Robot Vision Mini Project

Project Report

Group 18gr842

Aalborg University
Robot Vision

AALBORG UNIVERSITY

STUDENT REPORT

Robot Vision
Aalborg University
http://www.aau.dk

Title:
Robot Vision Mini Project

Theme:
Computer Vision

Project Period:
Spring Semester 2018

Project Group:
Group 18gr842

Participants:
Shagen Djanian
Niclas Hjorth Stjernholm

Supervisor:

Number of Pages: 15

Date of Completion:
April 26, 2018

Abstract:

The video example of the project is found on YouTube, following the link: https://youtu.be/79HsSZoeQIg.

The video example of the simulation is found on YouTube, following the link: https://youtu.be/1-KY_z2-Zf0.

Contents

Chapter 1

Introduction

For the mini project in the Robot Vision 2018 course the task of assembling Simpsons figures with a robot was given. The figures are made of LEGO®Duplo bricks stacked on each other as can be seen in Figure 1.1. The figures are the following:

- Homer: Yellow, Red, Blue
- Bart: Yellow, Orange, Blue
- Maggie: Yellow, Red
- Lisa: Yellow, Orange, Yellow
- Marge: Blue, Yellow, Green

Figure 1.1

To automatically solve this task a robot and a camera is needed in a setup like Figure 1.2. The camera takes pictures of the area where the bricks are located. The image is then processed to figure out where the bricks are located and how they are oriented. The robot is then ordered to assemble a figure e.g. Homer and can now do this automatically since it knows where the bricks are. The figure is then assembled and dropped off in a loading area and the robot prepares for a new order.

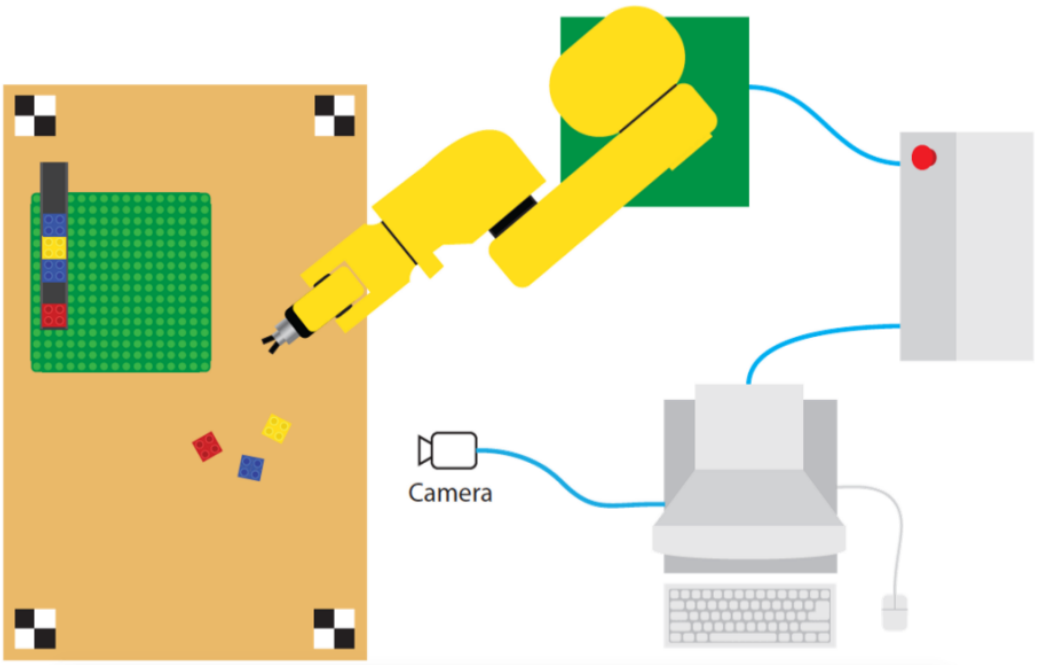


Figure 1.2

Chapter 2

Setup

To do the task laid out, the following equipment was used:

- 6 Degrees of Freedom Universal Robot
- Logitech Webcam 1080p
- MATLAB
- LEGO®Duplo bricks
- Parallel Gripper

The robot cell is shown in Figure 2.1. The red square marks the zone where the bricks are placed. The Universal Robot (UR) can then grab them with a parallel gripper. The green zone marks the drop off zone. Here the UR delivers the assembled figure.

