Development of a GPS-coordinates correction system for live broadcasting of orienteering competitions using the Kalman filter on Kotlin

Alexander Kovrigin[1] Mark Ipatov[1] Mikhail Senin[2*]

[1]Academic lyceum "Physical-Technical High School" named after Zh. I. Alfyorov
[2]O-GPS Center

Wednesday 30th December, 2020

Abstract

Many sports (for example, sailing, multi-racing, marathons, and orienteering) do not allow spectators constantly watch the athletes. To increase entertainment and control, trackers are usually used, the data from which is shown to viewers in real time. However, when conducting such online broadcasts, the organizers face the problem of inaccurate display of the position of the athlete. This can make it difficult for viewers to follow the dynamics of the event.

The goal of our work is to create a correction system that will correctly display the athlete's location using a specific model for a particular sport. Our work is devoted to methods of correction for orienteering.

To achieve the goal, we implemented and compared several correction methods: naive pulling the athlete to the road, Kalman filter [3], and hybrid options.

As a result, one of our hybrid options turned out to be 36% more accurate than the naive one. However, further improvement of this correction algorithm is possible.

Keywords: Kalman filter, GPS, Orienteering

Contents

*Scientific director

1 Introduction

When holding sports competitions, organizers often resort to using the means of online broadcasting of the event. It is beneficial for the competition, as more spectators will be able to watch it. In addition, in some sports (for example, orienteering, trails, rogaining, sailing, multi-racing, etc.) it is difficult for spectators to watch what is happening, except through online broadcasts.

Online broadcasts can be divided into two types:

- Video broadcasting - video filming of what is happening
- Tracking - displaying the position of the participants on a drawn map.

Organization of video broadcasts is much more expensive and technically more difficult than tracking since video broadcasts require a much faster data transfer method. Tracking, on the other hand, is cheaper and easier, and in many cases allows you to show the struggle of the competitors. Tracking also complements video broadcasts well.

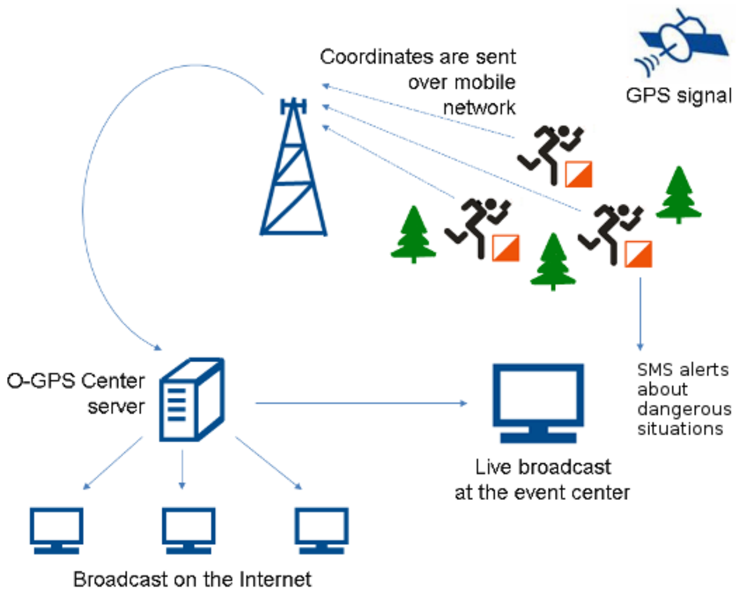

Coordinates are sent over mobile network
GPS signal
O-GPS Center server
SMS alerts about dangerous situations
Live broadcast at the event center
Broadcast on the Internet

Figure 1: Diagram of a tracking system

But for the broadcast to be spectacular, it must be free of artifacts caused by technical errors, such as:

- Crossing impassable objects (such as buildings or fences)
- Moving past checkpoints
- Running parallel to the road when it is known that the athlete was just running straight down the road

It is difficult to maintain a sense of immersion in the viewer when showing the athlete moving parallel to the road or crossing an impassable object.

That is why it is very important to make translations reliable and to eliminate such artifacts caused by errors. This can be done by using more accurate trackers, or by eliminating artifacts using the software. The second option is cheaper and complements the first.

