VC: Project 1

Project 1 – LEGO Brick Detection

António Santos up202008004
Pedro Silva up202004653
Fábio Morais up202008052

April 2024

1 Methodology

1.1 Pipeline

There are two ways of running the code: specifying a folder of images to run the detection on, or reading a JSON file with a list of paths to images to scan. If no option is selected we attempt to run a *input.json* file stored in the same location of the script, alongside the images. Our execution pipeline is comprised of the steps described in the following subsections.

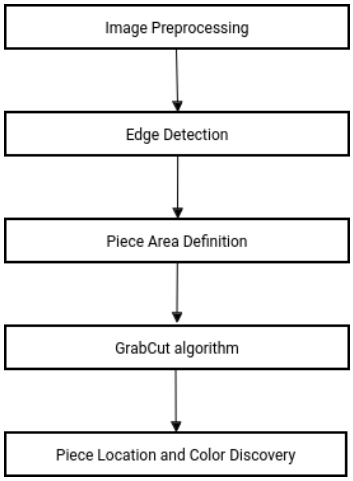


Figure 1: High-level flowchart of our pipeline

1.2 Preprocessing

For preprocessing, we convert the image to grayscale and execute the cv2.fastNlMeansDenoising() function which, as the name implies, uses the Non-local Means Denoising algorithm. After that, we also run histogram equalization and gamma correction, increasing the contrast of the image and improving its brightness. Finally, a bilateral filter is applied, in order to reduce the most amount of noise possible while still preserving edge strength.

1.3 Edge detection

For edge detection, we use a Canny edge detector on the preprocessed image, passing the parameters defined in the pipeline, which suffered several modifications throughout the project. Then we apply morphological transformations, more specifically, dillation and closing. Subsequently, we threshold the image in order to clearly separate the edges from the remaining content, then finally using cv2.findContours() to attempt to draw the bounding boxes around the LEGO pieces.

1.4 Finding piece areas

In order to find areas containing pieces in the images we need to make some assumptions:

- LEGO bricks are big enough to be seen in the image;
- it is better to find a bigger noisy area than no area (no brick);
- nearby areas, within some threshold, are possibly the same brick;

With these set, we ignore bounding boxes that are too small (in comparison to a ratio of the image's size), combine overlapping bounding boxes and boxes that are close to each other. This process might create bounding boxes which are more distant to the edges of the pieces than normal, but what matters in this step is that the pieces are all enclosed in some area.

1.5 GrabCut

Having the areas around pieces well-defined, we run the GrabCut algorithm on them to isolate the pieces from the background and potential noise. Having the pieces separated, defining their bounding boxes in the next step becomes a lot easier. The produced foreground mask is used on the same location of the bounding box in the original image to retrieve, what is hopefully at this point a well defined region of a potential brick.

1.6 Finding pieces and colors

The benefit of using GrabCut is now apparent, the combination of "noisy" rectangles will not reveal a piece, and in the instance that something is detected its size will be so small we should be able to safely ignore it. As such, we now perform preprocessing, edge detection, morphology transformations and bounding box definition again for each of the resulting masked images. There are however some differences from previous steps: preprocessing is simpler (no non-local denoising), boxes are not combined by overlap/proximity and edge detection uses much more lenient parameters for the Canny edge detector. Finally, with the definitive bounding boxes defined, we extract the dominant color from the image area by using K-means with 1 cluster. This value is returned in BGR and we set a threshold for each component of this color space when comparing each LEGO's color.

2 Explored methodologies

In order to land on the previously described pipeline many other methodologies were explored to varying success but still contributing to a better understanding of the task and challenges:

- cv2.connectedComponents() was initially used to find zones of interest before running edge detection, however it was too sensitive to noise. The idea was revisited later as a lightweight replacement for GrabCut but ultimately went unused;
- *"Heuristic-based GrabCut"* was an attempt at using Canny's number of results (upon exceeding a set number) and its result to apply a mask GrabCut on the entire image, it was extremely time consuming and unpredictable;
- Using HSV, use S and V to detect Black/White/Grey bricks and H for other colors. Whilst promising, it wasn't able to make outperform our final methodology;

3 Results

We were able to clearly identify most pieces and their color in all of the testing images, with some errors regarding close together and/or small bricks and rarely some noise still persists, this is most certainly due to the values currently being used for our minimum areas, Canny parameters, etc. Further tweaking of such parameters would be needed for better results, but current ones have left us with a good impression of the suitability of our approach. Some example images are available in the Appendix A.

