# 

# 

# 

# Image Processing

## Assignment plan

*Made by:*

**Tim Bakx** 3695301

**Roeland Krijgsman** 3832171

## Subject of the assignment

The target objects for our image processing assignment are LEGO bricks. LEGO bricks come in a lot of different shapes and sizes, we’re mostly interested in the shapes – size not so much. Almost anyone can picture a pile of LEGO bricks on the floor, looking for that one LEGO piece you saw 5 minutes ago. We’re looking to make that a little easier, by using image processing to detect different types of LEGO bricks, sorting most of it out for you. Of course, some LEGO bricks are very difficult to discern, especially when things overlap or when a build contains large custom (low-reusability) parts. As these are easy to spot anyway, but hard to process, we’re excluding these types of bricks in our assignment.

We’re assuming that the pile of bricks isn’t a literal pile, so as to prevent too much overlap. We’re using different degrees of *clutter*, which we define as the inverse of the amount of free space between objects (e.g. lots of free space = no/low clutter, almost no free space = high clutter). Since it’s important to define a few categories to use in this assignment, we’ve created five groups of LEGO bricks. We describe LEGO bricks by their size, always in the following manner: *x* by *y* by *z*, where *x by y* denotes the **smallest side** as *x* and the **biggest side** as *y*. The *z* denotes the height of the brick, ranging from ⅓ (plate) to 2 in our sample set.

### LEGO groups

#### Group A: rectangles

This is your run-of-the-mill LEGO block type. We’re also including the flat *plates* of LEGO, and the higher LEGO bricks. In picture 1, these bricks are represented by the 1x2x2 brick, and the 4x12x⅓ and 2x16x⅓ plates.

#### Group B: rectangles with extra

This is the first advanced type of LEGO brick. Mostly used to connect different types of LEGO, these are rectangular bricks with a little extra (e.g. a connector rod). Examples of these can be seen in picture 2.

#### Group C:

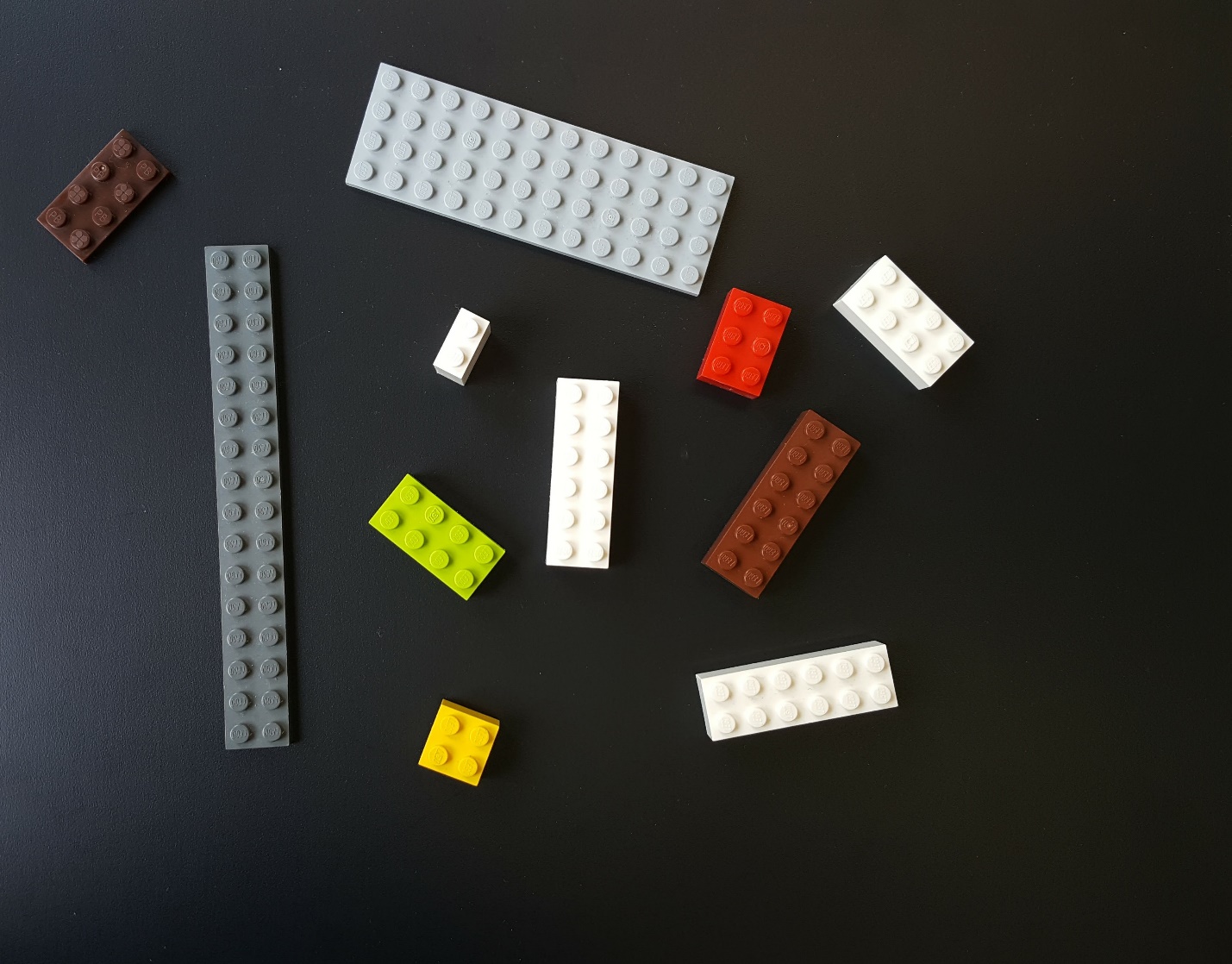
These are the advanced kinds of LEGO plates. They’re usually rectangular or almost rectangular, and are not covered all the way with studs. Examples of these can be seen in picture 3.

