**Навигация внутри помещений в мобильных приложениях.**

**Введение**

На сегодняшний день одной из актуальных тем в мобильных разработках является навигация внутри помещений. Это связано с тем, что здания становятся всё более объёмными и нередко имеют довольно сложную структуру, что затрудняет ориентацию в пространстве. Кроме того, решения, применяемые в навигации внутри помещений, помогают и в ориентировании вне зданий, на улице – там, где в условиях плотной застройки использование систем спутниковой навигации затруднено (нет спутников в прямой видимости, присутствует только отражённый/ослабленный/зашумленный сигнал GPS/Глонасc и т.д.).

Навигация внутри помещений отличается от обычной навигации, поскольку:

1. Местоположение внутри помещений необходимо определять быстро и с высокой точностью, чего  не позволяет GPS/ГЛОНАСС навигация.
2. Присутствует такой фактор, как этажность здания.
3. В помещении, как правило, нет дорог, по которым можно было бы прокладывать маршруты.

Таким образом, основными проблемами темы навигации внутри помещений являются определение местоположения с высокой точностью и построение маршрутов на открытой местности. Целью работы стали подбор и реализация алгоритмов для решения этих проблем.

В ходе изучения предметной области был сделан вывод, что поставленная задача на настоящий момент является актуальной и рабочих решений, удовлетворяющих всем требованиям, не существует.

Определению местоположения при помощи Bluetooth (Bluetooth Indoor Positioning) посвящена работа [1] Anja Bekkelien из Женевского Университета, однако ее задача заключалась в разработке подмодуля для Глобального Модуля позиционирования (GPM), в основе которого работала совокупность сенсоров, нам же необходимо реализовать самодостаточную навигационную систему, использующую в своей основе именно iBeacons. iBeacon – технология, представленная Apple в iOS 7, которая расширила возможности библиотеки для работы со службами геолокации в iOS. Вместо геолокации по широте и долготе iBeacon использует низкоэнергетический сигнал Bluetooth, который обнаруживается телефоном. Формат данных строго типизирован и детально описан на официальном сайте Apple для разработчиков.

Построение маршрутов на открытой местности изучала Anita Graser из Австрийского Технологического Института. Её работа [2] посвящена внедрению работы с открытыми пространствами в OpenStreetMap – веб-картографическом проекте по созданию географической карты мира. Однако, описанные в работе алгоритмы не учитывают препятствий на открытых пространствах, а реализация не учитывает специфики мобильных платформ.

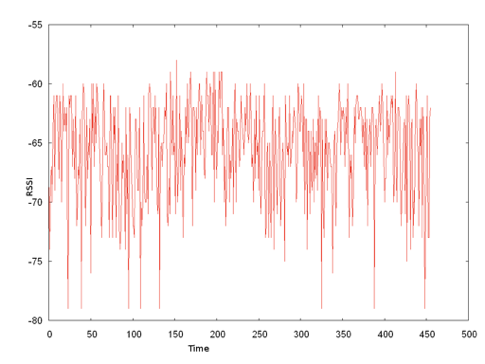
**Bluetooth Beacons**

Для определения местоположения мобильного устройства зачастую используются Bluetooth-маячки (beacons).

