Homework 2

16-311: Introduction to Robotics

Contents

Learning Objectives

In this homework, you will apply concepts from Lab 2 and expand your understanding by exploring filters and neural networks.

The primary objectives of this assignment are as follows:

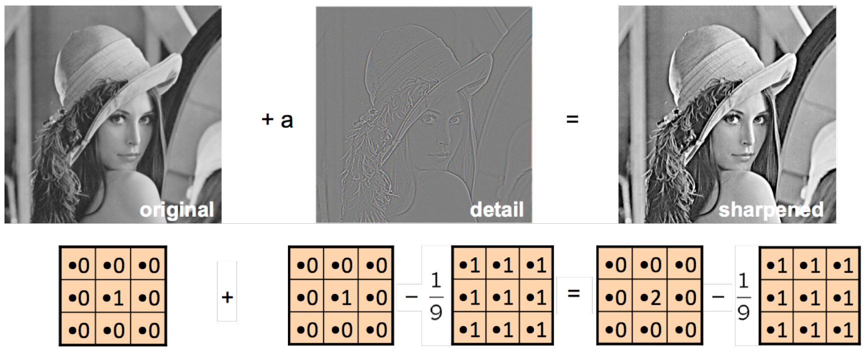
1. Develop practical skills in creating and utilizing filters for image processing.
2. Practice geometric methods, such as using similar triangles and stereo vision, to calculate distances.
3. Implement a neural network to classify images effectively.

Here is the point distribution for this assignment:

- Section 1 - 8 points
- Section 2 - 16 points
- Section 3 - 16 points
- Section 4 - 60 points

1 Filters

In this section, we will explore filters, also referred to as masks or kernels. Filters are fundamental tools in image processing and computer vision, used to manipulate pixel values within an image. They operate by systematically applying mathematical operations to a small, localized region of the image, often referred to as the filter's receptive field. This localized region is typically a small matrix, such as a 3×3 or 5×5 grid, that slides over the image to perform computations.



$$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} + \left(\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}\right) = \begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

1.1 What Are Filters and How Do They Operate?

Filters are matrices of numbers (weights) that are applied to a corresponding region of an image. The operation involves an element-wise multiplication between the filter and the pixels it overlaps, followed by summing the resulting values. This process, often referred to as convolution (or cross-correlation, depending on implementation), is repeated as the filter slides across the image. The resulting values form a new matrix, or output image, that reflects the operation performed by the filter.

Filters can highlight specific features, reduce noise, enhance edges, or extract patterns from images. For instance:

- **Smoothing filters** average pixel values to reduce image noise or blur details.

- **Edge detection filters** highlight areas of high contrast, identifying boundaries within the image.
- **Sharpening filters** enhance local details by amplifying differences between neighboring pixels.

1.2 Why Are Filters Used?

Filters serve numerous purposes in image processing, including:

- **Noise Reduction**: Smoothing filters, like Gaussian or average filters, reduce random variations in pixel intensity caused by noise.
- **Feature Extraction**: Edge-detection filters such as Sobel, Prewitt, or Laplacian kernels emphasize edges, which are vital for object recognition and analysis.
- **Data Interpolation**: When dealing with missing pixel data, filters can estimate values based on surrounding pixels.
- **Image Transformation**: Filters can transform an image's representation, such as enhancing specific patterns or detecting textures.

1.3 Exercises in Filter Operation

The following exercises demonstrate the practical applications of filters:

1. Design a 3×3 filter that computes the average of the nine pixels it overlaps with.
2. Create a 3×3 filter that computes the average using the center pixel and its four neighbors, based on 4-point connectivity.
3. Develop a 3×3 filter that calculates a weighted average of the nine overlapping pixels, assigning more weight to the center pixel. Ensure all weights sum to 1.
4. Construct a 3×3 filter capable of detecting vertical lines in an image.

2 Determining Distance Geometrically

Imagine you are working with a robot equipped with a Raspberry Pi Camera Module 3 that requires characterization. You aim to determine the distance between the lens and the image sensor (commonly referred to as the focal length) to use this information for future experiments involving distance estimation.

To approach this, you conduct an experiment:

- You position a one-inch cube exactly 12 inches (30.48 cm) away from the camera, ensuring the cube is centered in the camera's field of view.
- After capturing an image, you threshold and segment the image, counting the number of pixels corresponding to the cube's width.
- Repeating this process 10 times, you calculate an average width of 120 pixels for the one-inch cube.
- The camera's sensor resolution is known to be 4608×2592 pixels (horizontal and vertical respectively), with a sensor size of 6.45 mm horizontally.