At the same time, each specific sport, and its specifics can be taken into account. In our work, we have focused on ski orienteering and urban orienteering. The specificity of winter ski orienteering lies in the fact that athletes are more inclined to move along the ski track, which undoubtedly simplifies the task. In urban orienteering, an athlete can move between houses, pass under arches, and run along narrow lanes. In addition, the signal from the satellite is reflected numerously from the walls, creating an additional error. Because of this, a correction is needed that can determine where the athlete is.

In addition, the problem may be not only in the error of participant trackers but also in the methods of the mappers. For example, some turns on the map may be sharper than in reality. In the course of this work, we took into account these inaccuracies. The problem was solved by non-linearly transforming the map, comparing some prominent points on the map and the track (forks, turns, etc.)

The task of displaying an object on the road has been known for a long time and is widely used in navigators. However, in navigators with a very high probability, it is known that the movement is on the road. In ski orienteering, movement outside the tracks is possible, so it is necessary to estimate the probability of an athlete being on the track or outside the track by indirect signs. You can take into account the direction and speed, typical errors when passing intersections, and tracks of other athletes.

At the moment, there are no works to improve tracking specifically for orienteering. Nevertheless, there are works about the general elimination of the error and the prediction of the position of the observed object [6] [8]. One such method is the Kalman filter.

2 Problem formulation

Our goals for this work are as follows:

- Development of a working system for processing vector sports maps and tracks obtained via GPS from competitors, with subsequent correction of the latter (using the Kalman filter) to obtain tracks that are closest to the true ones
- Development of user interfaces that allow interaction with the correction system
- Development of a universal platform for writing track correctors
- Comparison of correctors built on this platform

3 Methodology

3.1 General

A parser for vector maps of the OCAD 11[1] format was written in the Kotlin programming language[4]. At the same time, the material on the Kalman filter was studied, for its subsequent implementation, the necessary libraries are connected. After creating a parser for vector maps and a system for their visualization, the first version of the corrector was written, based on the "pushing" of the track point to the nearest road. At the same time, a system was implemented for taking into account the corrective influence of external elements of the map (pushing track points out of places where they cannot be, pulling them to the road). After all of the above had been implemented, the second version of the corrector was implemented using the Kalman filter. The resulting version was tested and, in case of problems, corrected and supplemented. Throughout coding, tests were added that cover the maximum of the created code, in case of unforeseen errors during the editing process.

The Kotlin language was chosen because it is expressive, statically typed, allows one to use existing libraries for Java, and is well supported in the IDE. Moreover, there are no problems with portability, because Kotlin can run on the JVM.

3.2 Project structure

The project consists of 4 interconnected modules:

- **parser** - vector map parser
- **corrector** - various correctors and utilities for them (including the Kalman filter)
- **viewer** - A GUI that allows you to try out the corrector during development or for demonstration purposes.
- **cli** - a console interface that is used in production.

3.3 parser

The parser accepts a binary file in the OCAD 11 format as input, reading it byte by byte according to the format documentation. At the output, we get a set of isolines that form a map.

4 unit tests were written for this module. The code for this module is approximately 113 KB.

3.4 corrector

It is in this key module that various adjustment algorithms are implemented.

Each corrector has a wide range of parameters and settings (road search radius, variances for the Kalman filter and other hyper-parameters) that allow you to tune the behavior of the corrector. The best value of these parameters can be determined by various methods of global optimization. We have chosen the gradient descent method.

There were 24 unit tests written for this module. The code for this module is approximately 130 KB.

3.4.1 BaseCoordinateCorrector

The main class is BaseCoordinateCorrector. It is the abstract superclass for all other correctors. As input, it receives a map in the form of a set of contour lines (from parser). In the preprocessing process, we firstly convert them to Geometry from the JTS library[7]. Secondly, we divide the terrain map into square blocks with a fixed side. This was done to optimize calculations, since the calculation process often uses the intersection function from JTS, which, although very convenient, is quite resource-intensive. The speed of its work can be significantly increased if you limit the search area.

Methods are implemented in BaseCoordinateCorrector that allow the corrector to know the “athlete’s environment influence”, which is expressed as a vector. This is required, for example, in the Kalman filter to predict the skier’s next location. We have implemented three types of ”influences”:

1. **Road influence**, which pulls the athlete towards the track/road.
2. **Fence influence**, which pushes the athlete away from the fences.
3. **Building influence**, which pushes the athlete away from buildings if their location is outside of them; otherwise it just pushes the athlete out.