The camera was mounted to the tool, Figure 2.2, as there were no other places to mount it. The benefits of this approach are that the camera does not obstruct the robot when it is moving and that the cameras frame is known when the tool is known, except it is flipped on one of the axes.

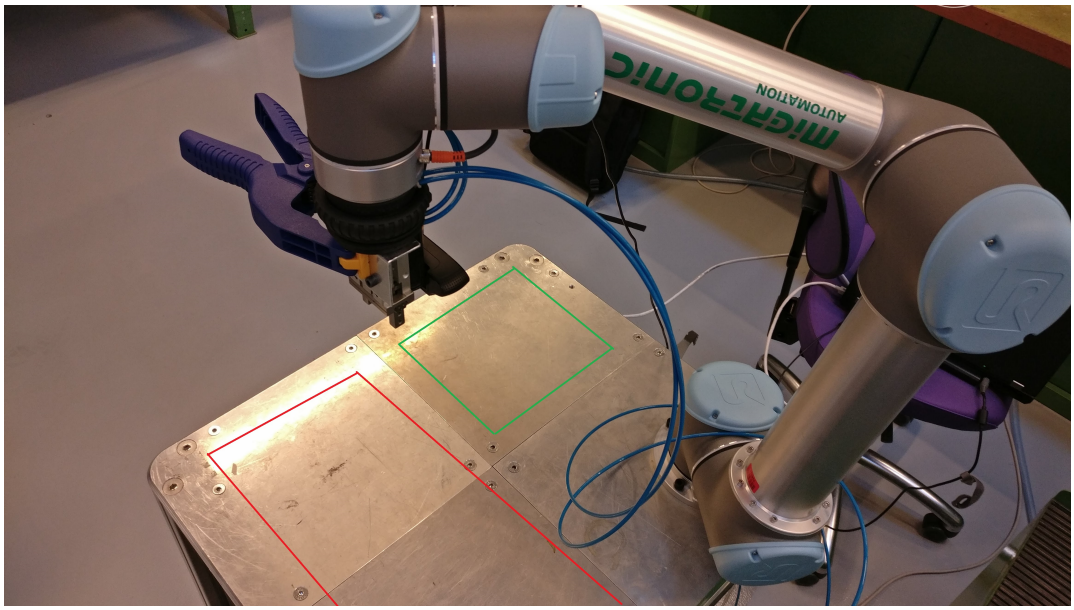


Figure 2.1

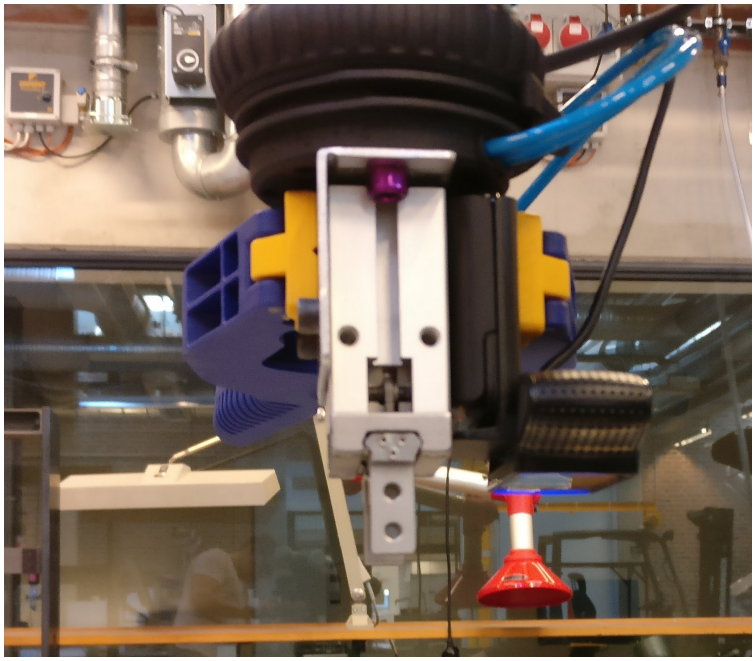


Figure 2.2

Chapter 3

Calibration

3.1 From image to camera

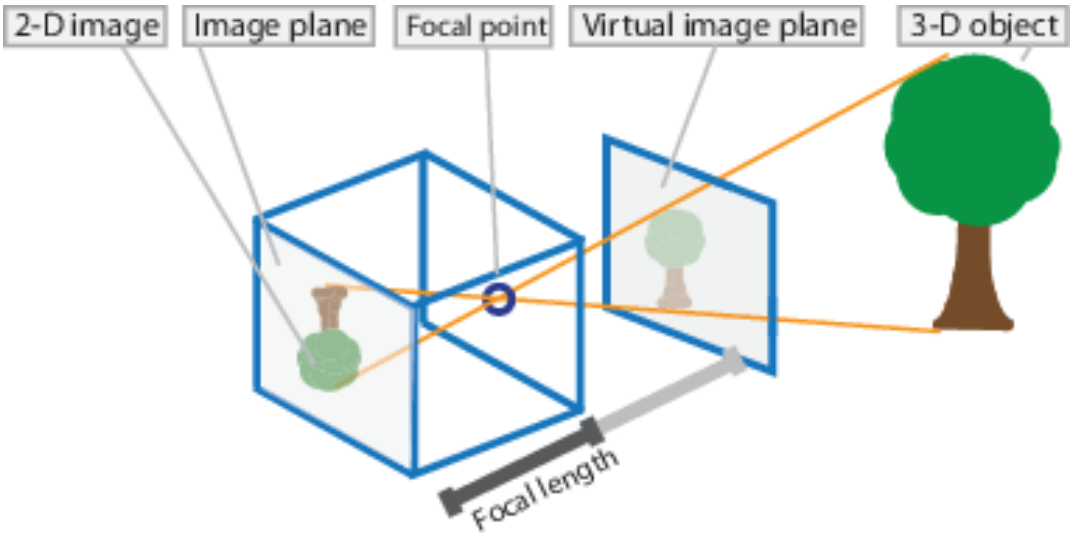


Figure 3.1

When taking an image for the use of detecting an object it is necessary to transform the image frame into the robot frame. First the image has to be transformed from world coordinates into camera coordinates and then to robot coordinates. When going from world to camera the pinhole camera model can be used. A graphical depiction of the model is shown in Figure 3.1. The camera is assumed to be a box where no light can enter except from a small hole called the focal point. The light that is cast on the back wall is an inverted 2D representation of the 3D world outside of the box. The 2D image depends on the distance from the focal point to the back wall, called the focal length. The transformation is split into two parts; from world coordinates to camera coordinates and from camera coordinates to image coordi-

nates. This is done by finding the intrinsic and extrinsic parameters. The intrinsic parameters look like shown in Equation 3.1.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} a_x & s & x_0 & 0 \\ 0 & a_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ z_s \\ 0 \end{bmatrix} \tag{3.1}$$