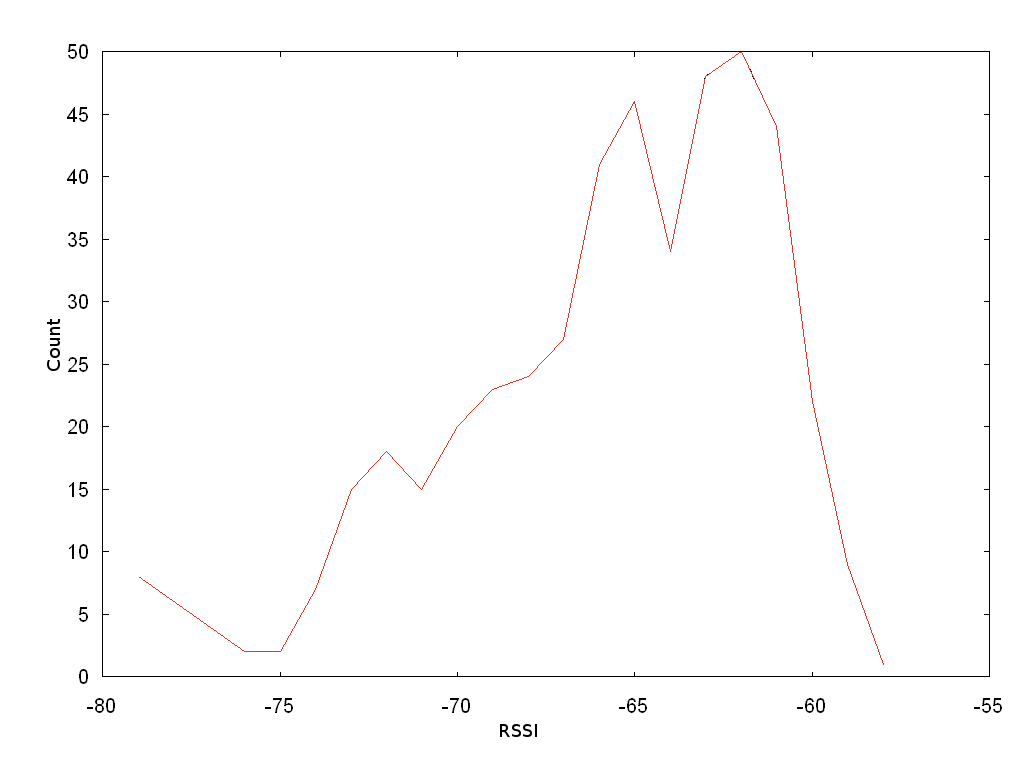
Bluetooth-маячки – это обыкновенные Bluetooth 4.0 LE устройства, то есть, их роль может с успехом выполнять любое устройство, оснащённое BLE-чипом, например, iPhone. Для этого необходимо установить маячки по периметру помещения, а координаты их расположения сохранить в базе данных. Маячки с заданной периодичностью будут производить широковещательную рассылку, содержащую идентифицирующую их информацию. Пользовательское приложение, циклично получая эти данные, сможет с помощью базы данных определить координаты маячков и, на основе силы посылаемого сигнала RSSI (Received Signal Strength Indicator), - своё местоположение [3].

**Алгоритм определения местоположения**

Получаемый RSSI параметр позволяет определить удаленность маячка от приемника сигнала – смартфона. Это происходит при помощи деления текущего RSSI на эталонный, измеренный производителем на расстоянии 1 метр. Значение RSSI обратно пропорционально удаленности смартфона от маячка. Однако, из-за физического эффекта интерференции сигналов, значение RSSI между двумя стационарными объектами непостоянно (графики 1, 2).

