Beeldverwerken

Robot Positioning Using an Omnidirectional Camera

Exercise 5

Verna Dankers (10761225), Adriaan de Vries (10795227)
Teaching Assistant Nick de Wolf

May 18, 2016

1 Introduction

When a robot acts in the real world, it should be able to move quickly and efficiently. Achieving those goals starts with creating an efficient positioning system, which is the task that will be discussed in this assignment. The robot at hand has an omnidirectional camera. When positioning itself, it matches its present view with a library of pre-recorded views at known locations. This can be implemented efficiently by the use of Principal Component Analysis (PCA), a technique for characterizing large data sets by their essence. This report discusses the theory behind PCA and the implementation of the positioning system for the robot.

2 Theory

Here we consider two similarity measures to compare two images and the theory behind PCA.

2.1 Similarity measures

Two images can be compared by calculating their correlation:

$$c = I_1 \circ I_2 = \frac{1}{K} \sum_{i=1}^{N} \sum_{j=1}^{N} I_1(i,j) I_2(i,j)$$

where K is a normalizing constant, N is the number of gray value pixels and I_1 and I_2 are the images. The larger c, the more similar I_1 and I_2 are. For the correlation measure to be meaningful both the images and the intensity values of the pixels should be normalized.

Another similarity measure could be the sum of the squared differences of the pixel values of both images:

$$c = \sum_{i=1}^{N} \sum_{j=1}^{N} (I_1(i,j) - I_2(i,j))^2$$

These two measures are related in the way that, if the pixel intensities have been normalized, the sum of squared differences is equal to $2 + I_1 \circ I_2$ for $K = -0.5$, as shown in Appendix A.

2.2 Principal component analysis

Now that we have introduced two similarity measures to compare two individual pictures, we need some efficient way to compare an image to a lot of images and to find its best match. We will use the PCA.

When a lot of comparisons have to be made for the positioning system of the robot, it is useful to represent the images with less dimensions to speed up the process. The PCA will provide us with a new basis, which is a linear combination of the original basis, that re-expresses the original data set in the best way. We will represent images in the eigenspace of its covariance matrix, because of the advantage that the components associated to the largest eigenvalues are the most significant ones for the representation of the image. That is, they minimize redundancy and maximize variance. We will work with images in a vector representation instead of the usual matrix representation. The image vectors can then be joined in a matrix:

$$X = \begin{bmatrix} (\vec{x_1} - \vec{\bar{x}}) & \dots & (\vec{x_n} - \vec{\bar{x}}) \end{bmatrix}$$

where $\bar{x}$ is the average over the entire data set. Now we can write each $\vec{x_i}$ as:

$$\vec{x_j} = \vec{\bar{x}} + \sum_{i=1}^{n} g_{ji} \vec{e_i}$$

where $\vec{e_1},\ldots,\vec{e_n}$ are the eigenvectors of the covariance matrix X^TX and $\vec{g_j}$ is the vector of the components of $\vec{x_j}$ in the eigenspace.

When we represent the image with less dimensions only the first k components are used, because the k largest eigenvalues are the most significant ones. Therefore we can approximate $\vec{x_j}$:

$$\vec{x_j} \approx \sum_{i=1}^{k} g_{ji}\vec{e_i} + \vec{\bar{x}}$$

Representing the images as described above allows us to calculate the correlation more efficiently. The Euclidean distance in eigenspace is equivalent to image correlation: maximizing the correlation is equivalent to minimizing distance, as shown in Appendix A. And when we approximate the distance by the distance in the k-dimensional eigenspace, we only have to compare k values instead of n. Because n is the total amount of pixels in the image, and k is a parameter with a relatively very small value, this makes comparing images much faster.

3 Algorithm

3.1 Principal Component Analysis

The PCA algorithm takes as input a training set X and a value k, which represents the amount of dimensions you want to reduce your data to. It first makes sure the data has a mean of zero, by subtracting its mean from it. Then it uses the eigenvectors of the covariance matrix with the k largest associated eigenvalues to find the most expressive basis in k dimensions. These eigenvectors form a matrix which, when multiplied by the matrix X, becomes a matrix that can be used to reduce the dimensionality of the data.

```
function pcaMatrix = our_pca(X, k)

    X = X - mean(X)

    n = size(X,2)
    Cov = (X * transpose(X))/(n-1)

    % getting the first k eigenvectors as a matrix
    EigVecs = Eigenvectors(Cov, k)

    pcaMatrix = EigVecs * X

end
```

3.2 Positioning with Nearest Neighbour

The nearest neighbor algorithm is a fairly simple one. It gets an image and a database of images (or in this case the dimension-reduced version of the images), and it compares the image with the database to see which image in the database resembles it most closely. Resemblance is measured in correlation as explained above.

```
function best_match = nearest_neighbor(image, database)

        % first we instantiate variables
        best_match = null
```

```
        best_correlation = 0

        for compare_image in database
                correlation = calc_correlation(image, compare_image)
                if correlation > best_correlation
                        best_match = compare_image
                        best_correlation = correlation
                end
        end
end
```

4 Experiments

The code for the experiments can be tested by deflating `assignment5.zip` and running `e5_adriaan_devries_verna_dankers` section by section.