where $x_s,\ y_s,\ z_s$ are the scene points in the camera coordinate system, u, v, w are the homogeneous image coordinates where w is a scaling factor. The parameters are a_x, a_y which are the focal length. x_0 and y_0 are to translate the image centre to be the actual image centre since there may be an offset due to fabrication issues. s is the pixel skew parameter but it is usually set to 0.

The extrinsic parameters are a translation and rotation that describe where in the world the camera is located and how it is oriented. The parameters can be expressed as: $\begin{bmatrix} R & -RC \\ 0_3^T & 1 \end{bmatrix}$ where R is the rotational 3×3 matrix and $-RC$ is the 3dimensional vector translation. To get the whole transformation it would look like shown in Equation 3.2.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} a_x & s & x_0 & 0 \\ 0 & a_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & -RC \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ z_s \\ 0 \end{bmatrix} \tag{3.2}$$

Lastly there can also be some radial distortion in the image caused by the lens, which causes lines that are linear in the world coordinate to appear non-linear in the image. This correction can be computed by using images with objects with known straight lines. To get all these parameters the Camera Calibration Toolbox for Matlab by Jean-Yves Bouguet can be used. Here 10 to 20 images are taken of a checker board pattern as shown in Figure 3.2, where the size of the squares is known, and from this it is possible to estimate the intrinsic and extrinsic parameters and distortion of the camera.