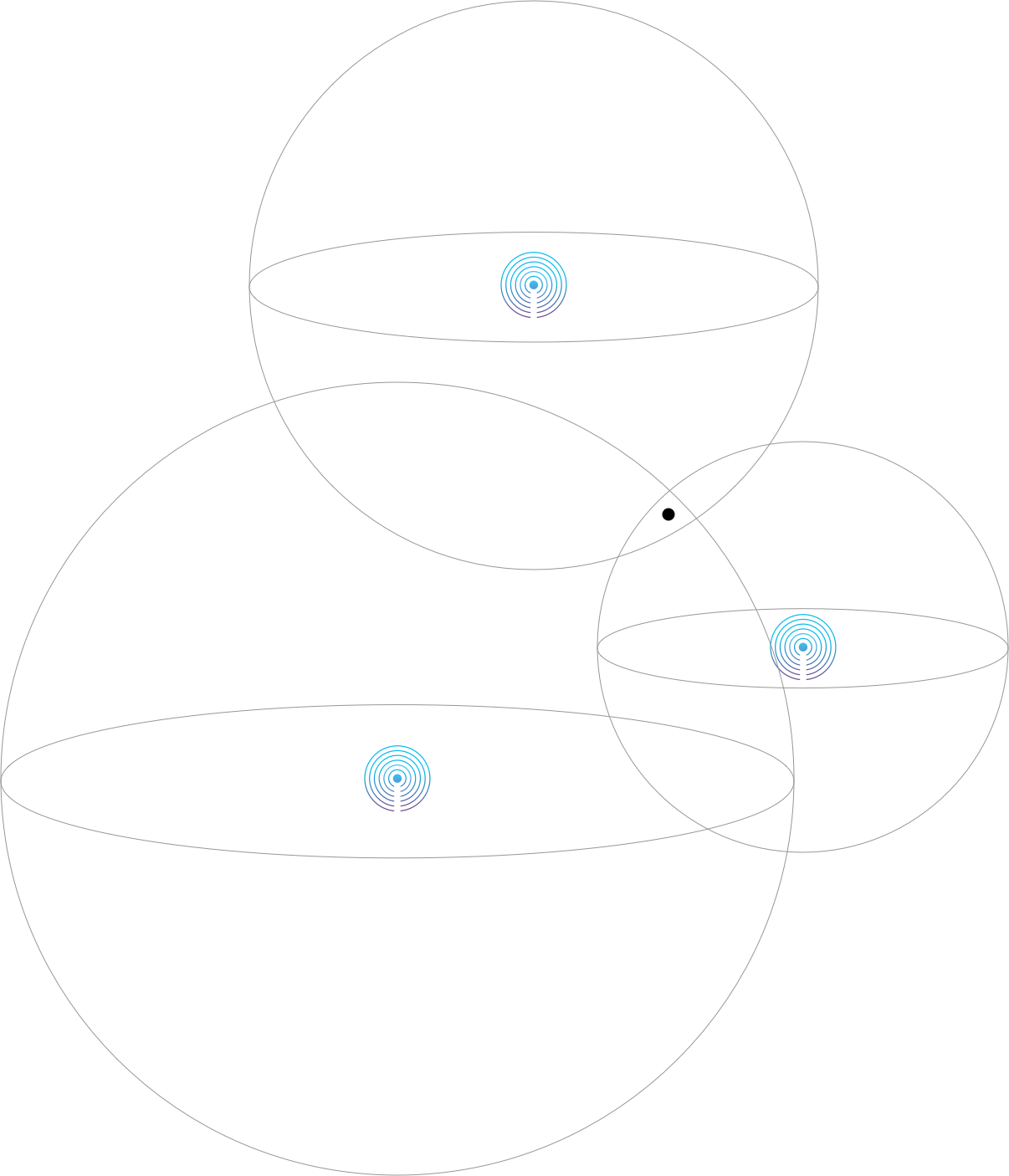


1. Значения RSSI между маячком и устройством на фиксированном расстоянии 3 м.



2. Гистограмма значений графика 1

Как видно из представленных графиков, при использовании одного маячка можно получать разную удаленность. Графически это можно представить как окружность с центром в точке с коорданитами расположения маячка и радиусом, равным вычисленному расстоянию до него. Радиус этой окружности даже при неподвижном состоянии приемника будет меняться с течением времени. Используя информацию от двух маячков, можно получить две окружности с изменяющимися радиусами и, возможно, одной или двумя точками пересечения. Если представить, что маячки дают точную информацию об удаленности, то, используя два маячка, точное местоположение можно рассчитать только находясь на одной линии с ними. При отклонении от этой линии получатся две точки пересечения и две возможные координаты. Однако, поскольку значение RSSI колеблется, точек пересечения может и не быть. Из чего можно сделать вывод, что для однозначного определения местоположения необходимо как минимум три маячка.



3. Алоритм трилатерации, графическое представление

Алгоритм нахождения координат по трем или более точкам и удалению от них называется алгоритмом трилатерации [4]. Задача трехмерной трилатерации решается при помощи нахождения координат пересечения трех сфер, которые определяются при помощи решения системы уравнений. Для начала составим систему уравнений для трех сфер, организованную следующим образом: так как любые 3 точки в пространстве образуют плоскость, поместим начало координат этой плоскости в центр одной из сфер, а центр второй сферы поместим на ось координат Ox.

В точке (x, y, z) будет располагаться приемник.

Таким образом, мы вывели уравнение для координаты приемника, используя информацию, полученную от Bluetooth-передатчиков. Очевидно, координата высоты может принимать два значения. Это обусловлено тем, что передатчики и приемник могут находиться не в одной плоскости. Для нашей задачи мы можем рассматривать только нижнее значение, так как маячки расположены выше пользовательских устройств.

В нашем случае существует “шум” и пересечением окружности может являться область или пересечений может не быть вообще. Поэтому необходимо модифицировать алгоритм, воспользовавшись методом итеративной трилатерации, описанным в [5].

