МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра Математического обеспечения ЭВМ

**ПОЗИЦИОНИРОВАНИЕ МОБИЛЬНЫХ УСТРОЙСТВ ВНУТРИ ПОМЕЩЕНИЙ**

Магистерская диссертация

по направлению 02.04.02 Фундаментальная информатика и

информационные технологии

магистерская программа «Программирование для мобильных устройств»

Допущен к защите в ГЭК \_\_\_\_\_\_\_

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_ д.ф.-м.н. доц. Махортов С.Д.

(подпись)

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_ Зонов А. В.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_ к.ф.-м.н. доц. Болотова С.Ю.

(подпись)

Воронеж 2017

СОДЕРЖАНИЕ

Введение 4

Постановка задачи 5

Глава 1. Теоретические основы определения местоположения пользователя 6

1.1. Основы работы существующих сервисов на примере функционирования GLONASS. 6

1.1.1. Рассмотренные технологии позиционирования внутри помещений 7

1.1.1.1. Навигация, используя Wi-Fi 7

1.1.1.2. Геомагнитное позиционирование 9

1.1.1.3. Ориентирование по базовым станциям операторов сотовой связи 9

1.1.1.3. Технология iBeacon 11

1.2. Форматы передачи данных 13

1.2.1. iBeacon 13

1.2.2. Bluetooth 15

1.3. Алгоритмы определения местоположения 16

1.3.1. Введение 16

1.3.2. Алгоритм трилатерации 16

1.3.3. Практическая реализация алгоритма трилатерации 19

1.3.4. Алгоритм итеративной трилатерации 22

1.3.5. Практическая реализация алгоритма итеративной трилатерации 24

1.3.5. Методы на основе оценки Байеса 26

Глава 2. Практическая реализация мобильного приложения 30

2.1. Выбор платформы и языка программирования 30

2.2. Организация взаимодействия с iBeacon маячками 33

2.3. Организация взаимодействия со встроенными сенсорами 35

2.4. Способы организации работы с CoreData 38

2.4.1. NSFetchedResultsController 40

2.5. Схема базы данных 41

2.6. Обеспечение безопасности пользовательских данных 42

2.7. Дополнительные инструменты 44

2.7.1. Система управления версиями файлов 44

2.7.2. Система отслеживания ошибок 44

2.7.3. Система сбора статистики 45

Заключение 47

Список использованных источников 48

Приложение 1. Экраны мобильного приложения 49

Приложение 2. Реализация взаимодействия с iBeacon 50

Приложение 3. Реализация взаимодействия с сенсорами 54

ВВЕДЕНИЕ

Задача создания мобильного приложения, позволяющего позиционировать девайс пользователя внутри помещений, актуальна. Существует множество примеров крупных зданий со сложной внутренней структурой, таких как аэропорты, торговые центры, университеты. В постройках такого типа ориентироваться могут лишь те, кто постоянно посещает их, а для человека, попавшего туда впервые, ориентирование в таких местах превращается в пытку. Кроме того, традиционные системы точного геопозиционирования не работают в таких ситуациях, где плотность застройки не позволяет использовать GPS спутники, так как при высотной плотной застройке GPS сигнал доходит до девайсов отраженный, ослабленный или зашумленный.

Актуальность данной задачи в том, что в современном мире люди привыкли использовать навигаторы для построения маршрутов, но как только человек попадает из открытой местности в здание, например, в аэропорт, ведение по маршруту заканчивается. Данным вопросом занимаются все крупные информационные корпорации, такие как Apple, Google, Yandex, 2Gis и другие. Практически все современные картографические сервисы предлагают поэтажные карты крупных торговых центров и аэропортов, но определить точное местоположение пользователя они не могут.

В результате проведенного исследования не было найдено библиотек или общепринятых практик для позиционирования внутри помещений в открытом доступе, хотя востребованность их велика, так как это большая маркетинговая ниша. В процессе исследования этого вопроса начали появляться первые приложения для конкретных выставок, которые предлагали пользователям скачать приложение со схематичным расположением павильонов, но реализация данных решений была закрытой и являлась строгоориентированной на определенное мероприятие.

ПОСТАНОВКА ЗАДАЧИ

Основная задача данной работы – поиск и сравнение различных технологий определения местоположения внутри помещений, знакомство с основами разработки приложений под управлением операционной системы iOS, изучение и сравнение различных алгоритмов по определению местоположения. Также ставится задача разработки приложения, позволяющего с помощью iPhone определить этаж, на котором находится утройство, а также местоположение устройства на этаже с точностью большей, чем позволяют существующие решения, использующие GPS и GPRS позиционирование внутри помещений. Приложение должно обладать следующими возможностями:

* + - * + Определение, в каком здании, на каком этаже и в каком месте находится устройство.
        + Получение подробной схемы здания.
        + Отображение карты этажа, на котором находится устройство.
        + Отображение текущего местоположения устройства на карте.
        + Обновление изменения местоположения в реальном времени и обновление карты.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ОПРЕДЕЛЕНИЯ МЕСТОПОЛОЖЕНИЯ ПОЛЬЗОВАТЕЛЯ.

**1.1. Основы работы существующих сервисов на примере функционирования GLONASS.**

GLONASS - глобальная навигационная спутниковая система, одна из двух функционирующих сегодня систем глобальной спутниковой навигации. Она состоит из 24 спутников, которые движутся в трех орбитальных плоскостях. Происходит это примерно на высоте 40 000 км. Все 24 спутника транслируют на землю несколько видов радиосигналов, а также общаются между собой и с наземной службой, за счет чего всегда знают свое местоположение. Сигнал, отправляемый каждым спутником, имеет строготипизированный формат, который содержит информацию о координатах самого спутника, а также время отправки сигнала. Как только мобильное устройство получает сигнал, оно его расшифровывает, получает координаты и время отправки, после чего, используя константную величину скорости сигнала, вычисляется расстояние. Но для определения местоположения нужно больше спутников, так как в спутниковом позиционировании используется трилатерация. Несколько спутников отправляют сигнал, результирующее расстояние расчитывается до каждого спутника. Если отложить отрезок от каждого спутника до точки совместного соприкосновения, то получим координаты устройства, принявшего сигнал. Этот принцип позиционирования считается достаточно точным, но тем не менее имеет погрешность, которая может достигать на открытой местности 13 метров. В помещениях же погрешность настолько велика, что использование глобальных систем позиционирования оказывается невозможным.

***1.1.1. Рассмотренные технологии позиционирования внутри помещений***

Как было рассмотрено выше, глобальная навигационная система не позволяет добиться достаточной точности в рамках решения задачи определения местоположения внутри помещений. Так как в постановке задачи было определено, что приложение должно работать на смартфонах под управлением опрерационной системы iOS, это наложило ограничения, но также предоставило широкий набор датчиков и сенсоров, которые можно использовать для решения задачи.

*1.1.1.1. Навигация, используя Wi-Fi*

Wi-Fi (Wireless Fidelity) – беспроводные сети на базе стандарта IEEE 802.11. На данный момент уже существуют системы позиционирования с использованием Wi-Fi, обычно это вспомогательные системы для GPS, так как для точного позиционирования по Wi-Fi необходимо достаточно много точек в зоне видимости без препятствий между трансмитером и приемником. Такие свойства позиционирования обусловлены способом определения местоположения, так как используется RSSI (received signal strenght indicator) и метод «Fingerprinting». Для определения местоположения необходимо иметь базу данных соответствия отпечатков каждой Wi-Fi точки и ее координаты. На основе этих данных, используя силу сигнала, можно определить расстояние до устройства, получающего сигнал, и по аналогии с GPS, используя трилатерацию и, как минимум, 3 Wi-Fi точки, определить координаты устройства. При реализации данного типа навигации на базе существующей инфраструктуры былы выявлены следующие проблемы:

1. Недостаточное пересечение областей покрытия Wi-Fi - очень малое количество мест общественного пользования оснащены достаточным количеством Wi-Fi роутеров. Даже если покрытие аэропорта достигало 100%, в каждом конкретном месте было, в основном, 1-2 видимых точки, что не позволяет однозначно определить метоположение пользователя (пересечение двух окружностей может дать две точки пересечения).
2. Недействительное соответствие значения RSSI и расстояния – из-за частоты работы в диапазоне между 2400-2500 MHz сигнал не обладает достаточной силой и перекрывается многими другими устройствами, также работающими в этом диапазоне. Вследствие чего, зачастую, даже при появлении трёх роутеров в зоне видимости RSSI был слишком мал. Именно по этой причине, пересечения трёх окружностей с центрами в координатах роутера и радиусом, равным посчитанному расстоянию на основе RSSI, не было. Погрешность вычисления в таком случае слишком велика.

Из-за этих особенностей была невозможна реализация проекта, согласно поставленной задаче. Тем не менее, этот способ позиционирования имеет свои преимущества и недостатки.

Преимущества:

1. Возможность использовать существующую инфраструктуру. При постановке задачи определения комнаты, в которой находится устройство, или любой другой задачи позиционирования устройств внутри помещений, не требующих высокой точности определения местоположения. Этот способ является самым дешевым при наличии покрытия Wi-Fi сетью.
2. Низкое энергопотребление. При использовании Wi-Fi позиционирования вместо GPS значительно уменьшается энергопотребление.
3. Достаточно большой радиус действия. Wi-Fi сигнал способен распространяться вплоть до 150 метров.
4. Возможность определить этаж. За счет уникального отпечатка каждого роутера есть возможность хранить в базе соответствие отпечатков и этажей, и, определяя самый сильный сигнал, возможно определить, на каком этаже находится устройство.

Недостатки:

1. Низкая точность. Точность позиционирования даже при специально подготовленной инфраструктуре будет не менее 15 метров.
2. Высокая цена улучшения инфраструктуры. Цена установки новой точки будет складываться из стоимости роутера, витой пары нужной длины, а также подведения питания и работ по установке.

*1.1.1.2. Геомагнитное позиционирование*

Геомагнитное позиционирование основано на магнитном поле земли, а именно на аномалиях в магнитном поле, которые используются в этом методе позиционирования. Погрешность при использовании данного подхода равна примерно 2 метрам. Перед осуществлением навигации, необходимо подготовить карту геомагнитных аномалий, а также перед каждым сеансом произвести корректировку магнетометра, после чего позиционирование будет готово к работе.

При реализации текущего подхода сразу были выявлены недостатки: магнитное поле в крупных зданиях не такое постоянное, и современная архитектура и обилие электронных устройств создают множество динамических аномалий, из-за которых построение постоянной карты здания становится невозможным. Таким образом, эмпирическим путем было выявлено несоответствие заявленной и реальной погрешностей.

К тому же, из-за того, что крупные здания обвиты проводкой, поле в которой меняется в зависимости от подключенной нагрузки, значительно изменяя конфигурацию магнитного поля вокруг себя, карта аномалий может меняться на протяжении дня.

Все это явилось причиной отказа от этого подхода. Тем не менее, данный подход широко используется в местах без доступа GPS сигнала и статичным полем аномалий, например, на подземных стоянках и складах, где расстояние до стен достаточно велико, а количество носимых устройств незначительно большое, чтобы динамически влиять на магнитное поле. Финский университет Уолу представил прототип описанной системы для использования в мобильных устройствах внутри помещений.

*1.1.1.3. Ориентирование по базовым станциям операторов сотовой связи.*

GSM - ориентирование основано на триангуляции сигнала сотовых вышек. Каждая базовая станция регулярно транслирует сигнал в открытый эфир, чтобы телефоны могли понимать, находятся ли они в зоне покрытия. Обычно в городах телефоны находятся в зоне покрытий сразу нескольких базовых станций, и каждое устройство выбирает базовую станцию своего оператора с наилучшим сигналом. Для упрощения доставки входящих звонков телефон с низкой периодичностью сообщает сети о том, какую базовую станцию он «слышит» лучше всего, когда сигнал с последней переданной станции становится слишком слабым. Это сделано для экономии заряда мобильных девайсов. Таким образом, процесс определения местоположения в данном случае должен осуществляться на мобильном устройстве. Этот метод подходит для решения поставленой задачи. Для определения своего местоположения достаточно знать координаты видимых базовых станций, которые можно получить из открытых источников, и, используя силу сигнала, вычислить расстояние между абонентом и базовой станцией. В случае одной видимой базовой станции получаем окружность с центром в точке расположения станции и радиусом, равным расстоянию между устройством и станцией, такая погрешность может доходить до 32 км, что является неприемлемым для поставленной задачи. Рассматривая лучший случай, когда есть две и более базовых станций, сила сигнала которых настроена на минимальные значения, получаем две и более окружностей, которые могут иметь или не иметь область пересечения. Такое поведение обусловлено физическими свойствами сигналов и большой погрешностью на закрытых площадях. В данном случае возможно сократить погрешность до 100 метров. Таким образом, можно сделать вывод, что данный подход нельзя использовать для решения поставленной задачи. Однако стоит отметить его преимущества и недостатки:

Преимущества:

1. Хорошее покрытие купных городов базовыми GSM - станциями.
2. Возможность использования базовых станций всех операторов связи.
3. Низкое энергопотребление.

Недостатки:

1. Большая погрешность вычислений.
2. Сложность улучшения инфраструктуры.
3. Невозможность определения высоты.

