

Документация по проекту
AR Navigation

Сопрачёв Андрей

Версия: 0.2.1
19 марта 2020 г.

Содержание

| | | |
|-----------|--|----------|
| I | Основное описание проекта | 2 |
| 1 | Система Event | 2 |
| II | Деление на модули | 2 |
| 2 | Вступление | 3 |
| 3 | Независимые модули | 3 |
| 3.1 | AR Unit | 3 |
| 3.1.1 | Некоторые примитивы | 4 |
| 3.1.2 | ARInterface | 4 |
| 3.1.3 | Эмуляция AR сессии | 5 |
| 3.1.3.1 | AREventRecorder | 5 |
| 3.1.4 | Алгоритм создания AR сцены в Unity | 5 |
| 3.1.5 | Пример AR сцены | 6 |
| 3.2 | GPS Unit | 6 |
| 3.2.1 | GPSInterface | 6 |
| 3.2.2 | Алгоритм создания GPS сцены в Unity | 6 |
| 3.2.3 | Пример GPS сцены | 7 |
| 3.3 | POS Unit | 7 |
| 3.3.1 | Задача позиционирования | 7 |
| 3.3.2 | Вычисление вектора линейного сдвига | 8 |
| 3.3.3 | Вычисление угла между системами | 9 |
| 3.3.4 | Решение задачи позиционирования с учётом погрешности | 9 |
| 3.3.5 | Вычисление вектора линейного сдвига с учётом погрешности | 10 |
| 3.3.6 | Вычисление угла между системами с учётом погрешности | 11 |
| 3.3.7 | Общий алгоритм позиционирования | 11 |
| 4 | Надстройки | 12 |
| 4.1 | Pos + AR subUnit | 12 |
| 4.1.1 | ARMapTool | 12 |
| 4.1.2 | Использование | 12 |
| 4.1.2.1 | Создание ARMapScriptable | 12 |
| 4.2 | Pos + GPS + AR subUnit | 13 |
| 4.2.1 | GPSMapTool | 13 |
| 4.2.2 | Использование | 13 |
| 4.2.2.1 | Создание GPSMapScriptable | 14 |

Часть I

Основное описание проекта

AR Navigation проект демонстрирует возможности бесшовной навигации в дополненной реальности. Проект создан в движке Unity3D.

Общую систему можно разделить на несколько подзадач:

- Интерфейс
- Поиск пути
- Позиционирование в пространстве

Интерфейс и поиск пути являются достаточно тривиальными задачами, по этому остановимся на рассмотрении алгоритма позиционирования.

1 Система Event

Вся внутренняя система разделена на модули, которые общаются между друг другом с помощью Event'ов.

Создаются *static* классы в которых определены делегаты, инвенты и функции их вызывающие.

```
static class Interface {
    public delegate void OnAction(Param param);
    public static event OnAction onAction;

    public static void DoAction(Param param) {
        onAction.Invoke(param);
    }
}
```

После чего из любого другого скрипта можно подписываться на события и вызывать их.

```
class Subscriber {
    void Start() {
        Interface.onAction += OnAction;
    }

    void OnAction(Param param) {
        //some code
    }
}

class Foo {
    Interface.OnAction(new Param());
}
```

Такой подход позволяет очень удобно отслеживать процессы происходящие в системе, а главное — записывать их, и в последствие многократно воспроизводить.