Let’s consider the method of calculating influence vectors. It is logical that objects outside a certain radius should not affect the result. At the same time, objects that are very close, too, should not influence infinitely strongly, but only with some finite force. To solve this problem, the `EasingFunction` helper enum class was written. Each entry is defined by some decreasing function $f(x) \begin{cases} f(0) = a \\ f(b) = 0 \end{cases}$, where a and b are the same for all easing functions and are the corrector parameters.

The following easing functions were implemented: `Linear`, `SinusoidalIn`, `SinusoidalOut`, `QuadraticIn`, `QuadraticOut`, `CubicIn`, `CubicOut`, `QuarticIn`, `QuinticIn`, `Exponential`, `Exponential10`, `Constant`.

`roadInfluence` is calculated in two ways: attraction to the nearest road attraction to the “average position of all roads” (hereinafter `medianInfluence`) within a certain radius around the athlete

As a result, the second method turned out to be more successful, since in the first case the direction of the influence vector can suddenly change (if another road has become the closest one), and in the second case the vector changes continuously.

Since when a person is directly on the road, it should not affect him in any way, an additional mode for `EasingFunction` was implemented - `bellLike`, which shifts the maximum of the function in this way to point $\frac{b}{2}$.

The resulting formula for the influence vector direction $\vec{a}$ with easing function f is:

$$\vec{a} = \int\limits_{|\vec{r}| \leq R_{max}} \hat{r} \cdot f\left(|\vec{r}|\right) \cdot dl \tag{1}$$

In this formula we consider all small bits of the road inside the R_{max} radius. $\vec{r}$ is the radius-vector that points to the center of the road piece. dl is the road piece length.

This formula turned out to be quite successful and convenient for our purposes. For example, its implementation is simple, since the roads are given in the form of a broken line, which greatly simplifies the calculation of the integral.