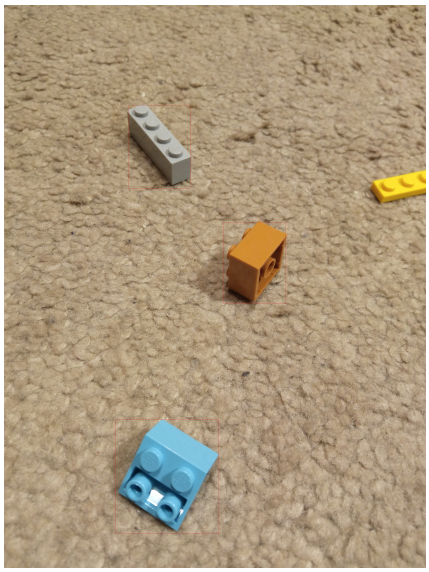
4 Conclusion

We feel that we have met the goals that were initially proposed for this project, being able to detect with some confidence the color, position and number of LEGO bricks in an image, even under non-ideal conditions. AI tools were used during the project exclusively to investigate some OpenCV functions inner-workings. Whilst the obtained results are satisfactory we could in the future potentially look into tweaking the currently available parameters, investigating Cascade Classifier's using Haar-like features and implementing some sort of AI assistance such as neural networks.

In conclusion we are proud of the developed work, not only was it an interesting exploration of image processing technologies and an effective and challenging introduction to Computer Vision tasks it has also sparked our interest for the upcoming project and the possibilities that neural networks opens for this area.

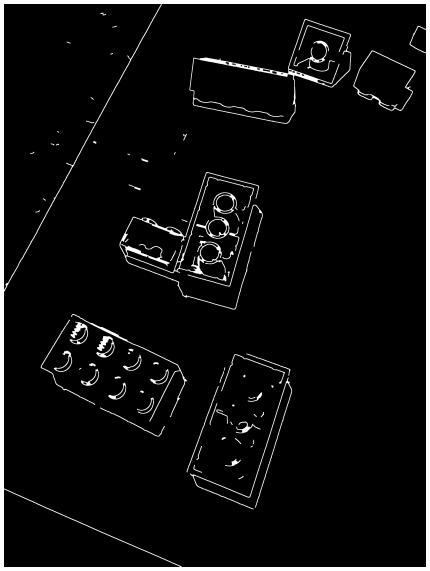
A Appendix

(a) Canny

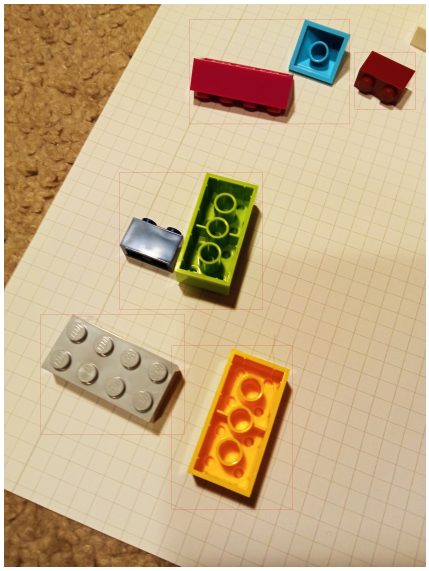


(b) Detection

Figure 2: Noise Reduction Case (IMG_20201127_234242)

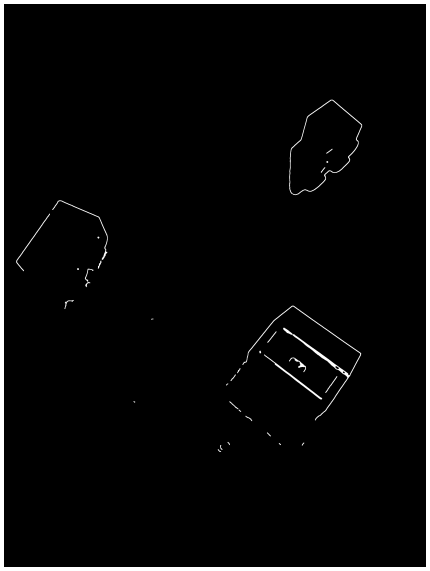


(a) Canny

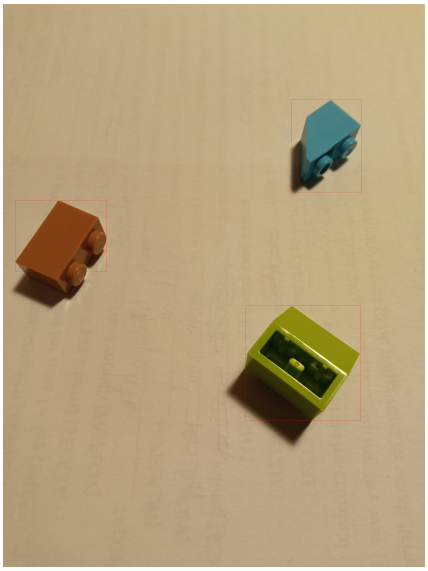


(b) Detection

Figure 3: Close Together Case (IMG_20201127_234607)



(a) Canny



(b) Detection

Figure 4: Simple Case (IMG_20201127_235047)