#### Group D:

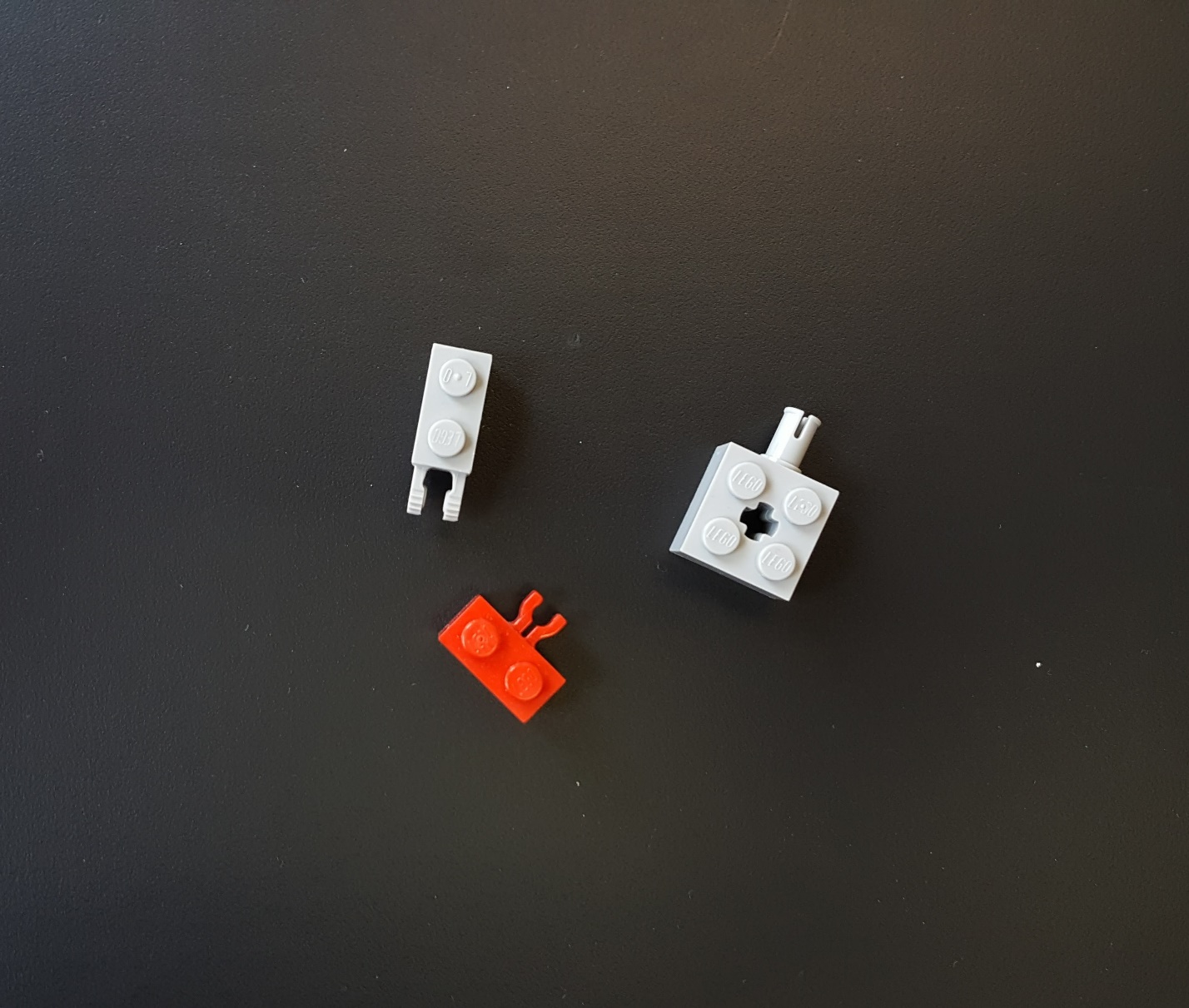
This group consists of distractor objects – either they’re too specific, or they’re things like LEGO minifigures, minifigure accessories and non-LEGO objects (see picture 4).

