<!-- 方法 1 -->
1.A，B,C,D...车辆已经按离各自的距离排列，即 B 离 A 最近，C 离 A 第二近....(为的是减少传感器测量距离增加而带来的误差)

从传感器获得车辆 AB 间的距离为 dB

从传感器获得车辆 BC 间的距离为 dC

CD 间距离为 dD

...

2. 定义车辆位置信息： car_positions = {'A': [x, y], 'B': [x, y], ...}

3. 定义车辆实际位置信息： car_positionsreal = {'A':[x, y], 'B': [x, y], ...}

4. 将车辆位置信息和车辆实际位置信息中的列表转换为 NumPy 数组：

```
for K in car_positions:
  car_positions[K] = np.array(car_positions[K])
  car_positionsreal[K] = np.array(car_positionsreal[K])
```

5. 定义计算欧几里得距离的函数：

```
def distance(A, B):
  return np.linalg.norm(np.array(A) - np.array(B))
```

6. 定义计算夹角的函数：

```
def angle(A, B):
  x = B[0] - A[0]
  y = B[1] - A[1]
  angle_rad = np.arctan2(y, x)
  angle_deg = np.rad2deg(angle_rad)
  return (90 - angle_deg) % 360
```

7. 定义车辆距离函数： 即为该车辆与前一个（已更新位置）的车辆间的距离

```
def length(K):
  if K in car_positionsreal:
    m = distance(car_positionsreal[chr(ord(K)-1)],
car_positionsreal[K])
  else:
    print('K' not found in car_positionsreal dictionary")
  return m
```

8. 定义预测位置函数：

```
def predict_position(pos, angle, dist):
  x = pos[0] + np.sin(math.radians(angle)) * dist
  y = pos[1] + np.cos(math.radians(angle)) * dist
  return x, y
```

9. 更新车辆位置信息：

记初始预测坐标 A,B（神经网络获得）间的欧式距离为 m1

记 err1=abs(m1-dB)

```
for car in car_positions:
  if car not in ['B','A']:
```

遍历其它车辆（除了 A、B 车辆），对于每辆车通过计算与 A 车辆的距离和角度信息，预测 A 车辆的位置。

```
      dist = distance(car_positionsreal['A'], car_positionsreal[car])
      angle = angle(car_positionsreal[car],car_positionsreal['A'])
      # 计算预测的位置,用车辆观察 A
      x, y = predict_position(car_positions[car], angle, dist)
      PRE_A = [x, y]
      # 计算预测位置与 B 车辆之间的欧氏距离
      m = distance(PRE_A, car_positions['B'])
      err 为预测的 AB 距离与实际传感器测量的欧氏距离之差的绝对值
      err=abs(m-dB)
      # 如果比之前的最小距离小，则更新最小距离和对应的 A 点位置
      if err < err1:
        err1 = err
        min_a_pos = PRE_A
  car_positions['A'] = min_a_pos # 最终确定 K 车辆的位置为更新后的最小误差对应的 A 点位置。
```

10. 通过观察车辆预测其它车辆的位置：
对于 K 在车辆位置中的每一个元素：
    如果 K 不是'A'
        对于 car 在车辆位置中的每一个元素：
            如果 car 不在['K']中
                计算 m1 为从 car_positions[K]到 car_positions[chr(ord(K)-1)]的距离（神经网络获得的坐标），即 K 车到离他最近的前一辆车的距离
                计算 dK 为车 K 和 K-1 之间的真实距离（传感器获得）
                计算 err 为 m1 和 dK 之间的差值的绝对值
                计算 dist 为从 car_positionsreal[K]到 car_positionsreal[car]的距离
                计算 angle1 为从 car_positionsreal[car]到 car_positionsreal[K]的角度
                计算 PRE_K 为基于 car 和 dist，angle1 预测的 K 点位置
                计算 m3 为从 PRE_K 到 car_positions[chr(ord(K)-1)]的距离（预测的 K 点坐标到已经更新过位置的（K-1）车的距离
                计算 err1 为 m3 和 dK 之间的差值的绝对值
                如果 err1 < err，则更新 min_a_pos 为 PRE_K，更新 car_positions[K]为 min_a_pos
        输出改进后的 car_positions[K]
11. 输出车辆位置信息：

```
  for K in car_positions:
    print(K, car_positions[K])
```

效果：对位置改进更好，即此时各车辆的位置更接近实际位置

<方法 2>
输入：

初始基准位置信息（神经网络获得）
初始传感器位置信息
测量距离
输出：

更新后的传感器位置信息
定义函数 fA(coords)、fB(coords)、fC(coords) 和 fD(coords) ，这些函数分别用于计算传感器 A、B、C 和 D 的位置误差。

