# AA 274A: Principles of Robot Autonomy I
# Problem Set 3: Individual work

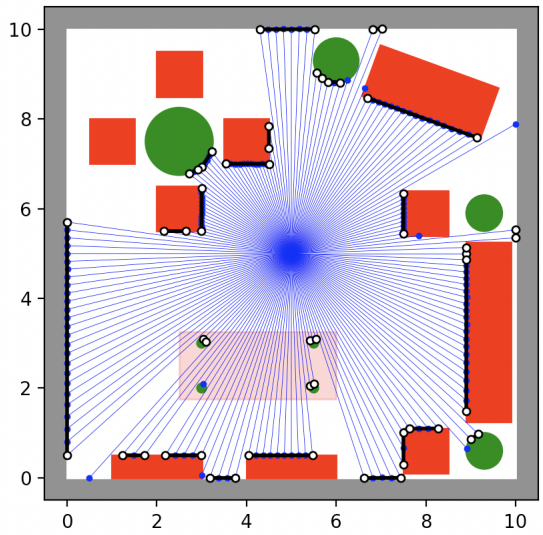
Name: Alban Broze
SUID: 06510556

11/11/2022

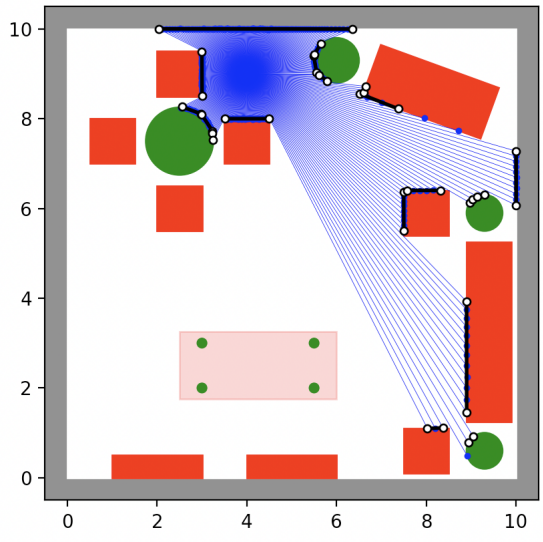
## Problem 2: Line Extraction

(ii) Below are the three plots showing the extracted lines for each of the data sets.

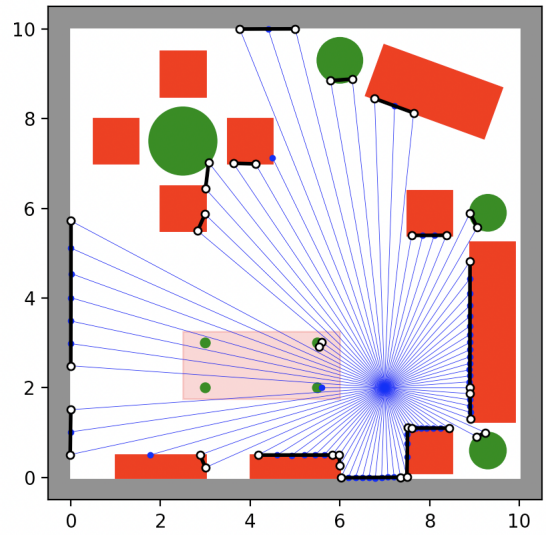
For rangeData_5_5_180.csv, the parameter choices are:
MIN_SEG_LENGTH = 0.01
LINE_POINT_DIST_THRESHOLD = 0.02
MIN_POINTS_PER_SEGMENT = 1
MAX_P2P_DIST = 0.5
The corresponding plot is:




For rangeData_4_9_360.csv, the parameter choices are:
MIN_SEG_LENGTH = 0.01
LINE_POINT_DIST_THRESHOLD = 0.02
MIN_POINTS_PER_SEGMENT = 1
MAX_P2P_DIST = 0.5
The corresponding plot is:




For rangeData_7_2_90.csv, the parameter choices are:
MIN_SEG_LENGTH = 0.01
LINE_POINT_DIST_THRESHOLD = 0.02
MIN_POINTS_PER_SEGMENT = 1
MAX_P2P_DIST = 0.5
The corresponding plot is:



# Problem 3: Linear Filtering

(i) (a)

$$G = \begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$$

(b)

$$G = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 7 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

(c)

$$G = \begin{bmatrix} -13 & -15 & -7 \\ -4 & -6 & -4 \\ 13 & 15 & 7 \end{bmatrix}$$

This filter computes edge detection along the x-direction (horizontally). This might be useful in computer vision to identify the boundaries of objects in an image. This could help reduce the amount of data to be processed since it filters out information that may be less relevant, while preserving important geometric properties of an image.

