Computer Vision - Project 1

Exploration

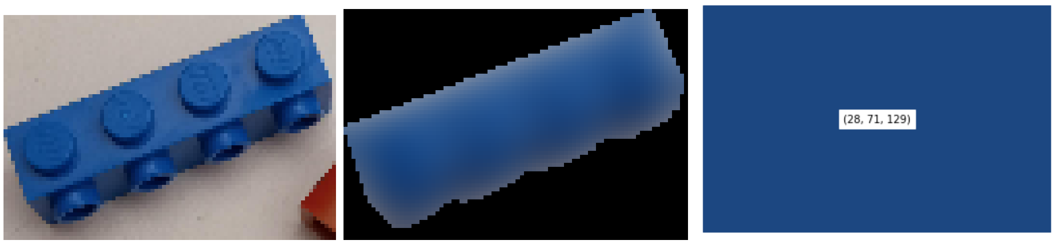
The first step of our exploration was the use of the Canny Edge Detection algorithm and Hough Lines, experimentING with different threshold values to see their impact on the results.

Unfortunately, by itself, edge detection was not nearly enough to obtain good results. We moved on to exploring different segmentation algorithms, such as: Otsu’s Method, Grabcut Algorithm, Adaptive Threshold, and K-means. We had some good results with some of them, but the K-means was by far the most consistent one, and hence, it was used in our final algorithm.

One of the main challenges during development was adjusting the values of the very sensible parameters for the algorithms used (some images worked perfectly with some values but, in turn, others were completely wrong). This was especially visible regarding the size of the kernel used in applying a Gaussian blur and the K-means parameters. Moreover, the color detection was a bigger obstacle than initially expected, requiring a lot of trial and error until satisfactory results for all images were achieved with the final model configuration.

Model Algorithm

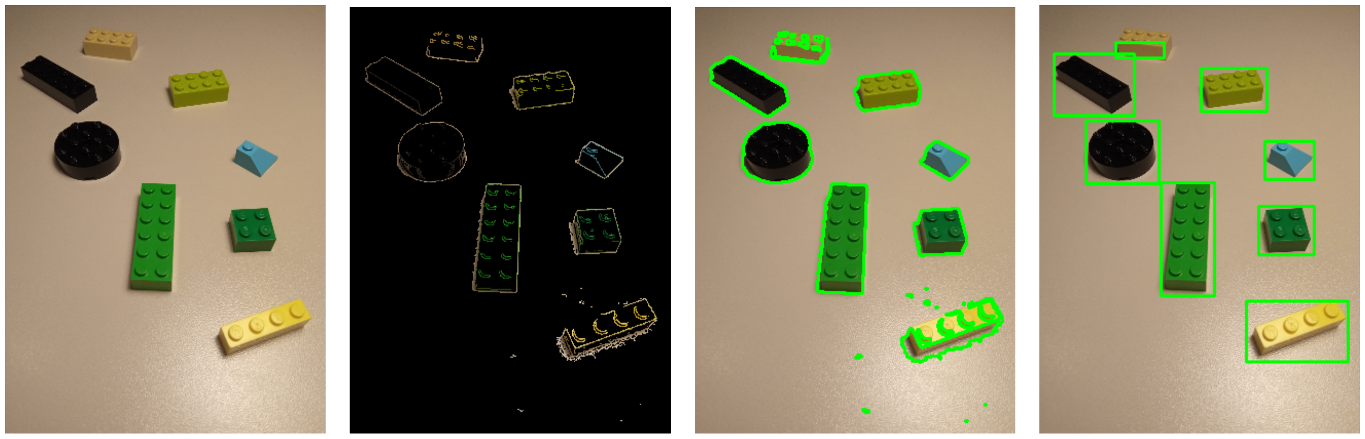
1. **K-means and Canny Edge Detection**
   a. Apply a small Gaussian Blur
   b. Multiple iterations of K-means with different K values and seeds to reduce color count
   c. Apply Canny Edge Detection Algorithm for each one of them
   d. Taking the edges for each iteration and choosing the overlapping ones across iterations according to a certain threshold for more precision on edge definition
2. **Contours**
   a. Get contours based on the edges, connecting nearby ones within a certain distance
3. **Bounding Boxes**
   a. Taking each contour, the algorithm tries to find the respective bounding box
   b. If there are too many boxes, it restarts from step 1 but with a bigger Gaussian Blur
   c. Bounding box culling to filter false positives: too small/big and overlapping boxes
4. **Color Detection**
   a. For each bounding box found earlier in Step 3:
      i. Apply K-means clustering to reduce the number of colors in the bounding box
      ii. Apply GrabCut to remove background and leave only the lego
      iii. Apply a big Gaussian Blur to ensure the lego color is very homogeneous
      iv. Analyze pixels to get the most common color values and store it on an *colors* array
   b. With an array of lego colors contained in each bounding box, leverage the LAB color space to compare all colors pairs more accurately than RGB and remove pairs that are too close (Euclidean Distance) according to a threshold

