Euclidean Distance for evaluating of different algorithms in terms of machine vision

I. The „scientific” method for comparision of Shi-Tomasi and my object tracking approach

In the case of machine visison it is a coner point of tracking objects on the scene. There is a crucial aspect of tracking, i.e. drawing the trajectory an object, namely the in resepct of self-driving car system, where the distance between objects, their sensing and based on that decison making about necessary manouver is an inevitable task.

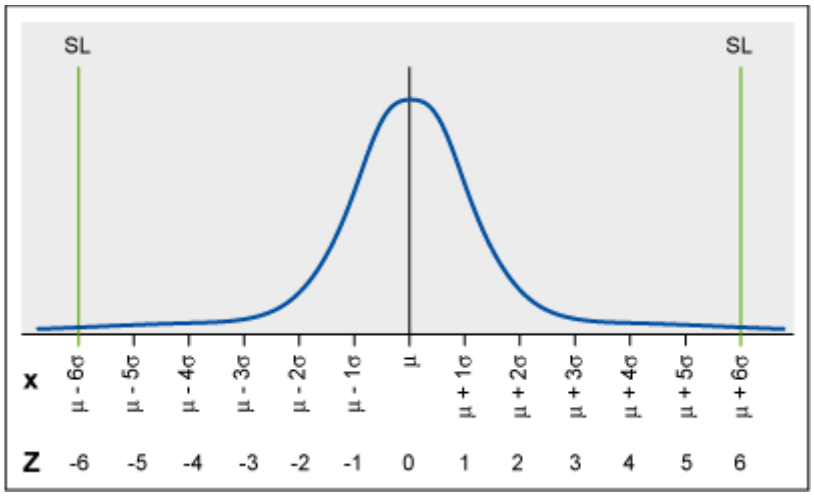
So concerning object tracking there are some well known methods, such as Shi-Tomasi Corner Detector, ….sajátot ide írni.

Since I have developed/used a non-Shi-Tomasi method in my work it is a must to compare the two methodes and try to validate my approach compared to Shi-Tomasi.

The most obvious appraoch can be the use of Euclidean distance between my points (which give the trajectroy of the objects) and the points of Shi-Tomasi Conrer Detector.

The question however is remain whether how we should compare the two approach and how should I validate my appraoch against Shi-Tomasi. Firstly I assume that Shi-Tomasi is a scientifically well established methos, this is the basis. If my approach has as small a deviation as possible from Shi-Tomasi then I consider it as a robust method.

The question is whether how we use the Euclidean Distance as a basis for validation/evaluation. My assumption is that if we compare Shi-Tomasi corner points with the geometric center of my object based on Euclidean Distance, and after that preparing a time series of the distances then we can have an insight into the behaviour of my modell. Namely if the change of the distances in time are not so significant then it can be handled as an off set. Of course if we see a significant changes in time then then my method is questionable from robustness point of view. Of course there is still a remaining question, how we should define the acceptable change of distances in time, i.e the measure of change which is not so significatnt and can be accepted as an off set. I have found the six sigma approach which can be a good basis for our evaluation:



If we create a histrogram of the distances and fits to a gaussian distribution we can calculate the variance and then the sigma of process. If the process (i.e. the time series of distance) exhibits such distribution which corresponds to six sigma rule then I propose to accept my approach.

Of course there are some „small" technical questions remaining:

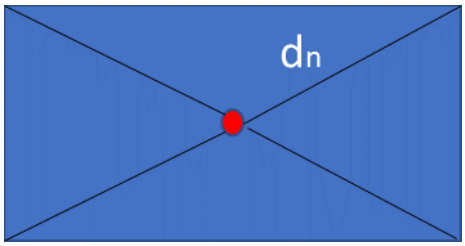
1. How should we choose the coordinate system
2. How should we define the distance
3. How should we handle the time
4. How should we handle the different shape of object
5. How many objects should be compared

Below I propose a workhypotesis:

