the presence of bottles over conveyor belt, a portion of the belt is hidden from camera. This information can be manipulated for the distance calculations.

The distance between camera and conveyor belt and the height of the bottles are always known. By setting a threshold value for the height difference, it is possible to detect the horizontal and the vertical bottles. The coordinates in spatial domain are obtained using

Hough Probabilistic Line Transform. It provides the starting and the endpoints of the detected line segments in the given frame.

A.3.3 Results:

1. It is possible to detect the vertical and the horizontal filled bottles but not the empty plastic bottles.
2. In the case of transparent bottle, LASER passes through the bottle without any reflections and the distance calculations are not sufficient.
3. It is not possible to detect dark bottles as they absorb LASER without sufficient reflections causing the distance calculation to be incorrect.
4. The technique is insufficient to perform the detections effectively.

A.4 Marker Based Labelling:

A.4.1 Goal for The Use:

The basic purpose of using this technique is to detect horizontal bottles. The aim is to calculate relative maxima of illuminations in an image for the bottles. If there are more than one relative maxima in an image, it means that there are some bottles present over the conveyor belt.

A.4.2 Explanation:

A sharp red LASER with the combination of ambient light is aimed at bottles over a conveyor belt causing the LASER to split into several small lines after falling over the bottles.

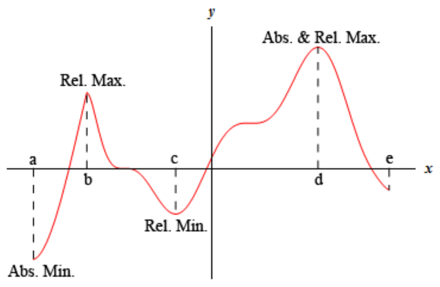


Figure A.3: Relative Maxima and Minima [R23]

Now, we have more than one relative maxima illuminations in an image as compared to only one maxima in an image in case of no bottles over conveyor belt. If there are more than one relative maxima in an image, it means that there are some bottle(s) over the conveyor belt.

A.4.3 Results:

1. The horizontal bottles as well as the empty bottles can be detected with this technique but with low accuracy. Because LASER will pass through the empty transparent bottles without any reflections.
2. It fails in case of dark bottles. Dark bottles absorb light and there is only 1 maxima illumination in image. It provides low accuracy for empty transparent bottles.
3. Because of these reasons, this technique is not sufficient enough to perform detections effectively.

A.5 Template Matching:

A.5.1 Goal for The Use:

Goal is to take image of conveyor belt without any bottle and teach system initially. Afterwards difference between teach and actual images (with bottles) is computed. If there is difference between these two images, then it can be assumed that there are some bottles present on conveyor belt.

A.5.2 Explanation:

The template is matched pixel by pixel by moving over image pixels in the actual image as in 2D convolution and compares the template with actual image. Images with bottles on the moving conveyor belt are captured. Afterwards the difference between actual image and the template is computed. If there is difference between template and the actual image, then it shows there are some bottles present on the conveyor belt. *[R24]*

A.5.3 Results:

1. The technique is error prone because conveyor belt is always in motion and there are always some difference between actual and reference image.

2. It requires a lot of complex calculations to fit the frames of template and actual image perfectly over each other which makes it relatively slower. It is not suitable for real time detections.

3. The detected results are of low accuracy. Hence, it is not suitable to pursue this technique further.

Appendix B: Development in Python Programming Language

A small subset of the project is also implemented in Python programming language. The goal is to reduce the configuration and setup overhead required with the usage of C++ programming language.

This implementation is primarily used to fetch the video stream from a smartphone camera (specifically, Google Nexus 6P and iPhone SE) and analyze the results without requiring the complete setup of the project. It allows to quickly iterate and identify the issues in the algorithm before implementing in C++ with proper class structure.

It should be noted that Python is not used for the full implementation of the project as it is an interpreted language with the runtime overhead and as such does not provide the speed and memory footprint requirements achieved by the usage of C++.

B.1 Fetch images from the directory:

```python
def fetch_images_from_folder(folder):
  _images = []
  for root, dir_names, file_names in os.walk(folder):
      for filename in fnmatch.filter(file_names, '*.bmp'):
          img = cv.imread(os.path.join(folder, os.path.join(root,
                                              filename)))
          if img is not None: _images.append(img)
  return _images
```

Listing B.1: Fetch images from the directory

B.2 Streaming video from IP Address:

```python
 stream = requests.get(ip_address, stream=True)
 if stream.status_code == 200:
   data_bytes = bytes()
   for chunk in stream.iter_content(chunk_size=2048):
       data_bytes += chunk
       a, b = data_bytes.find(b'\xff\xd8'),data_bytes.find(b'\xff\xd9')
       if a != -1 and b != -1:
           jpg = data_bytes[a:b + 2]
           data_bytes = data_bytes[b + 2:]
           frame = cv.imdecode(fromstring(jpg, dtype=uint8),
                   cv.IMREAD_COLOR)
           perform_algorithm(frame)
           if cv.waitKey(1) == 27:
               exit(0)
```

Listing B.2: Streaming video from IP Address

B.3 Reduce Image Density:

```python
@staticmethod
def get_region_of_interest(_reference_image, _x , _y, _width, _height):
  return _reference_image[
          int(_y):int(_y + _height), int(_x):int(_x + _width)]
@staticmethod
def reduce_density(_image, _min_thresh_value, _filter_kernel_size,
                   _method, caller):
  _image = cv.medianBlur(_image, _filter_kernel_size)
  ret, _image = cv.threshold(_image, _min_thresh_value, 255, _method)
  return _image
```