For $F'$, the edge detection will take place in the y-direction (vertically).

(d)

$$G = \begin{bmatrix} 3.5625 & 3.25 & 1.3125 \\ 5.25 & 5 & 2.25 \\ 4.3125 & 4.25 & 2.0625 \end{bmatrix}$$

This filter is a Gaussian Smoothing Filter. This means that all of the pixels are weighted by a Gaussian function (represented by $F$). For $F$ in this example, $\sigma = 0.85$. The advantage of the Gaussian filter is that it provides more weight to the neighboring pixels that are closer. This filter might be useful in computer vision to blur images and remove detail and noise. Furthermore, it's a realistic model of a defocused lens.

For $F'$, which is a Moving Average Filter, the neighboring pixels are weighted evenly instead of being weighted by a Gaussian function. Thus, the smoothing effect is different (in this case, the blurring effect is a little bit more intense with the MAF).

(ii)

$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} \sum_{w=0}^{c-1} F(u,v,w) \cdot \bar{I}(i+u, j+v, w)$ can be written as $G(i,j) = \mathbf{f}^T \mathbf{t}_{ij}$ since:

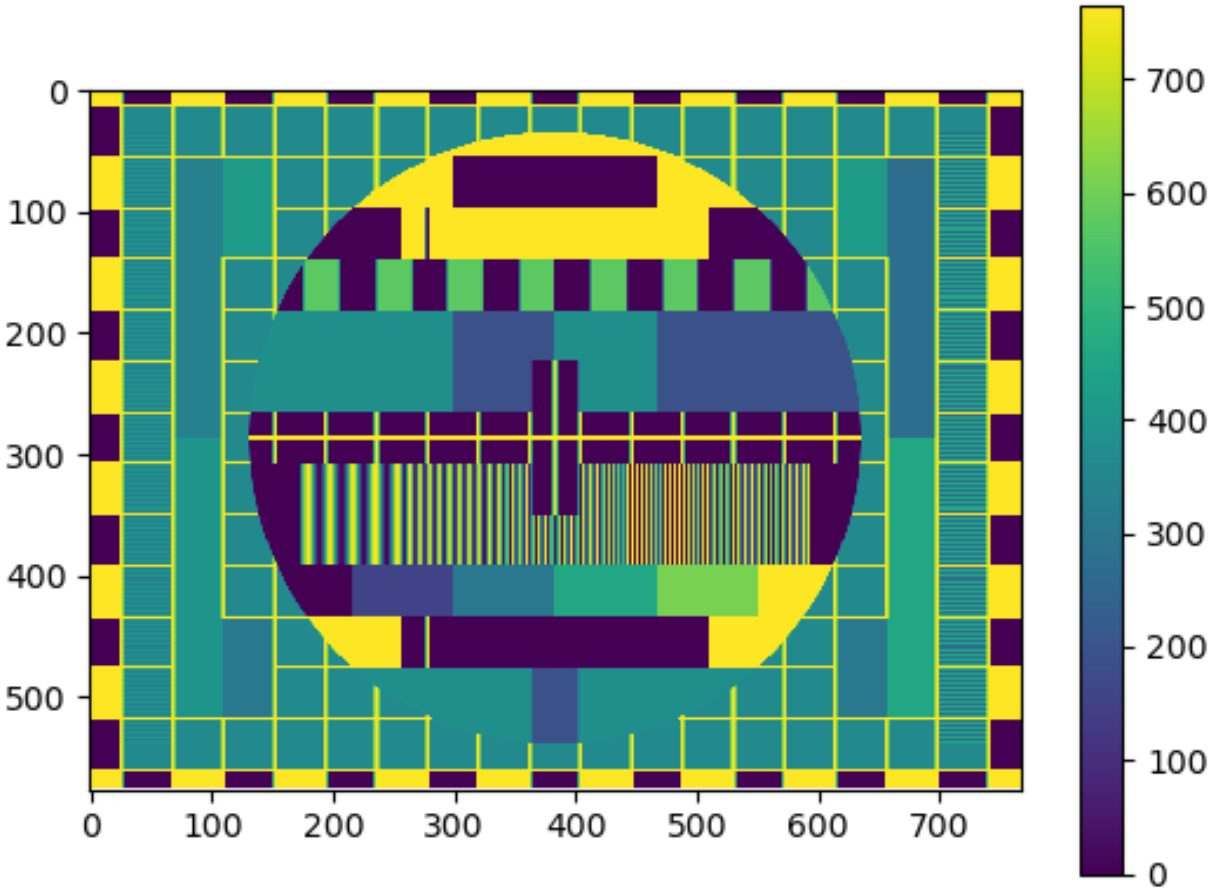
$$\begin{aligned} G(i,j) = {} & F(0,0,0) \cdot \bar{I}(i,j,0) \\ & + F(0,0,1) \cdot \bar{I}(i,j,1) \\ & + F(0,0,2) \cdot \bar{I}(i,j,2) \\ & \ldots \\ & + F(k-1, l-1, c-1) \cdot \bar{I}(i+(k-1), j+(l-1), c-1) \end{aligned}$$