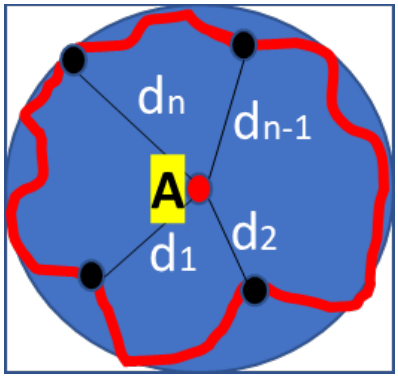
1. My tracking point should be in the center of a cartesian coordinate (the zero point). From computation point of view it seems to be a cost-effective procedure.
2. There are a lot of corner point generated by Shi-Tomasi, my proposal is to get the arithmetic mean of the distaneces between these points and my geometric center. This one measure will be the basis of the time series data.
3. Time: it is a good question becasue it depends on the length of example video. My proposal is to count the given pictures of the video, and take from it 5% of the pictures. Some statistical examination rules can be presented.
4. I peopose three basic shapes form the video which represent extremes. e.g. one shape should be a non convex type (crescent shaped), other ones can represent an other type of extremes (round shape).
5. As I proposed three type of objects should be scrutinised which represent extremes form shape point of vie, i.e. 3 objects will be involved.

Type of objects:

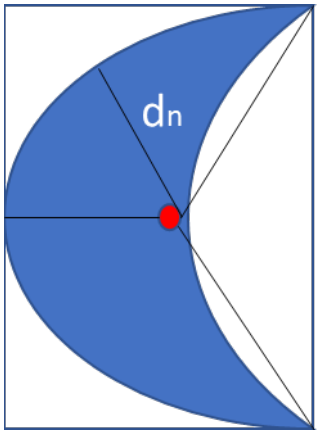
a. Similar to a rectangle



b. Similar to a round



c. crescent shaped



The cubature of each type of shape equals of my shape since I also worked with a cubature. The real Shi-Tomasi object of course are not a rectangle or round shape, but they also can be put a characteristic shape. The algorithms can be captured by basen on above legend in b point. The movement of A point (from time t1 to tm) will be compared with dn, dn-1…d2,d1 points. The square of distances: D1: A-d1; D2: A-d2…; Dn-1: A-dn-1, Dn: A-dn. The arithmetic mean of these distances will be displayed in time as a time series data, after that the histrogram can be prepared and variance of distribution also can be calculated.

II. In the below section I have quoated an article which examines the different approach of used distance methodology. One can find the advanteges and disadvanteges of the different type of distance methodology.

Distance metrics play a significant role in machine learning and deep learning. Machine learning algorithms like k-NN, K Means clustering, and loss functions used in deep learning depend on these metrics.

Thus, understanding the different types of distance metrics is very important to decide which metric to use when. For example, k-NN often uses euclidean distance for learning. However, what if the data is highly dimensional? Will euclidean distance still be valuable? No, it won't because, as we know, euclidean distance is not considered a good metric for highly dimensional space(refer to this link for more insight). So I guess you can relate now that knowing your distance measures can help you go from a poor classifier to an accurate model.

In this blog, we will walk through some of the most used Distance metrics and their use case and disadvantages, and how to implement them in python. The ones we will cover are as follows-:

- Euclidean Distance
- Manhattan Distance
- Chebyshev Distance

- Minkowski Distance
- Hamming Distance
- Cosine Similarity
- Jaccard Similarity
- Sørensen-Dice Index

Euclidean Distance

Euclidean Distance is one of the most commonly used distance metrics. Mathematically it is the square root of the sum of differences between two different data points. Following is the formula for calculating the distance between two k dimension vectors.

$$D(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2)}$$

Image By Author

Applications/Pros-:

- Highly intuitive, easy to understand, and an easy to use metric
- It should be used in algorithms like KNN or K Mean, where we have low dimensional data, and straight forward distance between data points is enough to gauge the similarities of these points.

