**LEGO Detection using MATLAB**

horizontal line

[**Image Processing 1**](#_vb7tei1dzrsw)

[**Methodology 2**](#_arolcxe0i15c)

[Pipeline Description: 2](#_vjirs6vej4nu)

[Logic Flowchart 3](#_qyf1fujxwieb)

[Algorithm/Technique Details: 3](#_utwsp3s39567)

[Preprocessing and Feature Extraction 3](#_4x5pfe9ragjw)

[Classification and Bounding Box Annotations 4](#_aub88irx8a2m)

[Limitations and Discussion: 5](#_yld24nxegw9l)

[Code Implementation: 6](#_ahrj2d2gyfv9)

[**Results 9**](#_5ypcb786z4zq)

[**References 9**](#_47q6id9wsp71)

# 

# Image Processing

The objective of this image processing project is to develop a **MATLAB-based Image processing pipeline** for detecting and identifying specific LEGO pieces within a dataset of 26 images. The problem includes locating similar LEGO pieces in images, annotating them with bounding boxes, labelling them with the corresponding part numbers, and classifying them within a class. The class corresponds to similar LEGO pieces.

This project focuses on recognizing and processing images of LEGO pieces by using image preprocessing techniques such as **grayscale conversion**, **binarization**, and **morphological operations** which we will go into more detail later. These techniques help reduce complexity and improve efficiency in detecting known LEGO parts. The project is designed to handle a predefined set of LEGO shapes, sizes, and part numbers.

# 

# 

# Methodology

## Pipeline Description:

The approach used to address the image processing task is as follows:

#### **Input Image Selection and Loading**

**Purpose:** The user is prompted to select an image from a predefined set (indexed from 1 to 26). This allows for controlled testing across a variety of images containing different LEGO pieces.

**Why it's important:** Providing flexibility in selecting input images ensures that the pipeline can be tested on various scenarios, such as different lighting conditions, backgrounds, and LEGO piece arrangements.

#### **Pre-processing**

**Purpose:** Pre-processing simplifies the image data for further analysis by applying operations that reduce noise and enhance the key features of LEGO pieces.

**Key techniques used:**

**Grayscale Conversion:** Converts the image to grayscale to eliminate colour information. This reduces computational load and focuses on shape and size, the most important features for LEGO detection.

*Why it’s important:* Color is not chosen as a distinguishing feature in this project even if it could be, so grayscale conversion removes unnecessary data while retaining the structural information of the LEGO pieces.

**Binarisation (Thresholding):** Converts the grayscale image into a binary image where LEGO pieces are represented as white (foreground) and the background as black.

*Why it’s important:* Simplifying the image into black and white allows for easy identification of object boundaries and reduces complexity when isolating LEGO pieces.

**Noise Removal (Morphological Operations):** Techniques such as erosion and dilation are applied to remove small noise elements from the image and enhance object boundaries.

*Why it’s important:* Removing noise helps avoid false detections and ensures that only significant objects (LEGO pieces) are detected.

#### **Feature Extraction**

**Purpose:** After pre-processing, the system identifies the distinct LEGO pieces within the image and extracts key features that describe their geometry.

**Key features extracted:**

* **Centroid:** The center point of each LEGO piece.
* **Bounding Box:** The rectangular region surrounding each LEGO piece.
* **Area and Perimeter:** These geometric properties help in classifying LEGO pieces based on size.
  + *Why it’s important:* These features are essential for uniquely identifying and distinguishing each LEGO piece from others. These are used later in the classification step to match the piece to a predefined pattern.

#### **Classification**

**Purpose:** Each detected LEGO piece is classified based on its geometric properties (such as width, height, and area) by comparing them to a set of predefined sizes.

**Key technique used:**

**Size Matching:** The extracted dimensions of each LEGO piece are compared to predefined geometric patterns that correspond to specific LEGO part numbers.

*Why it’s important:* By using predefined geometric patterns, the system can accurately identify LEGO pieces without relying on more complex techniques like machine learning, which simplifies the process and makes it more efficient for known shapes.

#### **Bounding Box Annotation**

**Purpose:** Once the LEGO pieces are classified, bounding boxes are drawn around them, and the corresponding part numbers are annotated on the image.

*Why it’s important:* Bounding boxes and labels provide a visual verification method, making it easier to confirm whether the correct LEGO pieces were detected and classified.