Listing B.3: Reduce Image Density

B.6 Dark Bottle Detection:

```python
@staticmethod
def get_dark_bottles(self, params):
  self.output_image = self.reduce_density(self.output_image,
                      dark_flag["min_threshold_value"],
                      dark_flag["filter_kernel"], cv.THRESH_BINARY_IN)
  element = cv.getStructuringElement(cv.MORPH_RECT,
      (1 * dark_flag["morph_size"] + 1,1 * dark_flag["morph_size"] + 1))
  for index in range(5):
      self.output_image = cv.morphologyEx(self.output_image,
                                          cv.MORPH_OPEN, element)
  detector = cv.SimpleBlobDetector_create(params)
  key_points = detector.detect(self.output_image)
  height, width, channel = self.output_image.shape
  for p in key_points:
      lower_boundary = p.size > dark_flag["min_blob_area"]
      upper_boundary = p.size < dark_flag["max_blob_area"]
      if lower_boundary and upper_boundary:
          x, y = p.pt[0], p.pt[1]
          x_boundary = x > dark_flag["tolerance"] and abs(
              x - width) > dark_flag["tolerance"]
          y_boundary = y > dark_flag["tolerance"] and abs(
              y - width) > dark_flag["tolerance"]
          if x_boundary and y_boundary:
              cv.drawMarker(self.input_image,
                            (int(x), int(y)),
                            (255, 0, 0),cv.MARKER_CROSS,
                            dark_flag["marker_size"],
                            dark_flag["marker_thickness"])
```

Listing B.6: Dark Bottle Detection

B.5 Hough Line Transform:

```
def apply_probabilistic_hough_transform(self, thresh, with_wait_key):
  hough_lines = cv.HoughLinesP(
                             thresh, 1,
                             180 * (np.pi / 180), 10, 10, 10 )
  self.output_image = self.input_image.copy()
  for hough_line in hough_lines:
      for x1, y1, x2, y2 in hough_line:
          cv.line(
              self.output_image,
              (x1, y1),
              (x2, y2),
              (0, 0, 255),
              1)
```

Listing B.5: Hough Line Transform

B.4 Cap Detection:

```
detector = cv.SimpleBlobDetector_create(params)
key_points = detector.detect(self.output_image)
unique_key_points = []
for point in key_points:
  if cap_flag["min_area"] < point.size < cap_flag["max_area"]:
      unique_key_points.append(point)
      cv.drawMarker(self.input_image,
                    (int(point.pt[0]),
                     int(point.pt[1])),
                    (0, 255, 0),
                    cv.MARKER_CROSS,
                    cap_flag["marker_size"],
                    cap_flag["marker_width"])
# draw caps points -- circle
cv.drawKeypoints(self.input_image,          # input image
               unique_key_points,           # keypoints found using blob
               self.input_image,            # output image
               (0, 255, 0),                 # colour for the points
               cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

Listing B.4: Cap Detection

Bibliography

[R1] Mark S. Nixon and Alberto S. Aguado. *Feature Extraction and Image Processing*. Academic Press, 2008, p. 86. < https://www.ppgia.pucpr.br/~facon/ComputerVisionBooks/2002FeatureExtractionAndImageProcessing.pdf >

[R2] Hochschule Rosenheim. Wagner Michael, Script_Image_Processing_for_Automated_Production.pdf

[R3] Hochschule Rosenheim. Wagner Michael Script_Image_Processing_for_Automated_Production.pdf

[R4] http://what-when-how.com/introduction-to-video-and-image-processing/blob-analysis-introduction-to-video-and-image-processing-part-1/

[R5] http://docs.opencv.org/trunk/d0/d7a/classcv_1_1SimpleBlobDetector.html

[R6] https://en.wikipedia.org/wiki/Ellipse

[R7] https://gamedev.stackexchange.com/questions/53142/decomposing-a-concave-mesh-into-a-set-of-convex-meshes/67901

[R8] https://en.wikipedia.org/wiki/Polarizer

[R9] https://stackoverflow.com/questions/37517983/opencv-install-opencv-contrib-on-windows

[R10] http://answers.opencv.org/question/25096/how-to-set-region-of-interest-roi/

[R11] http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur

[R12] http://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html

[R13] https://www.learnopencv.com/blob-detection-using-opencv-python-c/, Satya Mallick

[R14]http://docs.opencv.org/3.0beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

[R15] http://docs.opencv.org/3.1.0/d4/d70/tutorial_hough_circle.html

[R16] https://www.alliedvision.com/en/products/cameras/detail/Manta/G-032.html

[R17] https://www.1stvision.com/cameras/AVT/Manta-G-032-B-C.html

[R18] http://www.z-laser.com/en/products/product/machine-vision-lasers/zm18b/zm18

[R19] https://www.metaphase-tech.com/backlights/economical_backlight

[R20]http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descripto rs.html?highlight=moments

[R21] http://docs.opencv.org/trunk/d8/d01/tutorial_discrete_fourier_transform.html

[R22] D. Costineanu, Cristian Fosalau, Catalin Damian, O. Plopa "Triangulation-based 3D image processing method and system with compensating shadowing errors" < https://www.researchgate.net/publication/254191188 >

[R23] https://www.quora.com/What-is-local-maxima-and-local-minima-in-calculus-Is-there-a-way-to-determine-whether-the-local-maxima-or-minima-is-also-absolute-global-maxima-or-minima

[R24] http://docs.opencv.org/trunk/d4/dc6/tutorial_py_template_matching.html