*1.1.1.3. Технология iBeacon*

Определение местоположения при помощи Вluetooth iBeacon-маячков выглядит следующим образом: по всему периметру расставлены Bluetooth-метки, которые, придерживаясь протокола вещания iBeacon, производят широковещательную рассылку, которая, согласно типизации этого протокола, содержит идентифицирующую их информацию. Мобильное устройство принимает сигналы от всех маячков, которые до них доходят. Используя полученные данные, мобильное устройство может сопоставить уникальные идентификаторы устройств с координатами, которые были сохранены заранее. Для определния расстояния от устройства до маячка используется параметр RSSI.

RSSI (Received Signal Strength Indicator) – параметр силы сигнала, определенный в спецификафии Bluetooth. Этот параметр вычисляется пользовательским Bluetooth-приемником и обозначает силу принимаемого сигнала. Чем выше этот параметр, тем ближе устройство находится к маячку. Для того, чтобы определить расстояние в эмпирических величинах, например, в метрах, нам понадобится другой параметр – MP, который описан на уровне iBeacon.

MP (Measured Power) – уровень сигнала в 1 метре от передатчика. Данное значение передается вместе с остальной частью уникального идентификатора устройства. Данный параметр задается каждому маячку на заводе, и у маячков одного производителя и одной модели этот показатель может различаться.

Физически каждый iBeacon-маячок выглядит как небольшая плата с Bluetooth 4.0 LE (Low Energy). Протокол iBeacon – стандарт формата широковещательного сообщения, представленный Apple. Таким образом, для тестов возможно использовать любое устройство, оснащенное Bluetooth 4.0 и выше. Для практической реализации достаточно иметь недорогие iBeacon-маячки, которые состоят из батарейки, микроконтроллера, отвечающего за широковещание и периферийного модуля Bluetooth 4.0 LE.

Данная технология полностью удовлетворяет критериям поставленой задачи. Прототип системы, описанный с использованием данного подхода, показал достойные результаты. iBeacon хорошо подходит для решения задачи, но также, как и другие технологии, имеет свои преимущества и недостатки.

Преимущества:

1. Низкая стоимость оборудования.
2. Не требует подключения к сетям питания или Интернету.
3. Работоспособность до 3-х лет от одной батарейки.
4. Возможность прототипизирования системы с использованием мобильных телефонов без покупки iBeacon.
5. Небольшая погрешность вычислений (до 5 метров).
6. Легкое расширение инфраструктуры.
7. Определение этажей.
8. Низкое электропотребление пользовательского устройства.

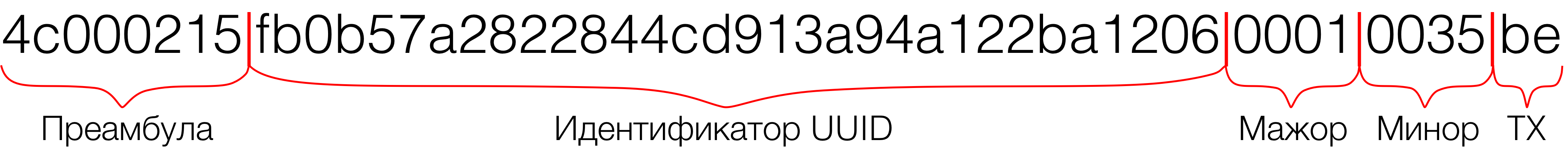
Минусы:

1. Необходимость контролировать состояние элементов питания во всех маячках.
2. Слабый сигнал, который блокируется перекрытиями.
3. Необходимость вручную развешивать маячки и сохранять их координаты, а также уникальный идентификатор в базе.
4. Отражение радиосигнала от поверхностей.
5. Зависимость от направленности излучения маячка и пользовательского устройства.

1.2. Форматы передачи данных

1.2.1. Формат данных iBeacon

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 разработчкиков.



*Рис 1. Формат данных iBeacon*

Преамбула – занимает 4 байта и является префиксным значением, определяющим, что это iBeacon маячок. Данный префикс всегда 4c000215.

Идентификатор UUID – на него отведено 16 байт, этот идентификатор не уникален и определяет конкретную группу маячков. Представим, нам необходимо установить iBeacon-маячки в главном корпусе ВГУ. Для решения данной задачи нам достаточно сгенерировать один UUID для всех маячков, таким образом, мы сможем подписываться на сообщения только от этих устройств и не будем получать другие, которые также могут быть установлены.

Мажор – на данную составляющую отведено 2 байта, это 2^16 = 65 536 различных значений. Мажор рекомендуется использовать для определения большой группы маячков, иденцифицируемой одним значением UUID. В примере с оборудованием главного корпуса ВГУ оптимально присвоить уникальное значение мажора каждому кабинету.

Минор – на него так же отведено 2 байта, этот параметр используется для однозначного определения маячка внутри группы.

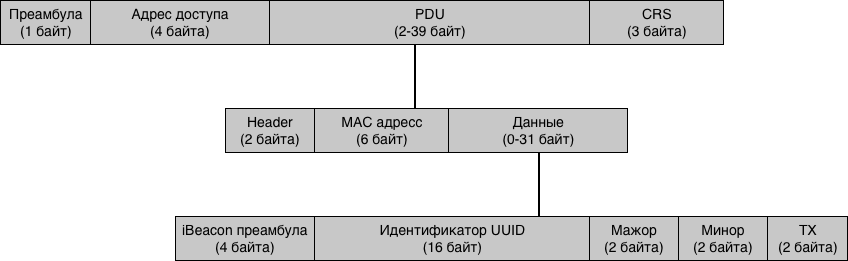
TX – занимает 2 байта в конце протокола и обозначает эталонное значение мощности маячка (значение RSSI), измеренное на заводе-изготовителе на расстоянии 1 метра от маячка. Первый бит - это знак, 1 соответствует отрицательный знак, а 0 - положительный. Именно благодаря этой константе, можно использовать метрические координаты в пространстве. В схеме, представленой на рисунке, значение TX равняется 0xBE, что соответствует числу 190 в десятичной системе счисления. Соответственно, эталонный RSSI на расстоянии 1 метра можно вычислить как 256-190 = -66 dBm.

Таким образом, получая связку UUID+мажор+минор, можно однозначно определить, от какого маячка был получен сигнал, после чего сделать выборку из базы, для получения координаты маячка с таким значением.

Также для решения задачи можно использовать несколько UUID, например, назначить уникальный UUID для каждого этажа. Используя данный подход, можно будет однозначно определить, какую часть карты стоит открыть, не делая запрос в базу.

1.2.1. Формат данных Bluetoth

iBeacon-формат является одним из стандартов данных, которые могут быть записаны в свободную область данных формата Bluetooth. Любое мобильное устройство, оснащенное Bluetooth-модулем, может считать информацию об iBeacon-маячках. Формат Bluetooth-пакета имеет следующую форму:



*Рис 2. Формат Bluetooth*

Как видно из рисунка, iBeacon - одно из заполнений области данных, слоя PDU в формате сообщения Bluetooth. Благодаря такому свободному формату, существуют аналоги формата iBeacon, например, формат Eddisone, разработанный в Google. Преимущество формата iBeacon - поддержка большим количеством платформ, чем другие форматы. iBeacon-формат не занимает всю область данных, отведенную в PDU, то есть при необходимости, можно расширить формат и использовать оставшиеся 5 байт в своих целях.

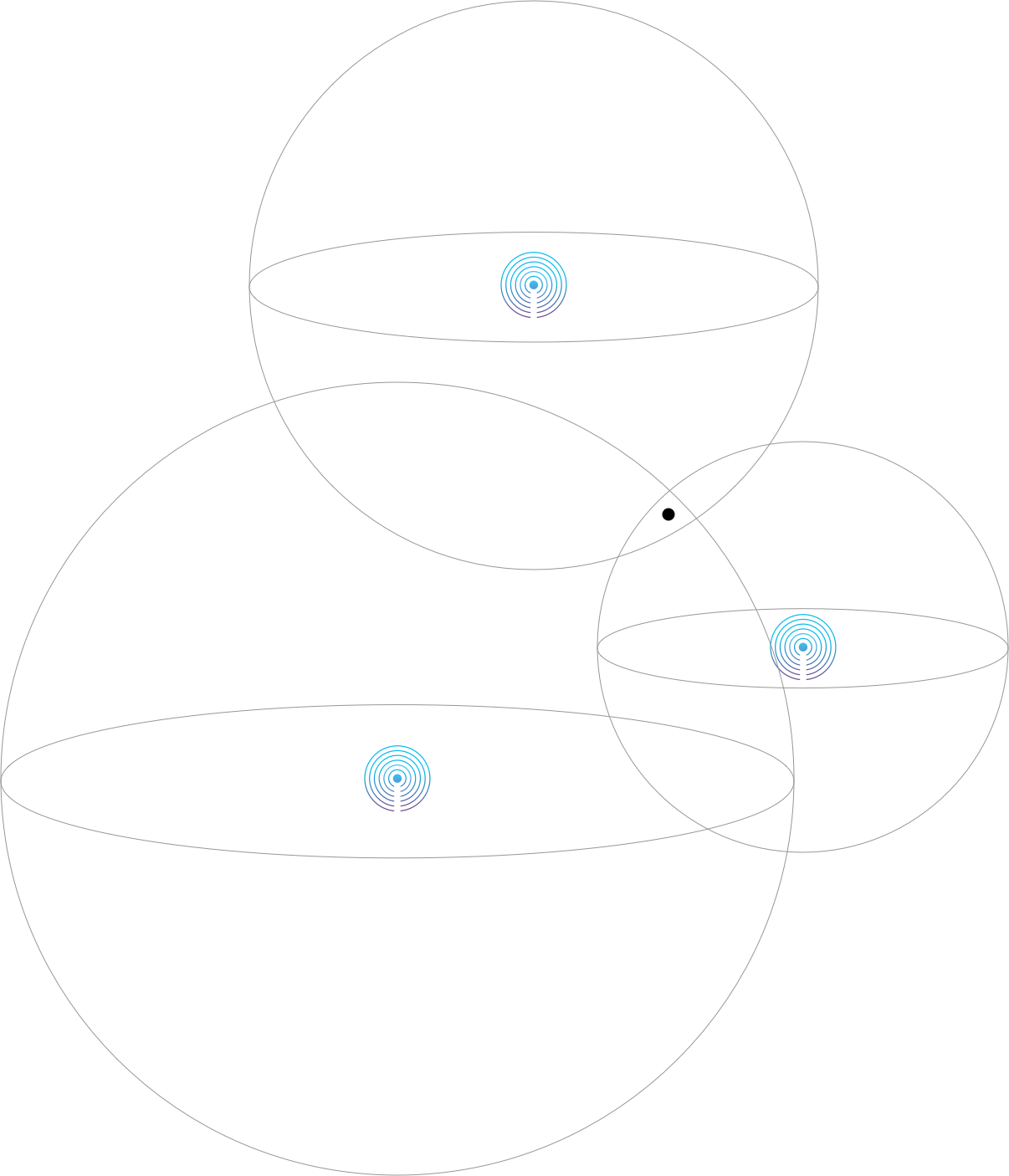
1.3. Алгоритмы определения местоположения.

1.2.1. Введение

Определение местоположения внутри помещений – фундаментальная проблема при программировании робототехники. Но в отличии от решений, применяемых в робототехнике, невозможно напрямую использовать встроенные сенсоры и команды для определения местоположения из-за того, что мобильные устройства приводятся в движение при помощи третьих лиц, в то время как в робототехнике возможно полагаться на команды движения самого робота. Сделав обзор применяемых технологий для определения местоположения, можно сделать вывод, что практически все они основаны на радиоволнах. Выбранная для решения поставленной задачи технология iBeacon так же использует радиоволны.

1.2.2. Алгоритм трилатерации

Трилатерация - это метод, позволяющий определить точку пересечения сигналов от передатчиков, путем построения в пространстве трех смежных треугольников, в которых известны длины их сторон. Этот метод основан на линейной засечке. Также часто применяются методы триангуляции и полигонометрии, но для этих методов необходимо обладать информацией об углах.



*Рис 3. Графическое представление задачи трехмерной трилатерации*

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. Для решения системы, вычтем второе уравнение из первого, благодаря чему найдем x:

Предположим, что две сферы имеют более одной точки пересечения, тогда имеет смысл следующее соответствие:

.

В этом случае, после подстановки x в уравнение первой сферы, находящейся в начале координат, находим уравнение окружности, которое является фигурой, пулучившейся в результате пересечения двух сфер.