The data used for the experiments comes from an omnidirectional camera and contains pictures of the building at Kruislaan 403, Amsterdam. The data set contains 550 pictures and all images have been annotated with a 2D position. The first picture of the data set is shown in Figure 1 to give you an idea of what the data looks like.

4.1 Principal Component Analysis experiments

The implementation of the PCA algorithm has been described in Section 3.1, Figure 2 shows the PCA vectors of the first nine pictures from the data set.

One important decision to make when applying PCA is how many principal components will be used to reduce the number of dimensions of the data. We would like to have both the best performance for the positioning system and maximum reduction of dimensions for efficiency. Therefore the choice for the k principal components corresponding to eigenvectors of the k largest eigenvalues should be based on results from the task at hand to find the optimal number. Figure 3 shows the values of the 50 largest eigenvalues. Only the first eigenvalue is missing because it was significantly larger. After the 25th eigenvalue the values of the eigenvalue differ so little that it may not help to add more principal components at that point: therefore we chose to use 25 principal components.

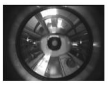

Figure 1: First picture from Kruislaan data set

Now that we know how to reduce dimensions and to how many dimensions we want to reduce, it is time to check how much faster the images can be checked for similarity after the application of PCA. In order to do this, we compared the first 50 images to every other image in the data set for both the naive similarity check and the check after application of PCA, while keeping track of the time. This resulted in an average runtime of 0.0956 seconds for the naive check and 0.0164 seconds for the improved check.

4.2 Positioning with Nearest Neighbour experiments

With the nearest neighbor algorithm a couple of experiments have been done. In all of these, the algorithm has to find the picture from the training set that most closely resembles the input picture. If the position of

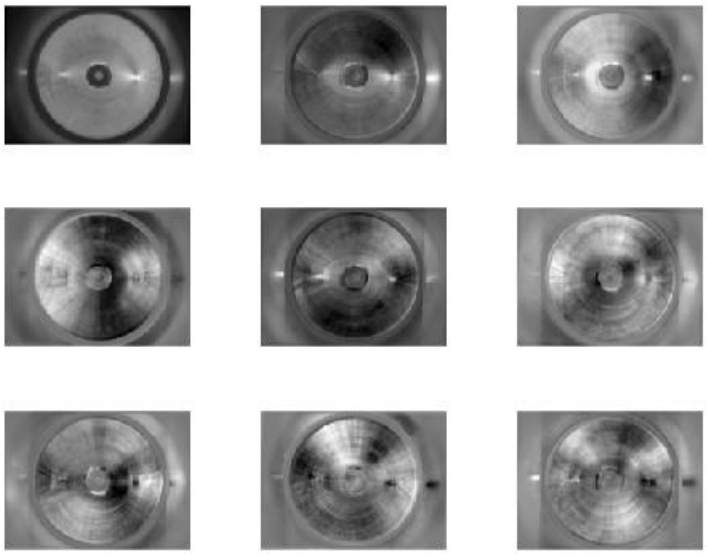

Figure 2: The PCA vectors of the first nine pictures

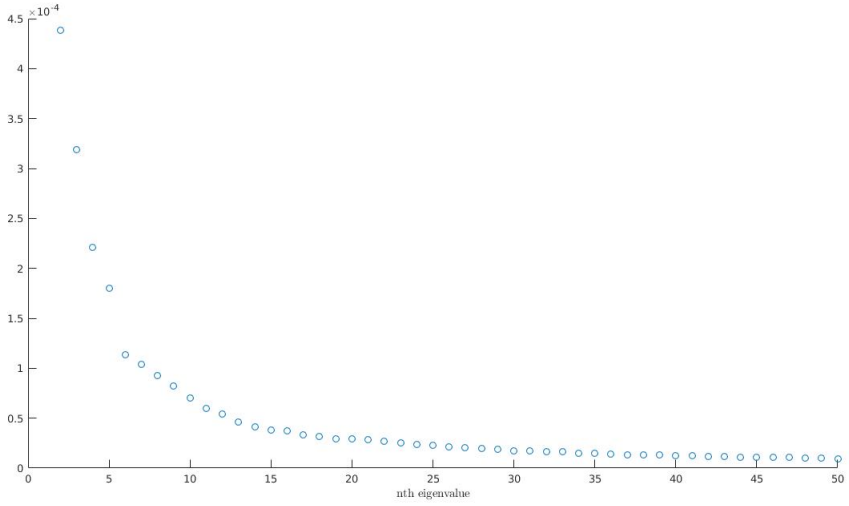

Figure 3: Largest eigenvalues from Kruislaan data set

the output image is within 150 units of the input image, it is deemed as correct.