**Output Image Generation**

**Purpose:** The final output is an image with the detected LEGO pieces highlighted by bounding boxes and annotated with part numbers. This serves as the primary visual result of the detection process.

*Why it’s important:* Output images offer a clear view of the system’s performance, enabling users to validate the detection accuracy.

#### **Text File Output**

**Purpose:** In addition to the output image, the system generates a text file that records important details of the detection process, such as the centroid coordinates, bounding box dimensions, and part numbers of detected LEGO pieces.

*Why it’s important:* Text file outputs provide a structured summary of the results, which can be useful for further analysis or integration into other systems.

#### **Testing and Validation**

**Purpose:** Multiple images are processed to assess the robustness and accuracy of the image processing pipeline.

*Why it’s important:* Testing ensures the system works across different scenarios and helps identify any limitations or areas for improvement.

# Logic Flowchart

# 

### Limitations and Discussion:

The provided pipeline relies on a predefined list of LEGO sizes and tolerance levels. This approach is efficient for known LEGO sets but might not handle entirely new or custom shapes not included in the list.

A more robust classification approach could involve training a machine-learning model on a large dataset of labelled LEGO images. This would allow for more generalisable classification across a wider variety of LEGO shapes and sizes. However, it would require gathering a substantial dataset and training a model, which can be time-consuming and computationally expensive.

Thresholding and morphological operations may not perform as well on all image backgrounds or lighting conditions. In some cases, more advanced segmentation techniques like watershed or active contours might be needed to achieve better object separation.

Overall, the design choices in the pipeline prioritize efficiency and simplicity for a specific scenario (known LEGO set). The limitations highlight potential areas for improvement if dealing with a broader range of LEGO shapes or more complex image backgrounds.

## Code Implementation:

This block is responsible for extracting the relevant features that will be used to classify the LEGO pieces

| % Store centroids, bounding box coordinates, area, and perimeter of detected LEGO pieces features = zeros(numRegions, 8); % Each row represents the features of one region |
| --- |