Для этого, обозначим начальные точки и полученное расстояние (истинная координата приемника и расстояние между передатчиком и приемником) как (*xi* , *yi* ) и *di* , соответственно. Полученные при помощи тривиального алгоритма начальные координаты обозначим (*xe*, *ye* ) . Разница или ошибка рассчитывается по формуле:

Теперь, применяя аппроксимацию при помощи ряда Тейлора первой степени, можно найти (∆ *x*,∆ *y*) , используя следующие равенства:

Где B вычисляется следующим образом

В результате получим:

Учитывая это, *fi* может быть рассчитан как:

Таким образом, получаем дробное значение, характеризующее ошибку. Итеративно необходимо повторять данные расчеты до тех пор, пока ошибка не уменьшится до заданной .

Однако, в результате все еще остается “шум”, который проявляется как изменение координат пользователей при отсутствии физического движения. Для фильтрации “шума” мы можем использовать встроенные в устройства сенсоры. Для данной задачи подходят методы на основе рекурсивной оценки Байеса.

В рамках данной работы были рассмотрены и фильтр Калмана [6] и многочастотный фильтр [7]. Фильтр Калмана предназначен для рекурсивного дооценивания вектора состояния априорно известной динамической системы, то есть для расчёта текущего состояния системы необходимо знать текущее измерение, а также предыдущее состояние самого фильтра. Алгоритм работает в два этапа. На этапе прогнозирования фильтр Калмана экстраполирует значения переменных состояния, а также их неопределенности. На втором этапе, по данным измерения (полученного с некоторой погрешностью), результат экстраполяции уточняется. Благодаря пошаговой природе алгоритма, он может в реальном времени отслеживать состояние объекта [8].

Многочастотный фильтр - последовательный метод Монте-Карло — рекурсивный алгоритм для численного решения проблем оценивания (фильтрации, сглаживания), особенно для нелинейных и негауссовских случаев. В отличии от фильтра Калмана многочастичные фильтры не зависят от методов линеаризации или апроксимации.

Экспериментальным путем установлено, что фильтр Калмана обеспечивает лучшее сглаживание.

**Построение маршрутов на открытой местности**

Перейдем к вопросу построения маршрутов. Поскольку здание обычно представляется не набором коридоров/лабиринтов, а некоторым количеством помещений, то, как было сказано ранее, изначально отсутствует заданное множество дорог. Решение данной задачи можно разбить на два этапа:

1. Представив помещение в виде полигона, произвести его триангуляцию.

2. Приняв вершины получившихся треугольников за вершины графа, а их стороны — за его рёбра, можно определить ближайшие к стартовой и конечной точкам вершины и найти кратчайший путь от одной до другой.

По определению, простой полигон — упорядоченная последовательность точек *V0, ..., Vn*. Соседние вершины соединены ребром *<Vi, Vi+1>*, где *0 ≤ i ≤ n-2*. Ребро *<Vn-1, V0>* соединяет первую и последнюю вершину. Каждая вершина принадлежит ровно двум рёбрам. Единственное место пересечения любых двух рёбер — вершина полигона.

Если полигон простой, то во время прохода по ребрам полигона, его внутренняя область всегда находится с одной и той же стороны. Скажем, что полигон направлен против часовой стрелке, если при проходе вдоль ребер, внутренняя область находится слева.

*Триангуляция* — разбиение геометрического объекта на симплексы, n-мерные обобщения треугольника. В нашем случае объект находится на плоскости, соответственно, полигон разбивается на треугольники.

Для триангуляции существует множество алгоритмов. Рассмотрим один из них (полностью описан в [9]), называемый ear clipping (отсекание ушей, ушная триангуляция). Сложность данного алгоритма O(n2).

«Ухом» полигона назовем треугольник, образованный тремя последовательными вершинами *Vi*0, *Vi*1, *Vi*2, где *Vi*1 является выпуклой вершиной (внутренний угол вершины меньше π), отрезок *<Vi*0*, Vi*2*>* полностью лежит внутри полигона, при этом ни одна вершина полигона, кроме *V* *i0*, *Vi1, Vi2* не принадлежит треугольнику. Отрезок *<Vi*0*, Vi*2*>* называется диагональю полигона. Вершина *Vi*1 называется кончиком уха. Треугольник (полигон, для которого *n = 3*) состоит из одного уха, однако кончиком уха можно считать любую из трех вершин. Полигон с четырьмя и более сторонами всегда имеет как минимум два непересекающихся уха.

