Лицей «Физико-техническая школа» им. Ж. И. Алфёрова

Санкт-Петербургского Академического университета

Курсовая работа (отчет о практике)

Разработка системы коррекции позиции участника соревнований по спортивному ориентированию с помощью фильтра Калмана на Kotlin

Работу выполнили:

ученик 11 класса Ипатов Марк,

ученик 11 класса Ковригин Александр

Научный руководитель:

Сенин Михаил Андреевич

Место прохождения практики:

ООО "Спортивные трекинговые системы"

Аннотация

Многие виды спорта (например, парусный спорт, мультигонки, различные марафоны и спортивное ориентирование) не позволяют зрителям постоянно наблюдать за спортсменами. Для повышения зрелищности и контроля используются трекеры, данные с которых показываются зрителям в режиме реального времени. Однако при проведении таких онлайн-трансляций организаторы сталкиваются с проблемой неточного отображения позиции спортсмена. Из-за этого зрителям может быть сложно следить за динамикой мероприятия.

Цель нашей работы – создать систему коррекции, которая будет правильно отображать местоположение спортсмена, используя специфичную модель для определенного вида спорта. Наша работа посвящена методам коррекции для спортивного ориентирования.

Для достижения цели мы реализовали и сравнили несколько методов коррекции: наивное притягивание спортсмена к дороге, фильтр Калмана[1], гибридные варианты.

В итоге один из наших гибридных вариантов оказался точнее наивного на 36%. Впрочем, возможно дальнейшее улучшение этого алгоритма коррекции.

[1] R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]," in IEEE Signal Processing Magazine, vol. 29, no. 5, pp. 128-132, Sept. 2012, doi: 10.1109/MSP.2012.2203621.

Содержание

Введение

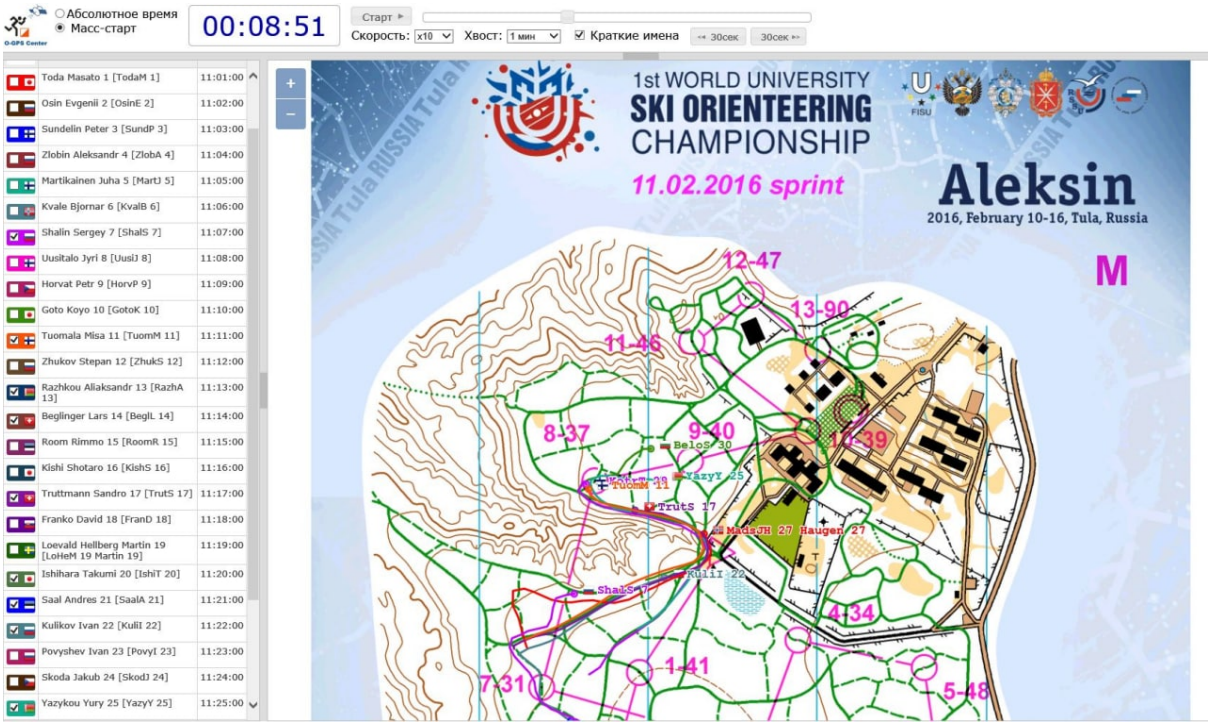
При проведении спортивных соревнований организаторы зачастую прибегают к использованию средств онлайн-трансляции мероприятия. Это выигрышно для соревнования, так как больше зрителей смогут наблюдать за его проведением. К тому же в некоторых видах спорта (например, в таких как спортивное ориентирование, трейлы, рогейны, парусный спорт, мультигонки и пр.) зрителям сложно наблюдать за происходящим, кроме как с помощью онлайн-трансляций.

Онлайн-трансляции можно разделить на два вида:

- видеотрансляции - видеосъемка происходящего
- трекинг - отображение положения участников на нарисованной карте.

Организация видеотрансляций существенно дороже и технически сложнее, чем трекинг, так как видеотрансляции требуют намного более быстрого метода передачи данных. Трекинг же дешевле и проще, и во многих случаях позволяет показать борьбу соревнующихся. Также трекинг хорошо дополняет видеотрансляции.

схема работы трекинговой системы



пример интерфейса трекинговой системы

Но для того, чтобы трансляция была зрелищной, нужно, чтобы в ней отсутствовали артефакты, вызванные техническими погрешностями, такие как:

- пересечение непреодолимых объектов (например, зданий или заборов)
- движение мимо контрольных точек
- движение параллельно дороге, когда известно, что спортсмен в действительности просто бежал ровно по дороге

Сложно сохранять чувство погружения у зрителя, если показывать как спортсмен перемещается параллельно с дорогой или пересекает непреодолимый объект.

Именно поэтому очень важно делать трансляции достоверными и устранять такие артефакты, вызванные погрешностями. Это можно сделать, используя более точные трекеры, либо устраняя артефакты с помощью программного обеспечения. Второй вариант дешевле и дополняет первый.

При этом в каждом конкретном виде спорта можно учитывать его специфику.