Часть II

Деление на модули

2 Вступление

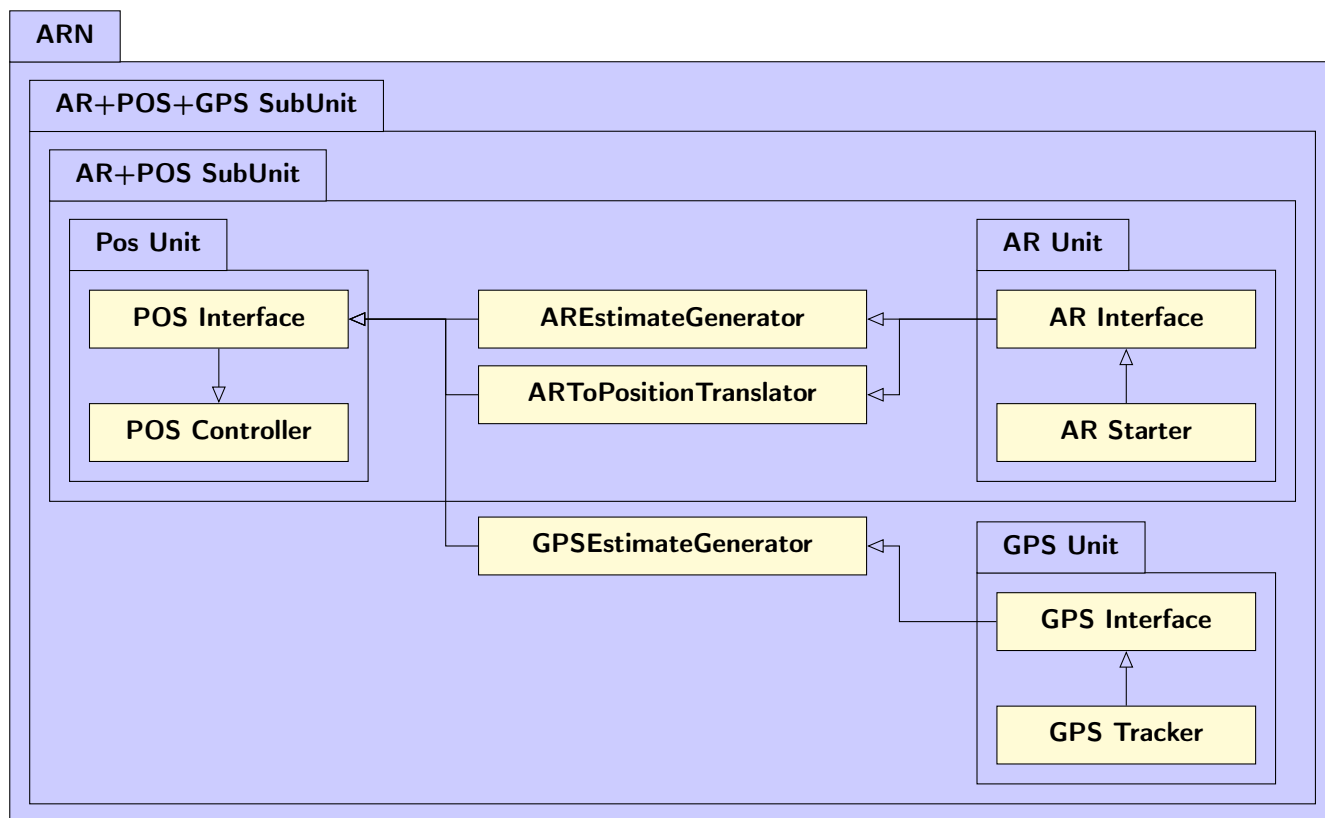
Работа приложения разделена на несколько функциональных модулей, некоторые из которых являются независимыми, а остальные – надстройками.

Независимые:

- AR Unit — модуль дополненной реальности, прослойка между нативными плагинами и общей системы
- GPS Unit — модуль позиционирования по GPS
- Pos Unit — модуль решающий задачу перевода координат из локальной в глобальную системы координат

Надстройки:

- Pos + AR SubUnit — надстройка над **Pos Unit** и **AR Unit** для проброски ивентов между ними
- Pos + GPS + AR SubUnit — надстройка над **Pos + AR SubUnit** и **GPS Unit** для проброски ивентов между ними



3 Независимые модули

3.1 AR Unit

Задача AR Unit — предоставить приложению уровень абстракции над ARKit и ARCore плагинами Unity.

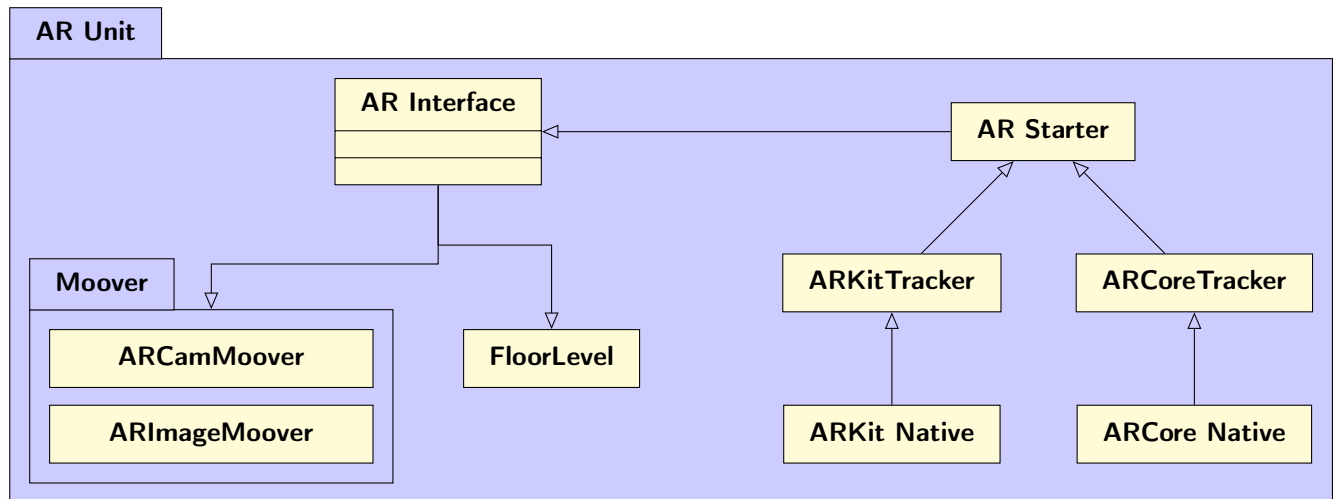
Является независимым модулем.