Предполагается рекурсивный подход к решению данной проблемы. Если возможно найти «ухо» в полигоне с *n ≥ 4* вершинами и «отсечь» его, то останется полигон из *n - 1* вершин и можно повторить процесс. Наивная реализация такого алгоритма приведет к сложности O(n3).

Однако, алгоритм можно реализовать со сложностью O(n2). Первый шаг заключается в том, чтобы хранить полигон в двусвязном кольцевом списке, чтобы иметь возможность быстрого удаления кончиков ушей. Создание такого списка — алгоритм со сложностью O(n). Вторым шагом является проход по всем вершинам и поиск ушей. Для каждой выпуклой вершины *Vi* возьмем треугольник *<Vi-1, Vi, Vi+1>* и проверим, существует среди остальных вершин полигона такая, которая лежит в этом треугольнике. Если ни одна из них не лежит внутри треугольника, треугольник будет являться «ухом».

Ещё одним шагом оптимизации рассматриваемого алгоритма может быть использование следующего факта. Очевидно, что если в проверяемый треугольник попадает какая-либо другая вершина многоугольника, то существует невыпуклая вершина, находящаяся внутри этого треугольника. Таким образом, проверять на нахождение внутри треугольника нужно только те вершины, углы которых являются невыпуклыми (внутренний по отношению к полигону угол больше 180 градусов).

Структура данных для полигона состоит из четырех двусвязных списков:

* вершины полигона хранятся в циклическом списке,
* выпуклые вершины хранятся в линейном списке,
* невыпуклые вершины хранятся в линейном списке,
* кончики ушей хранятся в циклическом списке.

Создав первые три списка, можно начать удалять уши по одному. Если *Vi* — удаляемая из полигона вершина, описание углов вершин *Vi-1* и *Vi+1* может измениться следующим образом [9]:

* Если один из этих углов выпуклый, то он останется выпуклым.
* Если один из углов *Vj* принадлежал другому «уху», то после удаления «уха» *Vi* «ухо» *Vj* может исчезнуть.
* Если один из углов невыпуклый, он может стать выпуклым, а также может создать новое ухо.

Таким образом, после удаления уха *Vi*, если соседняя вершина выпуклая, необходимо проверить, не образует ли она новое ухо, обойдя невыпуклые вершины и проверяя принадлежность этих вершин треугольнику уха. Всего есть O(n) ушей. Каждое обновление соседних вершин влечет проверку, процесс который занимает O(n) операций. Таким образом, весь процесс представляет собой алгоритм сложностью O(n2).

Ушной алгоритм также может быть применен к полигонам с дырками. Дыркой называется полигон, полностью лежащий внутри триангулируемого полигона. В таком случае перед применением алгоритма триангуляции необходимо соединить внешний полигон и все его дырки в одну последовательность точек.

Представим полигон с одной дыркой. Он состоит из внешнего и внутреннего полигонов. Порядок вершин во внешнем и внутреннем полигонах должны быть противоположными. Задача заключается в том, чтобы найти во внешнем и внутреннем полигонах вершины, между которыми можно построить отрезок, не пересекающийся ни с одним другим отрезком полигонов. В [9] описывается один из способов нахождения таких вершин:

1. Найти точку *M* внутреннего полигона с максимальным значением по оси *x*.
2. Найти самую ближнюю к точке *M* точку пересечения *I* луча *M + t(1,0)* со сторонами внешнего  полигона.
3. Если *I* — вершина внешнего полигона, искомая точка найдена.
4. Иначе *I* — точка отрезка *<Vi, Vi+1>*. Выбрать точку *P* — конец отрезка *<Vi, Vi+1>* с большим  значением по оси x.
5. Проверить невыпуклые вершины внешнего полигона (кроме *P*) на содержание внутри  треугольника *<M, I, P>*. Если ни одна из них не лежит внутри треугольника, то точка *P* является искомой.
6. Иначе хотя бы одна невыпуклая вершина лежит внутри треугольника. Среди таких вершин  выбрать такую вершину *R*, что угол *IMR* минимален. Если таких точек несколько, то все они лежат на одном луче, выходящем из *M*, и необходимо выбрать ближайшую к *M* точку. Точка *R* является искомой.

Вершины *M* и *P* необходимо дублировать и соединить полигоны в один двумя отрезками. Полученный полигон можно триангулировать, как обычный полигон без дырок.