В нашей работе мы сосредоточились на спортивном ориентировании на лыжах и городском спортивном ориентировании. Специфика зимнего ориентирования на лыжах, заключается в том, что спортсмены больше склонны перемещаться именно по лыжне, что несомненно упрощает задачу. В городском ориентировании же спортсмен может перемещаться между домами, проходить под арками, пробегать по узким переулкам. К тому же сигнал от спутника многочисленно отражается от стен, создавая дополнительную погрешность.

Из-за этого необходима коррекция, которая сможет определить, где находится спортсмен.

К тому же, проблема может быть не только в погрешности трекеров участников, но также в методах составителей карты. Например, некоторые повороты на карте были более крутыми, чем в действительности. В процессе выполнения курсовой работы мы учли эти неточности. Проблему удалось решить, нелинейно трансформировав карту, сопоставляя на карте и треке некоторые выделяющиеся точки (развилки, повороты и проч.)

Задача отображения объекта на дороге известна давно и широко используется в навигаторах. Однако в навигаторах с очень большой вероятностью известно, что движение идёт именно по дороге. В лыжном ориентировании возможно движение вне лыжней, поэтому нужно по косвенным признакам оценивать вероятность нахождения спортсмена на лыжне или вне лыжни. Можно учитывать направление и скорость, типичные погрешности при проезжании перекрёстков, треки других спортсменов.

На данный момент нет работ по улучшению трекинга именно для спортивного ориентирования. Тем не менее, существуют работы про общее устранение погрешности и предсказание положения наблюдаемого объекта.[23] Одним из таких методов является фильтр Калмана.

[2] KRASNER, Norman F. *Method for open loop tracking GPS signals*. U.S. Patent Nr. 6,633,255, 2003.
[3] MIURA, Shunsuke, et al. GPS error correction with pseudorange evaluation using three-dimensional maps. IEEE Transactions on Intelligent Transportation Systems, 2015, 16. Jg., Nr. 6, S. 3104-3115.

Постановка задачи

Создание рабочей системы обработки векторных спортивных карт и полученных через GPS треков участников соревнований, с последующей коррекцией последних (используя фильтр Калмана) для получения треков, наиболее близких к истинным. Создание пользовательских интерфейсов, позволяющие осуществлять взаимодействие с системой коррекции. Создание универсальной платформы для написания корректоров треков. Сравнение корректоров, построенных на данной платформе.

Методика

На языке программирования Kotlin [4] был написан парсер векторных карт формата OCAD 11. Параллельно с этим изучается материал по фильтру Калмана, для последующей его реализации, необходимые библиотеки подключаются. После создания парсера векторных карт и системы их визуализации, пишется первый вариант корректора, на основе «переноса» точки трека к ближайшей дороге. Параллельно реализуется система учёта корректирующего влияния внешних элементов карты (выталкивание точек трека из мест, где их быть не может, притягивание к дороге). После того, как всё вышеперечисленное реализовано, реализуется вторая версия корректора, с использованием фильтра Калмана. Полученный вариант тестируется и, в случае проблем, исправляется и дополняется. В процессе написания добавляются тесты, покрывающие максимум созданного кода, на случай непредвиденных ошибок в процессе редактирования.

По требованию код может быть предоставлен преподавателям для ознакомления. Репозиторий: https://ogps.jetbrains.space/p/ocdp/code/ocad-parser/

Язык Kotlin был выбран, т.к. он является выразительным, статически типизированным, позволяет использовать существующие библиотеки для Java и хорошо поддержан в IDE. Также не возникает проблем с переносимостью, т.к. Kotlin может исполняться на JVM.

Проект состоит из четырех взаимосвязанных модулей:

- **parser** – парсер векторных карт
- **corrector** – различные корректоры и утилиты к ним (включая фильтр Калмана)
- **viewer** – GUI, позволяющий опробовать корректор в процессе разработки или в целях демонстрации.
- **cli** – консольный интерфейс, который используется на production.

Ниже представлено описание каждого модуля.

parser

Парсер принимает на вход бинарный файл в формате `OCAD 11`, побайтово читая его согласно документации формата[5]. На выходе мы получаем набор изолиний, образующие карту.

Для этого модуля было написано 4 юнит-теста.
Код этого модуля занимает примерно 113 КБ.

---

[4] Kotlin Foundation 2020, Kotlin Documentation Reference, <https://kotlinlang.org/docs/reference>
[5] OCAD AG 2018, OCAD 11 Wiki, <http://www.ocad.com/wiki/ocad11/en/index.php?title=Main_Page>

corrector

Именно в этом ключевом модуле реализованы различные алгоритмы корректировки.

В каждом корректоре представлен широкий набор параметров и настроек (радиус поиска дорог, дисперсии для фильтра Калмана, коэффициенты различных уравнений и многое другое), позволяющие тонко настраивать поведение корректора. Лучшее значение этих параметров можно определять различными методами глобальной оптимизации. Мы выбрали метод градиентного спуска.

BaseCoordinateCorrector

Основным классом является `BaseCoordinateCorrector`. Он является абстрактным суперклассом для всех остальных корректоров. На вход он получает карту в виде набора изолиний (из parser). В процессе предобработки мы, во-первых, конвертируем их в `Geometry` из библиотеки `JTS`[6]. Во-вторых, разбиваем карту местности на квадратные блоки с фиксированной стороной. Это сделано для оптимизации вычислений, так как в процессе вычисления часто используется функция нахождения пересечения из `JTS`, которая, хоть и очень удобная, но довольно ресурсоемкая. Скорость её работы можно заметно увеличить, если ограничить зону поиска.

В `BaseCoordinateCorrector` реализованы методы, позволяющие корректору узнать “влияние окружения спортсмена”, которое выражается вектором. Это требуется, например, в фильтре Калмана для прогнозирования следующего местоположения лыжника. Видов “влияний” три:

1. Влияние от дорог (`roadInfluence`), которое притягивает спортсмена к лыжне/дороге.
2. Влияние от заборов (`fenceInfluence`), которое отталкивает спортсмена от заборов.
3. Влияние от зданий (`buildingInfluence`), которое отталкивает спортсмена от зданий, если его местоположение вне них; иначе просто отталкивает спортсмена наружу.