Модуль реализован на системе Events и предоставляет в использование **ARInterface**

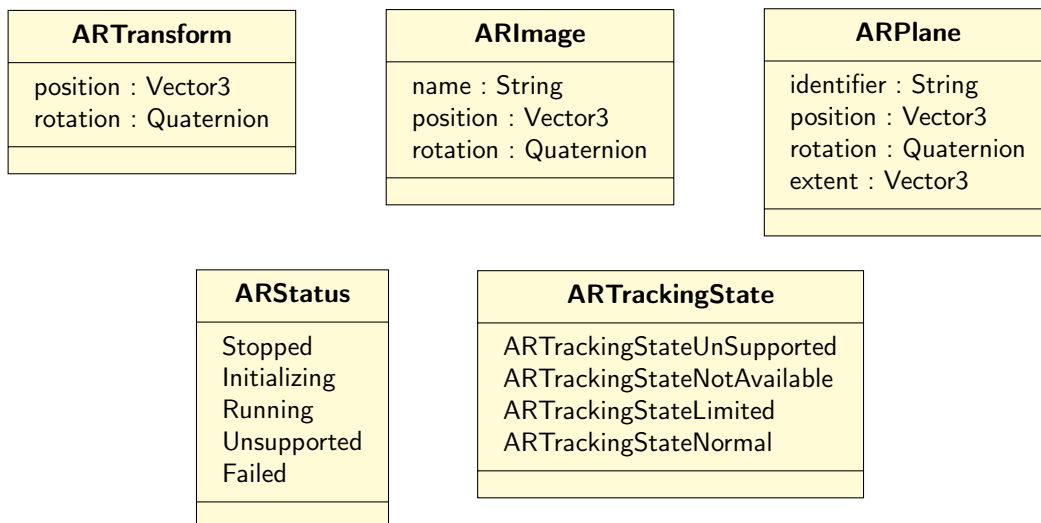
Для работы модуля на сцене необходим префаб **ARStarter** с дочерними **ARKitTracker** и **ARCoreTracker**. При запуске сцены **ARStarter** активирует "Tracker" соответствующий текущей платформе (ios — ARKitTracker, android — ARCoreTracker).

Tracker выполняет функцию проброски api между нативным Unity Plugin'ом и ARInterface.

На объекте Tracker выполняются настройки для запуска AR сессии конкретной платформы.



3.1.1 Некоторые примитивы



3.1.2 ARInterface

Разделён на две логические части — делегаты состояний и функции их вызывающие.

Делегаты:

- OnARTransformUpdate(ARTransform) — обновление координаты устройства в пространстве
- OnARCameraProjectionMatrixUpdate(Matrix4x4) — обновление параметров камеры (fov etc)
- OnImageAdd(ARImage) — первое появление AR метки в сцене
- OnImageUpdate(ARImage) — обновление положения существующей AR метки
- OnImageRemoved(ARImage) — удаление AR метки со сцены (≠ выход за пределы экрана, обычно вызывается в ARKit при остановке сцены)

- OnPlaneAdd(ARPlane) — первое появление ARPlane в сцене
- OnPlaneUpdate(ARPlane) — обновление положения существующей ARPlane
- OnPlaneRemoved(ARPlane) — обновление положения существующей ARPlane
- OnStatusChange(ARStatus) — изменение статуса AR сцены
- OnTrackingStateChange(ARTrackingState) — изменение статуса позиционирования
- OnTrackingStateReasonChange(ARTrackingStateReason) — информация о текущем статусе позиционирования (например: недостаточно освещения)
- OnStartSession() — запуск сессии
- OnRestartSession() — перезапуск сессии на лету
- OnStopSession() — остановка сессии
- OnSessionFailed() — критическая ошибка в сессии приводящая к её остановке (например: запрещён доступ к камере)
- OnChangePlaneMode(bool) — изменение состояния трекинга плоскостей

3.1.3 Эмуляция AR сессии

AR Unit предоставляет объекты для полной эмуляции всех событий AR сессии.

Все необходимые файлы находятся в Assets/Units/ARUnit/Fake все дальнейшие пути указаны относительно этой директории.

Для полной эмуляции перетащить в сцену Prefabs/FAKE_AR. Его дочерние объекты определяют поведение симуляции.

- FakeARMain — отвечает за эмуляцию статусов сессии
- Camera position AR generator — отвечает за эмуляцию положения камеры в пространстве
- FakeImage — отвечает за эмуляцию трекинга картинки

3.1.3.1 AREventRecorder Объект позволяющий записать и сохранить в файл все события происходящие во время AR сессии, а после этот файл воспроизводить. Для использования добавить на сцену Prefabs/SessionRecorder.

3.1.4 Алгоритм создания AR сцены в Unity

Все необходимые файлы находятся в Assets/Units/ARUnit все дальнейшие пути указаны относительно этой директории.