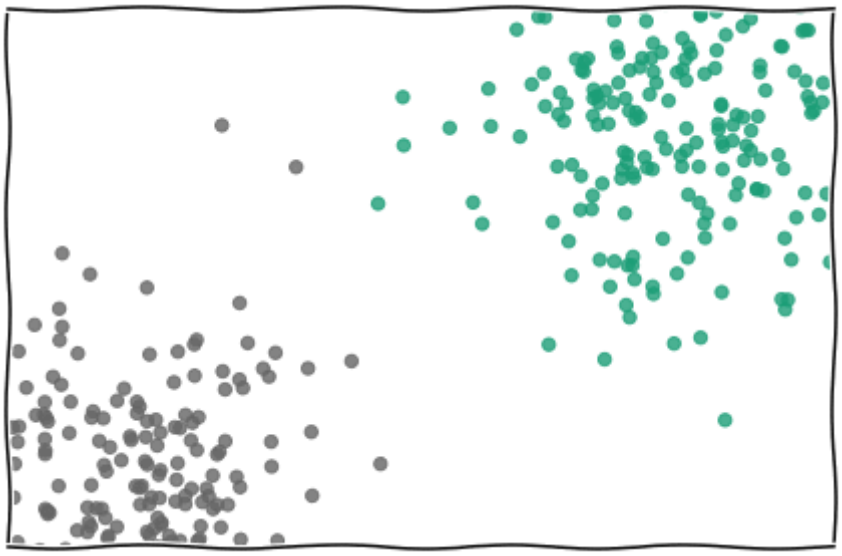


Image source

Disadvantages-:

- It can be influenced by the scale of input features, which means that the distance computed might get skewed depending on the units of each feature. Hence, it's **essential to normalize the input feature**.
- Not effective/useful for high dimensional data

Function to calculate Euclidean Distance in python:

```
from math import sqrtdef euclidean_distance(a, b):
    return sqrt(sum((e1-e2)**2 for e1, e2 in zip(a,b)))#ORfrom
scipy.spatial.distance import euclidean
dist = euclidean(row1, row2)
print(dist)
```