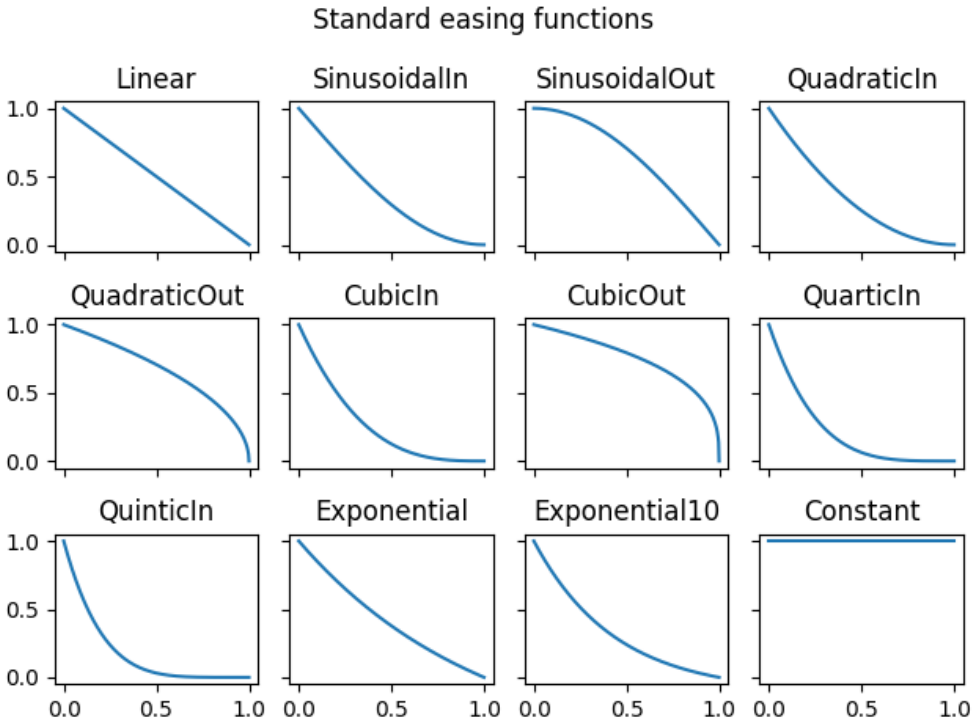
Рассмотрим метод расчета векторов влияния. Логично, что объекты, находящиеся за пределами некоторого радиуса не должны влиять на результат. В то же время, объекты, находящиеся очень близко, тоже не должны влиять бесконечно сильно, а лишь с какой-то конечной силой. Для разрешения этой проблемы был написан вспомогательный `enum`-класс `EasingFunction`. Каждый `entry` задается какой-либо убывающей функцией f(x) : f(0) = a и f(b) = 0, где a и b едины для всех `EasingFunction` и являются параметрами корректора.

[6] Locationtech 2020, JTS Documentation, <https://locationtech.github.io/jts/>

Были реализованы следующие вариации:

- `Linear`
- `SinusoidalIn`
- `SinusoidalOut`
- `QuadraticIn`
- `QuadraticOut`
- `CubicIn`
- `CubicOut`
- `QuarticIn`
- `QuinticIn`
- `Exponential`
- `Exponential10`
- `Constant`

Здесь представлено графическое отображение этих функций.

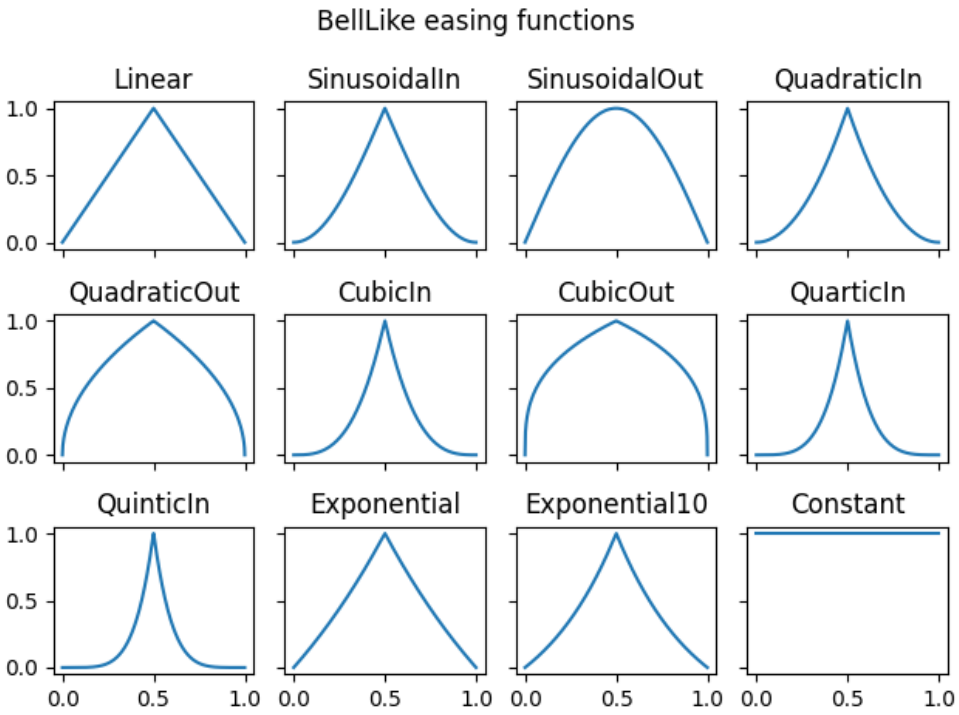


`roadInfluence` рассчитывается двумя способами:

1. притяжение к ближайшей дороге
2. притяжение к “среднему положению всех дорог” (далее `medianInfluence`) внутри некоторого радиуса вокруг спортсмена

В итоге второй метод оказался более удачным, так как в первом случае направление вектора влияния может неожиданно поменяться (если ближайшей стала другая дорога), а во втором случае вектор меняется непрерывно.

Поскольку при нахождении человека непосредственно на дороге, она никак на него влиять уже не должна, был реализован дополнительный режим для EasingFunction - bellLike, который сдвигает максимум функции таким образом в точку b/2.



Итоговая формула для направления влияния $\vec{a}$ представлена ниже, где f - EasingFunction.