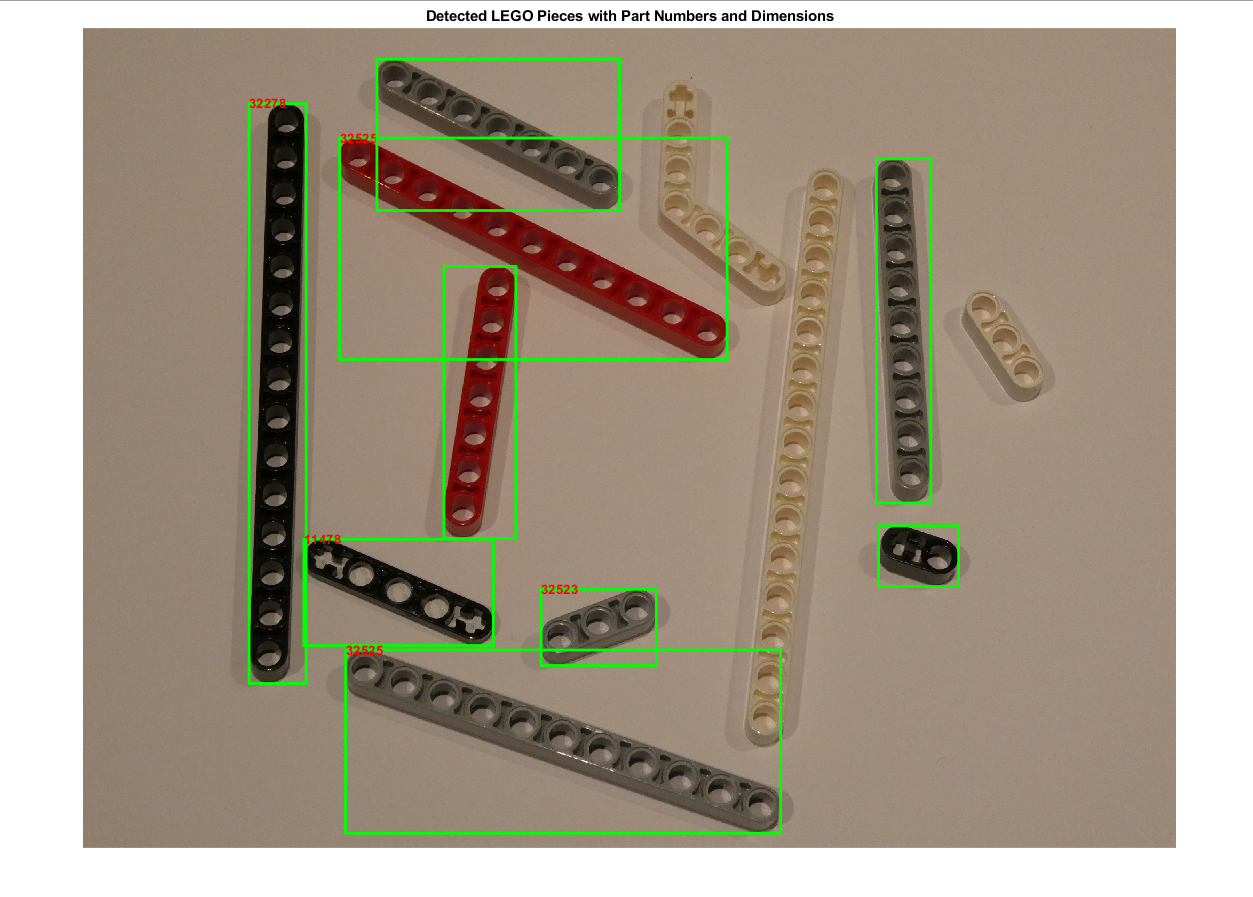
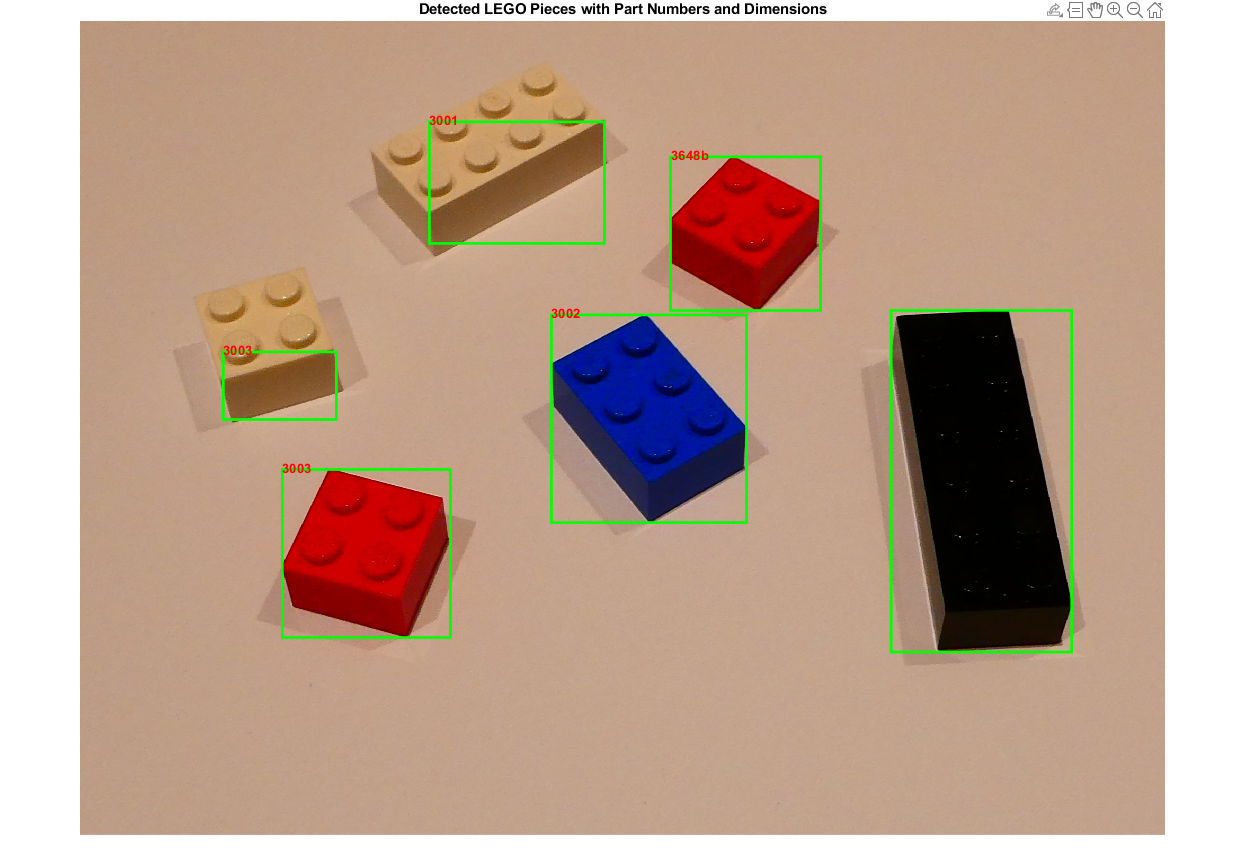
| for i = 1:numRegions  % Extract bounding box properties and area  bbox = stats(i).BoundingBox;  area = stats(i).Area;  perimeter = stats(i).Perimeter;  centroid = stats(i).Centroid;   % Filter out small bounding boxes  if area < minAreaThreshold  continue;  end    % Store centroid, bounding box coordinates, area, and perimeter  features(i, :) = [centroid, bbox, area, perimeter]; end  % Extract the widths and heights of the regions widths = features(:, 5) - features(:, 3); % Calculate width heights = features(:, 6) - features(:, 4); % Calculate height  % Transpose widths and heights for plotting sizes = [widths, heights]; % Combine into one matrix  % Create a bar plot to display the sizes figure; % Debug bounding box coordinates disp('Bounding Box Coordinates:'); disp('Region | XMin | YMin | XMax | YMax'); disp('----------------------------------'); for i = 1:numRegions  bbox = stats(i).BoundingBox; % Retrieve bounding box for the current region  xMin = round(bbox(1));  yMin = round(bbox(2));  xMax = round(bbox(1) + bbox(3));  yMax = round(bbox(2) + bbox(4));  fprintf('%6d | %5d | %5d | %5d | %5d\n', i, xMin, yMin, xMax, yMax);    % Calculate width and height  width = xMax - xMin;  height = yMax - yMin;    % Display width and height  disp(['Region ' num2str(i) ' Dimensions: Width = ' num2str(width) ', Height = ' num2str(height)]); end |
| --- |