函数 fA 等均为求其他车辆到某车辆的距离减去传感器测量的此距离的和：

即

```
fA=abs(np.sqrt((x-xa)**2+(y-ya)**2) - d12 +np.sqrt((x-xc)**2+(y-yc)**2) - d23 +np.sqrt((x-xd)**2+(y-yd)**2) - d24)
```

在 fA()、fB()、fC() 和 fD() 函数中计算每个车辆的位置误差，并将位置误差最小化。

用 minimize 函数在初始基准位置附近搜索使得 fA 最小的 A 车坐标，其他车辆同理

启动迭代循环，直到车辆的位置收敛或达到预设的最大迭代次数。

在每个迭代中，分别更新车辆 A、B、C 和 D 的位置，并检查是否已经收敛。

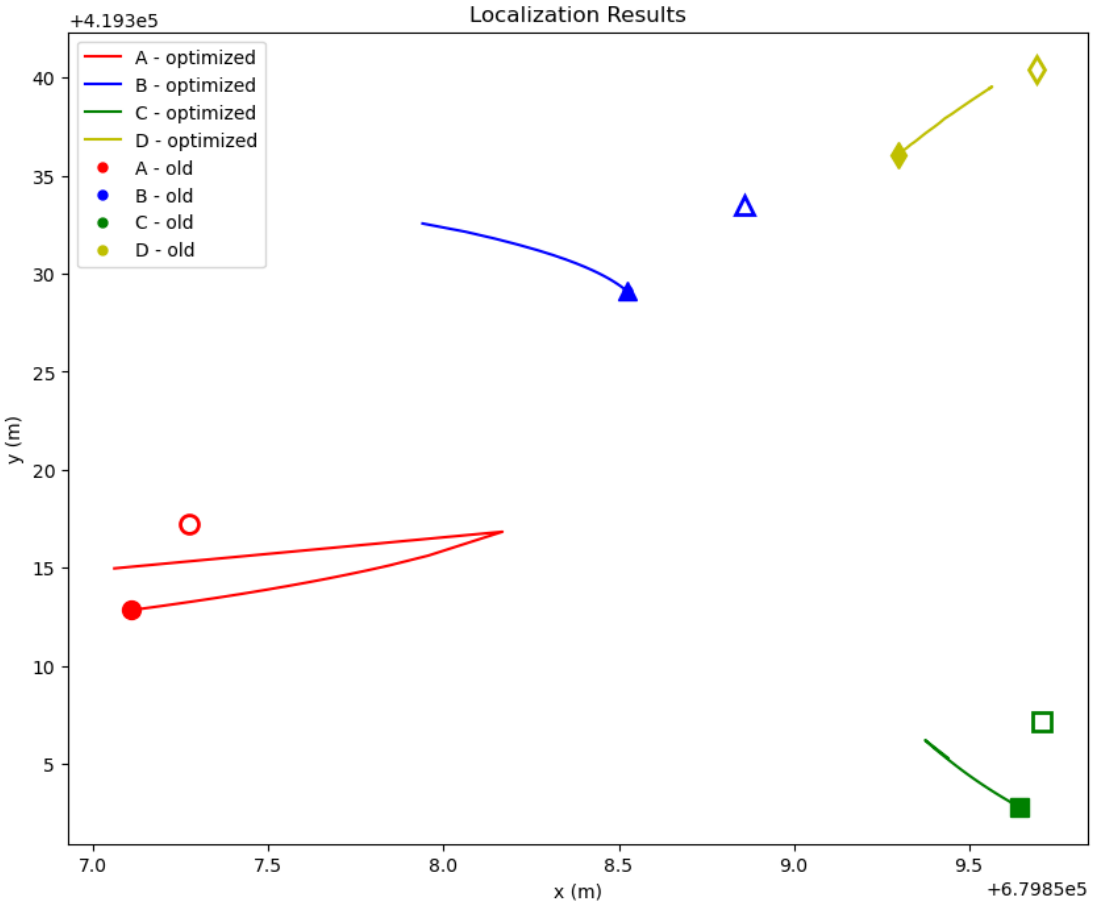
如果车辆位置没有收敛，则继续迭代。

在更新传感器位置后，计算每个车辆更新位置与其真实位置之间的欧氏距离，并输出结果。

如果所有车辆的位置已经收敛，则跳出循环。

输出更新后的车辆位置信息。

效果：1.对车辆间距离拟合的很好，即车辆间此时的彼此距离更加接近传感器测量的距离
2.定位精度不高,原因分析：每次取最小值要去照顾其他 3 个点，为解决此问题已经试过加权处理，效果虽然有改善但仍不如方法 1.



空心的四个点为初始点；实心四个点为迭代结束点