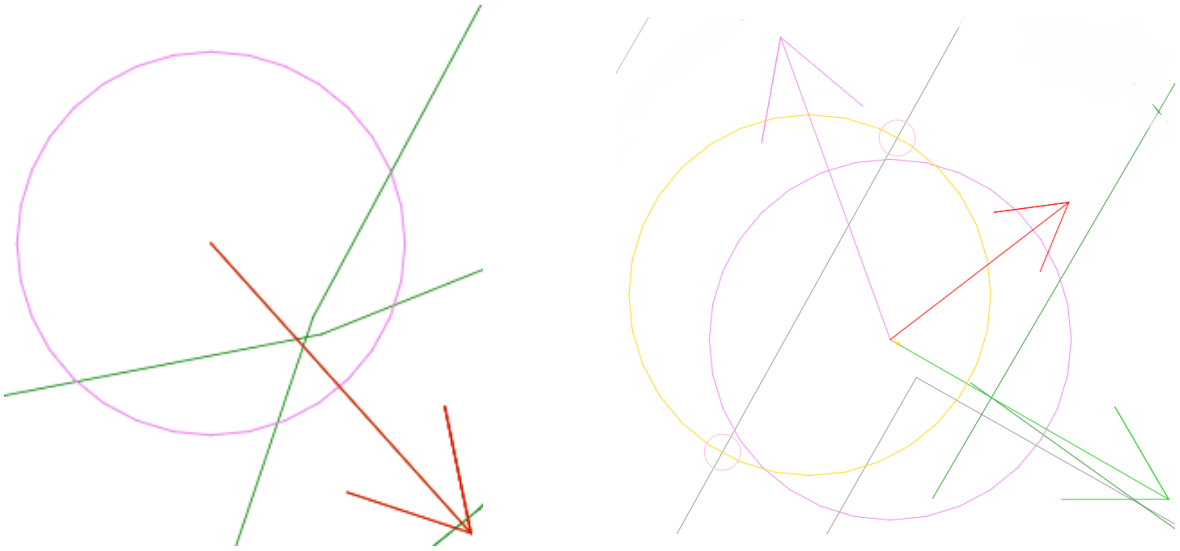
$$\vec{a} = \int\limits_{|\vec{r}| \leq R_{max}} \hat{r} \cdot f(|\vec{r}|) \cdot |\vec{dl}|$$

В этой формуле мы рассматриваем все маленькие кусочки дороги в радиусе R_{max}. $\vec{r}$ – радиус-вектор до центра данного кусочка. Модуль его вклада будет равен $f(|\vec{r}|) \cdot |\vec{dl}|$, а направление вклада задаёт радиус-вектор.

Данная формула оказалась довольно удачной и удобной для наших целей. Например, её реализация проста, так как дороги заданы в виде ломаной, что значительно облегчает вычисление интеграла.

Стоит ещё раз отметить, что $\vec{a}$ – лишь направление вектора влияния. Модуль реального вектора влияния равен $f(|\rho|)$, где ρ – расстояние до ближайшей точки дороги. Такой выбор был сделан для того, чтобы значение модуля вектора влияния было ограничено тем же a, но в то же время вектор влияния всё ещё изменялся непрерывно.

`fenceInfluence` и `buildingInfluence` высчитывается аналогично – через обычный `medianInfluence` (без bellLike; лишь с собственными R_{max}, a и b, а также вместо дорог рассматриваются геометрии заборов и зданий).



Демонстрация `influence` (они изображены стрелочками) через скриншоты из viewer
Красный – суммарное влияние; Зеленый – `roadInfluence`; Фиолетовый – `buildingInfluence`; Желтый – `fenceInfluence`

Многообразие корректоров на базе BaseCoordinateCorrector

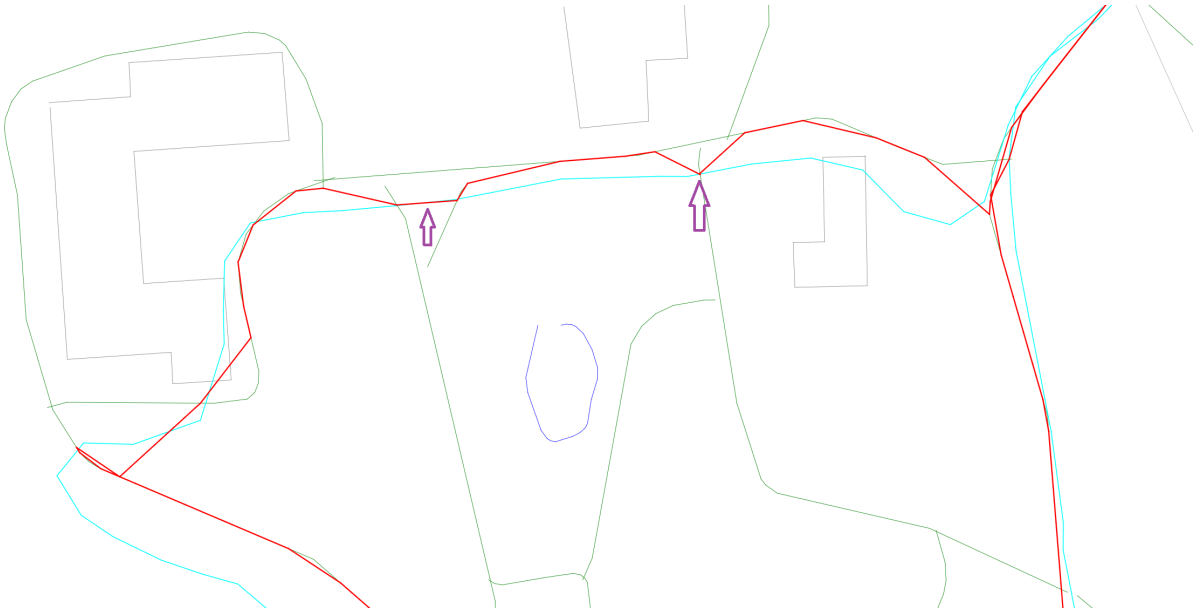
Нами были реализованы 5 следующих корректоров:

- `Basic`
- `Kalman`
- `Hybrid`
- `ReverseHybrid`
- `Augmenting`

Ниже дано описание каждого корректора

Basic

Самый простой и наивный корректор. Если на расстоянии меньше некоторого значения есть дорога, то спортсмен “притягивается” к ближайшей дороге.



пример работы Basic корректора
синее – исходный путь; красное – корректированный путь
В наивном методе сильно заметны искажения рядом с перекрестками

Kalman

Корректор, использующий классический фильтр Калмана, который мы реализовали для движения на плоскости.
В качестве физической модели для фильтра было использовано уравнение движения материальной точки с постоянной скоростью, а внешнее воздействие - в качестве ускорения этой точки. Таким образом, состоянием фильтра является вектор из четырёх элементов - координаты и скорости.