Using this information, what is the focal length of the camera in millimeters? Please include at least one symbolic equation that you used with your answer.

3 Stereo Vision

This section aims to help you practice implementing the geometry required to determine depth and position from stereo camera images. Consider the robot depicted in the diagram below, equipped with two forward-facing cameras. The cameras are spaced 10 cm apart, and they are modeled based on the Raspberry Pi Camera Module 3 specifications.

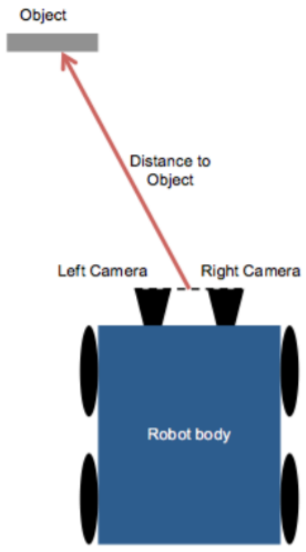


Figure 1: A top-down view of the robot and the object. Note: Diagram is not to scale.

The camera pair captures two frames, as shown below. In the image from the left camera, the object appears 30 pixels to the left of the image center. In the image from the right camera, the object is 50 pixels to the left of the image center.

The cameras have the following specifications: Focal length: 4.74 mm
Sensor size: 7.4 mm diagonal, 6.45 horizontal
Resolution: 4608 x 2592 pixels (horizontal and veritcal, respectively)
Pixel size: 1.4 μm x 1.4 μm

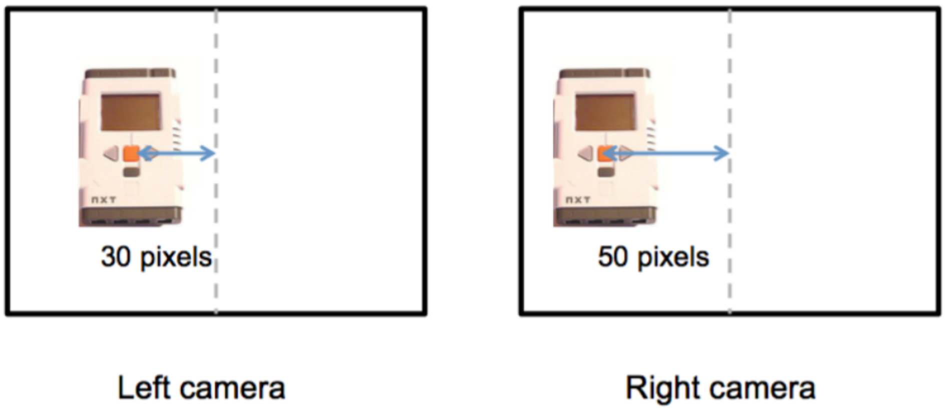


Figure 2: Images captured by the left and right cameras.

In your writeup, calculate the distance from the center of the front of the lenses to the center of the object in meters. Note that this requires considering the geometry in more than one dimension. Provide a clear explanation of your process. At a minimum, include two symbolic equations illustrating the geometric principles used to determine the distance.

4 Image Manipulation

The goal of this section of the assignment is to find a simplified Waldo figure in a series of images. You will create a Python function called `waldo` that takes the name of an image file as a parameter. This function should read a `.png` file (saved in the same directory as your program) and output a file called `output.txt` containing the x (horizontal) and y (vertical) coordinates of the center of each Waldo found in the image.

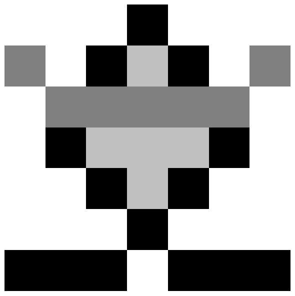


Figure 3: Large Waldo for illustrative purposes.