1. Создание сцены:

- Перетащить на сцену префаб /Prefabs/ARUnit
- При необходимости отключить объект ARFloorCalculate отвечающий за расчёт уровня пола
- На основную камеру добавить скрипт ARCamMoover и указать эту камеру в настройках ARKitTracker и ARCoreTracker

2. Настройка:

- IOS
 - Создать в проекте UnityARKitPlugin/ARReferensImagesSet и перетащить его на ARKitTracker в соответствующие поле

- ii. Создать в проекте UnityARKitPlugin/ARReferensImage для каждой желаемой метки, и указать ей текстуру и физический размер (ширину).
Заполнить ими созданный ReferensImagesSet.
 - iii. На объект трекинга добавить скрипт ARImageMover и в его имя указать имя метки
- (b) Android

- i. Создать в проекте GoogleARCore/SessionConfig и перетащить его на ARCoreTracker в соответствующие поле
- ii. В проекте выделить необходимые метки и создать GoogleARCore/AugmentedDataBase. Перетащить получившийся объект на созданный SessionConfig.
- iii. На объект трекинга добавить скрипт ARImageMover и в его имя указать имя метки

3. Запуск

Вызвать функцию ARInterface.StartARSession() из UI или другого скрипта. После инициализации ARInterface.ARStatus перейдёт в состояние Running и сессия будет успешно запущена.

4. Остановка

Для остановки сессии вызвать функцию ARInterface.StopARSession()

3.1.5 Пример AR сцены

Пример сцены расположен в Assets/Units/ARUnit/Example/FullARUnitExample в нём реализованы все возможности ARUnit

3.2 GPS Unit

Задача GPS Unit предоставить уровень абстракции над location servise.

Модуль реализован на системе Events и предоставляет в использование GPSInterface.

Для работы модуля на сцене необходим префаб GPSTracker.

3.2.1 GPSInterface

Разделён на две логические части — делегаты состояний и функции их вызывающие.

Делегаты:

- OnStartGPS(desiredAccuracyInMeters, updateDistanceInMeters) — запусе GPS трекинга с заданными параметрами погрешности
- OnStopGPS() — остановка GPS трекинга
- OnGPSStatusUpdate(GPSServiceStatus) — событие обновления GPS статуса
- OnGPSUpdate(GPSInfo) — событие обновления координаты
- OnStartCompass() — запусе компаса
- OnStopCompass() — остановка компаса
- OnGPSCompassUpdate(GPSCompassInfo) — событие обновления азимута

3.2.2 Алгоритм создания GPS сцены в Unity

Все необходимые файлы находятся в Assets/Units/GPSUnit все дальнейшие пути указаны относительно этой директории.

1. Создание сцены

Перетащить на сцену префаб /Prefabs/GPSTracker

2. Запуск

Вызвать функцию `GPSInterface.StartGPS()` для отслеживания позиционирования и `GPSInterface.OnStartCom` для отслеживания азимута

3. Отслеживание

Подписаться на события `GPSInterface.OnGPSUpdate` и `GPSInterface.OnGPSCompassUpdate`

4. Выключение

Вызвать функцию `GPSInterface.StopGPS()` и `GPSInterface.StopCompass()`

3.2.3 Пример GPS сцены

Пример сцены расположен в `Assets/Units/GPSUnit/Example/FullGPSUnitExample` в нём реализованы все возможности `GPSUnit`

3.3 POS Unit

3.3.1 Задача позиционирования

Зная координату P относительно запуска приложения, вычислить P' относительно известной системы координат.

ARUnit в любой момент времени позволяет получить координату в системе координат связанной с точкой запуска трекинга. При этом начало координат O этой системы находится в точке запуска, ось Y направлена против гравитации, ось Z — по проекции нормали экрана устройства на плоскость перпендикулярную Y в момент запуска трекинга. Назовём эту систему — **локальная система координат**.

Заданная система координат — связана с физической картой, где центр O' определяется разработчиком, ось Y' по нормали к карте, оси $North$ и $East$ направлены по соответствующим сторонам света. Назовём её **глобальная система координат**.

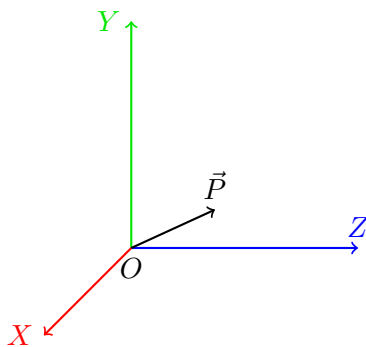


Рис. 1: Локальная система координат

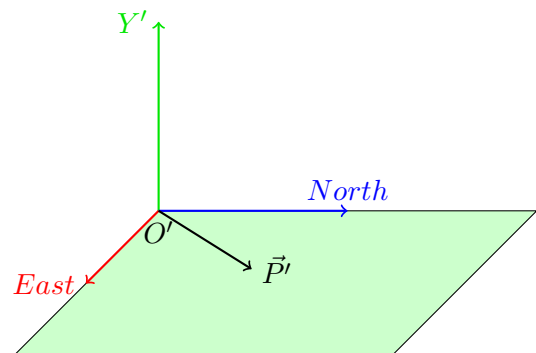


Рис. 2: Глобальная система координат

Соноправленность осей Y и Y' позволяет свести задачу к двумерному случаю.