Подставим в уравнение третьей сферы равенство:

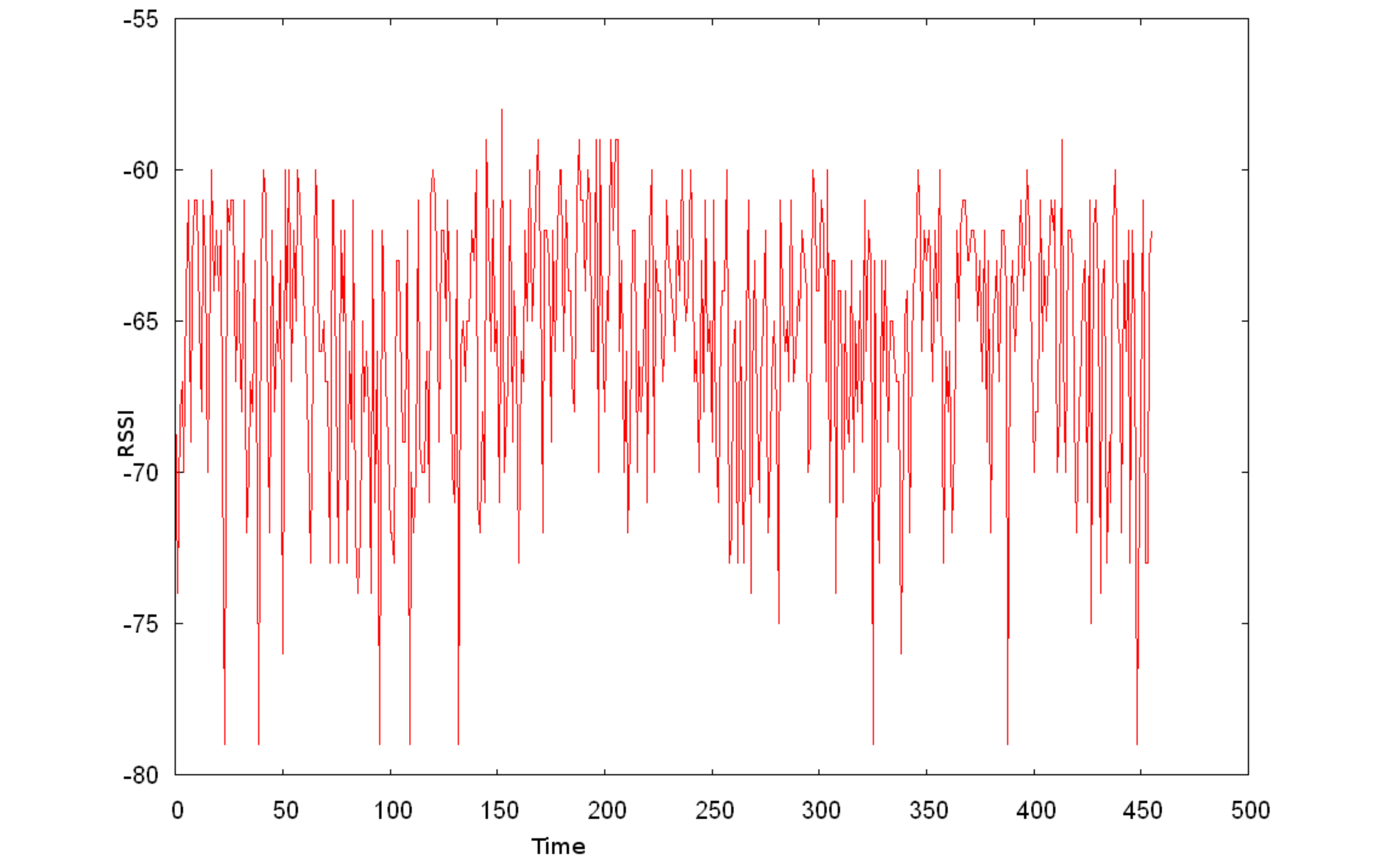
И найдем значение y:

Зная координаты x и y, можно найти координату z:

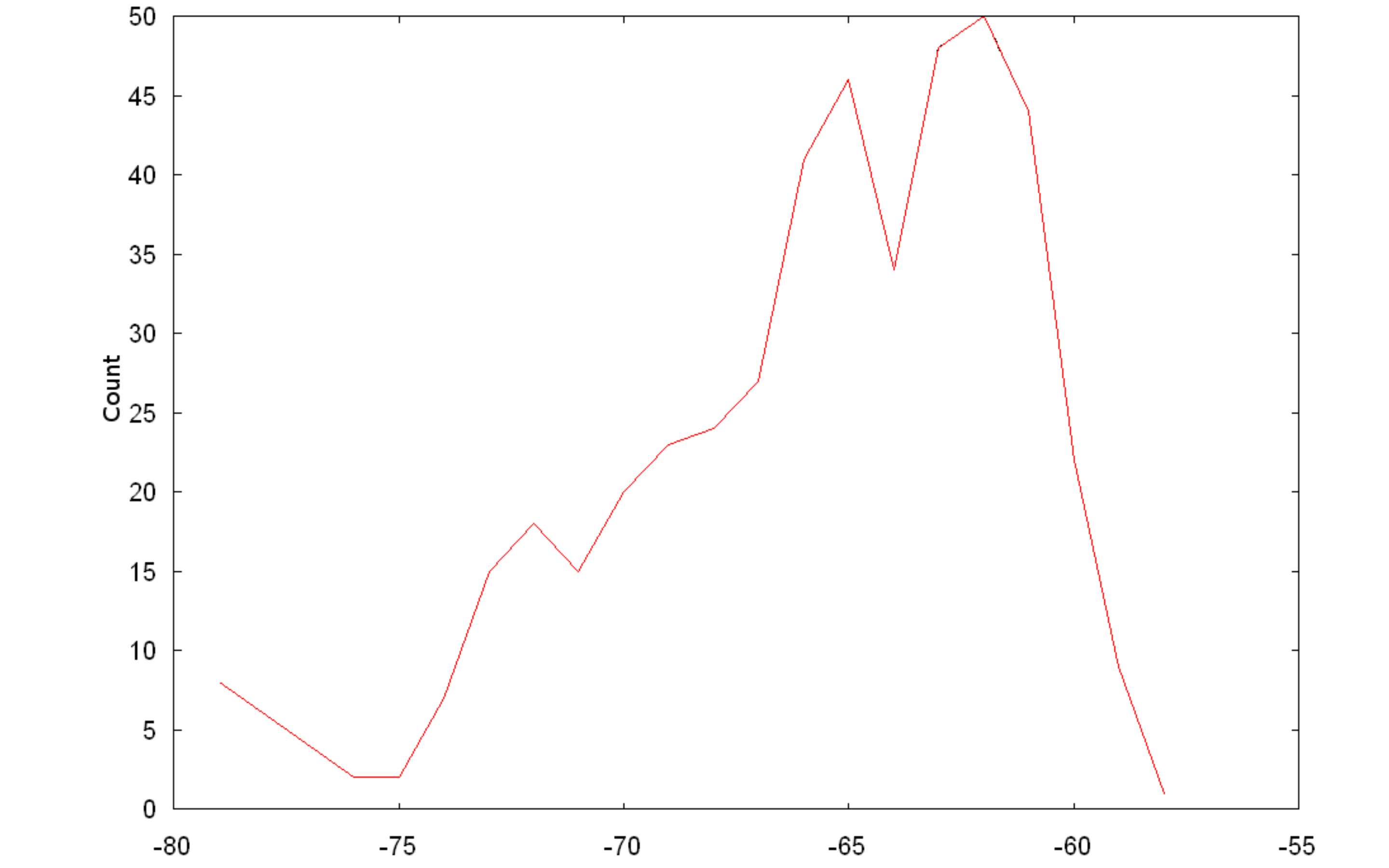
Таким образом, выведено уравнение для координаты приемника, используя информацию, полученную от Bluetooth-передатчиков. Очевидно, координата высоты может принимать два значения. Это обусловлено тем, что передатчики и приемник могут находиться не в одной плоскости. Для нашей задачи можно рассматривать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

1.2.3. Практическая реализация алгоритма трилатерации

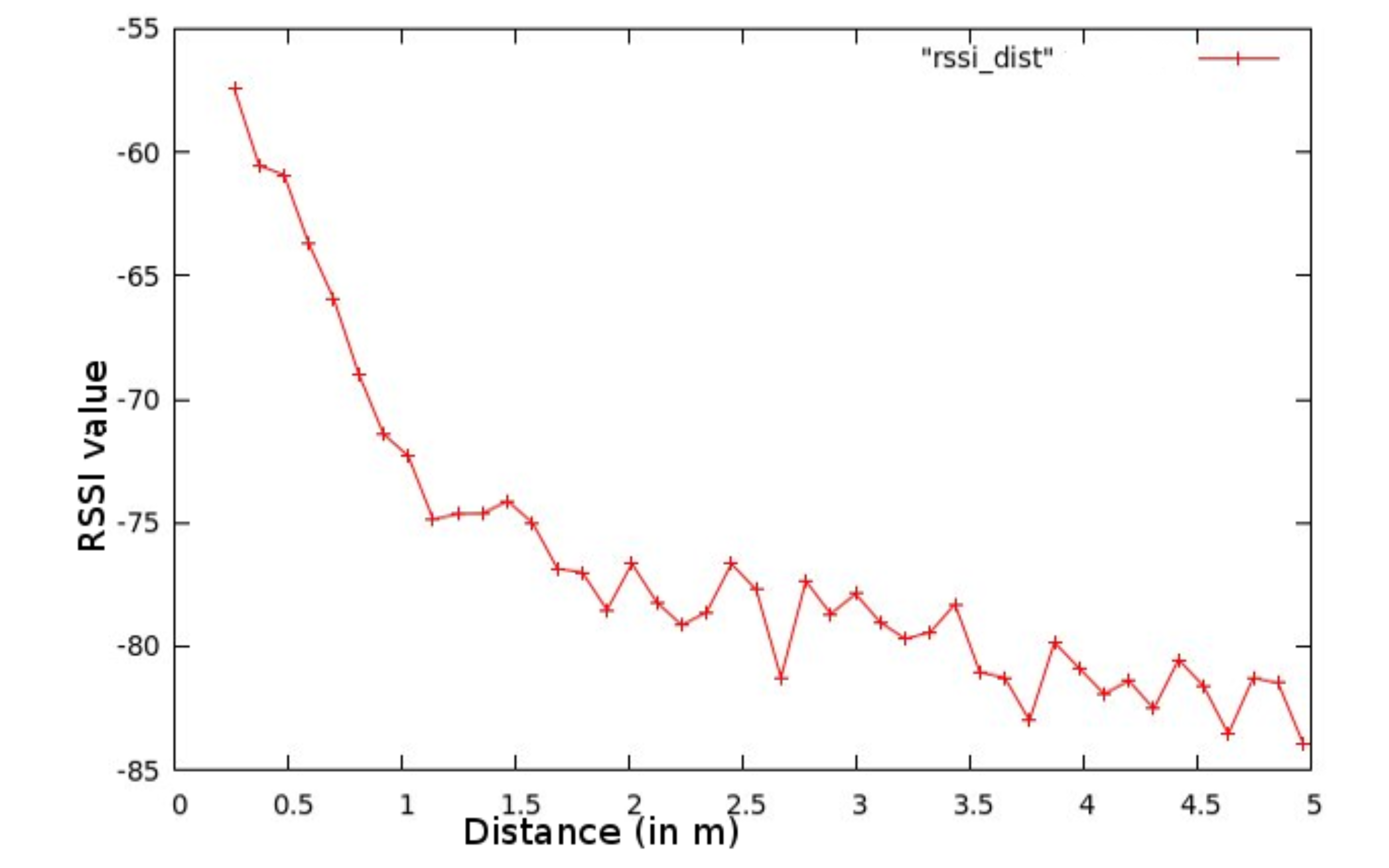
Получаемый от маячков параметр RSSI (Received Signal Strength Indicator) – параметр, позволяющий определить удаленность маячка от приемника сигнала. Это происходит следующим образом: маячок широковещает свой уникальный идентификатор, в хвосте которого находится параметр TX. Используя параметр, обозначающий эталонный RSSI на расстоянии 1 метр, производится деление текущего значения RSSI на эталонный. В теории мы должны получить точное расстояние в метрах между приемником – мобильным устройством и передатчиком (Bluetooth-маячком) с форматом передачи данных iBeacon. Однако из-за физического эффекта интерференции сигналов значение RSSI между стационарно-установленными приемником и передатчиком непостоянно, что можно увидеть на графиках.



*Рис 4. График изменения значения RSSI при статичном размещении приемника и передатчика*



*Рис 5. График моды изменения значения RSSI при статичном размещении приемника и передатчика*



*Рис 6. График значений RSSI на разном расстоянии*

На представленных графиках видно, что при использовании одного маячка можно получать разную удаленность при статичном расположении приемника и передатчика. Таким образом, задача трилатерации в чистом виде не позволяет определить местоположение устройства. В данных условиях можно получить в пересечении трех сфер область либо не получить точек пересечения сфер вообще. Для решения данной задачи необходимо модифицировать алгоритм трилатерации, используя итеративный подход.

1.2.4. Алгоритм итеративной трилатерации.

