Report for Lab exercise 5 for Computer Vision

- Distance Function and Gaussian Function

```
# I found that computing sqrt of sqare sum is faster than using norm function.
def distance(x, X):
    return torch.sqrt((X - x)[:, 0]**2 + (X - x)[:, 1]**2 + (X - x)[:, 2]**2)
# For vectorization version, weight can be calculated by F(exp(dist)), it would
return a vector too.
def gaussian(dist, bandwidth):
    weight = 1/bandwidth/np.sqrt(2) * torch.exp(-dist ** 2 / 2 / (bandwidth**2))
    return weight/torch.sum(weight)
```

- Normalization term in gaussian

First time I did not add the normalization term in gaussian function:

Then I found that when updating points, the value of X becomes larger and larger, achieving:

```
X = {Tensor: (3675, 3)} tensor([[ 6.3774e+09, -2.7340e+09,  3.7408e+09],\n       [ 4.9379e+02, -2.1302e+02,  2.9430e+02],\n       [ 1
```

Finally it leads to an error because the values now surpass the boundary and will not converge a steady state

- Update_point Function

```
    x_update = torch.mm(weight.double(), X.double())
```

- Acceleration

Using for loop in gaussian (vectorization code is above)

```
    for i in range(len(dist)):
        weight[i] = 1/bandwidth/np.sqrt(2) * torch.exp(-dist[i] ** 2 / 2 /
(bandwidth**2))
```

This is so damn slow whatever using gpu or cpu, and my computer runs forever and doesn't get a outcome in hours.

Using Vectorization with cpu

```
Elapsed time for mean-shift: 13.434001445770264
```

Using Vectorization with gpu

```
Elapsed time for mean-shift: 16.866063594818115

In[3]:
```

In conclusion, it's easy to see vectorized-based mean-shift is way faster than for-loop-based mean-shift.

And it's weird that using gpu is slower than using cpu. It might because that this is a very simple calculation, which means that gpu is as the same as cpu calculation. Gpu is a little slower just because the extra copying the data from cpu to cuda.

- Result