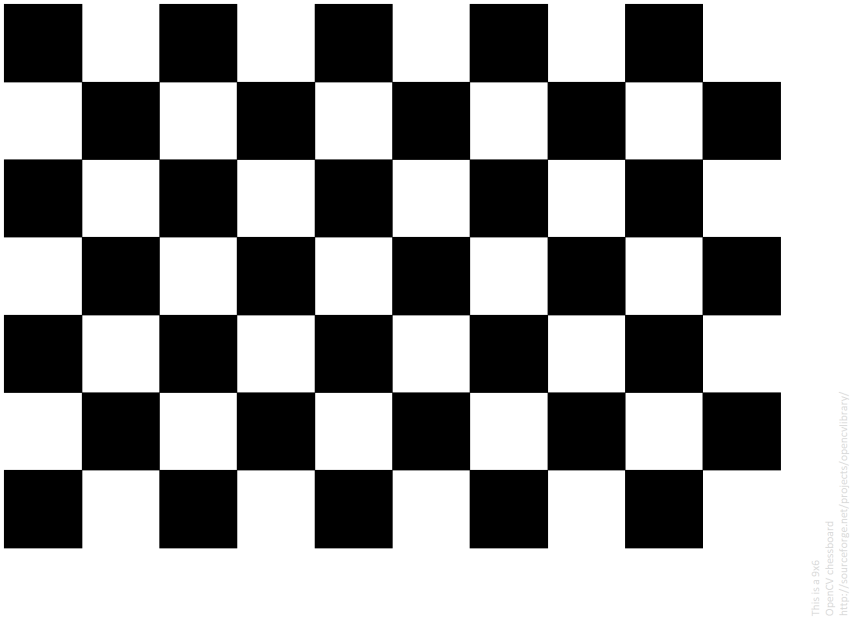


Figure 3.2

3.2 From Image to Robot

Once the parameters are known they need to be transformed into robot coordinates. For this the transformation matrix from the camera coordinates to robot is needed. A way to simplify this is to use a different approach than finding the camera parameters. If the camera is orthogonal to the planar space that the robot needs to operate in it can be assumed that image is simply a version of the planar space with constant scaling. If a point is known in pixel coordinates and in robot coordinates it is possible to compute a simple projection between the points. Then it is not necessary to get the camera parameters and know the robots frame as long as the mapping is known. The mapping can be found as shown in Figure 3.3 with Equation 3.3.

$$\begin{aligned} x_{\text{rob}} &= \theta_0 + \theta_1 * x_{\text{img}} + \theta_2 * y_{\text{img}} \\ x_{\text{rob}} &= (1 + *x_{\text{img}} + y_{\text{img}}) * \vec{\theta} \end{aligned} \tag{3.3}$$

Where x_{rob} is the x coordinate for the robot in a place in the image space. x_{img} and y_{img} are the pixel coordinates of the exact same point with θ_1 and θ_2 as scaling factors for when the robot moves in the x direction in robot coordinates how much does it move in x and y of the pixel coordinates. θ_0 is the offset that comes from the robot's base not having the same origin as the image. When done for both x and y the result is found using **??**

$$\begin{aligned} x_{\text{rob}} &= (1 + *x_{\text{img}} + y_{\text{img}}) * \vec{\theta} \\ y_{\text{rob}} &= (1 + *x_{\text{img}} + y_{\text{img}}) * \vec{\phi} \end{aligned} \tag{3.4}$$

Where θ is the x parameters and ϕ is the y parameters. These can be solved for by using multiple known points and their accompanying robot and pixel coordinates like shown in **??**

$$\begin{bmatrix} x_{\text{rob1}} \\ x_{\text{rob2}} \\ x_{\text{rob3}} \end{bmatrix} = \begin{bmatrix} 1 & x_{\text{img1}} & y_{\text{img1}} \\ 1 & x_{\text{img2}} & y_{\text{img2}} \\ 1 & x_{\text{img3}} & y_{\text{img3}} \end{bmatrix} * \vec{\theta} \tag{3.5}$$