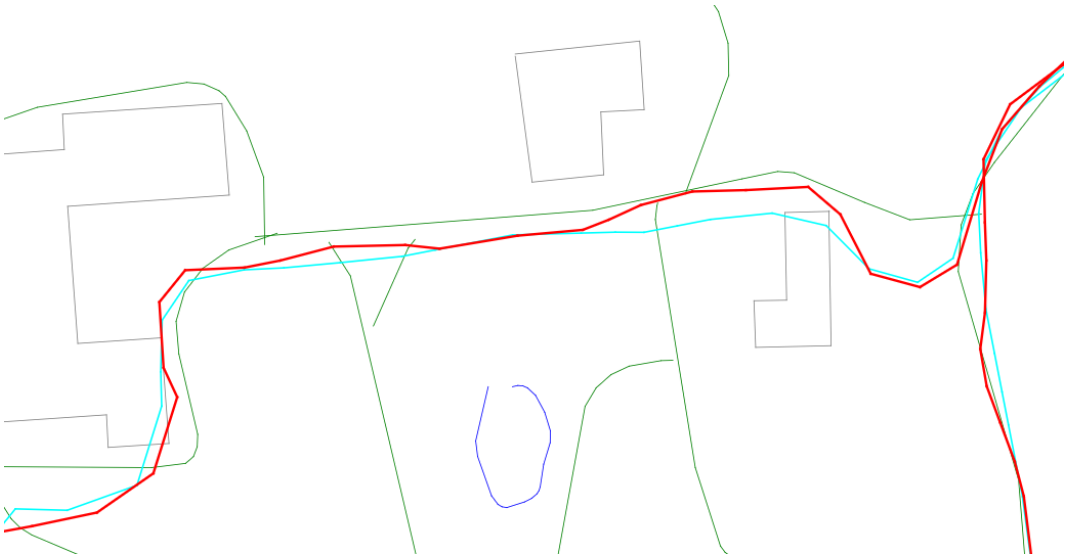
$$\begin{bmatrix} x & y & v_x & v_y \end{bmatrix}$$

Матрица эволюции системы имеет следующий вид:

$$\begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Внешнее воздействие передаётся в виде вектора, получаемого функцией `externalInfluence` следующим образом:

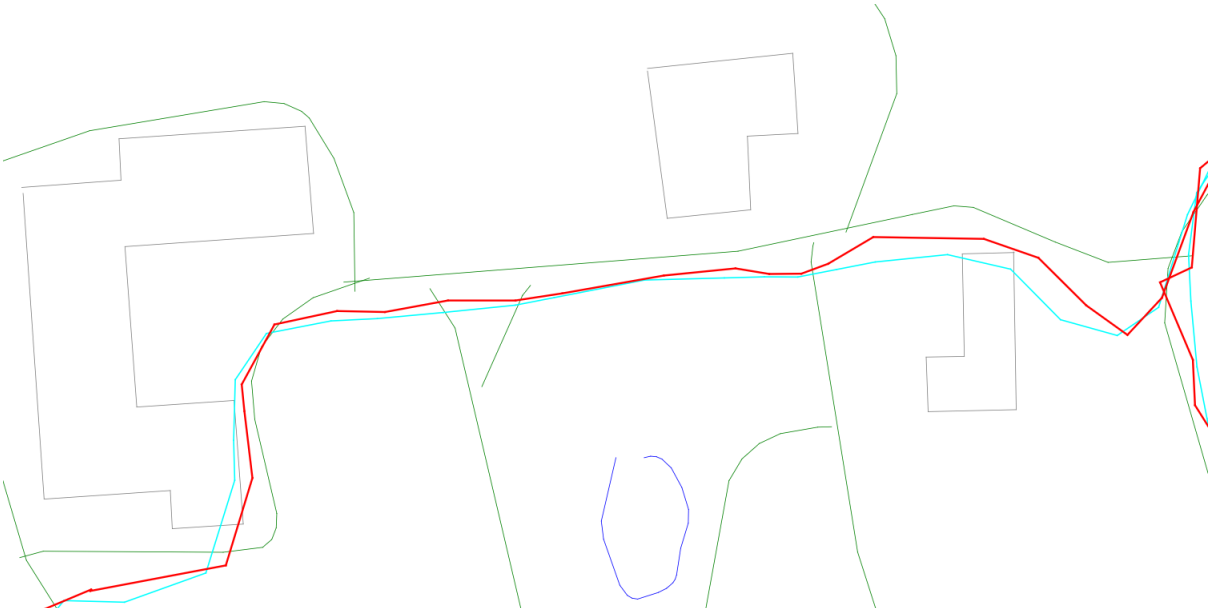
$$\begin{bmatrix} 0 & 0 & a_x & a_y \end{bmatrix}$$



пример работы Kalman корректора
синее – исходный путь; красное – корректированный путь

Augmenting

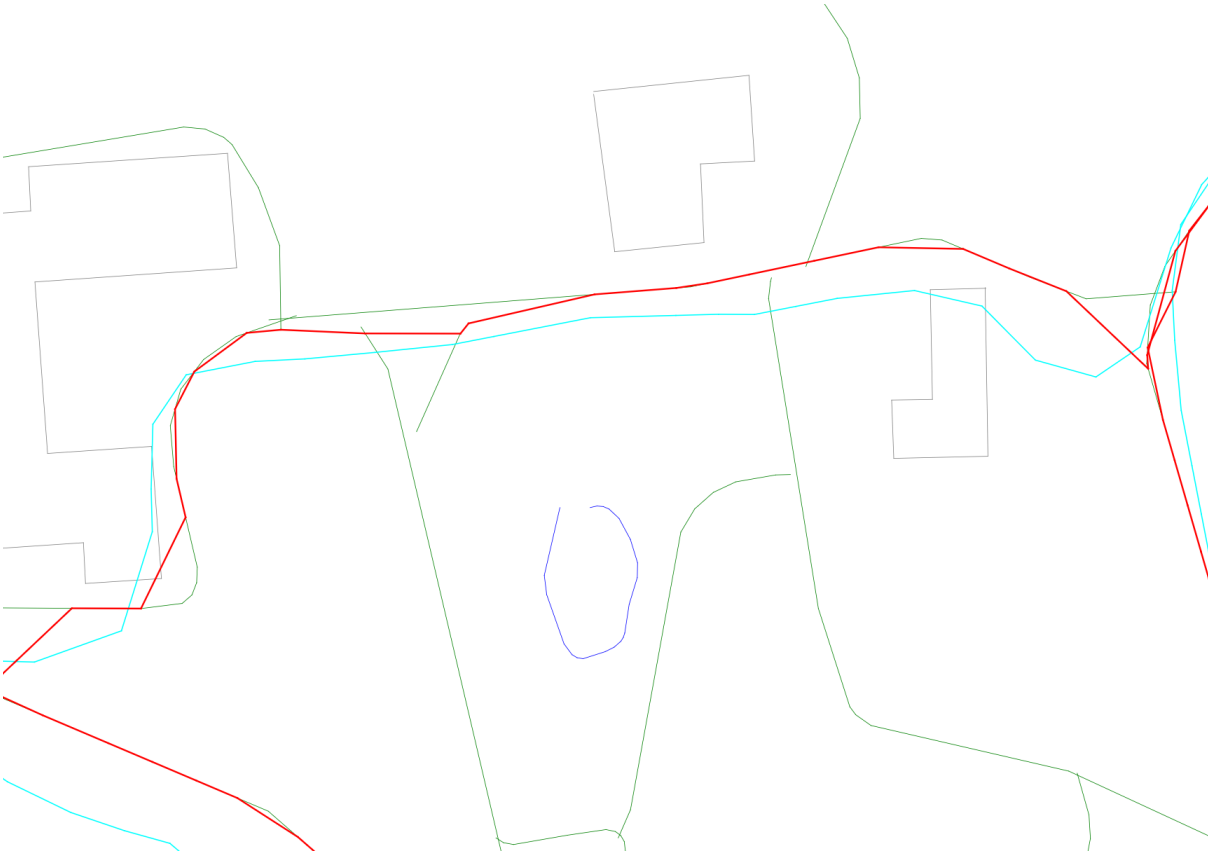
Модификация `Kalman`. В процессе использования предыдущих корректоров было обнаружено, что GPS-трекеры иногда длительное время не посылают сигнал. Из-за этого у фильтра Калмана случаются проблемы с прогнозированием. Мы попробовали решить эту проблему, добавляя фиктивные точки с фиксированным временным периодом между известными положениями спортсмена.