Обозначим начальные точки и полученное расстояние 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

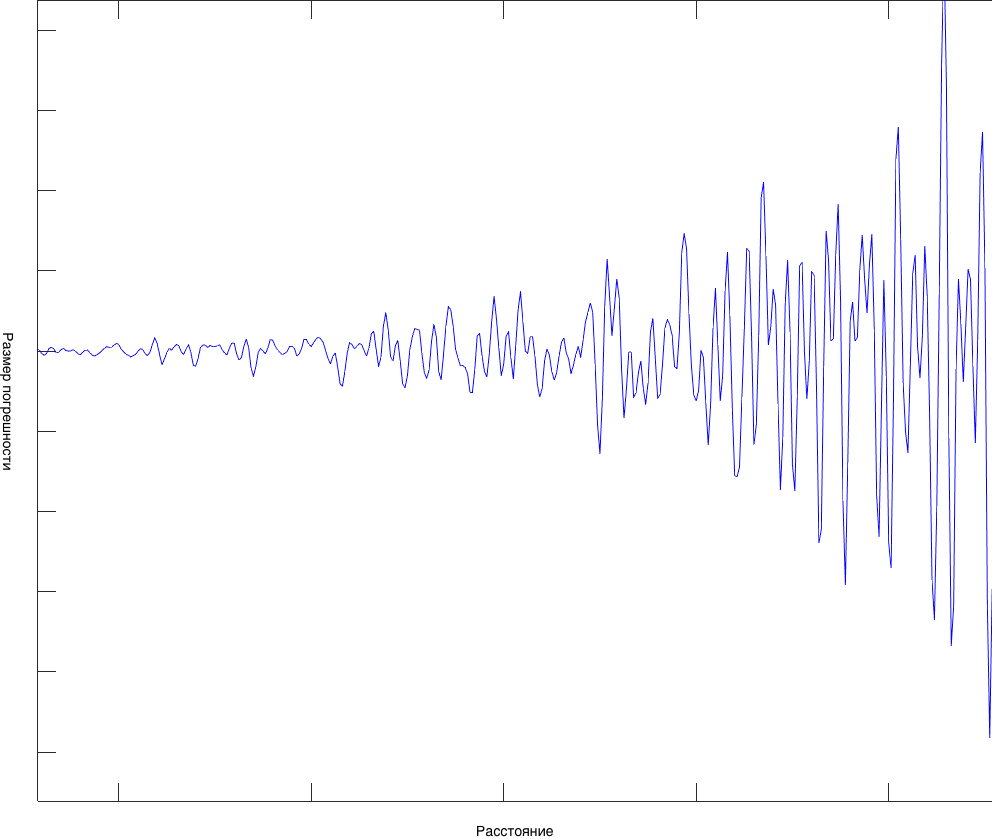
В результате получим:

Учитывая это, *fi* может быть рассчитан

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

1.2.5. Практическая реализация алгоритма итеративной трилатерации.

В результате практической реализации алгоритма итеративной трилатерации удалось решить проблемы отсутствия решений системы уравнений или же получения области решений уравнения. Для достижения целевой точности необходимо четко определить минимальное доверительное значение RSSI. Это необходимо сделать, исходя из технических характеристик маячков. Более простые решения, применяемые в дешевых маячках, имеют недостаточную силу сигнала. Точность определения расстояния обратно пропорциональна растоянию, то есть чем дальше мы находимся, тем меньше точность определения. Стоит отметить, что отношение значения погрешности к расстоянию выглядит очень похожим на экспоненциальную функцию.



*Рис 7. График погрешности измерения значения RSSI в зависимости от расстояния*

То есть, зная допустимую погрешность, можно определить минимальный уровень доверия маячку. Также стоит учитывать, что маячки должны быть расположены таким образом, чтобы в любой точке помещения находилось как минимум три маячка с уровнем RSSI выше порогового допустимого значения для точности. При наличии же физических препятствий или отражателей сигналов, таких как столбы, шкафы, зеркала, необходимо размещать больше 3-х маячков с достаточной силой сигнала. Этот подход позволяет получить несколько значений и рассчитать их среднее. Все возможные сочетания определяются по формуле:

где k=3 - элементы, необходимые для трилатерации местоположения, n – различные маячки, причем nk. Таким образом, увеличив количество видимых маячков с достаточной силой сигнала до четырёх, получаем 4 координаты, усреднив значения которых, определяем более точное значение. При пяти маячках количество координат увеличивается до десяти. Такое количество маячков позволяет определить местоположение с достаточной точностью даже в самых сложных условиях.

Используя описанные решения на прототипе приложения, вычисляется местоположение пользователя с достаточной точностью. Однако существует такая проблема как «шум», который проявляется в виде изменения координаты пользователя на дистанцию до 5 метров при отсутствии физического движения. Для фильтрации «шума» недосточно только информации от маячков. Понять, находится ли устройство в покое или в движении, можно при помощи встроенных в устройство сенсоров, магнетометра, гироскопа и акселерометра. Для того, чтобы учитывать внешние атрибуты, подходят методы на основе рекурсивной оценки Байеса.

1.2.6. Методы на основе оценки Байеса.

Для интеграции данных о передвижении устройства, получаемых от сторонних сенсоров, были рассмотрены и реализованы многочастотный фильтр и фильтр Калмана. Фильтр Калмана предназначен для рекурсивного дооценивания вектора состояния априорно известной динамической системы, другими словами, для расчета текущего состояния системы необходимо знать предыдущее состояние фильтра. Алгоритм ориентирован на двухэтапную работу:

1. Этап прогнозирования. На первом этапе фильтр Калмана экстаполирует значения переменных состояния, а также их неопределенности.
2. Этап уточнения. 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.

Для интеграции фильтра Калмана в систему необходимо определить динамическую составляющую системы последовательных трилатераций. Управляющим воздействием будет информация о векторе движения, получаемая от набора датчиков, и множество последовательных измерений местоположения методом трилатерации маячков для формирования реальной оценки состояния.

Координату мобильного устройства можно описать законом:

*(1)*

Из-за невозможности опираться только на сенсоры, необходимо добавить случайную величуну .

*(2)*

Выше описан алгоритм получения координат от внешних маячков. Для использования его с фильтром Калмана добавим к этим координатам ошибку распространения навигационного сигнала, обозначенную за , которая будет являться случайной величиной. В результате, получим ошибочные данные вычисления координаты:

*(3)*

Итоговая задача фильтрации будет состоять в том, чтобы, используя неверные показания сенсора , добиться достаточно точного приближения к

В результате, получаем уравнение системы для мобильного устройства с использованием набора сенсоров. Вычисление позиции по маячкам будет иметь следующий вид:

(4)

где – известная временная, контролирующая изменение системы, она определяется из вектора движения, полученного от датчиков.

и – ошибки изменения вектора системы и датчиков, соответственно.

Если для k-го шага известно значение сенсора , отфильтрованное таким образом, что приближает истинную координату достаточно, то на следующем шаге становится известно неточное показание сенсора

Задача интеграции фильтра Калмана для использования с определением местоположения при помощи внешних iBeacon-маячков и набора внутренних сенсоров состоит в получении наилучшего приближения к истинной кооординате , из ошибочных показаний сенсора и предсказаний истинного значения - . Для этого необходимо добавить коэффициент, определяющий вес сенсора K, а предсказанному значению (1-K). В результате чего, будет получена следующая формула:

(5)

где K – коэффициент Калмана шага.

Коэффициент Калмана должен быть выбран так, чтобы получившееся оптимальное значение координаты было наиболее близко к истинной координате Представим, что в данном здании маячки установлены с высокой плотностью и количество препятствий, ослабляющих сигнал минимально. В таком случае, определение координат при помощи iBeacon-маячков будет обладать высокой точностью. Для того, чтобы доверять координатам результата трилатерации маячков, значению коэффициента K устанавливается больший вес. Если в области определения местоположения сигнал маячков слаб или количество маячков небольшое, либо недостаточное (меньше трех), то ориентацию в этой части помещения лучше производить, используя вектор движения, полученный от встроенных датчиков, то есть увеличить коэффициент Калмана для вектора сенсоров.

Для нахождения точного значения коэффициента Калмана необходимо произвести минимизацию ошибки:

*(6)*

Используя уравнение (4), получим:

(7)

Для рассматриваемой системы критерием минимизации можно считать среднее значение от квадрата ошибки:

(8)

Таким образом, применяя уравнение (7) и критерий минимизации (8), получаем:

(9)

Минимальное значение уравнение (9) принимает при

*(10)*

Подставив в среднеквадратичкую ошибку , миниммизирующее значение коэффициента Калмана , получаем:

(11)

Таким образом, полностью интегрирована фильтрация Калмана по данным датчиков в существующий алгоритм трилатерации.

Формула (11) является итерационной и позволяет находить коэффициент Калмана.

Формулы (5) и (11) позволяют найти приближения расчитанной координаты к вектору скорости, путем подсчета коэффициента Калмана.

Многочастотный фильтр (последовательный метод Монте-Карло) – рекурсивный алгоритм для численного решения проблем оценивания: фильтрации и сглаживания, особенно для нелинейных и негауссовских случаев. В отличии от фильтра Калмана, многочастотные фильтры не зависят от методов линеализации и апроксимации. Детально рассматривать данный подход не имеет смысла, так как эмпирическим путем было выявлено, что благодаря своей пошаговой природе, фильтр Калмана лучше справляется с устранением шума. К тому же, при движении он обеспечивает оптимальное сглаживание за счет вектора состояния.

ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ.

2.1. Выбор платформы и языка программирования

Для выбора платформы мобильного приложения необходимо определить, что и для кого нужно написать. Требуется разработать алгоритм для определения местоположения мобильного устройства с использованием маячков iBeacon и встроенных сенсоров. Так как для работы данных алгоритмов нужен минимальный набор сенсоров в телефоне, состоящий из гироскопа, магнетометра и акселерометра, можно сделать вывод что необходимо разработать нативное приложение для смартфонов под управлением наиболее популярных мобильных операционных систем, а именно iOS и Android.

Операционные системы iOS и Android имеют различные инструменты для нативной разработки. Подходы при разработке приложения для мобильных устройств можно разделить на три типа:

1. Веб приложения. Самый быстрый способ разработать мобильное приложение. Благодаря принципам гибкой разметки и стандартам современных веб - технологий, становится возможным реализация веб - приложений с элементами, характерными нативной разработке, такими как кеширование, оффлайн работоспособность, возможность получить доступ к большому числу API операционной системы, такими как доступ к галерее, воспроизведение музыки, использование внутренных сенсоров и многое другое. Основными преимуществами данного подхода является отсутствие необходимости установки приложения и кроссплатформенность. Недостатками является ограниченость оффлайн функциональности, сложность поддержки всех браузеров, ограничения в использовании API, например, в результате исследования, данный подход оказался не подходящим для решения поставленной задачи, так как современные веб - фреймворки не дают доступ к необходимому API для работы с Bluetooth, и в частности с iBeacon.
2. Нативные приложения. Такими приложениями обычно называют приложения, требующие установки и разрабатываемые отдельно для каждой платформы. Для нужных нам платформ, iOS и Android, необходимо использовать разные принципы и подходы разработки, начиная от дизайна и заканчивая принципами проектирования. Так же для реализации нативной разработки нужно использовать различные языки, инструменты и среды разработки. Для разработки приложений для операционной системы iOS можно использовать такие языки как Objective-C, Swift, среду разработки Xcode и менеджер зависимостей Cocoapods. Для разработки приложения под операционную систему Android необходимо использовать такие языки, как Java или Kotlin, среду разработки Android Studio и менеджер зависимостей Gradle. К тому же для разработки интерфейсов приложений необходимо придерживаться принципам построения интерфейсов, Human Interface Guidelines для iOS и Material Design Principles для Android. Несмотря на большие временные затраты, в сравнении с веб приложениями, нативная разработка обладает такими преимуществами, как безопасность, скорость работы, поддержка всех современных механизмов, доступных при обновлении системы, работа в фоне, работа без доступа к сети.
3. Гибридные. Создание гибридных приложений находится всегда между веб и нативной разработками. Существует множество инструментов, одни из которых делают упор на веб сторону, другие на нативность. Основной принцип создания гибридного приложения – создать одно приложение, унивесальное для всех платформ с доступом к нативному API. До недавнего времени, несмотря на востребованность гибридной разработки, она применялась, в основном, в простых приложениях, без повышенных требований к безопасности, скорости работы, пользовательскому опыту использования и работе в оффлайн режиме. Но в середине 2015 года мир гибридных приложений переродился благодаря разработанной на хакатоне Facebook технологии, получившей в дальнейшей название React native. Этот подход к разработке не пытается предлагать создавать одно приложение для всех операционных систем, а предполагает вынесение основной бизнес - логики и модели в отдельный переиспользуемый модуль, написанный на javascript. В отличии от других подходов к гибридной разработке мобильных приложений React native предлагает описывать интерфейс пользователя, отрисовку анимаций с использованием нативных языков и средств разработки, что качественно влияет на пользовательский опыт, безопасность и работу в оффлайн режиме.

Очевидно, что для написания кроссплатформенного продукта использование React Native является оптимальным, и первый прототип приложения был написан именно на этой технологии, но у этого подхода есть минус, о котором все стараются молчать, а именно синхронный мост между нативным кодом и кроссплатформенным, то есть все сообщения по обработке пользовательских касаний, прокрутки карты и передачи данных о текущем местоположении пользователя идут синхронно в главном потоке, что приводит к замораживанию пользовательского инферфейса при большом объеме передаваемых данных между блоками приложения, а это происходит при появлении большого количества маячков. Причина этого заключается в том, что модуль получения информации о маячках возможно описать только на уровне нативного приложения, а алгоритм расчета местоположения должен быть вынесен в кроссплатформенный модуль.

В итоге, для решения данной задачи был интегрирован подход ручной синхронизации уровня подсчета координаты для двух платформ. Эта задача стала решаемой с появлением двух похожих языков разработки для iOS и Andoid, а именно Swift и Kotlin соответственно, разница в этих двух языках на столько незначительна, что путем произведения набора операций по поиску и замене можно перевести логику, написанную с одного языка на другой. На данный момент есть открытые библиотеки, которые могут делать это автоматизировано.