To then go to robot coordinates from pixel coordinates Equation 3.6 is used.

$$\begin{bmatrix} x_{\text{rob}} \\ y_{\text{rob}} \end{bmatrix} = \begin{bmatrix} 1 & x_{\text{img}} & y_{\text{img}} \end{bmatrix} \begin{bmatrix} \theta_0 & \phi_0 \\ \theta_1 & \phi_1 \\ \theta_2 & \phi_3 \end{bmatrix} \tag{3.6}$$

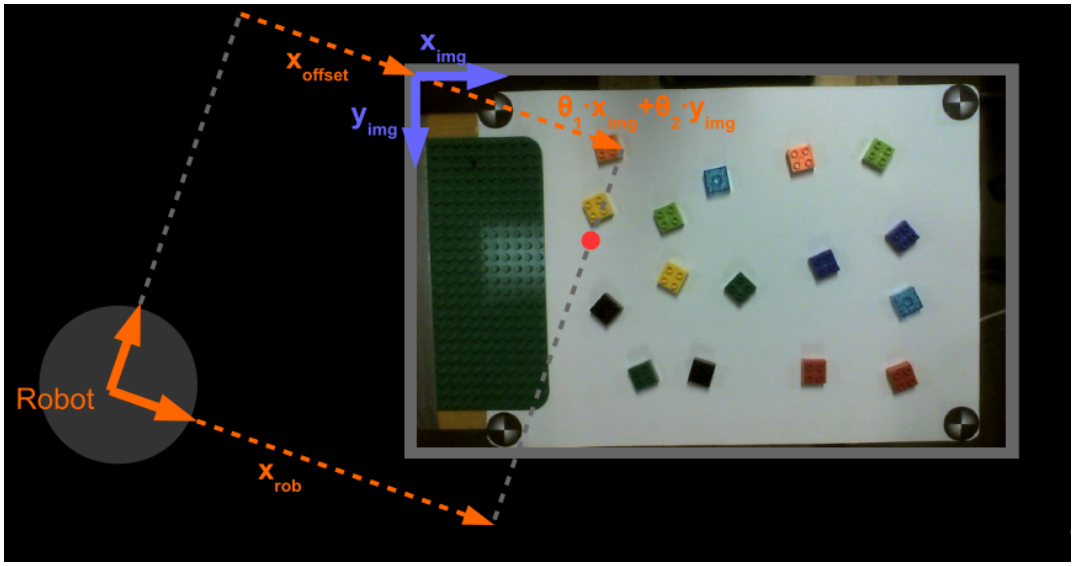


Figure 3.3

This procedure will increase in precision with the increasing number of points used. In the project it is chosen to use three stationary points in the workspace frame of the operating table.

Chapter 4

Computer Vision

The computer vision part is the image processing part of the project. Here the machine differentiates the different LEGO®bricks and their individual colours. Furthermore their pixel position is found to be converted to world coordinates later in the program. This is done to let the robot use the knowledge of the different colours to automatically assemble the figures with the right coloured bricks. To find the bricks in the picture, several steps are made to differentiate the colours:

- Format conversion
- Background subtraction
- Colour thresholding
- Noise removal
- BLOB analysis
- Feature extraction

The original picture processed is shown in Figure 4.1.

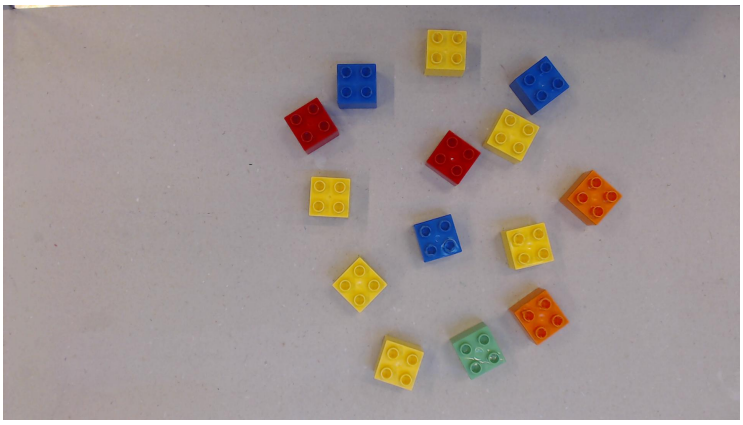


Figure 4.1: Original picture taken by the webcam attached to the robot.