First, the nearest neighbor algorithm ran on the images as a whole, so without reducing the dimensions first. This gave an accuracy of 74%, with a runtime of slightly over 12 seconds. Although 12 seconds is not unreasonably long, it is not hard to imagine tasks where time reduction is necessary (such as tasks with higher resolution images).

Then, PCA was used to first bring the images down to a vector with k dimensions. The effects of this on the runtime and accuracy can be seen in Figure 4. We notice that the time it takes to complete does not really differ between $k = 10$ and $k = 50$, if we assume the peaks at 26 and 40 to be anomalous. This

is most likely because the actual comparing of images is almost instant with so few dimensions. Of course the runtime would increase with $k \gg 50$, but at $k = 50$ the accuracy seems to not be increasing any further. from the plots we conclude that an ideal value for k should be at least 25, as any lower value shows a drop in accuracy.

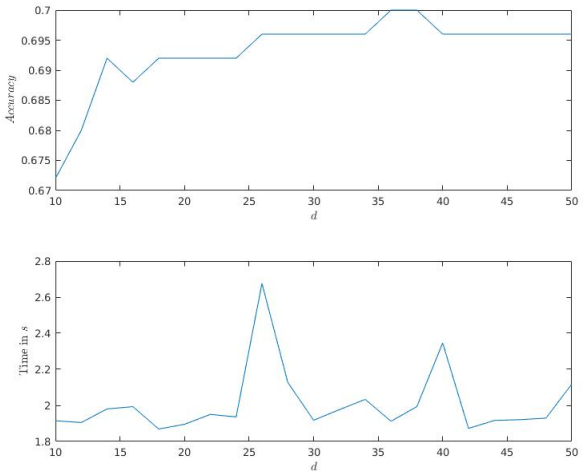

Figure 4: Accuracy and elapsed time for the nearest neighbour comparisons

Finally, the algorithm was also used on images which had the (mostly dark) edges cut off. For this experiment we used PCA to bring down the dimensionality of the vectors to 20. When an edge of 5 pixels wide was removed from the pictures, the accuracy remained the same, with 69%. Cutting off 10 pixels still did not change the accuracy, and at 20 pixels the accuracy dropped to 67%. However, as we have low resolution images, taking away an edge of 20 pixels wide removes about half the pixels. This is far exceeding the initial goal of taking away some of the darker edges.

We can conclude that taking off a part of the edge does not help the algorithm, as PCA already does a good job of ignoring this irrelevant data, because of its low variance.

5 Conclusion

With an accuracy of 0.7 and negligible run-time, PCA gives an inexpensive method for comparing images, in this case to find the position of a robot. However, 0.7 is not sufficiently near 1 to be completely relied upon. Any system using this exact method will have to take into account frequent errors.

A Theory questions

1. Correlation:

$$I_1 \circ I_2 = \frac{1}{K} \sum_{i=1}^{N} \sum_{j=1}^{N} I_1(i,j) I_2(i,j)$$

   Sum of squared differences:

$$c = \sum_{i=1}^{N} \sum_{j=1}^{N} (I_1(i,j) - I_2(i,j))^2$$

$$\sum_{i=1}^{N} \sum_{j=1}^{N} I_1(i,j)^2 - 2I_1(i,j)I_2(i,j) + I_2(i,j)^2$$

   Now, if the images are normalized so that $\sum_{i=1}^{N} \sum_{j=1}^{N} I(i,j)^2 = 1$, the sum of squared differences is equal to $2 + I_1 \circ I_2$ if we set $K = -0.5$. So the condition that needs to hold is that the pixel intensities have to be normalized. If we take $K = 1$, so ignore the normalizing term altogether, we end up with a negative proportionality between the correlation and the sum of squared differences.

   In conclusion, the sum of squared differences is usable as a correlation-measure if the input images are normalized.

2. The images seem to not be normalized. But not to worry! We can do that!

3. The correlation is simply the sum of the product of the corresponding entries of the matrices. A dot product is also the sum of the product of corresponding entries, but defined for vectors. If the matrix is a vector, the correlation is the dot product.

4. 150 times 112 does indeed equal 16800.

5. Trucco, on page 266, mentions that X is an $N^2 \times n$ matrix. N is the width or height of a square image, and n is the amount of images. Thus, a row in X represents the measurements of a single pixel for all the images. A column represents the measurements of all pixels in a single image. This is equivalent to the explanation of X in Shlens page 5.

6. (Note that this is where the theory questions stop being questions.)
   This is not really a question, but it has most certainly been read.

7. We will look up what we forgot.

8. Thanks for the explanation.