2.2. Организация взаимодействия с iBeacon маячками

API для работы с iBeacon маячками было представлено на Apple World Wide Developers Conference в 2013 году вместе с форматом iBeacon. Этот API позволяет устойствам под управлением оперционной системой iOS определять, в зоне действия каких маячков они находятся.

Для того, чтобы добавить отслеживание iBeacon маячков, нужно подключить к приложению библиотеку CoreLocation. Api, встроенный в Core Location, способен извещать устройство о том, что оно вошло в регион вещания конкретного маячка или вышло из региона его вещания. Вся эта функциональность доступна, начиная с iOS 7 и iPhone 4.

Так как iBeacon является частью Core Location и позволяет приложению отслеживать местоположение пользователя, для использования API пользователю нужно будет разрешить приложению доступ к его геоданным. Существуют различные пользовательские истории для реализации взаимодействия с iBeacon маячками. Вначале необходимо зарегистрироваться на отслеживание маячков с определенными уникальными идентификаторами (UUID), максимальное количество UUID, на отслеживание которых может подписаться одно приложение 20. После чего возможно зарегистрировать нотификацию о появлении в зоне видимости зарегистрированных маячков, это уведомление появится даже в случае, если приложение было выгружено из памяти. Apple рекомендует использовать эти уведомления для сообщения пользователю, что приложение обладает возможностью навигации в здании, в которое он зашел. Для регистрации в центре уведомлений приложения необходимо так же получить от пользователя разрешение на совершение данного типа действий.

Пример 1-1 Создание и регистрация iBeacon региона

- (void)registerBeaconRegionWithUUID:(NSUUID \*)proximityUUID andIdentifier:(NSString\*)identifier {

// Создание iBeacon региона для мониторинга

CLBeaconRegion \*beaconRegion = [[CLBeaconRegion alloc] initWithProximityUUID:proximityUUID

identifier:identifier];

// Регистрация Маячка в менеджере локации

[self.locManager startMonitoringForRegion:beaconRegion];}

Пока пользовательское устройство находится в области вещания маячка, приложение может начать мониторинг информации о маячках в регионе, используя метод startRangingBeaconsInRegion:, описанный в классе CLLocationManager.

Пример 1-2 Определение расстояния между iBeacon маячками и устройством

// Метод делегата из протокола, описанного в CLLocationManagerDelegate.

-(void)locationManager:(CLLocationManager \*)manager

didRangeBeacons:(NSArray \*)beacons

inRegion:(CLBeaconRegion \*)region {

if ([beacons count] > 0) {

//самый близкий

CLBeacon \*nearestExhibit = [beacons firstObject];

if (CLProximityNear == nearestExhibit.proximity) {

//Отображение информации о маячке к которому подошел пользователь

[self presentExhibitInfoWithMajorValue:nearestExhibit.major.integerValue];

} else {

//Скрывание информации о маячке

[self dismissExhibitInfo];

}

}

Так же для поиска других устройств с приложением рядом можно использовать встроенный в мобильное устройство Bluetooth, превратив его в iBeacon маячок.

Пример 1-3 Перевод мобильного устройства в режим iBeacon маячка

NSUUID \*proximityUUID = [[NSUUID alloc] initWithUUIDString:@"39ED98FF-2900-441A-802F-9C398FC199D2"];

CLBeaconRegion \*beaconRegion = [[CLBeaconRegion alloc] initWithProximityUUID:proximityUUID identifier:@"ru.vsu.myregion"]

NSDictionary \*beaconPeripheralData = [beaconRegion peripheralDataWithMeasuredPower:nil];

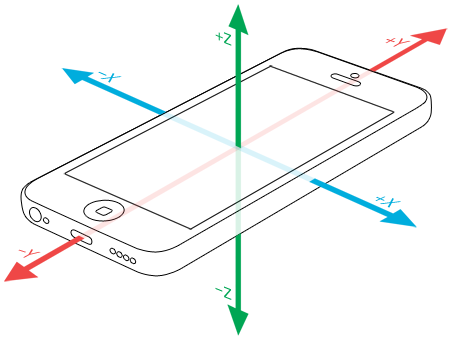
CBPeripheralManager \*peripheralManager = [[CBPeripheralManager alloc] initWithDelegate:selfqueue:nil options:nil];

[peripheralManager startAdvertising:beaconPeripheralData];

2.3. Организация взаимодействия со встроенными сенсорами

Встроенная библиотека Core Motion в операционной системе iOS позволяет получать и обрабатывать события, зафиксированные широким набором сенсоров. На данный момент основной набор сенсоров для телефона — это акселерометр, гироскоп и магнетометр. Каждый из перечисленныйх датчиком имееет особенности в его отслеживании. В случае, если телефон оборудован сопроцессорами M7 или M8, обрабатывающими постоянно информацию с сенсоров, то можно использовать информацию об активности пользователя, такую как шаги, количество пройденных этажей и тип активности (прогулка, бег, поездка на велосипеде…). Логически можно объединить акселерометр и гироскоп в одну группу, оба этих сенсора представляют получаемую информациию в виде трехосевой системы координат, центр которой проходит через само устройство. Например, для iPhone в портретной ориентации:

ось Ox проходит через устройство слева направо, Oy снизу-вверх, а Oz проходит перпендикулярно сзади вперед:



*Рис 8. Расположение системы координат в портретной ориентации*

Для работы со всеми сенсорами необходимо создать объект класса CMMotionManager. Этот класс предоставляет доступ ко всей информации о движениях мобильного устройства. Как и все API, предоставляющие персональную информацию, приложение получит доступ к информации и движениях только в случае, если пользователь при первом запуске приложение даст права на это. Класс CMMotionManager предоставляет подобные интерфейсы для каждого типа датчиков, акселерометра, гироскома и магнетометра.

let manager = CMMotionManager()

if manager.isGyroAvailable {

manager.gyroUpdateInterval = 0.1

manager.startGyroUpdates()

}

Для добавления вектора движения и фильтрации состояния покоя, используя фильтр Калмана, нам необхоимо знать, когда устройство находится в состоянии покоя, а когда движется, а так же поворачивать карту туда, куда направлено устройство. Для этого нам нужно подписаться на обновления данных акселерометра:

if manager.isAccelerometerAvailable {

manager.accelerometerUpdateInterval = 0.01

manager.startAccelerometerUpdates(to: .main) {

[weak self] (data: CMAccelerometerData?, error: Error?) in

if let acceleration = data?.acceleration {

let rotation = atan2(acceleration.x, acceleration.y) - M\_PI

self?.mapView.transform = CGAffineTransform(rotationAngle: rotation)

}

}

}

Каждый пакет CMAccelerometerData включает x, y, z занчения, отражающие количество ускорения в g (единица измерения гравитации) для каждой из осей. Например, если пользователь будет держать телефон в портретном режиме, вектор (x, y, z) будет выглядеть (0,-1,0), лежащий на столе телефон дает вектор (0,0,-1), а повернутый на 45 градусов (0.707, -0.707, 0). Для сглаживания движения нужно использовать данные гироскопа, для этого нужно аналогично у инстанса класса CMMotionManager вызвать метод startGyroUpdates.

if manager.isDeviceMotionAvailable {

manager.deviceMotionUpdateInterval = 0.01

manager.startDeviceMotionUpdates(to: queue) {

[weak self] (data: CMDeviceMotion?, error: Error?) in

if let gravity = data?.gravity {

let rotation = atan2(gravity.x, gravity.y) - M\_PI

self?.mapView.transform = CGAffineTransform(rotationAngle: rotation)

self?.imageView.transform = CGAffineTransform(rotationAngle: rotation)

}

}

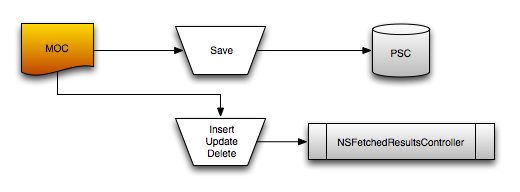
}

Остается проблема определения вектора движения, так как при равномерном движении ускорение может не изменяться, в этом случае, если пользователь запустил приложение, уже находясь в равномерном движении, можно использовать класс CMPedometerData, задав промежуток времени, в конфигурации которого можно получить доступ к количеству шагов пользователя, тип его текущего движения, узнать расстояние, которое он прошел за прошедший промежуток времени и много другое. Таким образом, беря промежуток времени 20 секунд до запуска приложения, зная тип движения, количество шагов и пройденное расстояние, можно расчитать среднюю длинну шага. На основе данных гироскопа возможно узнать, в каком направлении движется пользователь, и величину шага на основе расчетов сопроцессора серии M, таким образом можно построить вектор движения, который необходим для фильтрации Калмана.

***2.4. Способы организации работы с CoreData***

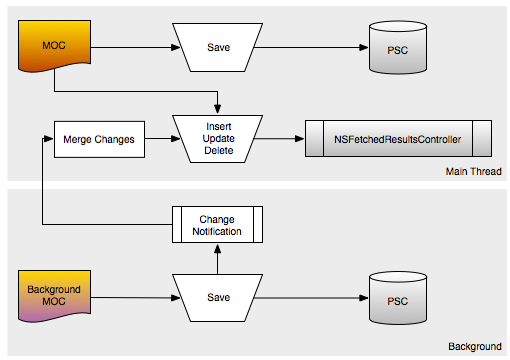
В данном проекте были рассмотрены несколько способов работы с CoreData, от самого простого, до самого быстродейственного:

1. Работа в главном потоке - самый простой способ, но относительно логики - неправильный, так как модель меняется до сохранения, кроме того, сохраняя в главном потоке, замораживается интерфейс пользователя на время сохранения



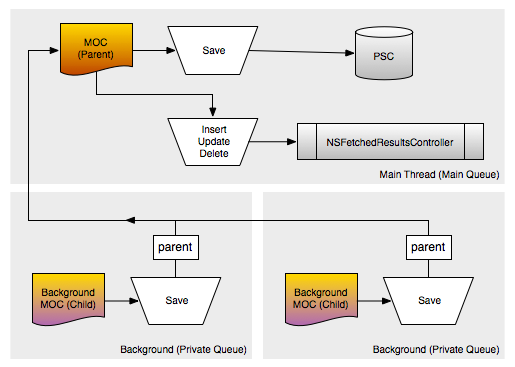
*Рис 9. Схема работы в главном потоке*

1. Работа в многопоточном режиме - паттерн, рекомендуемый apple в ее документации для параллельной работы с CoreData. Каждый поток имеет свой private контекст, каждый раз создает managed context в потоке, в котором он должен использоваться. Эта модель себя хорошо зарекомендовала, но в следующем обновлении iOS появилась более совершенная модель.



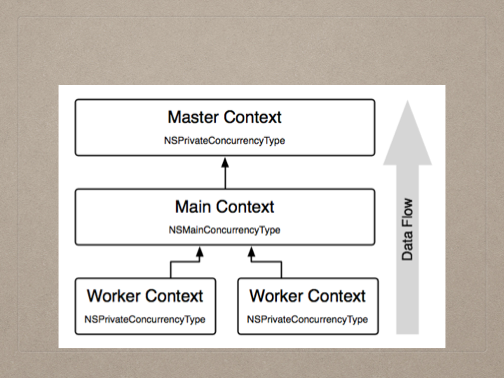
*Рис 10. Схема работы в многопоточном режиме*

1. Родительские контексты - Apple так же добавила возможность соединять в цепь managed object contexts вместе, устанавливая parentContext property для другого context. Таким образом, можно сделать изменения в Core Data и потом определиться, нужно ли их сохранить или откатить. При сохранении child context изменения автоматически распространяются на уровень выше, в представленном случае - на родительский контекст. Такой подход дает возможность обрабатывать добавление большого количество данных в фоновом потоке без блокировки главного потока(интерфейса).



*Рис 11. Схема работы с родительскими контекстами*

Но любой fetch request приведет к операциям чтения или записи с диска в главном потоке, сохранению данных в фоновом потоке и потом сохранению в главном потоке, что все равно заблокирует главный поток. После долгих поисков был найден паттерн лучше. Этот паттерн используется Apple UIManagedDocument и было найдено много рекомендаций использования этого паттерна, даже в известной книге Marcus Zarra "CoreData Book".



*Рис 12. Финальная структура организации работы с CoreData*

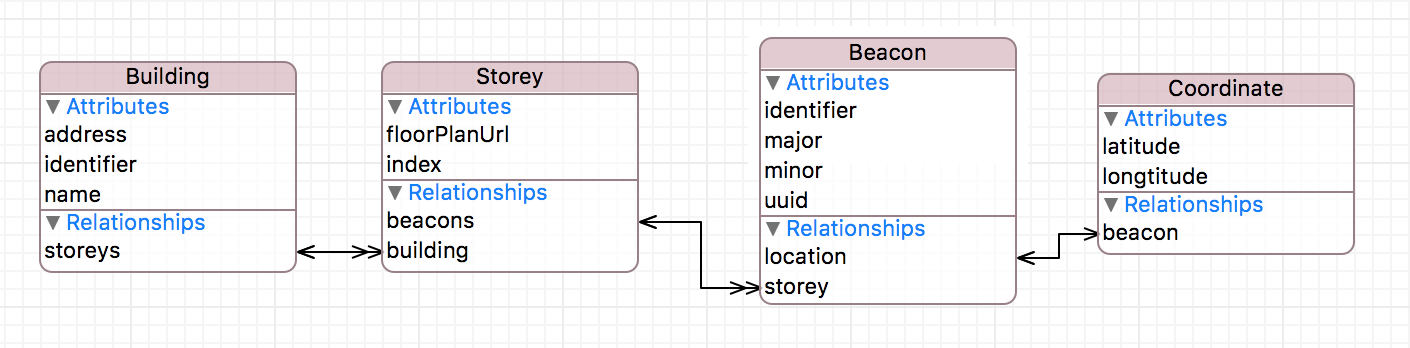
Идея заключается в том, что master managed object context присоединяется к persistent store с приватным распараллеливанием (операции в фоновом потоке), после чего создается managed object context в главном потоке as a child of this master context. Серьезные задания, такие как добавление крупных данных, происходят в worker contexts, которые устанавливаются как child contexts для main context с private concurrency type.