4.1 Format Conversion

The original picture taken by the webcam is in RGB and is converted to a chromaticity image. This is done using Equation 4.1:

$$r = \frac{R}{R+G+B},\ g = \frac{G}{R+G+B},\ I = \frac{R+G+B}{3} \tag{4.1}$$

Chromaticity is given from the chromaticity plane. This is shown in Figure 4.2. This is done since the colours are being normalised to avoid influence by illumination. Then the colour description is supplemented by an intensity as calculated in Equation 4.1. This will be referred to as RGI and the conversion is RGB to RGI.

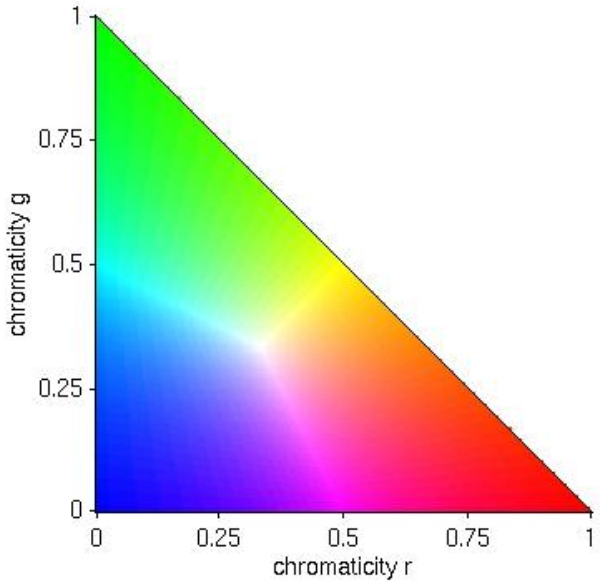


Figure 4.2: Chromaticity plane

The advantage of this is, that this should allow for colour thresholding on any image as it disregards illumination.

4.2 Background Subtraction

The background of the image is subtracted to ease the localisation of the brick edges. This is done by taking a picture of the surface without any bricks as a background image from the exact same position as the original image. To avoid unwillingly removing any objects from the new image generated a threshold is set fro specific pixel values.

Firstly the RGB image of the background is converted to an RGI image. The subtraction is then performed pixel-wise using the formula shown in Equation 4.2. The subtraction is done using the two RGI images.

$$F(x,y) = I(x,y) - B(x,y) \tag{4.2}$$

The thresholding is done by setting a threshold a value and doing a pixel-wise comparison. Should any of the channels have a higher value than the threshold, the pixel will be kept as a non-background pixel value.

4.3 Colour Thresholding

The colour thresholding is done to separate the individual brick colours from each other. The thresholding is done by comparing the pixel values of the RGI image to a set threshold for each channel. Firstly a threshold is applied to the intensity channel. This is done to rule out coloured pixels which may be indistinguishable because of too small an intensity.

The colour thresholding is then made on the image. This is done for every brick colour used in the project by setting thresholds for both r and g channels to match the desired colour of a brick. A small pseudocode example is shown below:

```
if:
        TH_r_min < r < TH_r_max AND
        TH_g_min < g < TH_g_max
then: object pixel
else: non-object pixel
```