Manhattan Distance

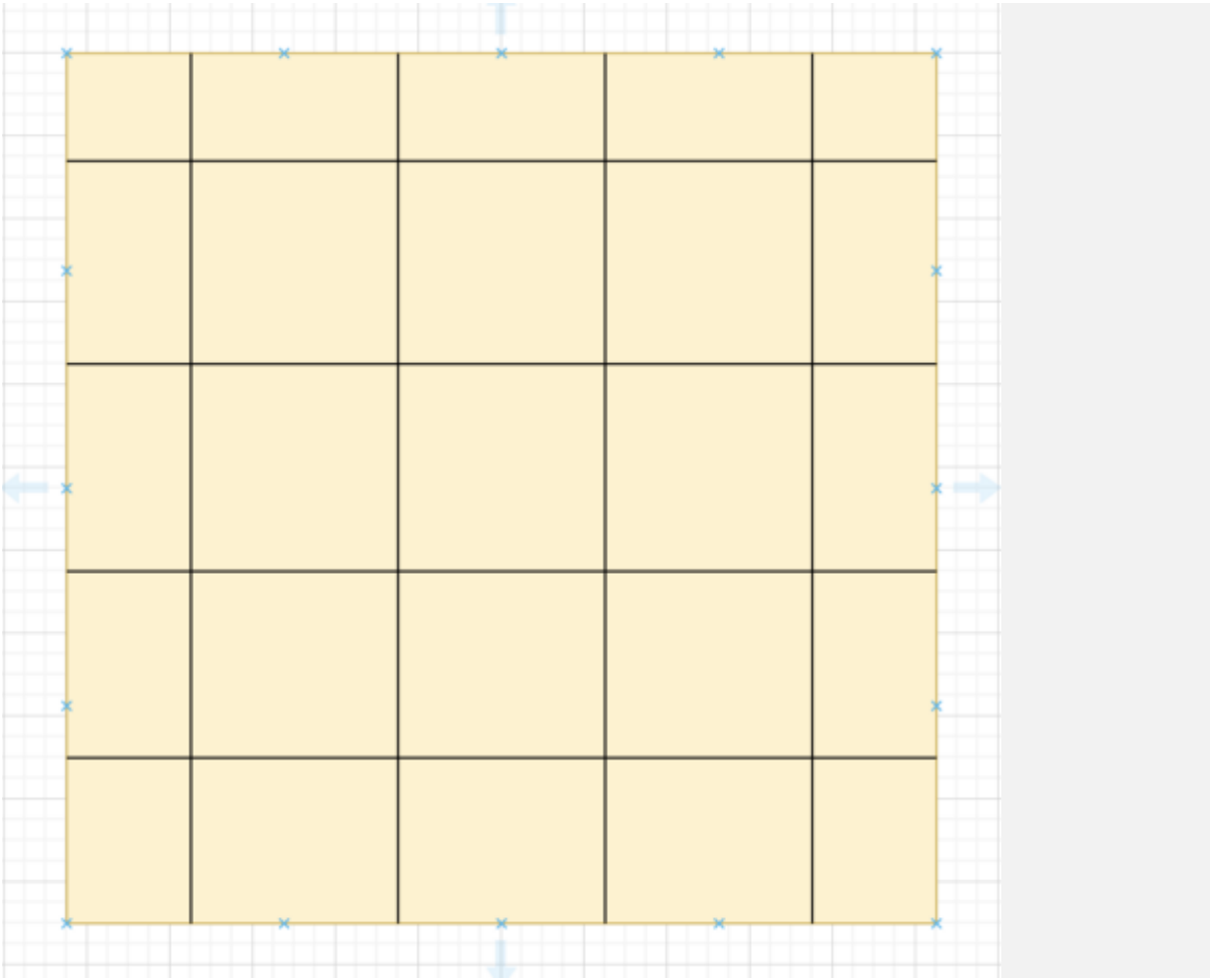


Image By Author

It is often referred to as Taxicab distance or City Block distance. Imagine you are in a city block with two sets of parallel roads, horizontal and vertical, as shown. So, if you want to travel from one point to another, you can only move at right angles.

The distance that is then calculated between two points represents the Manhattan distance.

$$D(x, y) = \sum_{i=1}^{k} |x_i - y_i|$$

Image by Author

Application/Pros-:

- When discrete/binary attributes are present in the dataset, the manhattan distance metric is more effective as it accounts for the path that can be realistically taken from the given values of attributes.

Disadvantages -:

- It does not represent optimal distance in the case of floating attributes in our dataset.
- For high dimensional data, it works better than Euclidean distance, but it’s still not the best option performance-wise.

Function to calculate Manhattan Distance in python:

```
def manhattan_distance(a, b):
    return sum(abs(e1-e2) for e1, e2 in zip(a,b))#ORfrom
scipy.spatial.distance import cityblock
dist = cityblock(row1, row2)
print(dist)
```

Chebyshev Distance

Chebyshev distance is defined as the maximum difference between two vectors among all coordinate dimensions. In other words, it is simply the maximum distance along each axis

$$D(x, y) = \max_i \left(|x_i - y_i|\right)$$

Image By Author

Application/Pros-:

- This metric is usually used for logistical problems. For example, to calculate minimum steps required for a vehicle to go from one place to another, given that the vehicle moves in a grid and thus has only eight possible directions (top, top-right, right, right-down, down, down-left, left, left-top)

Disadvantages-:

- It can only be used for particular problems. It can’t be applied for any general-purpose problem like euclidean can.

Function to calculate Chebyshev Distance in python:

```
def chebyshev_distance(a, b):
    return max(abs(e1-e2) for e1, e2 in zip(a,b))#ORfrom
scipy.spatial.distance import chebyshev
dist = chebyshev(row1, row2)
print(dist)
```

Minkowski Distance

Minkowski generalizes all the above-discussed distance metrics like Euclidean, Manhattan, and Chebyshev. It is also called p-norm vector as it adds a parameter called the “p” that allows different distance measures to be calculated. The formula is -:

$$D(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Image by Author

For p=1 — Manhattan Distance

For p=2 — Euclidean Distance

For p=infinity — Chebyshev Distance

Since this is a more generic distance metric, different values of p will cause different pros and cons. We have already discussed some of them above.