This corresponds to the sum of the element-wise multiplication of two vectors. These vectors are defined as follows:

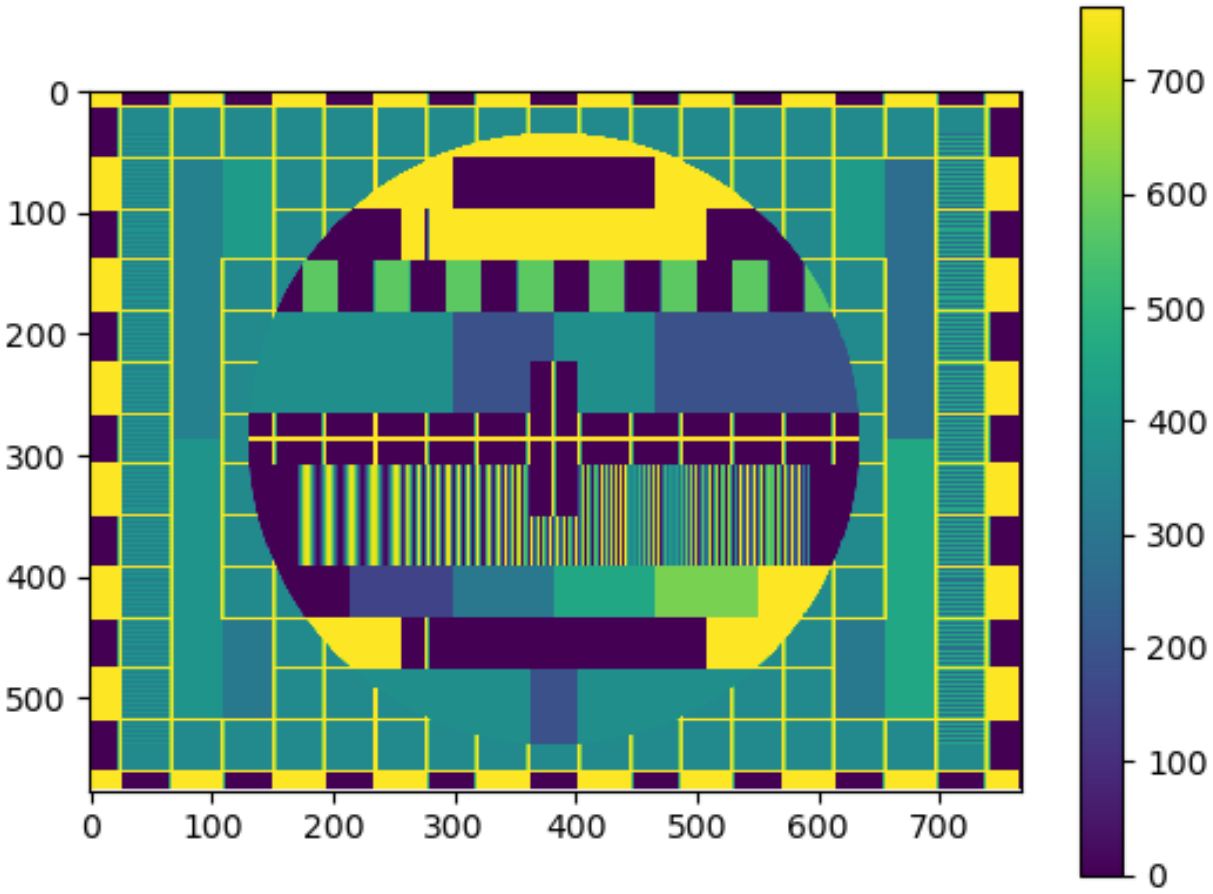
$$\mathbf{f} = \begin{bmatrix} F(0,0,0) \\ F(0,0,1) \\ F(0,0,2) \\ \dots \\ F(k-1,l-1,c-1) \end{bmatrix} \text{ and } \mathbf{t}_{ij} = \begin{bmatrix} \bar{I}(i,j,0) \\ \bar{I}(i,j,1) \\ \bar{I}(i,j,2) \\ \dots \\ \bar{I}(i+(k-1),j+(l-1),c-1) \end{bmatrix}$$