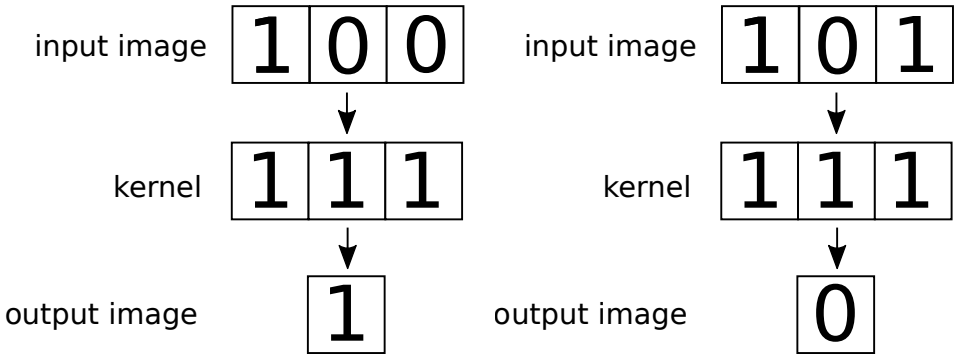
If the pixel satisfies the set thresholds for the two channels, it is marked as a binary 1 in an output image.

To set the thresholds fro the different colours in the r and g channels the RGI image is analysed. By cropping the picture to match each brick colour and analysing each for the chosen colour the thresholds are found by taking the 5% and 95% percentiles are found and used as the thresholds for each colour.

4.4 Noise Removal

The noise removal also known as morphology is done as the images generated from thresholding can contain some outliers which is not entire bricks but is recognised inside the threshold limits. Furthermore the bricks recognised contain some holes which is preferred to be filled.

To do this morphology the *hit* and *fit* operations are taken in use also known as dilation and erosion. These are firstly used to close the holes in the bricks and then opening to remove noise. For this kernels for both dilation and erosion are defined. The functionality of both dilation and erosion is shown in Figure 4.3



(a) Dilation kernel **(b)** Erosion kernel

Figure 4.3: Example of dilation and erosion kernel.

4.5 BLOB Analysis

The BLOB analysis is made to find the location of the bricks but also to find the rotation of the bricks to pick them up correctly. To do this a classification is necessary. This is done using the Grassfire algorithm with 4-connectivity. This enumerates each pixel belonging to a BLOB on the images for each of the brick colours.

The next step is finding the centre of the BLOBs or bricks, done by using centre of mass. This is calculated using the formulas shown in Equation 4.3.

$$x_m = \frac{1}{N} \sum_{i \epsilon \text{object}} x_i, \; y_m = \frac{1}{N} \sum_{i \epsilon \text{object}} y_i \tag{4.3}$$

To be able to pick up the bricks properly, the rotation of the bricks must be found. By calculating the distance from each pixel, forming a BLOB of a brick, to the centre of the BLOB the four corners are found by finding the four pixels furthest away from the centre. By defining four vectors from the centre of the brick to each corner the angles are calculated and by that the rotation. This is found by finding the mean of the four angles. In illustration of the vectors defined of the BLOB is shown in Figure 4.4.

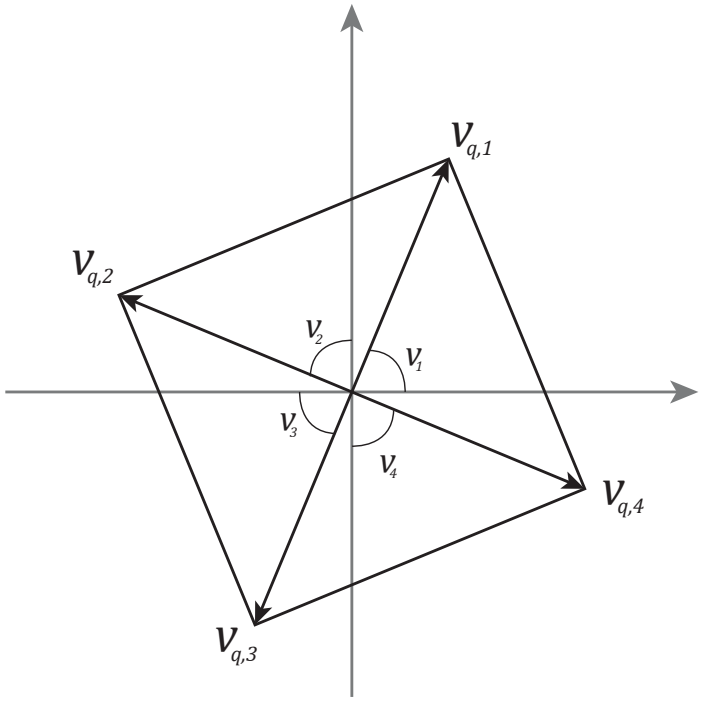


Figure 4.4: Rotations found from vectors and corners in a BLOB

The final result of the computer vision part yields five arrays with brick information of each colour and brick. This includes position and rotation used to locate and pick up the bricks. A final informational image of this including a directional vector showing the rotation is shown in Figure 4.5.