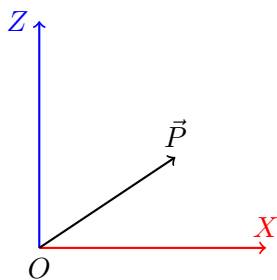


Рис. 3: Локальная система координат 2D

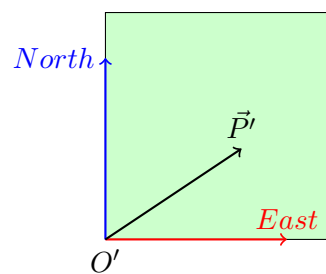


Рис. 4: Глобальная система координат 2D

Переобозначим оси: $North \rightarrow Z'$ и $East \rightarrow X'$. И разделим задачу на 2 части:

1. Ось Z сонаправленна с Z' , но центр O не совпадает с O' , задача найти вектор смещения \vec{d}
2. Центр O совпадает с O' , но направление Z не совпадает с Z' , задача – найти угол α между этими осями

Для однозначного решения этой задачи исходных данных недостаточно. Необходимо знать координаты одной и той же точки в обеих системах координат. Введём объект **Estimate**

| Estimate |
|--|
| localPos : Vector3 globalPos : Vector3 correctAngle : float angleAcc : float posAcc: float |
| |

- localPos — координата точки в локальной системе координат
- globalPos — координата этой же точки в глобальной системе координат
- correctAngle — предполагаемый угол коррекции между для приведения одной системы координат к другой (в градусах)
- angleAcc — погрешность угла коррекции (в градусах)
- posAcc — погрешность определения соответствующих точек

Получить **Estimate** можно разными путями, например перевести показания GPS в глобальную систему координат и записать в **globalPos**, а текущую координату ARUnit записать в **localPos**.

3.3.2 Вычисление вектора линейного сдвига

Пусть у нас есть один Estimate, обозначим его *localPos* точкой E , а *globalPos* точкой E' . И решим первую задачу пренебрегая погрешностями измерения. При этом по условию задачи оси систем координат сонаправлены $Z \uparrow Z'$ и $X \uparrow X'$, а центры различны $O \neq O'$.

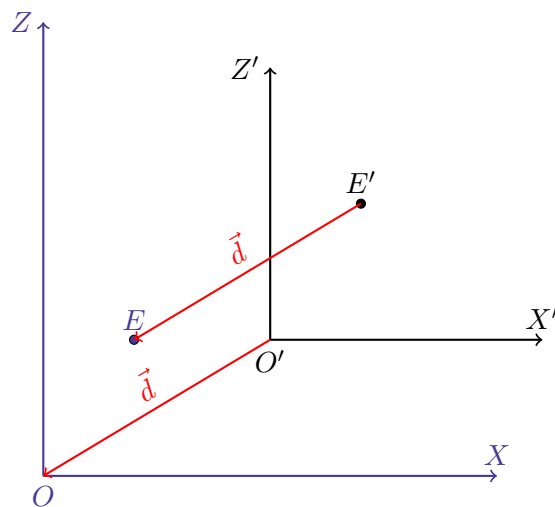


Рис. 5: Определение вектора линейного сдвига \vec{d}

Вычитая из координат точки E , координаты E' получим вектор \vec{d} который и будет искомым сдвигом между O и O' .

3.3.3 Вычисление угла между системами

Пусть у нас есть один Estimate, обозначим его *localPos* точкой E , а *globalPos* точкой E' . И решим **вторую** задачу пренебрегая погрешностями измерения. При этом по условию задачи центры систем координат совпадают $O = O'$, а оси — нет.

То есть одна система повернута относительно другой на угол α его и надо найти.

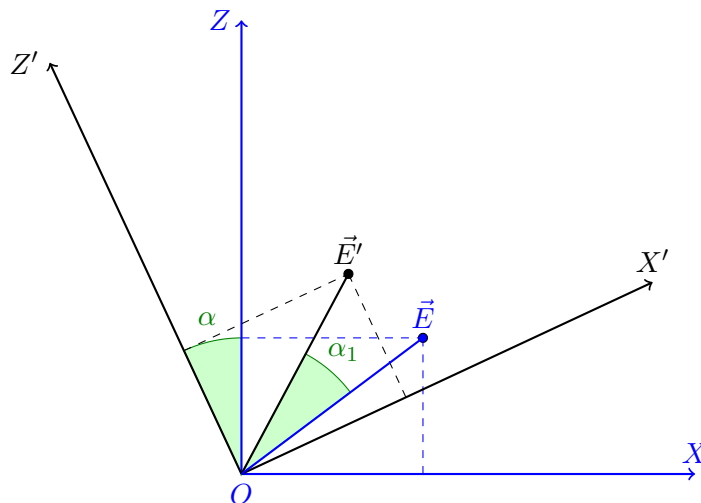


Рис. 6: Определение угла поворота α

Построим вектора \vec{OE} и $\vec{OE'}$, и угол $(\vec{OE}, \vec{OE'}) = \alpha_1 = \alpha$ — искомый угол.