The dimensions were displayed in the output windows and used to manually correspond the part numbers of the detected LEGO pieces in the LegoSize array.

| % Display the labeled image with bounding boxes, part numbers, and dimensions figure; imshow(selected\_image); hold on;  for i = 1:numRegions  % Extract bounding box properties  bbox = [features(i, 3), features(i, 4), features(i, 5), features(i, 6)];    % Calculate bounding box coordinates  xMin = round(bbox(1));  yMin = round(bbox(2));  width = round(bbox(3));  height = round(bbox(4));  xMax = xMin + width;  yMax = yMin + height;    % Define region dimensions  regionDimensions = [width, height];    % Display dimensions  disp(['Region ' num2str(i) ' Dimensions: Width = ' num2str(width) ', Height = ' num2str(height)]);    % Initialize part number  partNumber = '';   % Iterate through predefined LEGO sizes  for j = 1:size(legoSizes, 1)  pieceSize = legoSizes{j, 1}; % Extract predefined size    % Check if dimensions match within tolerance  if numel(regionDimensions) == numel(pieceSize)  sizeDiff = abs(regionDimensions - pieceSize); % Compare width and height    if all(sizeDiff <= [tol\_width, tol\_height])  partNumber = legoSizes{j, 2}; % Assign part number  break; % Exit loop once a match is found  end  else  % Handle incompatible sizes (optional: display a warning)  disp(['Warning: Incompatible sizes for piece ' legoSizes{j, 2}]);  end  end    % Draw bounding box  rectangle('Position', [xMin, yMin, width, height], 'EdgeColor', 'g', 'LineWidth', 2);    % Label with part number and dimensions  text(xMin, yMin - 5, partNumber, 'Color', 'r', 'FontSize', 10, 'FontWeight', 'bold'); end  title('Detected LEGO Pieces with Part Numbers and Dimensions'); hold off; |
| --- |

# Results

The LEGO detection system successfully identified most LEGO pieces within the provided dataset, drawing bounding boxes around them and displaying the correct part numbers. However, some pieces that were not in the predefined list were misidentified, and there was an issue with the output image generation where bounding boxes failed to display in the final annotated images



# References

Citations:

[1] Russ, J. C. (2016). The Image Processing Handbook (Sixth Edition). CRC Press.

[2] Jähne, B. (2013). Digital image processing: concepts, algorithms, and applications. Springer Science & Business Media.

[3] Soille, P. (2003). Morphological image analysis: principles and applications. Springer Science & Business Media.

# 

## 