*2.4.1. NSFetchedResultsController*

NSFetchedResultsController представляет собой контроллер, предоставляемый фрэймворком Core Data для управления запросами к хранилищу. Этот класс идеально подходит для отображения выборок в таблицу, через него осуществляется запрос к БД, объекты загружаются в оперативную память по необходимости. Также большим преимусществом является возможность отслеживать изменения в результатах запроса, то есть достаточно добавить/изменить/удалить запись из базы, а FRC сам отследит эти изменения и в случае изменения перезагрузит нужные ячейки с обновленными данными.

***2.5. Схема базы данных***

Схема базы данных достаточно компактная, но это гарантирует всю нужную функциональность при максимальной производительности.



*Рис 13. Схема базы данных*

На рисунке 17 представлена схема локальной базы данных, основная сущность Beacon. Все атрибуты сущности являются обязательными, атрибут identifier является конкатенацией трех сторк uuid+major+minor, обеспечивающим уникальный идентификатор каждого маячка, этот атрибут используется для построения индекса в таблице. Маячок связан с вспомогательной сущностью Coordinate, это сделано для удобства работы с объектным представлением схемы. Сущность Storey – представляет информацию об этаже, а именно ссылку на схему этажа и индекс этажа. Связь этажа с маячками каскадная, следовательно при удалении этажа вызовется удаление всех маячков, которые с ним связаны. Сущность Building представляет собой здание, индекс построен по атрибуту identifier, который равняется идентификатору beacon, на который нужно подписаться для навигации в этом здании. Связь с этажами также каскадная, при удалении здания, будут удалены все этажи и все маячки каскадом.

***2.6. Обеспечение безопасности пользовательских данных***

Задача обеспечения безопасности очень актуальна в современном мире. Схема работы с маячками организована таким образом, что маячок транслирует сигнал, а приложение его принимает, сам же маячок не знает о нахождении пользователя рядом с ним.

Для хранения необходимой программе информации о посещенных местах в программе используется Keychain Services. Согласно документации Apple, Keychain **-** безопасный контейнер, предоставляющий возможность безопасно хранить пароли, ключи, сертификаты, заметки и подобное. Встроенные средства операционной системы Apple хранят в Keychain сетевые пароли для Wifi, учетные записи Vpn и т. д. Эти данные зашифрованы и хранятся в базе данных sqlite в файле /private/var/Keychains/keychain-2.db. Основные возможности, предоставляемые Keychain:

1. Поиск.
2. Изменение.
3. Добавление
4. Удаление

Эти действия можно производить над объектами keychain.

Для сохранения информации в keychain - используется класс KeychainItemWrapper. При записи объекта необходимо указать идентификатор. Также при записи объекта в Keychain необходимо указать значение kSecAttrAccessible, которое регулирует уровень доступа к данным keychain. Существует 6 возможных вариантов ограничения уровня доступа:

1. kSecAttrAccessibleAfterFirstUnlockК - к информации нет доступа после перезагрузки устройства, пока устройство не разблокировано пользователем. Записи с этим атрибутом мигрируют на новое устройство при использовании зашифрованных резервных копий
2. kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly - к информации нет доступа после перезагрузки устройства. Записи с этим атрибутом не мигрируют на новое устройство.
3. kSecAttrAccessibleAlways - данные будут доступны всегда, даже если устройство заблокировано. Записи с этим атрибутом мигрируют на новое устройство.
4. kSecAttrAccessibleAlwaysThisDeviceOnly - данные будут доступны всегда, даже если устройство заблокировано. Записи с этим атрибутом не мигрируют на новое устройство.
5. kSecAttrAccessibleWhenUnlocked - данные будут доступны только тогда, когда устройство разблокировано. Записи с этим атрибутом мигрируют на новое устройство.
6. kSecAttrAccessibleWhenUnlockedThisDeviceOnly - данные будут доступны только тогда, когда устройство разблокировано пользователем. Записи с этим атрибутом не мигрируют на новое устройство.

Для целей данного проекта выбран kSecAttrAccessibleWhenUnlocked, так как доступ нужен только активному приложению и миграция на другие устройства всегда удобна.

**2.7. Дополнительные инструменты**

***2.7.1. Система управления версиями файлов***

В качестве системы управления версиями файлов использовалась git. При разработке програмного продукта важно хранить все изменения, желательно на нескольких машинах. При разработке проекта над ним, как правило, работают несколько людей, возникает необходимость иметь каждому разработчику последнюю версию продукта со всеми изменениями. Эти функции выполняют системы контроля версиями. Самыми популярными версиями таких систем являются SVN и GIT. При работе над данным проектом был выбран GIT из-за ряда факторов:

* Наличие удобного механизма для работы с ветками.
* Распределенность рабочих копий.
* Хранение метаданных а не файлов.

***2.7.2. Система отслеживания ошибок***

В качестве системы отслеживания ошибок использовалась Jira. Системы отслеживания ошибок существуют для того, чтобы документировать процесс разработки. Если рассматривать Jira, то при ее использовании заводятся задачи, которые переводятся в список тех, которые нужно сделать. Задачи могут быть разного типа:

* Задания
* Эпики
* Стори
* Ошибки

Каждая задача имеет ряд состояний, например, «нужно сделать» - «в процессе» - «проверяется» - «готова». Данный подход позволяет систематизировать процесс разработки. Рассматриваемая система отслеживания ошибок помогает с планированием работ. Новые задачи могут добавлять разные пользователи и назначать исполнителя, который, в свою очередь, может оценить затраты времени на задачу, оставить свой комментарий или переводить на другого исполнителя; каждое действие фиксируется на общем экране, таким образом, руководителю проекта не нужно следить за каждым участником и узнавать его статус, а достаточно лишь смотреть на общий статус работ.

***2.7.3. Система сбора статистики, аналитики и обратной связи***

В процессе разработки, тестирования, а особенно после выпуска продуктов очень важно собирать данные пользователей. Их вопросы, отзывы, ошибки и проблемы. Системы аналитики мобильных приложений активно развиваются и конкурируют, большинство из них предоставляются бесплатно пока приложением пользуется небольшое количество людей. В текущем проекте интегрированы следующие системы:

* Mixpanel
* Flurry
* Crashalytycs
* Testflight

Каждая система отвечает за свою часть действий:

*Mixpanel* собирает информацию об активности пользователей, о том, как часто они пользуются приложением, куда они заходят в приложении, что не используют. Этот инструмент позволяет анализировать проблемные места приложения, выявлять недостатки и добавлять/удалять возможности приложения, которые требуются/не используются. Так же данный инструмент используется для A/B тестирования. Общий прицип данного тестирования состоит в том, что часть пользователей пользуются одной версией программы, а часть другой, и исходя из данных можно понять, какие изменения и как повлияли на удобство использования приложением.

*Flurry* данный инструмент используется для агрегирования обратной связи пользователей, если у человека возникают проблемы, вопросы или приложения, он может написать в обратную связь, и вместе с письмом отображается его активность, ошибки, с которыми он сталкивался и вся информация, которая в дальнейшем может понадобиться разработчикам.

*Crashalytycs* - система обработки ошибок приложения, позволяет отлавливать все ошибки на устройствах пользователей. Этот инструмент по каждой неисправности формирует отчет, в котором содержится информация о версии, конфигурации программы, а также, при каких манипуляциях и даже при вызове какого метода пошел сбой. Этот инструмент позволяет распространять тестовые версии продукта между тестировщиками.

*Testflight* был выбран для тех же целей, что и Crashalytycs, но гораздо раньше. В процессе создания проекта Apple купила Testflight и интегрировала его в свой сервис публикации приложений itunesConnect.com. К сожалению, этот сервис не поддерживает iOS 7, и заменой сервису Testflight выступил Crashalytycs*.*

ЗАКЛЮЧЕНИЕ

В результате проделанной работы были изучены подходы для решения задачи определения местоположения внутри помещений. Был разработан прототип системы, позволяющей успешно выполнять навигацию внутри помещений.| Прототип удовлетворяет требованиям по точности определения местоположения.

