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. Later we found that K-means also allowed for a more accurate and sharp Grabcut.

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

Below presented are the four main steps of the final developed algorithm/model:

1. **K-means and Canny Edge Detection**
   a. Apply a small Gaussian Blur to make pixel color more uniform/blended
   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

Models Steps Explanation

Step 1 - It's important to state that all images are converted to 1/10th the size of model input to make it faster (mainly for this step) without affecting model quality significantly (by our testing).

This 1st step aims at reducing the color space of the image for better edge detection via multiple iterations that are performed with different K values and random seed points because different images require different K values for a good Canny Edge Detection output. By making many iterations and aggregating the most common results in a final *edges* variable, we can cover a more diverse image set.

Note: The use of random K-mean seeds results in the model having better runs than others.

Step 2 - The aggregated edges may be very "spaghettified" and not connected into a single unit. This step ensures close enough edges are melted into a single contour shape that hopefully encapsulates the whole lego piece, instead of detecting very small legos inside an actual piece (specially due to shadows).

This, however, has a drawback of sometimes connecting two very close pieces into a single one.

Step 3 - Through our testing with the different images, we observed a very large amount of bounding boxes (bb) caused mainly by irregularities in the background (e.g. rug fibers, shadows and squared paper being considered pieces). The model applies a culling step to check which bb are too small/big (and hence implausible matches). To further avoid cases where the circles that allow legos to connect to each other are considered pieces within pieces, overlap and contained bb can also be culled. Finally, if the model still finds a large amount of bb, it may mean the background is very noisy, creating false matches, and thus, we restart from step 1 with a bigger blur to reduce said noise and get only good matches. This bigger blur sometimes leads to legos with similar color to background to be ignored.

Step 4 - With each bb encapsulating each lego, the easier way found to successfully and consistently extract its color was to reduce color space again with K-means so the following GrabCut can be more precise to distinguish between background and lego. With only the lego pixels defined, a very big blur makes the colors very blended and uniform so lights and shadows are less impactful in the overall most common pixel color. Finally, similar colors in LAB color space are discarded (being this very subjective).

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 a bigger blur needs to be applied
- Pieces that are too close are sometimes considered a single piece due to their shadows connecting the contours

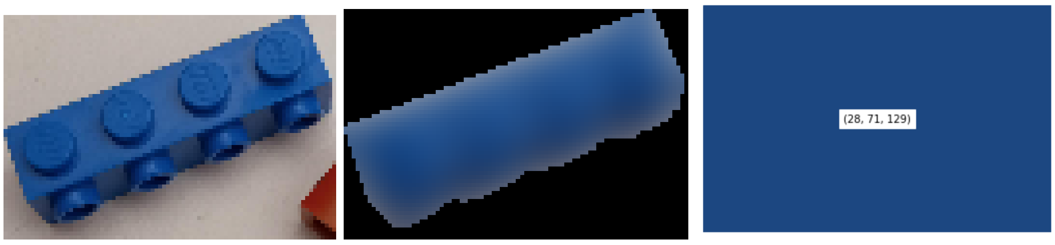
Some instances of the script failing to identify the lego pieces due to above depicted issues can be found on the annex. Notwithstanding, the model achieves, on the total 50 images, rather satisfactory accuracy both in lego and respective color detection.

Group

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

Annex - Examples

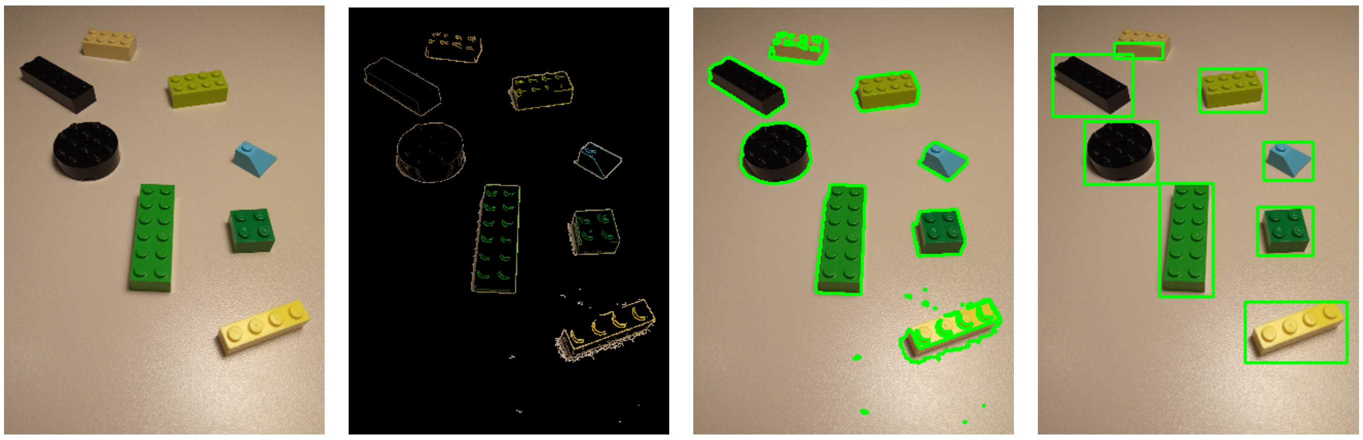
Color Detection

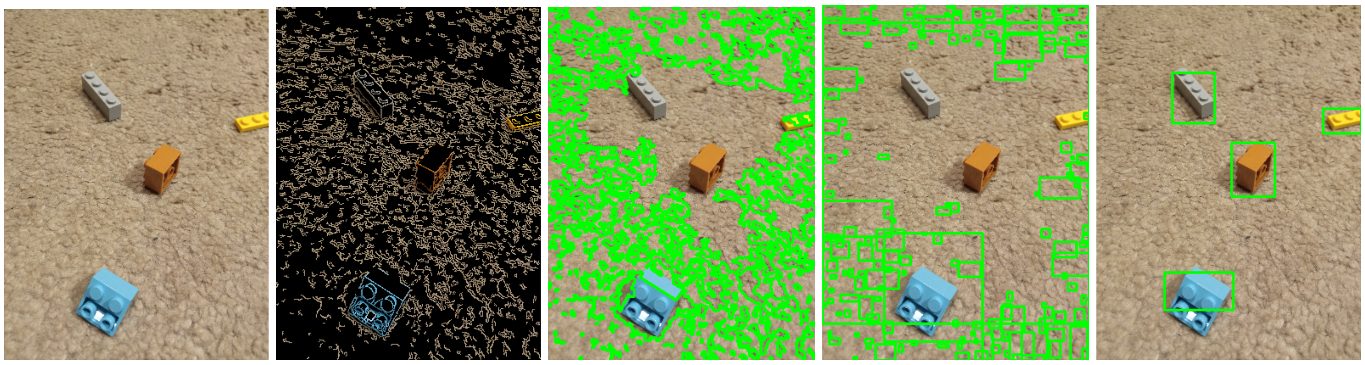


Model Step 4 Example of color extraction for a single lego

Execution 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