#### Group E:

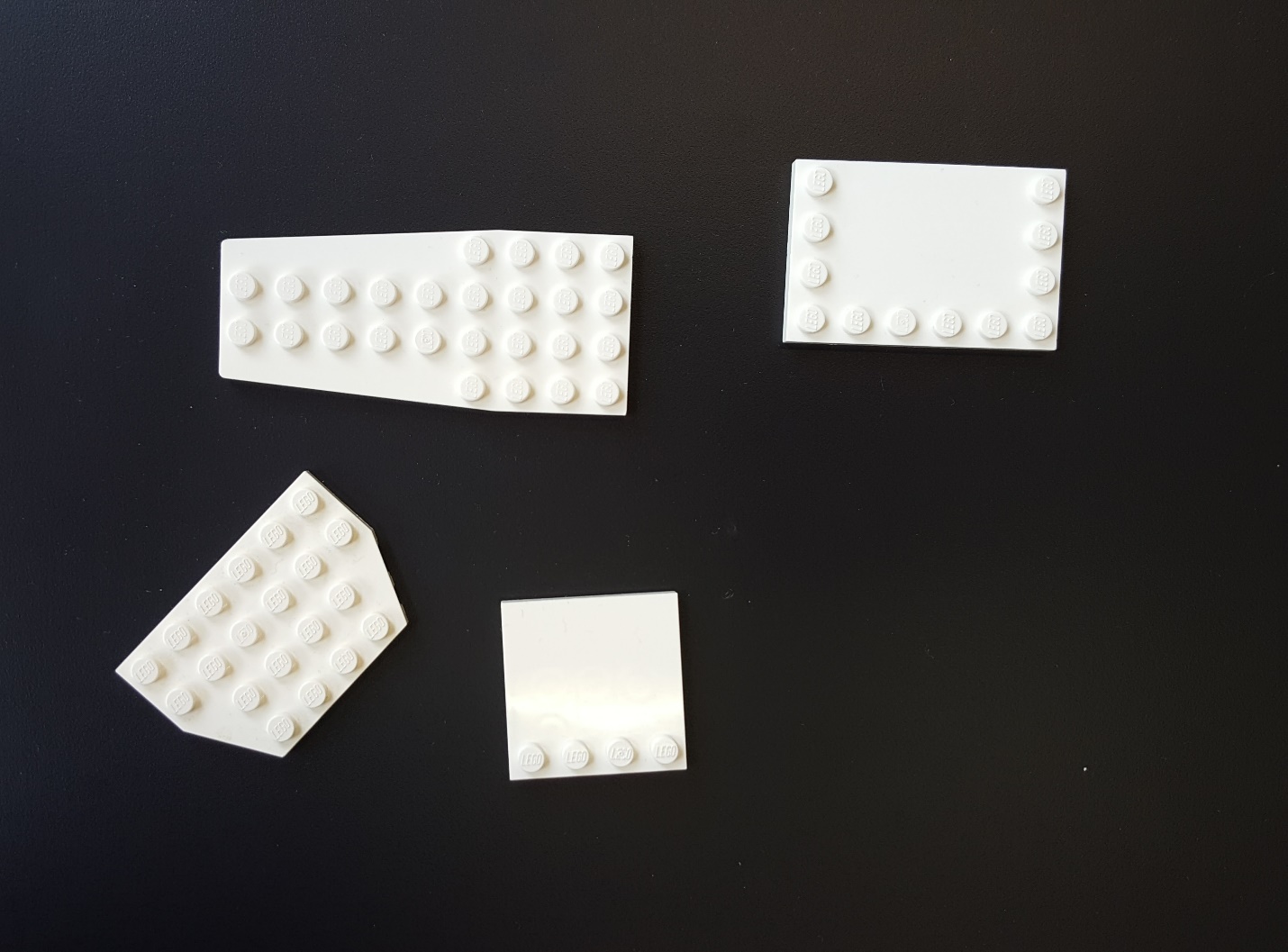
This group consists of differently-shaped LEGO bricks. These include the housing/roof bricks, and slanted bricks. These can be found in picture 5.