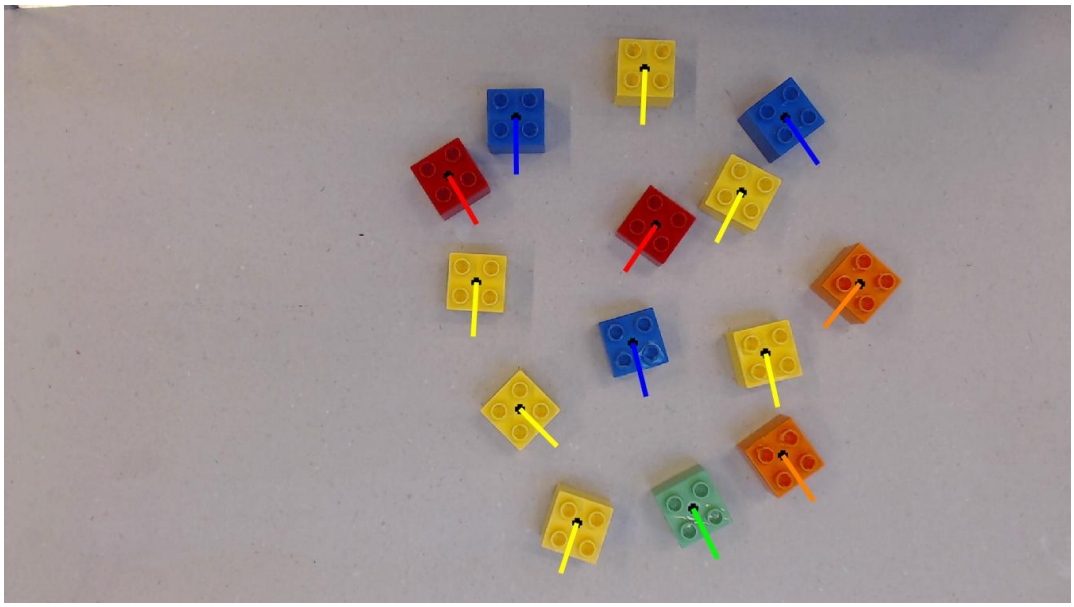


Figure 4.5: Final informational image including brick centre and rotation directional vector.

Chapter 5

Assembly

5.1 Robot Control

The robot was controlled by using a Matlab script that creates an interface to the UR robot. This is done with a TCP/IP connection. The robot is then controlled linearly or by joint values. Since the location of the brick is known, inverse kinematics can be used get the robot to that location. As long as the location and orientation of where the end effector is supposed to be, the internal software of the robot takes care of the inverse kinematics.

5.2 Final Assembly

The assembly is made on the platform where the Universal Robot (UR5) is mounted. Here a piece of cardboard is laid out on the platform to occlude the shiny metal background, easing the image processing as this will give a plain background to process instead of a reflective surface.

For each initialisation of the program a new picture is taken of the background, then bricks are placed in the workspace and a picture is taken. The positions and orientation of the bricks are found using the image processing described in chapter 4.

When the image processing is done it is possible to order any of the five Simpsons characters from the MATLAB command window. Because of functionality issues with the gripper on the robot it was not possible to physically assemble the figurines. Instead the robot marks the two or three bricks used for the specific character, taking the theoretically height into account as well and moves to the drop-off point. A red brick was used as the middle piece in the Homer character as there was not any white bricks available. The amount of bricks was limited to the option of building one of each character. Had there not been any issues with the gripper the characters could have been assembled and placed in the drop-off area, as proved in the example video.