Both vectors have length $k$ x $l$ x $c$. Furthermore, this dot product has to be computed for each $(i, j)$ pair.

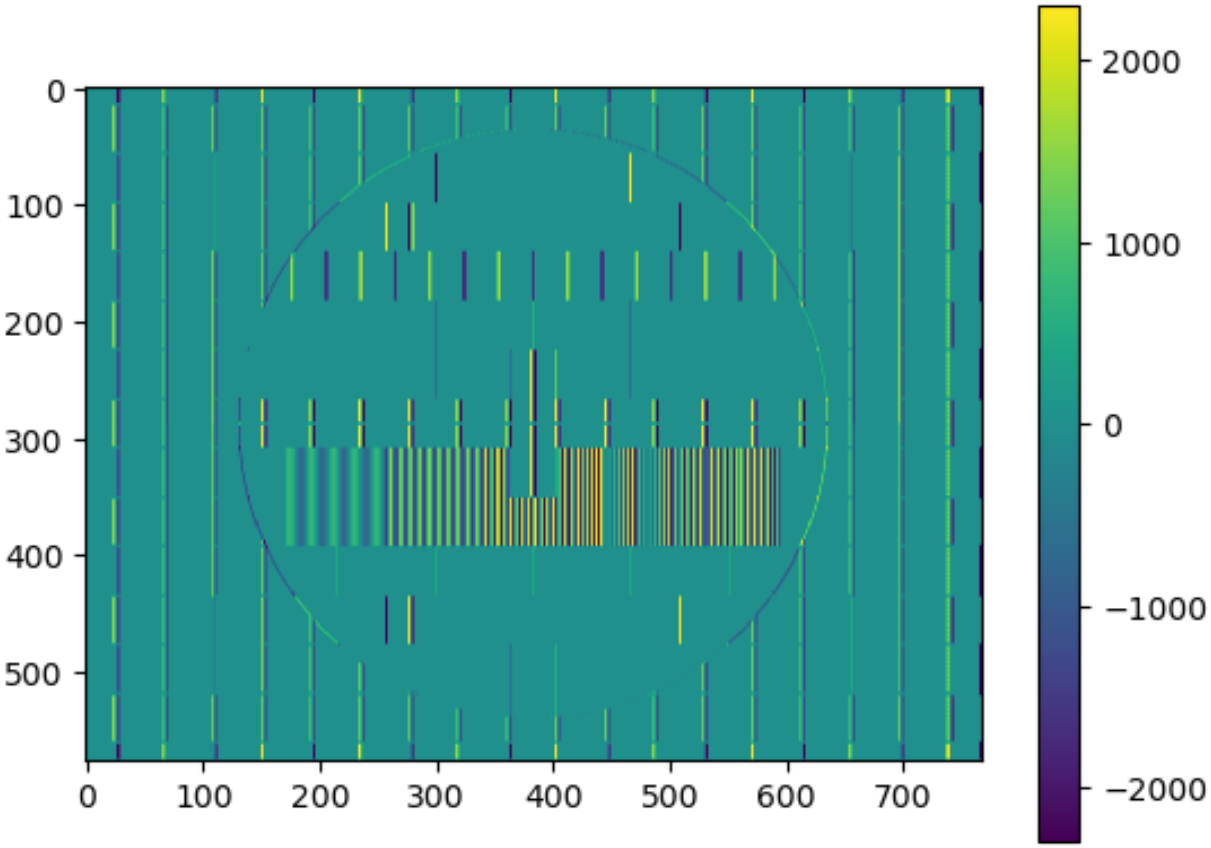
(iii) Resulting images of the corr function in linear_filter.py can be found below.