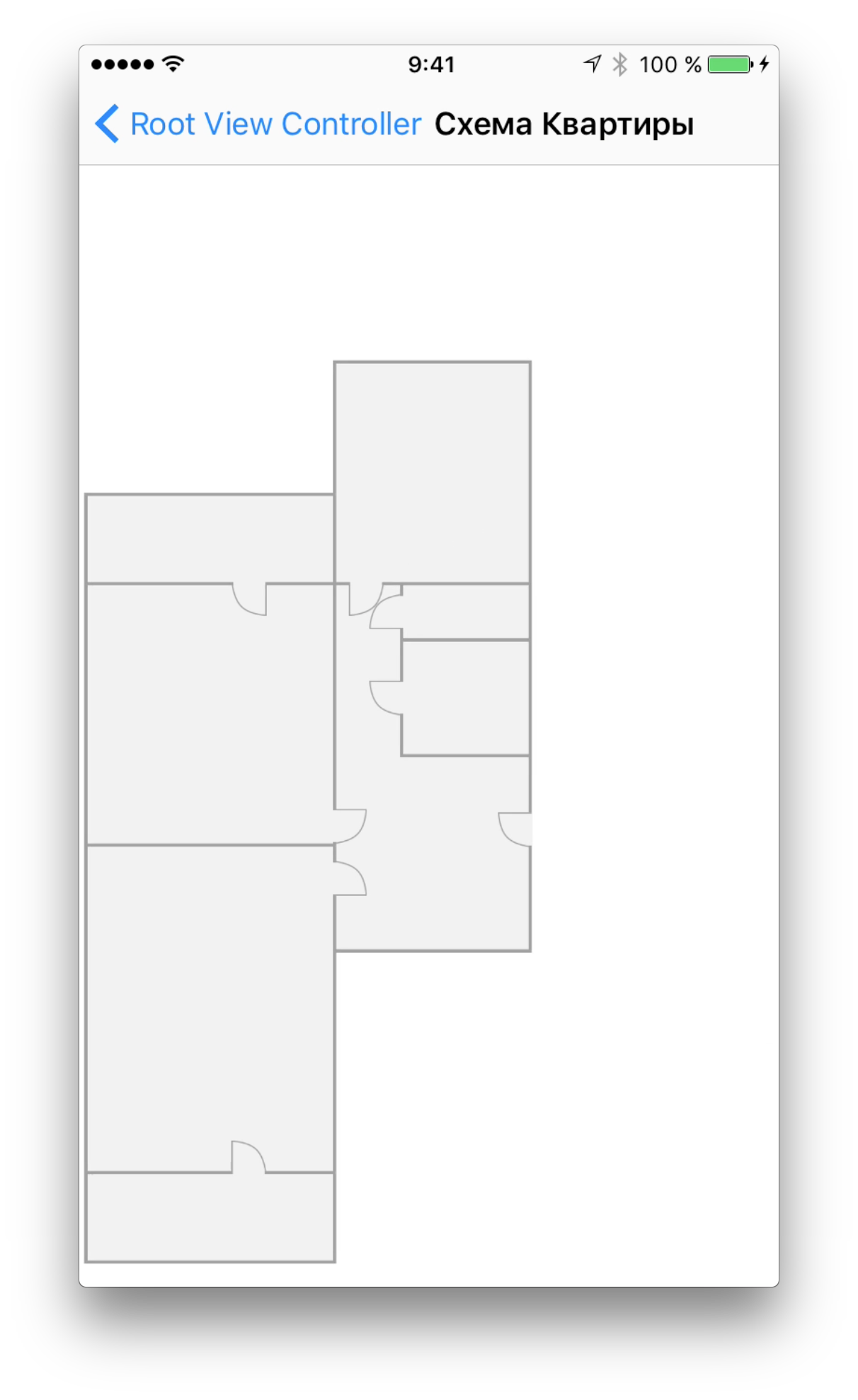
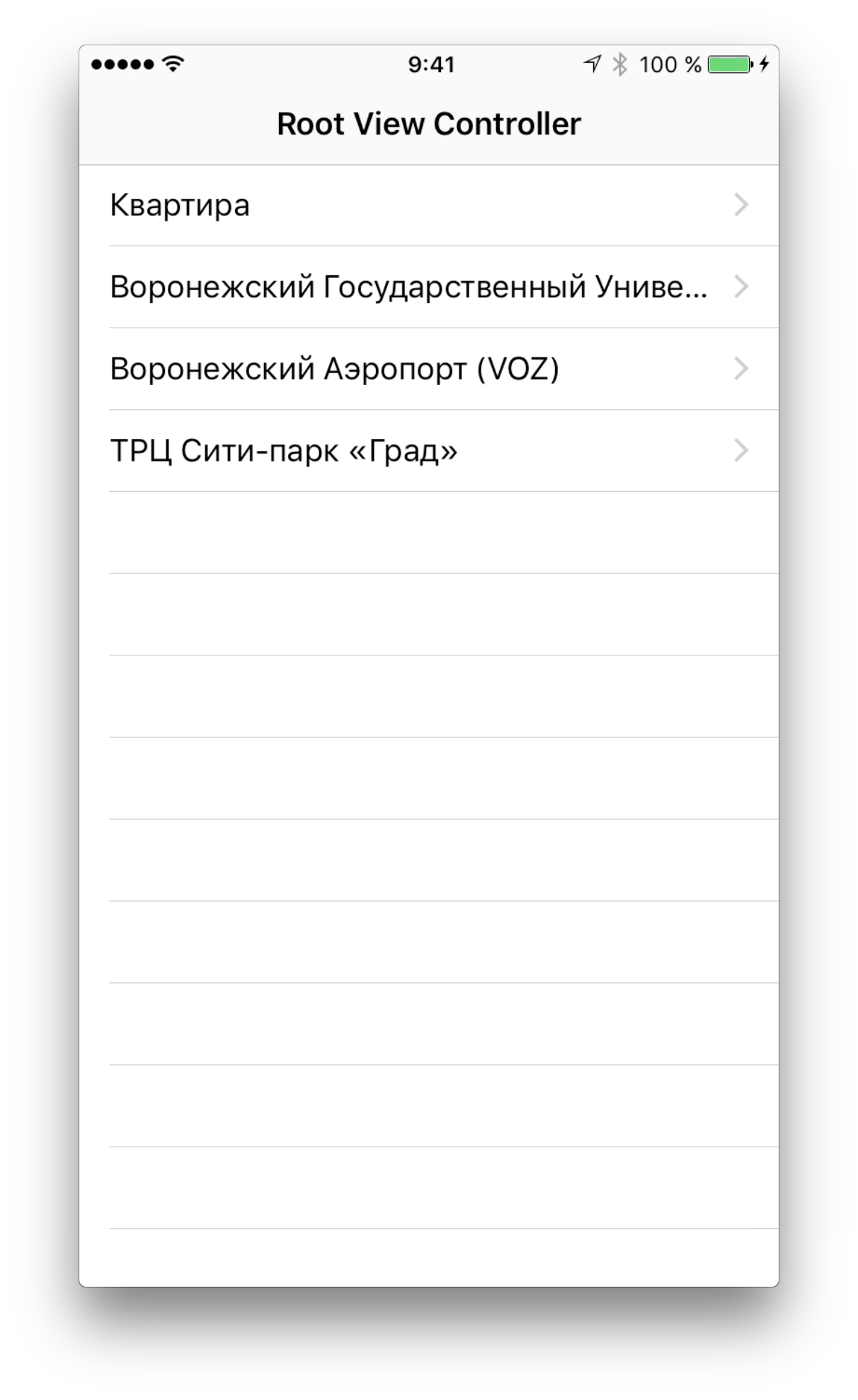
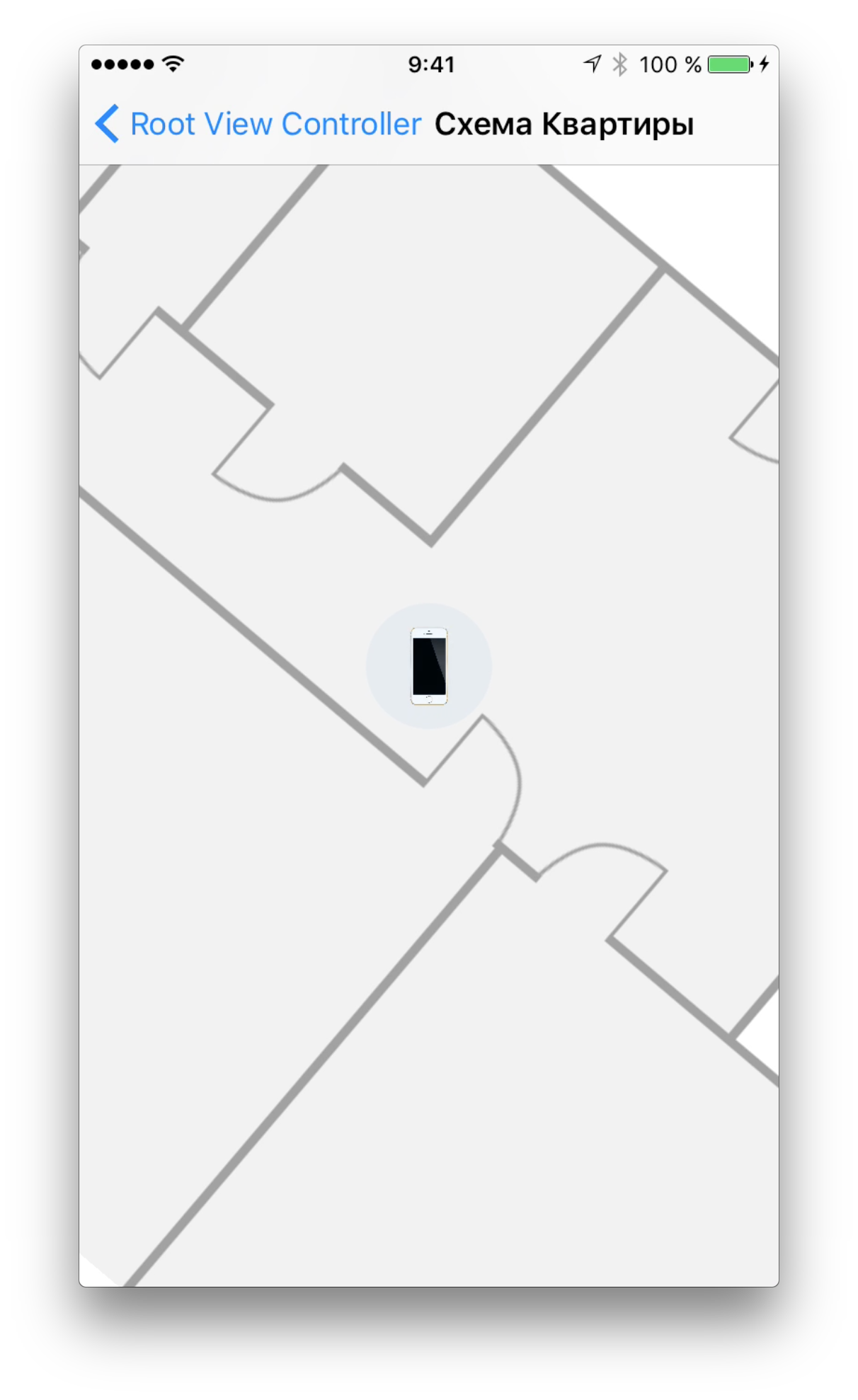
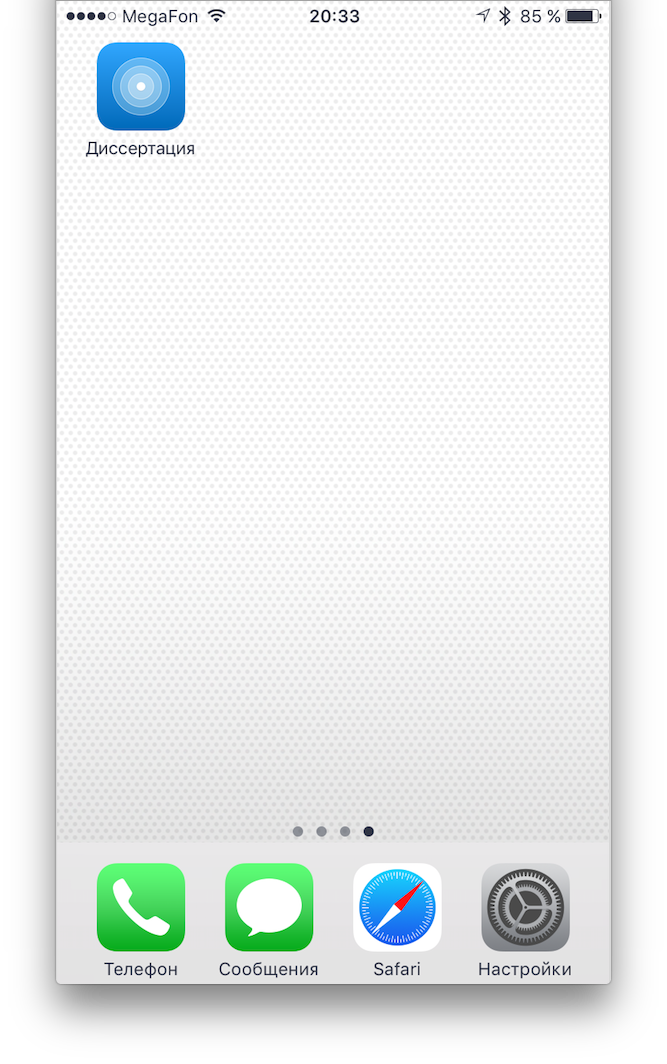
В ходе решения поставленых целей были решены следующие задачи:

* Рассмотрены существующие технологии, применяемые в навигации внутри помещений.
* Реализованы подходы, наиболее отвечающие решению поставленной задачи.
* Проведена оценка работоспособности технологий, выделены плюсы и минусы каждого
* Реализован алгоритм итеративной трилатерации
* Интегрирован фильтр Калмана с использованием встроенных сенсоров для оценки состояния системы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация Apple: [сайт] - (URL: <http://developer.apple.com/>) (дата обращения 15.09.2016).
2. Bing-Fei Wu, Cheng-Lung Jen, and Kuei-Chung Chang. Neural fuzzy based indoor localization by Kalman filtering with propagation channel modeling, 2007. - P 812-817.
3. Erin-Ee-Lin Lau and Wan-Young Chung. Enhanced RSSI-based real- time user location tracking system for indoor and outdoor environ- ments, 2007. - P 1213-1218.
4. Sebastian Thrun, Wolfram Burgard and Dieter Fox, Probabilistic Robotics, 2005. - P 12-34.
5. Zhe Chen, Bayesian filtering: From Kalman filters to particle filters, and beyond, 2003. – P 6-48.
6. Ioannis M. Rekleitis, A particle filter tutorial for mobile robot localization, 2004. – P 46-93.
7. J. Carpenter, P. Clifford, and P. Fernhead, An improved particle filter for non-linear  problems, 1997. – P 114-117.
8. Аналитические методы калмановской фильтрации для систем с априорной неопреде- ленностью / М. З. Згуровский, В. Н. Подладчиков. –1995. -С 230– 298.
9. Оценивание состояния динамической системы в условиях неопределенности / В. И. Ширяев, В. И. Дол- бенков, Е. Д. Ильин, Е. О. Подивилова. 2011. – C. 234–243.
10. Кац, И. Я. Минимаксная многошаговая фильтрация в статистически неопределенных ситуациях / И. Я. Кац, А. Б. Куржанскии. 1978. – No 11. – С. 79–87.
11. Климченко, В. В. Планирование измерений параметров контролируемых технических объектов / В. В. Климченко. 2011. – Т. 1. – С. 46–48.

ПРИЛОЖЕНИЕ 1. ЭКРАНЫ МОБИЛЬНОГО ПРИЛОЖЕНИЯ



ПРИЛОЖЕНИЕ 2. РЕАЛИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ C IBEACON

const struct WCBeaconRangerNotifications WCBeaconRangerNotifications = {

        .beaconsRanged = @"WCBeaconRanger=>BeaconsRanged"

};

1. @interface WCBeaconRanger()

@property (strong, nonatomic, readonly) CLLocationManager \*locationManager;

@end

@implementation WCBeaconRanger

{

    CLLocationManager \*\_locationManager;

    id<WCCSVDataGrabber> \_grabber;

}

* #pragma mark - Initialization

- (instancetype)initWithCSVGrabber:(id<WCCSVDataGrabber>)grabber

{

    if ((self = [super init])) {

        \_grabber = grabber;

    }

    return self;

}

* #pragma mark - Overridden getters/setters

- (CLLocationManager \*)locationManager

{

1. if (!\_locationManager) {

        \_locationManager = [CLLocationManager new];

1. \_locationManager.delegate = self;

    }

1. return \_locationManager;

}

* #pragma mark - Implementation of the protocol

- (void)startTrackingRegion:(CLBeaconRegion \*)region

{

    [self.locationManager startMonitoringForRegion:region];

}

- (void)stopTrackingRegion:(CLBeaconRegion \*)region

{

    [self.locationManager stopMonitoringForRegion:region];

}

* #pragma mark - CLLocationManager - region monitoring

- (void)locationManager:(CLLocationManager \*)manager didStartMonitoringForRegion:(CLRegion \*)region

{

    DDLogDebug(@"Monitoring has been started for region %@", region.identifier);

    [manager performSelector:@selector(requestStateForRegion:) withObject:region];

}

- (void)locationManager:(CLLocationManager \*)manager didDetermineState:(CLRegionState)state forRegion:(CLRegion \*)region

{

    if ([region isKindOfClass:[CLBeaconRegion class]]) {

        CLBeaconRegion \*beaconRegion = (CLBeaconRegion \*) region;