Picture 1: group A LEGO bricks



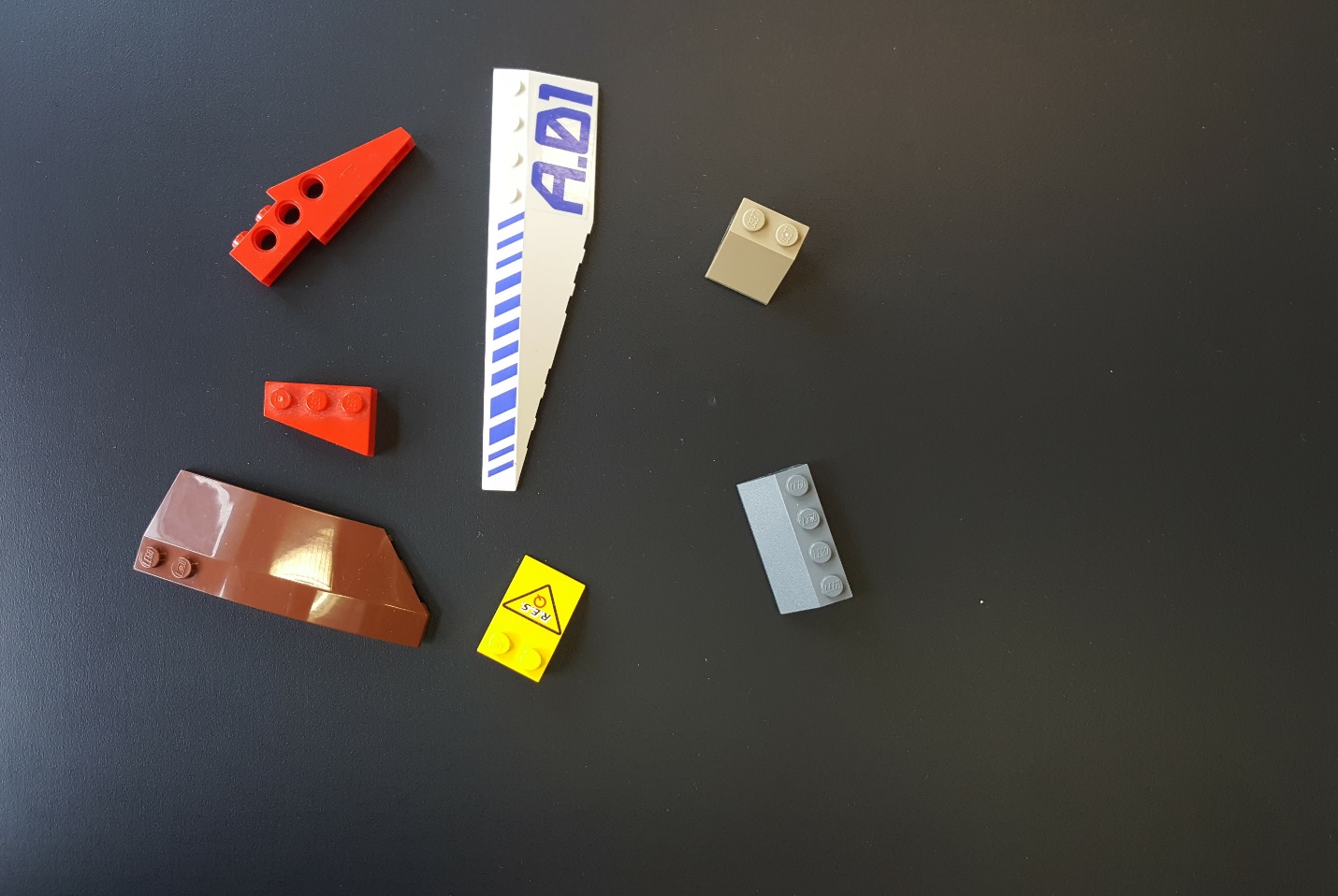
Picture 2: group B LEGO bricks



Picture 3: group C LEGO bricks



Picture 4: group D objects

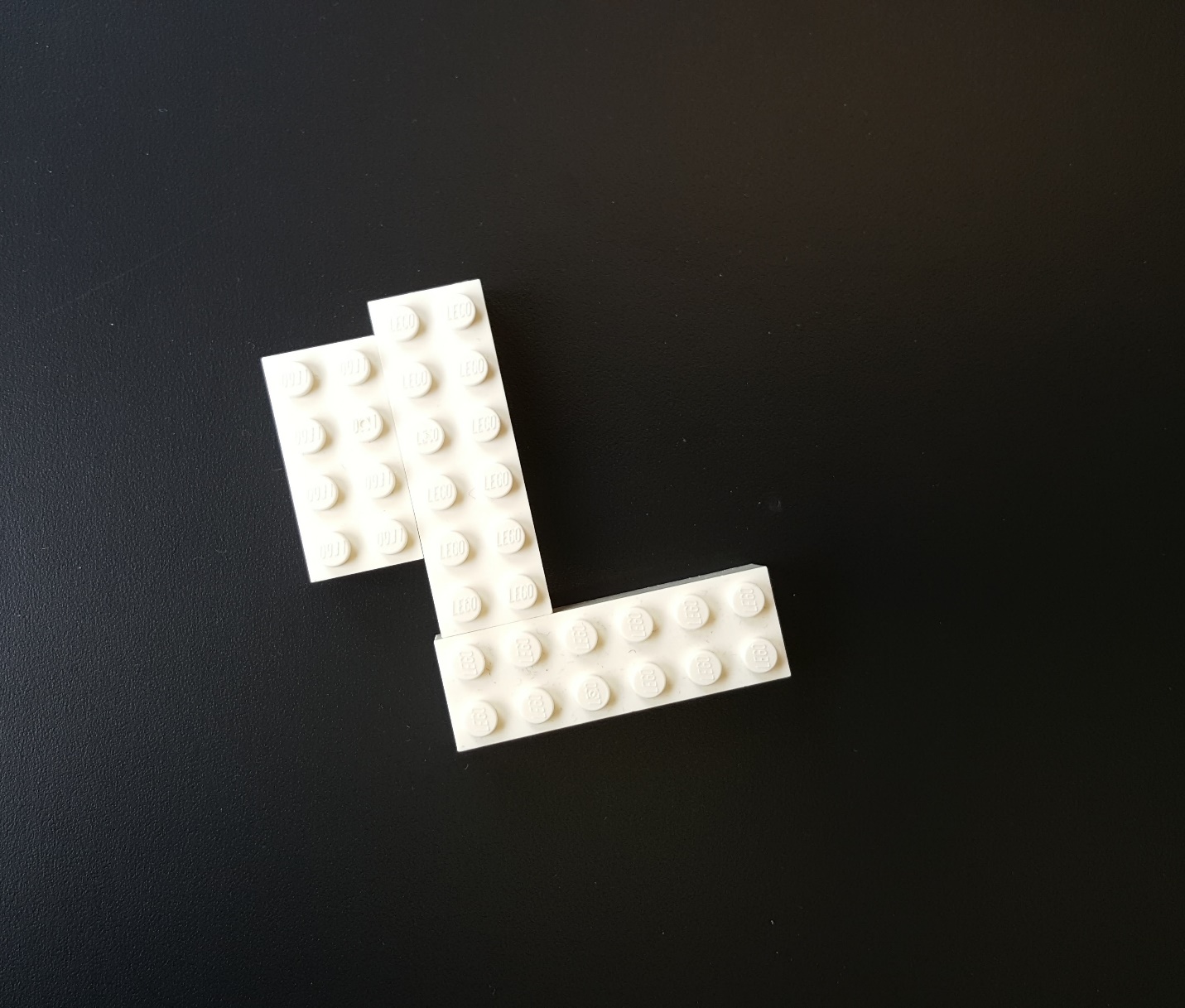


Picture 5: group E LEGO bricks

## Context

### Orientation

LEGO bricks look quite different when upside down. We’re only interested in looking at upright LEGO bricks, so we can use the studs – the little round bumps on top – for detection. Furthermore, we can’t magically detect edges where the human eye can’t even see edges, so we’re excluding bricks of the same colour lying adjacent to each other (see picture 6, especially at the top of the edge between the 2x4 and the 2x6).



Picture 6: obscured edges in adjacent LEGO bricks of the same colour

### Background

We’re using a black background, which is not without noise. There are different lighting backgrounds (some have more of a gradient background), shadows, bits of dust, etc. These will all have to be filtered out when detecting the different defined groups.