пример работы Augmenting корректора
синее – исходный путь; красное – корректированный путь

Hybrid

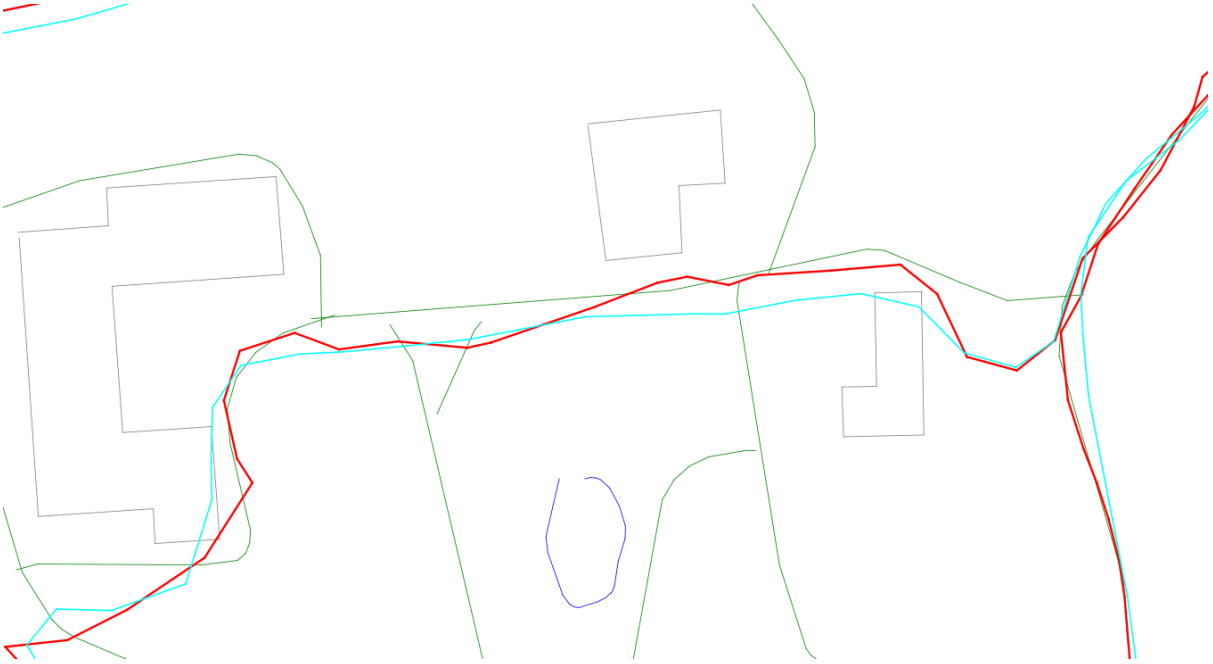
Комбинация `Kalman` и `Basic`. Сначала производится вычисление координаты по Калману, а затем производится “оттягивание” от зданий (откидывание на границу, если внутри) и “притягивание” к дороге. Этот вариант лучше работает в ситуациях, когда спортсмен двигался не по лыжням.



пример работы Hybrid корректора
синее – исходный путь; красное – корректированный путь

ReverseHybrid

Комбинация `Basic` и `Kalman`. Сначала производится “оттягивание” от зданий (откидывание на границу, если внутри) и “притягивание” к дороге, а лишь затем производится вычисление координаты по Калману. Этот вариант лучше работает в ситуациях, когда спортсмен движется строго по лыжне.



пример работы ReverseHybrid корректора
синее – исходный путь; красное – корректированный путь

Для этого модуля было написано 24 юнит-теста.
Код этого модуля занимает примерно 130 КБ.

