Описание алгоритма (GetTriplets)

В исходной версии BLE Navigator использовался алгоритм расчета комбинации из трех маячков, расстояния до которых затем применялись для расчета положения пользователя при помощи Трилатерации. К сожалению, исследования показали, что часто данные комбинации меняются, в результате чего маркер положения совершает достаточно большие визуальные скачки по карте.

Исследования также показали, что часто в качестве одного из маячков в триплете выбирается удаленный от текущего положения пользователя маячок. Объясним данную ситуацию на примере. На рис. 1 красной окружностью показано реальное положение пользователя.

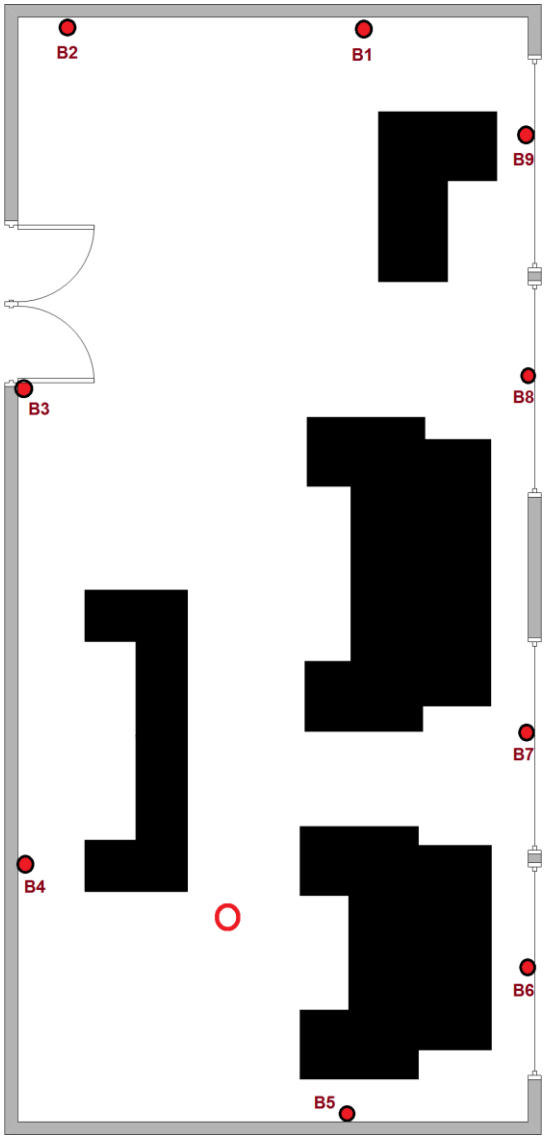

Рис. 1 – Пример

В этом случае разумно, если в триплете будут присутствовать только маячки из ближайшей окрестности вокруг положения пользователя. В случае примера, это маячки №4-7, возможно иногда №3 и №8. В реальности часто в триплет попадают маячки №1 и №9. В этом случае происходит значительное увеличение ошибки позиционирования.

Для того чтобы справится с данной трудностью, придумали несколько доработок:

1. Ввести весовые коэффициенты, пропорциональные расстоянию от маячка с максимальным RSSI до всех других маячков:

$$D(i) = \sqrt{\Delta_X^2 + \Delta_Y^2 + \Delta_Z^2}, \quad (1)$$

где $\Delta_X = beacX(i) - beacX(posMax)$, $\Delta_Y = beacY(i) - beacY(posMax)$, $\Delta_Z = beacZ(i) - beacZ(posMax)$;

$posMax$ – номер маячка по порядку (как в json-файле карты), который на данный момент времени обладает наибольшим RSSI;

i – номера всех остальных маячков;

$beacX(i), beacY(i), beacZ(i)$ – соответственно координаты X, Y и Z i-го маячка.