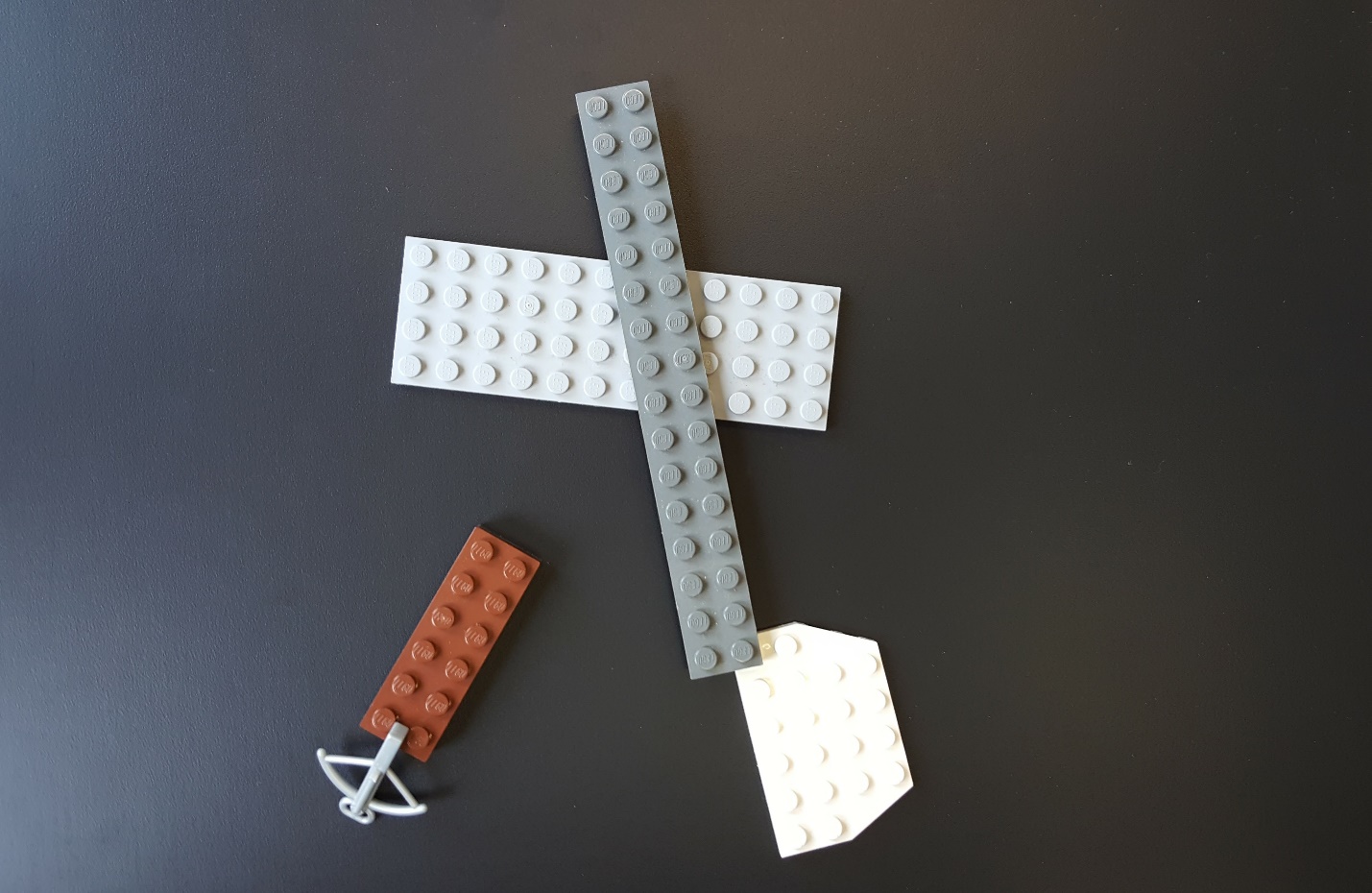
### Perspective

First of all, we have to note that group E can’t be detected in a perspective shift when comparing objects from this group to group A. We’re going to work on detecting these, and should we not succeed, we’re replacing the testing images for group E. We’ll file group E objects under group D, and add additional images for different perspectives when necessary.

As noted previously, we don’t care much about size. We mostly focus on **type** and **proportion**.

