<!-- 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。

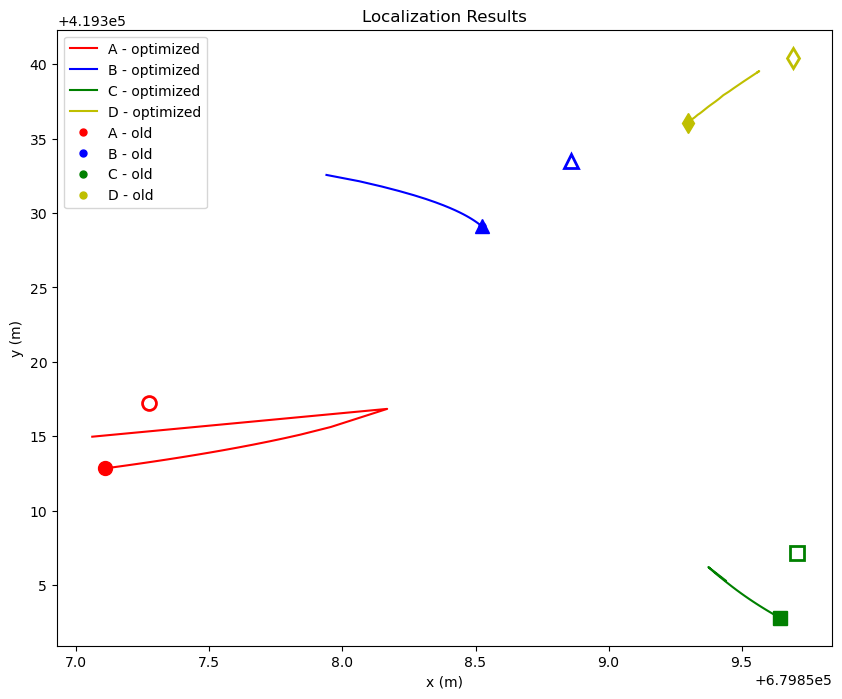
在更新传感器位置后，计算每个车辆更新位置与其真实位置之间的欧氏距离，并输出结果。

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