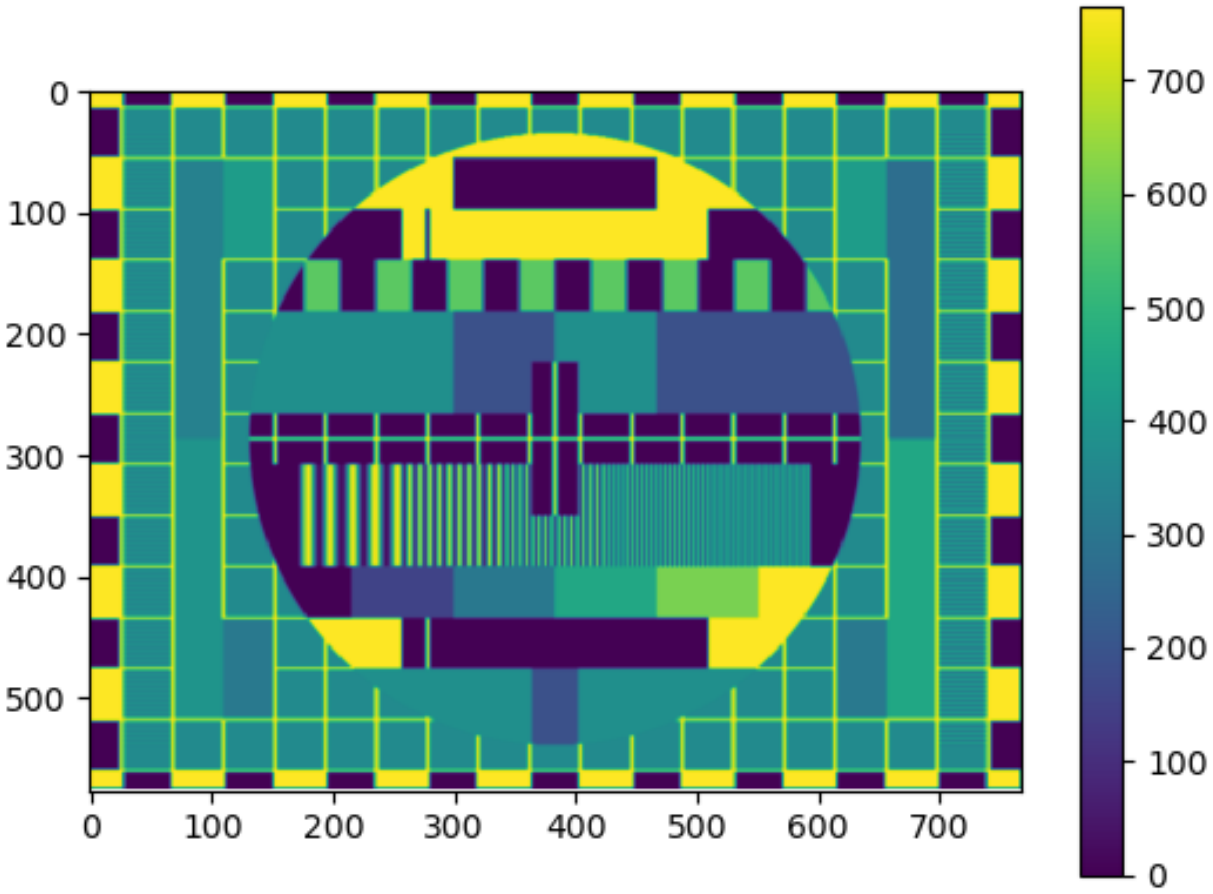

Using filt1


0
100
200
300
400
500
0
100
200
300
400
500
600
700
700
600
500
400
300
200
100
0