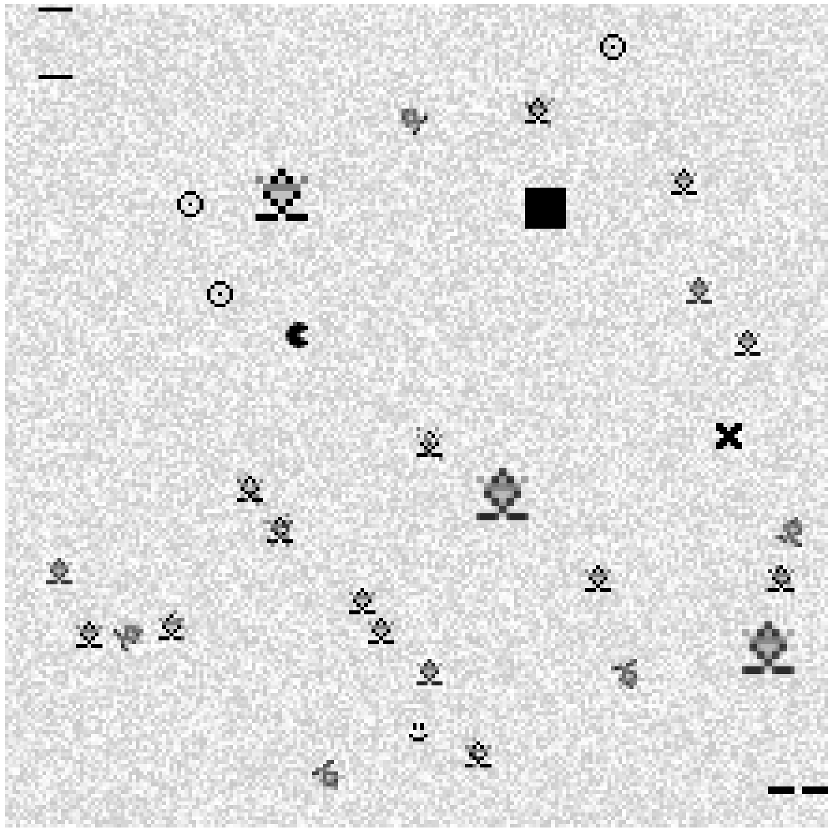


Figure 4: Sample environment for finding Waldos.

Function Specifications

- **Input:** The function should take a single parameter - the filename of the search image (e.g., `'waldoScene1.png'`).
- **Output:** The function should create a file named `output.txt` in the same directory.
- **Coordinate System:** The origin $(1, 1)$ is at the top-left corner of the image. X increases to the right, and Y increases downward.
- **Output Format:** Each line in the output file should contain two space-separated integers representing the x and y coordinates of a found Waldo. Do not include headers, extra text, or punctuation in the `.txt` file.

You can download the regular sized Waldo image from here: `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/homework/hw1/waldo.png`. This image will be in the same directory as your code for grading. Figure 4 shows a sample environment that you will be tasked to find waldo in.

Additionally, you can download the sample environment from here: `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/homework/hw1/waldoScene1.png` and a text file with the positions of Waldo is here: `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/homework/hw1/waldoSearch1output.txt`.

Grading Criteria

You will be graded based on an image that contains:

- 5 exact copies of Waldo
- 4 blurred copies of Waldo
- 7 copies of Waldo with noise
- 5 rotated copies of Waldo
- 1 double-size copy of Waldo
- 2 double and blurred copies of Waldo
- 7 not Waldos (like a smiley face)

Points will be awarded as follows:

- 40 points for finding between 1 and 3 Waldos
- 45 points for finding 4 Waldos
- 50 points for finding 5 Waldos
- 60 points for finding all Waldos from three different varieties

A position is considered accurate if it is within 2 pixels of the center of a small Waldo or within 4 pixels of the center of a double-sized Waldo.

You are not permitted to use library functions that trivialize the problem. This includes functions like `cv2_resize()`, `cv2_rotate()`, `skimage_feature_match_template()`, etc. You are allowed to use basic image processing functions like `np.array()`, `Image.open()`, `cv2_cvtColor()` (for color space conversions), and basic NumPy operations. Your code should work with an input image of any size.

You can start from this Python code: `https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/homework/hw1/waldo.py` You may change this code as much as you want or start from scratch with you own function. Whatever changes you make please ensure you are reading an image from the same folder as the function and writing to 'output.txt' your final (x, y) coordinate pairs.

Additional example: `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/homework/hw1/waldoScene2.png` and a text file with the positions of Waldo is here: `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/homework/hw1/waldoSearch2output.txt`.

What To Submit

Submissions are due on Gradescope by the date specified in the Syllabus.

Please submit the following items:

1. **PDF Document:** Include written answers for sections 1, 2 and 3. Name the file `hw2.pdf` and upload it to Gradescope.
2. **Code Submission:** Submit a folder containing all code files along with a `README` file that includes clear instructions on how to run the code on Gradescope.