### Overlap

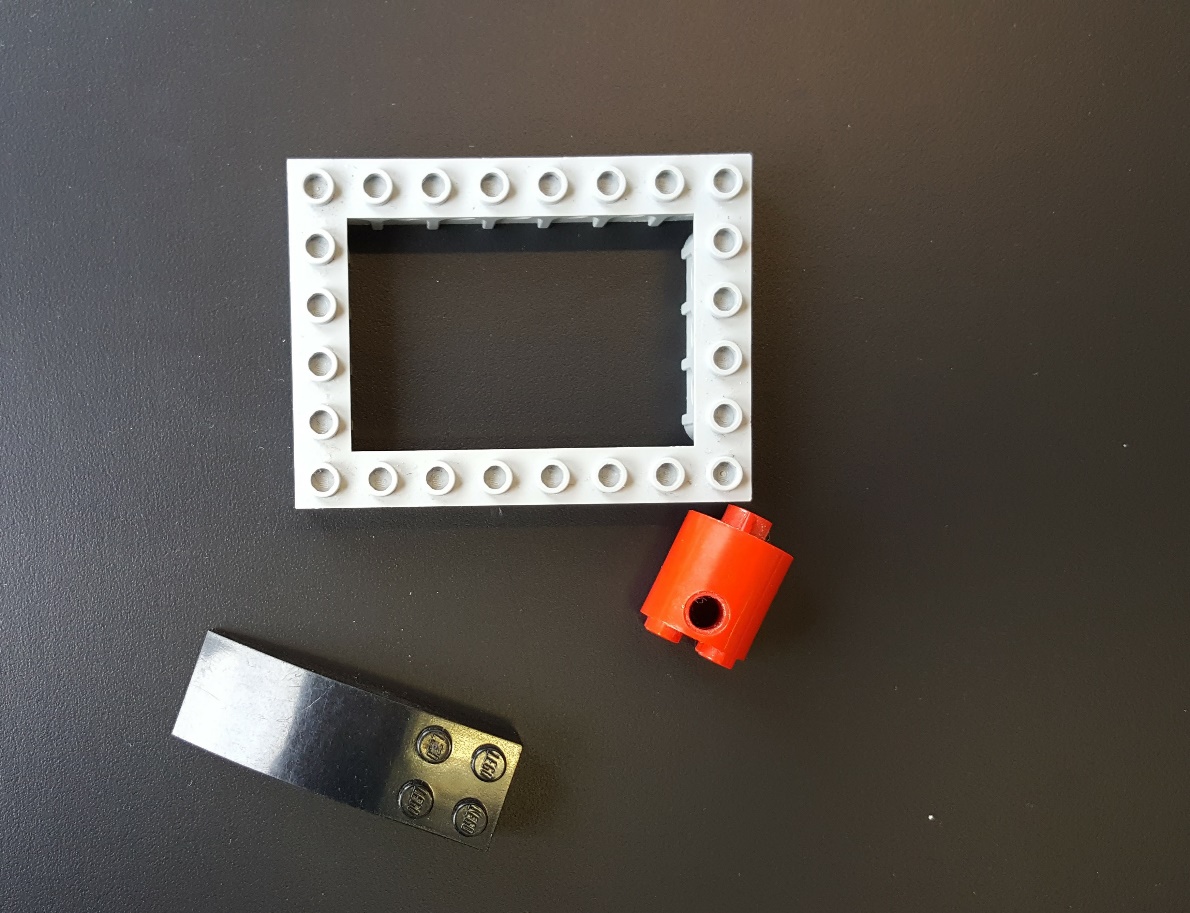
We haven’t addressed overlap yet. When using image processing, things like overlap can really inflate the difficulty level of labelling. Sometimes, it’s not clear where an object ends, and a new object starts, or whether it was actually the same object, but partially obscured. If it’s not too obscured, defining *too obscured* by having an overlap of max. 5% of the object being overlapped, we’ll have to be able to label the objects correctly. Should this exceed the maximum 5% overlap, then the object being overlapped will not be counted towards any statistics, as it’s unlikely that we can label them correctly. This is illustrated in picture 7, where the bottom two overlapping objects doesn’t exceed the 5% maximum, and the top overlap does exceed the maximum.



Picture 7: overlap in LEGO bricks

## Excluded bricks

We’ve excluded some types of difficult LEGO bricks, illustrated by picture 8. These are deemed too difficult to label for this assignment. These bricks all have elements that are so different from the other groups, that we don’t think it’s feasible for this assignment to label these as well as the other groups.



Picture 8: excluded bricks

## Algorithm

We’re going to use the following method to get from an input image to a labelled image:

### Grayscale

First, we detect the background and grayscale the whole image. We’ll filter out basic imperfections in this step, using methods like Gaussian blur

### Thresholding

We’ll use Closing/Windowing to select interesting objects/areas

### Watershed/similar methods

We’ll use Watershed or combinations of other methods like Hough transform and others to detect individual edges/borders of objects, and find objects by using these edges

### Edge object removal

Remove all objects that cross the image boundary (as we can’t determine whether they are bigger outside of the image, and don’t want to make assumptions about the rest outside of the image)

### Labelling

We’ll start labelling the objects using our five defined groups

### Filter

Filter the resulting image from the previous steps using the labels. Map these label values to grayscale values, so the groups can be seen in the image

### Original with result

Using the previous step, highlight the selected group in the image by grey-scaling the original image for every pixel except for those that are contained within the target group (keep original colour value here).