Using filt2


0
100
200
300
400
500
0
100
200
300
400
500
600
700
2000
1000
0
−1000
−2000

Using filt3




Using filt4

(iv) The runtime for the corr function for each filter:

```
(base) albanbroze@DN0a229276 Problem_3 % python3 linear_filter.py
Correlation function runtime: 1.0070269107818604 s
Correlation function runtime: 1.0138239860534668 s
Correlation function runtime: 1.021338939666748 s
Correlation function runtime: 1.0501599311828613 s
```

We can thus see that the runtime is around 1 second for our implementation, which is about 50 times slower than the typical runtime of 20 milliseconds.

Two suggested ways to speed up the runtime could be:

(a) Since the computations for each pixel are independant, there is no need to compute them sequentially. A very effective way to improve the runtime would be by using parallel computing (e.g. CPUs or GPUs).

(b) Reduce the number of operations to be performed. Consider the grayscale case where we want to apply correlation to an image $I$ of size $m \times n$ with a filter $F$ of size $k \times k$. By using the approach described in part $(i)$, we need $(2k^2 - 1)mn$ operations to compute $G(i, j)$. However, if the filter we're using can be written as an outer product $F = \mathbf{f}\mathbf{f}^{\mathrm{T}}$, we can reduce the number of operations needed to compute $G(i, j)$ to $2(2k - 1)mn$. This is true because we can then run the correlation

on the image with each filter vector ($\mathbf{f}$ and $\mathbf{f}^{\mathrm{T}}$) successively and separately (one after the other). We can thereby reduce the number of operations by an order $O(k)$. Reducing the number of operations will reduce the runtime of the correlation function.

(v) If a filter can be written as $F = \mathbf{f}\mathbf{f}^{\mathrm{T}}$, it is easy to find vector $\mathbf{f}$. Let's imagine $\mathbf{f}$ = [a,b,c]$^{\mathrm{T}}$. Then, we know that $F$ should have the following format:

$$F = \mathbf{f}\mathbf{f}^{\mathrm{T}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{bmatrix} a & b & c \end{bmatrix} = \begin{bmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{bmatrix}$$

The format above is the mathematical condition for $F$ to be written in the form F = $\mathbf{f}\mathbf{f}^{\mathrm{T}}$. If this condition is satisfied, we can find the vector $\mathbf{f}$ by taking a vector composed of the square roots of the diagonal terms of $F$. This also means that, if $F$ can be represented in the same format as the matrix above, we can reduce the number of operations significantly (by an order $O(k)$, see above) and thus the runtime too. This method works for a vector $\mathbf{f}$ with any length, as long as $\mathbf{f}$ and $\mathbf{f}^{\mathrm{T}}$ represent the same vectors where one is the transpose of the other. Should this not be the case, on can compute the singular value decomposition (SVD) of $F$. The mathematical condition that holds on F is that it has to have rank 1. The vector $\mathbf{f}$ is then equal to the eigenvector that corresponds to the non-zero eigenvalue of $F$.

(vii) Convolution can be implemented with correlation by first rotating the filter by 180° (once vertically and once horizontally).

# Extra Problem: Template Matching

(iii) Template matching performs poorly on other images. Four ways to improve template matching are:

(a) Randomized rotation of the template

(b) Change template colors

(c) SIFT (Scale-Invariant Feature Transform)

(d) Deep Learning with Convolutional Neural Networks (CNNs)

The first method is about running the template matching algorithm multiple times with the template being rotated randomly at each iteration. This would allow to match templates to images even if they don't have the same orientation.

The second method is about running the template matching algorithm multiple times with the color of the template being modified. This would allow to better detect the template even for different brightness/luminosity cases. It's also possible to change both the image and the template to a greyscale to make them color-invariant, thereby removing a variable in the matching process.

The third method is about identifying interesting points on the object and extract them to provide a "feature description" of the object to be recognized. Such points can generally be found on high-contrast regions of an image (e.g. object edges and contours).

The fourth method is about training a neural network to recognize our object on a training data set. The purpose is then for our model to be able to recognize our specific object on new images. These methods are two examples of methods that can be used to improve template matching without re-scaling our image multiple times.