3.3.4 Решение задачи позиционирования с учётом погрешности

Погрешность может возникнуть как в исходных данных, так и накопиться со временем.

- Погрешность исходных данных появляется при неточности соответствия точки в локальных и глобальных координатах, например в случае с GPS, точной координате *localPos*, соответствует координата GPS с погрешностью, то есть *localPos* может быть удалена от координаты GPS на некую величину погрешности ϵ

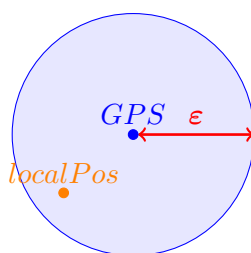


Рис. 7: Погрешность исходных данных

- Накопительная погрешность. Для определения координаты в пространстве ARUnit интегрирует показания акселерометра в этом расчёте появляется погрешность зависящая от пройденного расстояния и пользовательского устройства. Например на iPhone10 такая погрешность составляет 5 см на 1 метр, что означает, что с каждым пройденным метром координата полученная из ARUnit "ошибается" на 5 см.

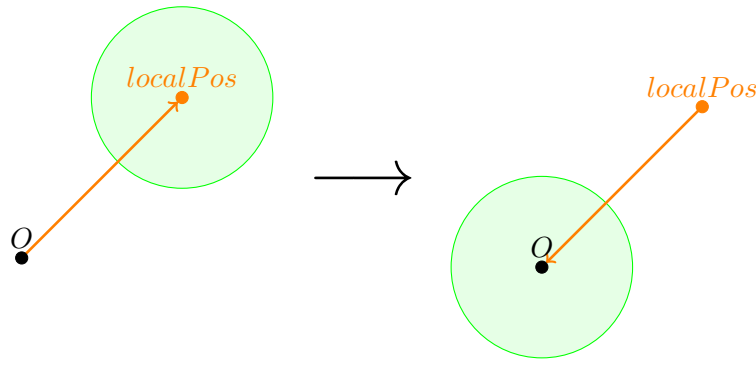


Рис. 8: Накопительная погрешность. Относительный переход

Благодаря относительности систем координат, увеличение погрешности *localPos* относительно центра можно интерпретировать как увеличение погрешности центра относительно координаты. А значит можно свести обе погрешности к одному. Для этого следует увеличивать погрешность всех ранее полученных *Estimate*'ов с изменением *localPos*

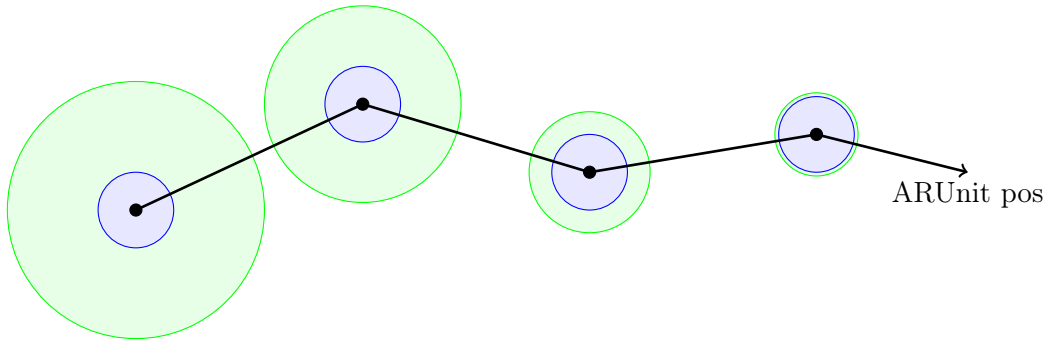


Рис. 9: Увеличение погрешности с изменением *localPos*

За эту суммарную погрешность отвечает поле *posAcc* объекта *Estimate*.

3.3.5 Вычисление вектора линейного сдвига с учётом погрешности

Вектор \vec{d} — разница между *localPos* и *globalPos* отдельно взятого *Estimate*'а, назовём этот *estimate* — *pivot*.

На вход поступает поток *Estimate*'ов, а значит для достижения максимальной точности достаточно выбрать *Estimate* с минимальной горизонтальной погрешностью (*posAcc*). Однако, при приближении к пороговому показателю точности *Estimate* (например для GPS — минимально возможная погрешность в текущих условиях) и пересчёте \vec{d} по минимальной погрешности, в позиционирование могут возникать скачки, по этому в целях стабильности целесообразно, при малых погрешностях не производить поиск нового *pivot*, пока *localPos* последнего *Estimate* находится в радиусе погрешности его *globalPos*.