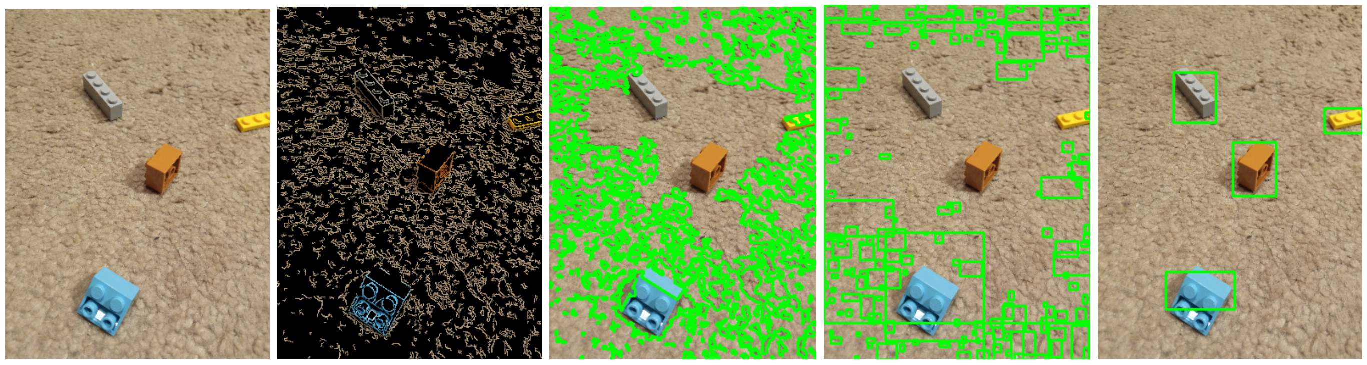


Step 4 Example of color extraction for a single lego

Results

The following images represent the multiple stages of the program running. In the second sequence, there is one more image, due to an extra iteration with a larger blur to reduce the hitbox count.

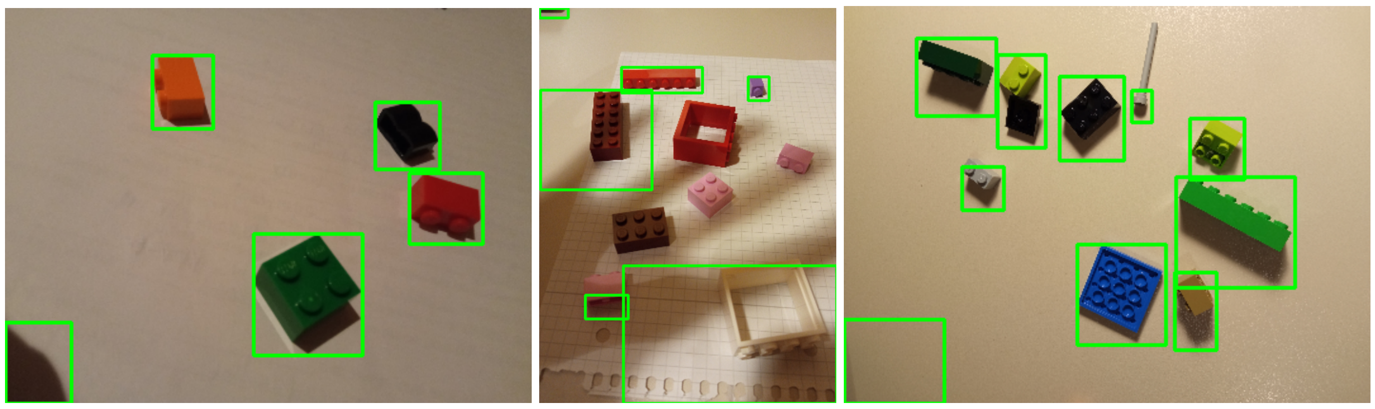




Known Issues

The model used has a couple of identified issues:

- The results can take a while (up to 40s) to compute mainly due to the iterations of Step 1
- There are some detection problems related with false positives in shadows and corners or others where the Canny edges for shadows overlap with the legos and big bounding boxes are defined
- Some light colors (gray, pink) or that are very similar to background (gray, white, transparent) are sometimes not found as pieces when if a bigger blur needs to be applied
- Pieces that are too close are sometimes considered a single piece (like on the 3rd image) due to their shadows connecting the contours



Group

João Alves - 202007614 Marco André - 202004891 Rúben Monteiro - 202006478