Далее для каждого маячка вычисляются модифицированные значения RSSI:

$$RSSI_{mod}^{i}(t) = D(i) * RSSI_{filt}^{i}(t), \quad (2)$$

где $RSSI_{filt}^{i}(t)$ – значение RSSI i-го маячка на выходе фильтра Калмана для RSSI.

В идеальном случае значение $RSSI_{mod}^{i}(t)$ должно зависеть как от величины расстояния между маячками, так и от $RSSI_{filt}^{i}(t)$, т.е., если расстояние небольшое, но значение $RSSI_{filt}^{i}(t)$ стремится к -100 дБм (аномальное поведение), то такой маячок должен находиться в конце отсортированной выборки, не должен выбираться. И наоборот.

После запуска данной модификации оказалось, что значения $D(i)$ гораздо больше по величине, чем $RSSI_{filt}^{i}(t)$ для всех маячков. В результате значения $RSSI_{filt}^{i}(t)$ не оказывали практически никакого влияния на величину $RSSI_{mod}^{i}(t)$.

Для устранения данного недостатка выражение (2) модифицировали следующим образом:

$$RSSI^i_{mod}(t) = D(i)^p * RSSI^i_{filt}(t), \quad (3)$$

где p – показатель степени.

Экспериментально выбрали p=0.1 (это значение по умолчанию). Его значение стоит вынести в настройки SDK.

2. Лучшие результаты в позиционировании получаются, когда реальное положение пользователя попадает в треугольник, организованный выбранными тремя маячками. Для ситуации, показанной на рис. 1, это соответствует выбору для определения положения маячков 4, 5 и 7, 4, 6 и 7 или 4, 5 и 6. Однако на практике часто выбираются маячки 6, 7, 8 или 5, 6, 7. В этом случае в оценку положения вносятся большие ошибки, что проверяли визуальным анализом.

Для того чтобы выбирать треугольник маячков более качественно, реализовали следующую идею. После выполнения преобразования 1, описанного выше, по выбранному триплету маячков производится расчет углов треугольника, который ими (маячками) создается - метод "bool **isAnglesCorrect**()".

Условие корректности треугольника – ни один из его углов не должен превышать 110 градусов. Метод **isAnglesCorrect**() возвращает true, если это так.

Если метод **isAnglesCorrect**() вернул false, т.е. один из углов треугольника больше 110 градусов, то из триплета маячков убирается 3й маячок и на его место ставится маячок, который стоит на следующем (в данном случае четвертом) месте в отсортированной выборке значений $RSSI^i_{mod}(t)$. И метод **isAnglesCorrect**() запускается еще раз для обновленной тройки маячков. И так далее, пока не будет найден корректный треугольник, либо не перебраны все возможные подстановки последнего маячка.

Если ни один вариант не дал положительного результата, то дальше передаются три маяка, которые были выбраны в самом начале.

Алгоритм работы метода isAnglesCorrect()

Исходные данные: координаты (x; y) выбранных маяков A, B и C.

Шаг 1: Определяем длины трех векторов: AB, BC и AC как

$$AB = [x_B - x_A \quad y_B - y_A]$$
$$CB = [x_B - x_C \quad y_B - y_C]$$
$$AC = [x_C - x_A \quad y_C - y_A]$$

где точки A, B и C – точки расположения соответственно первого, второго и третьего маяков на карте.

Шаг 2: Определяем угол между векторами AB и AC как

$$< AB\_AC = acos\left(\frac{\overline{AB} \cdot \overline{AC}}{\overline{|AB|} \cdot \overline{|AC|}}\right)$$

и угол между векторами AB и CB:

$$< AB\_CB = acos\left(\frac{\overline{AB} \cdot \overline{CB}}{\overline{|AB|} \cdot \overline{|CB|}}\right)$$

Тогда третий угол можно определить как:

$$< AC\_CB = 180 - < AB\_CB - < AB\_AC$$

Шаг 3: Если хотя бы один из полученных углов больше 110 градусов, то проверка треугольника на корректность считается не пройденной и метод возвращает false. Иначе, true.