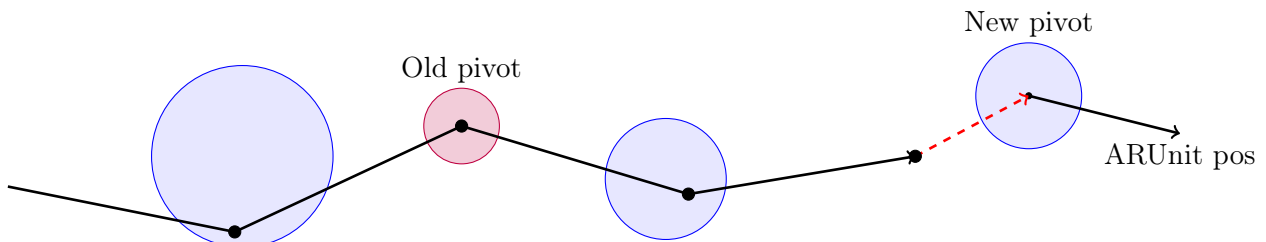


Рис. 10: Определение *pivot* по потоку *Estimate*

3.3.6 Вычисление угла между системами с учётом погрешности

Угол α между системами вычисляется по координатам двум Estimate'ам. Однако эти координаты определены с известной погрешностью, а значит можно определить и погрешность $\Delta\alpha$.

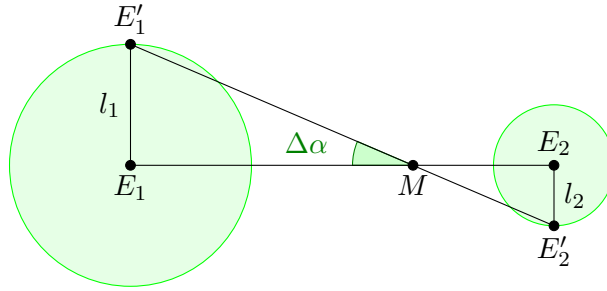


Рис. 11: Определение погрешности $\Delta\alpha$

Имеем два Estimate'а с центрами в точках E_1 и E_2 и погрешностями l_1 и l_2 соответственно. Рассмотрим худший вариант, когда погрешность максимальна, и реальная позиция максимально отдалены от её измерения и находятся в точках E'_1 и E'_2 . Такое диаметрально противоположное расположение даёт максимальный угол отклонения $E'_1E'_2$ от E_1E_2 , а значит и максимальную погрешность при расчёте α .

$$\begin{aligned} E_1E_2 \cap E'_1E'_2 &= M \\ \angle E'_1ME_1 &= \Delta\alpha \\ \tan \Delta\alpha &= \frac{l_1 + l_2}{2|E_1E_2|} \\ \Delta\alpha &= \arctan \frac{l_1 + l_2}{2|E_1E_2|} \end{aligned}$$

Далее мы имеем два возможных варианта:

- Перебор всех возможных пар Estimate'ов и определение пары с минимальной погрешностью $\Delta\alpha$ и расчёт угла α по этой паре
- Перебор всех возможных пар Estimate'ов, и расчёт угла α :
 - по среднему значению между n лучших пар
 - по среднему значению между всеми парами с весом $p = f(\Delta\alpha)$. Например $p = \frac{1}{\Delta\alpha}$

Практические эксперименты показали, что оптимальным является определение α по одной лучшей паре.

3.3.7 Общий алгоритм позиционирования

Опираясь на вышеперечисленные пункты можно сформулировать общий алгоритм реализованный в POS Controller.

Запуск алгоритма производится при каждом получении нового Estimate.

1. Получаем новый Estimate и сохраняем его в массив ранее пришедших Estimate'ов
2. При необходимости обновляем pivot
 - по минимальной погрешности, если погрешность велика
 - по выходу за пределы новой погрешности, если погрешность мала
3. При необходимости обновляем угол между системами

- по *Estimate.correctAngle*, если *Estimate.angleAcc* меньше погрешности всех пар и текущей погрешности угла системы
- по углу α , если $\Delta\alpha$ меньше *Estimate.angleAcc* и текущей погрешности угла системы

А так же после каждого получения новой координаты от ARUnit увеличиваем погрешность всех ранее пришедших Estimate на расстояние пройденное с прошлого измерения умноженное на погрешность на метр $Estimate.posAcc = Estimate.posAcc + Distance(lastPos, newPos) * accByMeter$

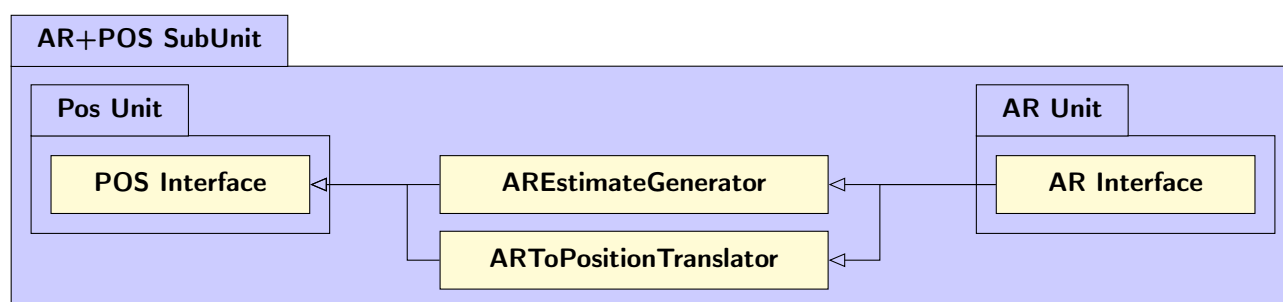
4 Надстройки

4.1 Pos + AR subUnit

Задача надстройки — пробросить события между ARUnit и PosUnit.