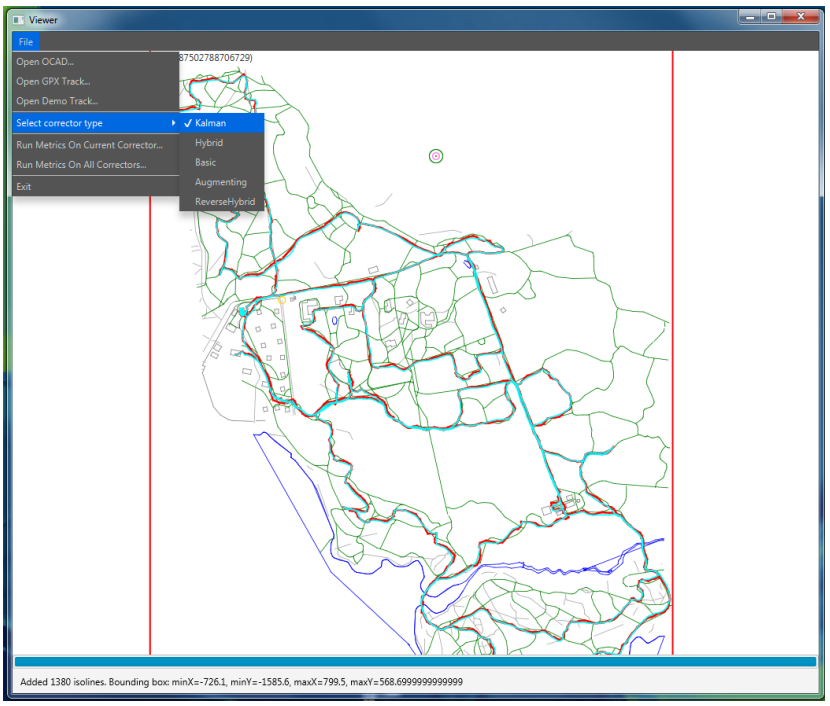
viewer

Этот модуль является простым GUI, который использует популярную библиотеку JavaFX[7]. Из-за этого это приложение является кроссплатформенным.

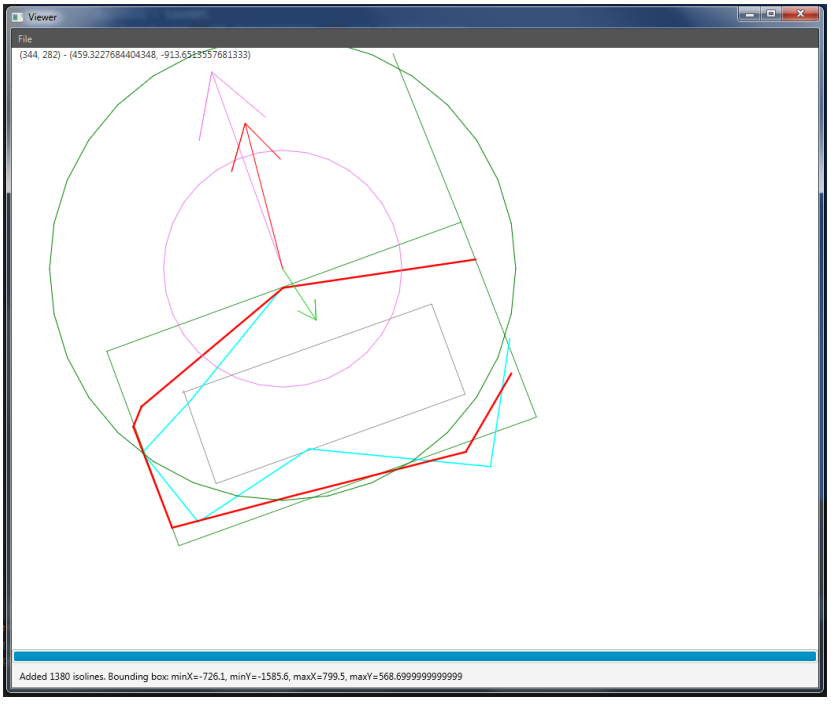
Его функции включают:

- просмотр OCAD карт
- отображение влияния окружающей среды в виде направленного отрезка
- выбор используемого корректора
- корректировка трека из файла формата GPX [8] и отображение результата на экране
- демонстрационные треки (`CorrectorDemoTrack`), которые заданы на малой карте для целей демонстрации и отладки во время разработки
- запуск подсчета метрики (см. раздел Подсчёт метрики – оценка корректора) для текущего или всех корректоров.

[7] OpenJFX 2020, JavaFX, <https://openjfx.io/>
[8] TopoGrafix 2004, “GPX: the GPS Exchange Format”, <https://www.topografix.com/gpx.asp>



общий функционал приложения



пример демонстрационного трека

Код этого модуля занимает примерно 53 КБ.

cli

Этот модуль является консольным интерфейсом для нашей программы. Для него использовалась библиотека `clikt`[9].
Его главное предназначение – генерация корректора по заданным настройкам. Каждый параметр любого корректора можно настроить через cli, благодаря работе с `kotlin-reflection`[10]: все параметры корректоров автоматически парсятся по всем классам и им присваиваются соответствующие названия для настройки в cli. Её основное предназначение – загрузка параметров из другой программы (например, из веб-сервера).

```
waleko@AlexKovrigin:~/o3/cli-shadow-local_build/bin$ ./ocad-corrector --help
Usage: ocad-corrector [OPTIONS] COMMAND [ARGS]...

  This script provides a command-line interface for OCAD Corrector

Options:
  -v, --version  Show the version and exit
  -h, --help     Show this message and exit

Commands:
  convert  This command converts an OCAD file (.ocd) and builds a
           CoordinateCorrector from it
  config   Configures settings for CoordinateCorrector and convert command
```

основное меню помощи

```
waleko@AlexKovrigin:~/o3/cli-shadow-local_build/bin$ ./ocad-corrector convert --help
Usage: ocad-corrector convert [OPTIONS] SOURCE [CORRECTORTYPE]

  This command converts an OCAD file (.ocd) and builds a CoordinateCorrector
  from it

Options:
  -o, --output PATH               Output path (default: same as source but
                                  with .geo extension)
  -c, --config FILE               Json configuration file
  --config_augmenting_regression FLOAT
                                  (default: 0.66)
  --config_augmenting_timeDelayInMilliseconds INT
                                  (default: 200)
  --config_basic_blocksPreprocessingMode [MultipleBlocks|SingleBlocks|NoPreprocessing]
                                  Mode of preprocessing blocks that original
                                  map will be divided into (default:
                                  MultipleBlocks)
  --config_basic_building_easingFunction [Linear|SinusoidalIn|SinusoidalOut|QuadraticIn|CubicIn|QuarticIn|QuinticIn|Quad
raticOut|Exponential|Exponential10|Constant]
                                  Easing function for calculating the impact
                                  of this symbol class (default: Linear)
  --config_basic_building_externalInfluenceSearchRadius FLOAT
                                  Maximum radius for external influence
                                  search for this symbol class (default:
                                  10.0)
  --config_basic_building_maxExternalAcceleration FLOAT
                                  Maximum external influence acceleration for
                                  this symbol class (default: 25.0)
```

все параметры на этом скриншоте
были автоматически сгенерированы с помощью `kotlin-reflection`

[9] AJ Alt 2020, clikt, <https://ajalt.github.io/clikt/>
[10] Kotlin Foundation 2020, Kotlin Reflection, <https://kotlinlang.org/docs/reference/reflection.html>

```
waleko@AlexKovrigin:~/o3/cli-shadow-local_build/bin$ ./ocad-corrector config --help
Usage: ocad-corrector config [OPTIONS] COMMAND [ARGS]...

  Configures settings for CoordinateCorrector and convert command

Options:
  -h, --help  Show this message and exit

Commands:
  set     Configures an individual option for CoordinateCorrector
  list    Prints out current settings for CoordinateCorrector
  load    Loads all settings for CoordinateCorrector from the given file
  export  Exports internal configuration file
```

операции с конфигурацией

Для этого модуля было написано 14 юнит-тестов.
Код этого модуля занимает примерно 47 КБ.