        DDLogInfo(@"App tracked a change for region %@ (UUID %@)", beaconRegion.identifier, beaconRegion.proximityUUID.UUIDString);

        switch (state) {

            case CLRegionStateInside:

                DDLogInfo(@"Starting ranging beacons");

                [manager startRangingBeaconsInRegion:beaconRegion];

                break;

            case CLRegionStateOutside:

                DDLogInfo(@"Stopping ranging beacons");

                [manager stopRangingBeaconsInRegion:beaconRegion];

                break;

            default:

                DDLogInfo(@"State is not determined");

                break;

        }

    }

}

- (void)locationManager:(CLLocationManager \*)manager didEnterRegion:(CLRegion \*)region

{

    if ([region isKindOfClass:[CLBeaconRegion class]]) {

        CLBeaconRegion \*beaconRegion = (CLBeaconRegion \*) region;

        DDLogInfo(@"App tracked an entry to region %@ (UUID %@)", beaconRegion.identifier, beaconRegion.proximityUUID.UUIDString);

    }

}

- (void)locationManager:(CLLocationManager \*)manager didExitRegion:(CLRegion \*)region

{

    if ([region isKindOfClass:[CLBeaconRegion class]]) {

        CLBeaconRegion \*beaconRegion = (CLBeaconRegion \*) region;

        DDLogInfo(@"App tracked an exit from region %@ (UUID %@)", beaconRegion.identifier, beaconRegion.proximityUUID.UUIDString);

    }

}

- (void)locationManager:(CLLocationManager \*)manager monitoringDidFailForRegion:(CLRegion \*)region withError:(NSError \*)error

{

    DDLogWarn(@"Location manager has failed to monitor region %@ due to an error: %@ (%@)", region.identifier, error.localizedDescription, error);

}

* #pragma mark - CLLocationManager - Beacon ranging

- (void)locationManager:(CLLocationManager \*)manager didRangeBeacons:(NSArray<CLBeacon \*> \*)beacons inRegion:(CLBeaconRegion \*)region

{

    DDLogDebug(@"Ranged %tu beacons from region %@", beacons.count, region.identifier);

    if (beacons.count > 0) {

        NSNumber \*timestamp = @([NSDate date].timeIntervalSince1970);

        for (CLBeacon \*beacon in beacons) {

            [\_grabber writeValues:@[

                    timestamp.stringValue,

                    [NSString stringWithFormat:@"%@.%@", beacon.major.stringValue, beacon.minor.stringValue],

                    @(beacon.proximity),

                    @(beacon.accuracy),

                    @(beacon.rssi)

            ]];

        }

1. [[NSNotificationCenter defaultCenter] postNotificationName:WCBeaconRangerNotifications.beaconsRanged object:self];

    }

}

- (void)locationManager:(CLLocationManager \*)manager rangingBeaconsDidFailForRegion:(CLBeaconRegion \*)region withError:(NSError \*)error

{

    DDLogWarn(@"Beacon monitoring in region %@ failed due to an error: %@, (%@)", region.identifier, error.localizedDescription, error);

}

* #pragma mark - CLLocationManager - General methods

- (void)locationManager:(CLLocationManager \*)manager didChangeAuthorizationStatus:(CLAuthorizationStatus)status

{

    DDLogInfo(@"Location manager's state has changed to %d", status);

    switch (status) {

        case kCLAuthorizationStatusNotDetermined:

            [manager requestAlwaysAuthorization];

            break;

        case kCLAuthorizationStatusAuthorizedAlways:

            DDLogDebug(@"Location services are already authorized for this app, continuing...");

            break;

        default:

            DDLogWarn(@"Unsuitable response received from location services. Go to Settings to authorize");

            break;

    }

}

- (void)locationManager:(CLLocationManager \*)manager didFailWithError:(NSError \*)error

{

    DDLogWarn(@"Location manager failed with an error: %@ (%@)", error.localizedDescription, error);

}

@end

ПРИЛОЖЕНИЕ 3. РЕАЛИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ C СЕНСОРАМИ

const struct WCSensorType WCSensorType = {

        .Accelerometer = @"Accelerometer",

        .Attitude = @"Attitude",

        .Compass = @"Compass",

        .DeviceAcceleration = @"DeviceAcceleration",

        .DeviceAccelerationStabilized = @"DeviceAccelerationStabilized",

        .GPS = @"GPS",

        .Gravity = @"Gravity",

        .GravityStabilized = @"GravityStabilized",

        .Gyroscope = @"Gyroscope",

        .MagneticField = @"MagneticField",

        .MagneticFieldStabilized = @"MagneticFieldStabilized",

        .Magnetometer = @"Magnetometer",

        .Motion = @"Motion",

        .Rotation = @"Rotation"

};

const struct WCSensorFusionNotifications WCSensorFusionNotifications = {

        .measurementCompleted = @"MeasurementCompleted"

};

1. #pragma mark - Additional declarations

@interface WCSensorFusion() <CLLocationManagerDelegate>

* @property (strong, nonatomic, readonly) id<WCCSVDataGrabber> dataGrabber;
* @property (strong, nonatomic, readonly) CLLocationManager \*locationManager;
* @property (strong, nonatomic, readonly) CMMotionManager \*motionManager;
* @property (strong, nonatomic, readonly) NSOperationQueue \*handlerQueue;
* @property (strong, nonatomic, readonly) NSTimer \*timer;
* @end

@implementation WCSensorFusion

{

    CLLocationManager \*\_locationManager;

    CMMotionManager \*\_motionManager;

    NSOperationQueue \*\_handlerQueue;

    NSTimer \*\_timer;

}

1. #pragma mark - Initializers

- (instancetype)initWithDataGrabber:(id<WCCSVDataGrabber>)dataGrabber

{

    if ((self = [super init])) {

        \_dataGrabber = dataGrabber;

    }

* return self;

}

1. #pragma mark - Overridden getters/setters

- (CLLocationManager \*)locationManager

{

    if (!\_locationManager) {

        \_locationManager = [CLLocationManager new];

        \_locationManager.delegate = self;

    }

    return \_locationManager;

}

- (CMMotionManager \*)motionManager

{

    if (!\_motionManager) {

        \_motionManager = [CMMotionManager new];

        \_motionManager.showsDeviceMovementDisplay = YES;

        NSTimeInterval defaultInterval = 1 / 60.0;

        \_motionManager.deviceMotionUpdateInterval = defaultInterval;

        \_motionManager.accelerometerUpdateInterval = defaultInterval;

        \_motionManager.gyroUpdateInterval = defaultInterval;

        \_motionManager.magnetometerUpdateInterval = defaultInterval;

    }

    return \_motionManager;

}

- (NSOperationQueue \*)handlerQueue

{

    if (!\_handlerQueue) {

        \_handlerQueue = [NSOperationQueue new];

        \_handlerQueue.maxConcurrentOperationCount = 2 \* [NSProcessInfo processInfo].processorCount;

    }

    return \_handlerQueue;

}

- (NSTimer \*)timer

{

    if (!\_timer) {

        \_timer = [NSTimer scheduledTimerWithTimeInterval:1.0

                                                  target:self

                                                selector:@selector(timerFired:)

                                                userInfo:nil

                                                 repeats:YES];

    }

    return \_timer;

}

1. #pragma mark - Logging methods

- (void)writeValues:(Vector3D)vector forSensor:(NSString \*)sensor

{

    NSNumber \*timestamp = @([NSDate date].timeIntervalSince1970);

    [\_dataGrabber writeValues:@[

            timestamp.stringValue,

            sensor,

            @(vector.x),

            @(vector.y),

            @(vector.z)

    ]];

}

1. #pragma mark - Timer handler

- (void)timerFired:(NSTimer \*)timer

{

    [[NSNotificationCenter defaultCenter] postNotificationName:WCSensorFusionNotifications.measurementCompleted

                                                        object:self];

}

1. #pragma mark - WCSensorFusion - implementation of protocol

- (void)startGatheringSensorData

{

    // Starting the queue

    self.handlerQueue.suspended = NO;

    // Starting location manager updates

    [self.locationManager startUpdatingLocation];

    [self.locationManager startUpdatingHeading];

    // Starting motion manager updates

* typeof(self) \_\_weak that = self;

    [self.motionManager startAccelerometerUpdatesToQueue:self.handlerQueue

                                             withHandler:^(CMAccelerometerData \*accelerometerData, NSError \*error) {

                                                 if (error != nil) {

                                                     DDLogWarn(@"Failed to receive accelerometer data due to an error: %@ (%@)", error.localizedDescription, error);

                                                 } else {

                                                     [that writeValues:FromAccelerometerData(accelerometerData) forSensor:WCSensorType.Accelerometer];

                                                 }

                                             }];

    [self.motionManager startGyroUpdatesToQueue:self.handlerQueue

                                    withHandler:^(CMGyroData \*gyroData, NSError \*error) {

                                        if (error != nil) {

                                            DDLogWarn(@"Failed to receive gyroscope data due to an error: %@ (%@)", error.localizedDescription, error);

                                        } else {

                                            [that writeValues:FromGyroscopeData(gyroData) forSensor:WCSensorType.Gyroscope];

                                        }

                                    }];

    [self.motionManager startMagnetometerUpdatesToQueue:self.handlerQueue

                                            withHandler:^(CMMagnetometerData \*magnetometerData, NSError \*error) {

                                                if (error != nil) {

                                                    DDLogWarn(@"Failed to receive magnetometer data due to an error: %@ (%@)", error.localizedDescription, error);

                                                } else {

                                                    [that writeValues:FromMagnetometerData(magnetometerData) forSensor:WCSensorType.Magnetometer];

                                                }

                                            }];

    [self.motionManager startDeviceMotionUpdatesUsingReferenceFrame:CMAttitudeReferenceFrameXTrueNorthZVertical

                                                            toQueue:self.handlerQueue

                                                        withHandler:^(CMDeviceMotion \*motion, NSError \*error) {

                                                            if (error != nil) {

                                                                DDLogWarn(@"Failed to receive accelerometer data due to an error: %@ (%@)", error.localizedDescription, error);

                                                            } else {

                                                                typeof(that) ref = that;

                                                                [ref writeValues:AccelerationFromMotion(motion) forSensor:WCSensorType.DeviceAcceleration];

                                                                [ref writeValues:StabilizedAccelerationFromMotion(motion) forSensor:WCSensorType.DeviceAccelerationStabilized];

                                                                [ref writeValues:GravityFromMotion(motion) forSensor:WCSensorType.Gravity];

                                                                [ref writeValues:StabilizedGravityFromMotion(motion) forSensor:WCSensorType.GravityStabilized];

                                                                [ref writeValues:MagneticFromMotion(motion) forSensor:WCSensorType.MagneticField];

                                                                [ref writeValues:StabilizedMagneticFromMotion(motion) forSensor:WCSensorType.MagneticFieldStabilized];

                                                                [ref writeValues:RotationFromMotion(motion) forSensor:WCSensorType.Rotation];

                                                                [ref writeValues:FromAttitude(motion.attitude) forSensor:WCSensorType.Attitude];

                                                            }

                                                        }];

    // Starting timer

    [self.timer fire];

}

- (void)stopGatheringSensorData

{

    // Stopping location manager

    [self.locationManager stopUpdatingLocation];

    [self.locationManager stopUpdatingHeading];

    // Stopping motion manager

    [self.motionManager stopAccelerometerUpdates];

    [self.motionManager stopGyroUpdates];

    [self.motionManager stopMagnetometerUpdates];

    [self.motionManager stopDeviceMotionUpdates];

    // Cancelling the rest of operations in operation queue

    [self.handlerQueue cancelAllOperations];

    self.handlerQueue.suspended = YES;

    // Timer invalidation

    [self.timer invalidate];

    \_timer = nil;

}

1. #pragma mark - CLLocationManagerDelegate implementation - Access

- (BOOL)locationManagerShouldDisplayHeadingCalibration:(CLLocationManager \*)manager

{

* return YES;

}

- (void)locationManager:(CLLocationManager \*)manager didChangeAuthorizationStatus:(CLAuthorizationStatus)status

{

    DDLogInfo(@"Location manager's state has changed to %d", status);

    switch (status) {

        case kCLAuthorizationStatusNotDetermined:

            [manager requestAlwaysAuthorization];

            break;

        case kCLAuthorizationStatusAuthorizedAlways:

            DDLogDebug(@"Location services are already authorized for this app, continuing...");

            break;

        default:

            DDLogWarn(@"Unsuitable response received from location services. Go to Settings to authorize");

            break;

    }

}

1. #pragma mark - CLLocationManagerDelegate implementation - Heading

- (void)locationManager:(CLLocationManager \*)manager didUpdateHeading:(CLHeading \*)newHeading

{

    [self writeValues:FromHeading(newHeading) forSensor:WCSensorType.Compass];

}

1. #pragma mark - CLLocationManagerDelegate implementation - Location

- (void)locationManager:(CLLocationManager \*)manager didUpdateLocations:(NSArray<CLLocation \*> \*)locations

{

    [self writeValues:FromLocation(locations.lastObject) forSensor:WCSensorType.GPS];

}

1. #pragma mark - CLLocationManagerDelegate implementation - Error handling

- (void)locationManager:(CLLocationManager \*)manager didFailWithError:(NSError \*)error

{

    DDLogWarn(@"Location manager failed with an error: %@ (%@)", error.localizedDescription, error);

}

* @end