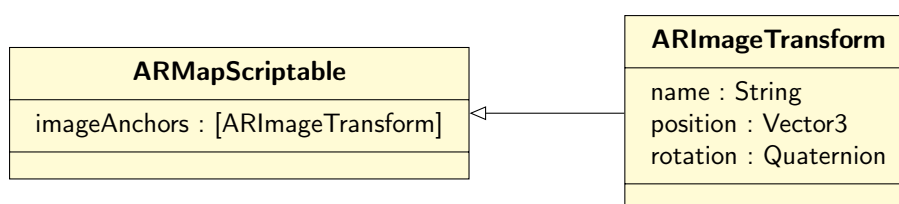
Состоит из двух объектов:

- AREstimateGenerator — отправляет событие OnEstimateAdd при обнаружение метки
- ARToPositionTranslator — отправляет событие обновления координаты устройства в локальной системе координат



4.1.1 ARMapTool

Для корректной работы AREstimateGenerator необходимо знать координаты меток в глобальной системе координат, для этого используется ARMapScriptable настраиваемый с помощью ARMapTool.



4.1.2 Использование

Для работы проброски ивентов перетащить префабы AREstimateGenerator и ARToPositionTranslator на сцену.

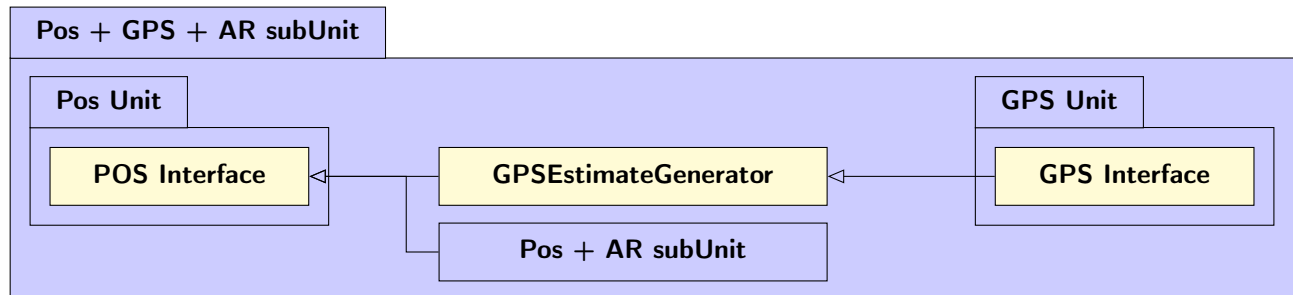
4.1.2.1 Создание ARMapScriptable

1. Создать в проекте ARMapScriptable.
2. Создать на сцене GameObject и добавить на него компонент ARMapTool
3. Указать в TargetScriptable объект созданный в первом пункте
4. Перетащить нужное количество префабов A4 Target на сцену дочерним к ARMapTool

5. Назначить им названия и координаты на сцене
6. На объекте ARMapTool нажать кнопку **Set Anchors**

4.2 Pos + GPS + AR subUnit

Задача надстройки — пробросить события между **GPSUnit** и **PosUnit** для этого используется **AREstimateGenerator**



4.2.1 GPSMapTool

Для корректной работы **GPSEstimateGenerator** необходимо знать связку между глобальной и геодезической системами координат, для этого используется **GPSMapScriptable** настраиваемый с помощью **GPSMapTool**.

| GPSMapScriptable |
|--|
| latitude : float longitude : float altitude : float localPos : Vector3 width : float height : float filter : Texture2D |

- latitude — широта
- longitude — долгота
- altitude — высота
- localPos — координата в локальной системе
- width — ширина карты в метрах
- height — высота карты в метрах
- filter — фильтр погрешности GPS, картинка в красный канал которой, записан коэффициент умножения текущей погрешности

4.2.2 Использование

Для работы проброски ивентов перетащить префаб **GPSEstimateGenerator** на сцену.

4.2.2.1 Создание GPSMapScriptable

1. Создать в проекте GPSMapScriptable.
2. Перетащить на сцену префаб GPSMapTool
3. Изменить спрайт карты на свой
4. Указать в TargetScriptable объект созданный в первом пункте
5. Указать в инспекторе координаты PivotMain и PivotScale
6. Объекты PivotMain и PivotScale установить на сцене в нужные координаты по спрайту карты
7. Нажать кнопку **Set size** для масштабирования карты и приведения её глобального размера к геодезическому
8. Нажать кнопку **Set map** для записи в GPSMapScriptable