Подсчёт метрики – оценка корректора

Когда мы получили большое количество корректоров, нашей целью стало определение наиболее качественного из них.

Для этого мы создали метрику, позволяющую численно оценить эффективность каждого корректора.

Сначала мы взяли истинные треки – то, как на самом деле бежал спортсмен. Добавляем случайной погрешности к координатам и отдаём этот трек корректору. Чем точнее корректор восстановит маршрут, тем он лучше, и, соответственно, тем меньше его значение метрики.

Мы решили ввести числовую характеристику такой метрики по следующей формуле:

$$score = \frac{\sum_{i=1}^{N} dist^2(perfect_i, corrected_i) + buildingInfluence^2(corrected_i) + fenceInfluence^2(corrected_i)}{N}$$

Стоит отметить, что здесь `fenceInfluence` и `buildingInfluence` имеют одинаковые параметры, чтобы их значение в одинаковой точке было одинаковым.

Посчитав такую характеристику для всех корректоров получаем следующий результат:

Корректор	Значение метрики
Kalman	349.78165476372806
Hybrid	**180.38854570850103**
Basic	281.866790514779
Augmenting	628.8243648778987
ReverseHybrid	**230.28450833775446**

Таким образом, наилучшие результаты достигаются **Hybrid** корректором. Это логично, так как в нем сочетается и простая идея притяжения к дороге (так как лыжники с большей вероятности на лыжне) и фильтр Калмана, позволяющий сделать сглаживание, убирающее случайные погрешности.

Впрочем, стоит отметить, что другой гибридный метод (**ReverseHybrid**) тоже получил хороший результат. Из этого можно сделать вывод, что гибридные алгоритмы – действительно лучшее решение нашей задачи.

Полученные результаты и выводы

В итоге были написаны четыре модуля для нашего проекта. Была сделана платформа в виде `BaseCoordinateCorrector` для создания корректоров, к которой было написано много юнит-тестов, покрывающие большую часть кода, позволяющие защитить кодовую базу от случайных ошибок. В этой платформе создана высококонфигурируемая система учёта внешнего влияния. На этой платформе было создано 5 корректоров, 4 из которых используют фильтр Калмана. Было произведено их сравнение, и выявлены лучшие – **Hybrid** и **ReverseHybrid**.

Дальнейшие планы

Очевидным продолжением является создание новых корректоров на нашей платформе с помощью других модификаций фильтра Калмана или других методов уменьшения погрешности треков[11][12]. Помимо этого можно также использовать трехмерные карты местности для предсказания изменения скорости за счет изменения потенциальной энергии спортсмена.

Также есть возможность дальше оптимизировать параметры более продвинутыми методами глобальной оптимизации[13].

Кроме того, возможно дальнейшее усовершенствование метрик в зависимости от конкретного вида спорта.

Также одним из вариантов дальнейшего развития является добавление обработки и получения не только показаний с gps спутника, но и с акселерометра и гироскопа, которые есть в большинстве смартфонов и, вероятно, в некоторых передатчиках. Используя показания от них для более точного получения текущего вектора ускорения, что, как нам кажется, значительно увеличит точность предсказания и модели для фильтра Калмана.

---

[11] KRASNER, Norman F. *Method for open loop tracking GPS signals*. U.S. Patent Nr. 6,633,255, 2003.
[12] MIURA, Shunsuke, et al. GPS error correction with pseudorange evaluation using three-dimensional maps. IEEE Transactions on Intelligent Transportation Systems, 2015, 16. Jg., Nr. 6, S. 3104-3115.
[13] TÖRN, Aimo; ŽILINSKAS, Antanas. *Global optimization*. Berlin: Springer-Verlag, 1989.

Список используемых источников

KRASNER, Norman F. *Method for open loop tracking GPS signals*. U.S. Patent Nr. 6,633,255, 2003.

MIURA, Shunsuke, et al. GPS error correction with pseudorange evaluation using three-dimensional maps. IEEE Transactions on Intelligent Transportation Systems, 2015, 16. Jg., Nr. 6, S. 3104-3115.

R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]," in IEEE Signal Processing Magazine, vol. 29, no. 5, pp. 128-132, Sept. 2012, doi: 10.1109/MSP.2012.2203621.

Honghui Qi and J. B. Moore, "Direct Kalman filtering approach for GPS/INS integration," in IEEE Transactions on Aerospace and Electronic Systems, vol. 38, no. 2, pp. 687-693, April 2002, doi: 10.1109/TAES.2002.1008998.

CRASSIDIS, John L. Sigma-point Kalman filtering for integrated GPS and inertial navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 2006, 42. Jg., Nr. 2, S. 750-756.

TÖRN, Aimo; ŽILINSKAS, Antanas. *Global optimization*. Berlin: Springer-Verlag, 1989.

Kotlin Foundation 2020, Kotlin Documentation Reference, <https://kotlinlang.org/docs/reference>

OCAD AG 2018, OCAD 11 Wiki, <http://www.ocad.com/wiki/ocad11/en/index.php?title=Main_Page>

Locationtech 2020, JTS Documentation, <https://locationtech.github.io/jts/>

Geotools 2020, Geotools Documentation, <http://docs.geotools.org/>

OpenJFX 2020, JavaFX, <https://openjfx.io/>

TopoGrafix 2004, “GPX: the GPS Exchange Format”, <https://www.topografix.com/gpx.asp>

AJ Alt 2020, clikt, <https://ajalt.github.io/clikt/>

Kotlin Foundation 2020, Kotlin Reflection, <https://kotlinlang.org/docs/reference/reflection.html>

Благодарности

Выражаем благодарность:

- Лицею «Физико-техническая школа» им. Ж. И. Алфёрова Санкт-Петербургского Академического университета за включенную в школьную программу возможность проведения исследовательской деятельности
- ООО "Спортивные трекинговые системы" за методическую поддержку
- Компании JetBrains за язык программирования Kotlin и среду разработки IntelliJ IDEA