Function to calculate Minkowski Distance in python:

```
def minkowski_distance(a, b, p):
    return sum(abs(e1-e2)**p for e1, e2 in
zip(a,b))**(1/p)#ORfrom scipy.spatial import minkowski_distance
dist = minkowski_distance(row1, row2)
print(dist)
```

Hamming Distance

It is the simplest one so far. It is equal to the number of values that are different between two data points. So let’s say the two data points x and y are as follows-:

x = [1,2,3,0,2,4]

y = [1,3,3,0,1,4]

Then hamming distance = 2 as for index (assuming indices start from 0) 1 and 4 the values are different in x and y. It is usually used for calculating the distance between two binary strings.

Advantages/Pros-:

- It is generally used in error detection when data is sent from one computer to another computer.

Disadvantages-:

- Since it’s used to find the difference between two data points by comparing each value, it is mandatory that they have equal length.
- It does not incorporate the actual value/magnitude of the data point. Hence it’s not advised to use when the magnitude of feature plays an important role.

Function to calculate Hamming Distance in python:

```
def hamming_distance(a, b):
    return sum(abs(e1 - e2) for e1, e2 in zip(a, b)) /
len(a)#ORfrom scipy.spatial.distance import hamming
dist = hamming(row1, row2)
print(dist)
```

Cosine Similarity

It is also one of the most commonly used distance metrics. It is used to find the similarity between two vectors/data points by calculating the cosine angle between them.

$$D(x, y) = cos(\theta) = \frac{x \cdot y}{\|x\| \ \|y\|}$$

Image By Author

For more understanding, if two vectors coincide or have the same orientation, then cosine similarity between them would be 1, and if they are exactly opposite, it would be -1.

Advantages/Pros-:

- Works excellent with high-dimensional data and should be used for them ideally.
- It is used to find the similarity between many things like text embedding, picture embedding, or fingerprint embedding.

Disadvantages-:

- Similar to the hamming distance, the magnitude of the vector is not taken into account in cosine similarity; only their direction is considered.

Function to calculate Cosine Similarity in python:

```
from numpy import dot
from numpy.linalg import normdef cosine_similarity(a,b):
    return dot(a, b)/(norm(a)*norm(b))#ORfrom
scipy.spatial.distance import cosine
dist = 1 - cosine(row1, row2)
print(dist)
```

Jaccard Similarity

Jaccard similarity is used to calculate similarity, and it emphasizes on the similarity between two finite sample sets instead of vectors. It is defined as the size of the intersection of the sets, divided by the size of the union of these sets.

$$D(x,y) = \frac{|x \cap y|}{|x \cup y|}$$

Image By Author

For example, if two sets have two entities in common and have a total of 7 unique entities, then the similarity will be 2/7.

Advantages/Pros-:

- The Jaccard similarity is used as a metric to evaluate image segmentation models in deep learning, where the model has to classify each pixel of the image. It is used to calculate how accurately we have segmented the required entity compared to the ground truth. It can similarly be used in other applications.

Disadvantages-:

- Jaccard similarity is highly dependent on the size of the data. Large datasets can significantly impact the similarity, as in this case, the union could increase substantially while the intersection stays low.

Function to calculate Jaccard Similarity in python:

```
import numpy as npdef jaccard_similarity(x,y):
    intersection = np.logical_and(x, y)
    union = np.logical_or(x, y)
    similarity = intersection.sum() / float(union.sum())
    return similarity#ORfrom sklearn.metrics import
jaccard_scoresimilarity = jaccard_score(a, b)
print(similarity)
```

Sørensen-Dice Index

The Sørensen-Dice index is very similar to Jaccard similarity. The Sørensen-Dice index is a bit more intuitive because it can be seen as the percentage of overlap between two sets, which is a value between 0 and 1:

$$D(x, y) = \frac{2\,|x \cap y|}{|x| + |y|}$$

The Sørensen-Dice index is also used as a metric for image segmentation.

Conclusion

In this blog, we covered various distance metrics that are widely used in practice. Each metric is advantageous in some situations and disadvantageous in others. Hopefully, this blog will help you decide which metric would better fit your use case.