Полигон может иметь несколько «дырок». В таком случае нужно применить описанный выше алгоритм к каждой из них последовательно, выбирая из списка оставшихся такую, которая содержит вершину с максимальным значением *x*.

Было замечено, что при малом количестве углов полигона, после триангуляции может получиться достаточно разреженный граф, что не позволит построить оптимальный маршрут через его вершины. Поэтому было решено произвести уплотнение полигональной сетки. Это можно сделать при помощи *тесселяции*.

Тесселяция — процесс разбиения полигональной сетки на более мелкие многоугольники. В результате тесселяции граф получается более плотным, и кратчайшие пути станут гораздо более естественными для движения пешехода.

Итак, поскольку необходимо произвести триангуляцию, данные карты должны представлять набор полигонов, где координаты углов внешних стен помещения должны быть перечислены по часовой стрелке, а координаты углов внутренних стен и других преград («дырок» в полигоне) — против часовой, или наоборот.

Поскольку помещение может представлять невыпуклый многоугольник, а внутри его могут быть дополнительные преграды, необходим такой алгоритм триангуляции, который̆ бы учитывал эти два момента.

На выходе алгоритма триангуляции и, если нужно, тесселяции должен получиться набор вершин треугольников, который затем необходимо преобразовать в неориентированный взвешенный граф: вершины треугольников становятся вершинами графа, а стороны — его рёбрами, причем длина стороны становится весом соответствующего ребра.

Для построения маршрута достаточно использовать один из алгоритмов поиска кратчайшего пути, например алгоритм Дейкстры.

В результате описанных выше операций получим маршрут, который представляется на карте ломанной линией. В качестве дальнейшего улучшения пользовательского опыта можно рассмотреть сглаживание полученного маршрута, например, при помощи кубических сплайнов.

**Заключение**

В результате нашего исследования был создан работоспособный прототип приложения, который удовлетворяет требованиям к производительности, точности определения местоположения, а так же умеет строить оптимальные маршруты внутри помещений. Исходя из проведенных экспериментов, были получены следующие результаты:

1. Описанные алгоритмы обладают достаточной производительностью, для того чтобы производить все расчеты в реальном времени, учитывая ограничения мобильных устройств, на которых прототип тестировался.
2. Придерживаясь данных техник погрешность вычисления координат составляет 15 метров, для уменьшения погрешности необходимо учитывать в первую очередь именно внутренние датчики.
3. Погрешность вычисления увеличивается при движении и поворотах, из-за чего, используя текущую реализацию, невозможно избежать небольших подергиваний точки, обозначающей местоположение пользователя на карте при движения такого типа.
4. Удается построить достаточно правдоподобный, оптимальный маршрут в помещении, не имеющем определенных дорог.

Литература.

1. Anja Bekkelien, Bluetooth Indoor Positioning, University of Geneva, pages 14-17, March 2012
2. Graser Anita, Integrating Open Spaces into OpenStreetMap Routing Graphs for Realistic Crossing Behaviour in Pedestrian Navigation. AIT Austrian Institute of Technology, 2016.
3. Bing-Fei Wu, Cheng-Lung Jen, and Kuei-Chung Chang. Neural fuzzy based indoor localization by Kalman filtering with propagation channel modeling. In IEEE International Conference on Systems, Man and Cybernetics, 2007 , pages 812-817, Oct. 2007.
4. Erin-Ee-Lin Lau and Wan-Young Chung. Enhanced RSSI-based real-time user location tracking system for indoor and outdoor environ- ments.In ICCIT 07: Proceedings of the 2007 International Conference on Convergence Information Technology , pages 1213-1218, Washing- ton, DC, USA, 2007. IEEE Computer Society.
5. Sebastian Thrun, Wolfram Burgard and Dieter Fox, Probabilistic Robotics, September 2005
6. Zhe Chen, Bayesian filtering: From Kalman filters to particle filters, and beyond, Technical report, McMaster University, 2003.
7. Ioannis M. Rekleitis, A particle filter tutorial for mobile robot localiza- tion, Technical Report TR- CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Quebec, Canada H3A 2A7, 2004.
8. J. Carpenter, P. Clifford, and P. Fernhead, An improved particle filter for non-linear problems,tech. rep., Department of Statistics, University of Oxford, 1997.
9. David Eberly, Triangulation by Ear Clipping, Geometric Tools, LLC, 2015.