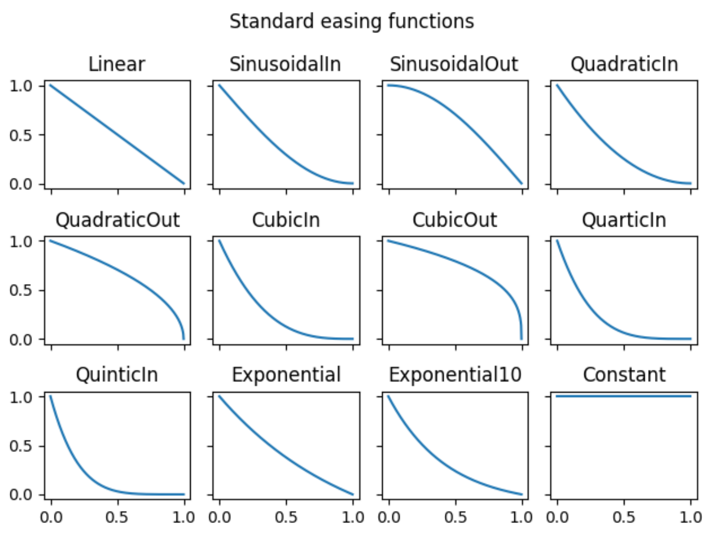


Figure 2: Easing functions plot

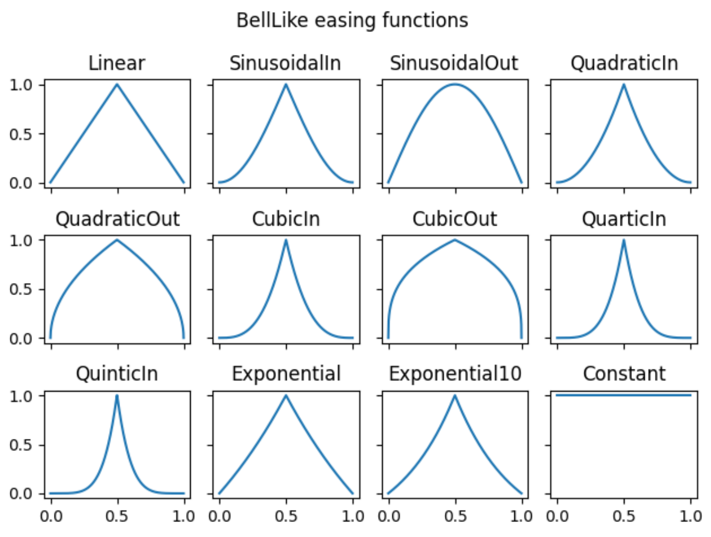


Figure 3: `bellLike` easing functions plot

It is worth noting once again that $\vec{a}$ is only the direction of the vector of influence. The modulus of the real influence vector is $f(\rho)$, where ρ is the distance to the nearest point of the road. This choice was made so that the value of the modulus of the influence vector would be limited to the same a, but at the same time the influence vector would still change continuously.

`fenceInfluence` and `buildingInfluence` are calculated in the same way - through the usual `medianInfluence` (without `bellLike`; only with their own R_{max}, a and b, and instead of roads, fence and building geometries are considered).

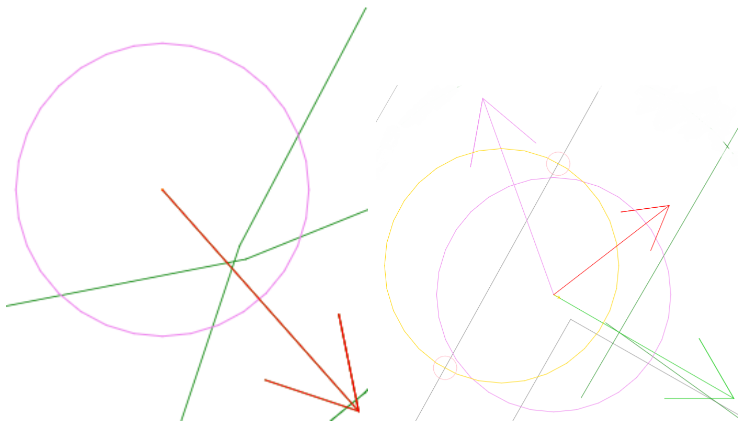


Figure 4: Influence demo (shown as arrows in the viewer app). Red denotes the total influence; Green – road influence; Purple – building influence; Yellow – fence inflience

3.4.2 Correctors based on BaseCoordinateCorrector

We have implemented the following 5 correctors:

- Basic
- Kalman
- Hybrid
- Reverse Hybrid
- Augmenting

3.4.3 Basic

The simplest and most naive corrector. If there is a road at a distance less than a certain value, then the athlete is "pulled" to the nearest road.

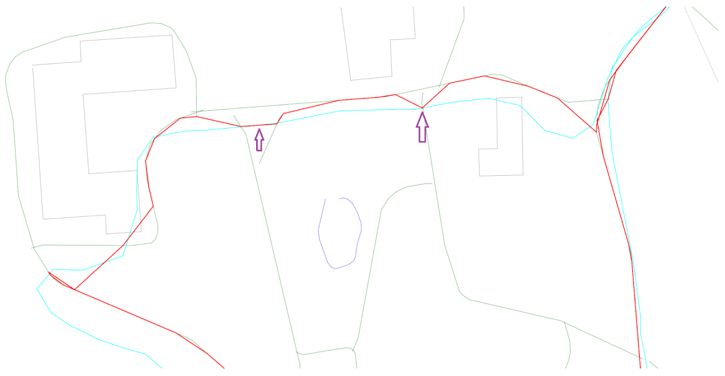


Figure 5: An example Basic corrector run. Blue denotes the recorded track; Red – corrected track

In the figure 5 we can see serious deviations from a straight line near crossings.

3.4.4 Kalman

Corrector using the Kalman filter, which we implemented for movement on a plane.

As a physical model for the filter, the equation of motion of a material point with a constant speed was used, and the external influence was used as the acceleration of this point. Thus, the filter state is a vector of four elements - coordinates and velocities: $\begin{pmatrix} x & y & v_x & v_y \end{pmatrix}$.

Evolution matrix then has is as follows: $\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

External influence is passed as a vector obtained by the `externalInfluence` function as follows: $\begin{pmatrix} 0 & 0 & a_x & a_y \end{pmatrix}$.

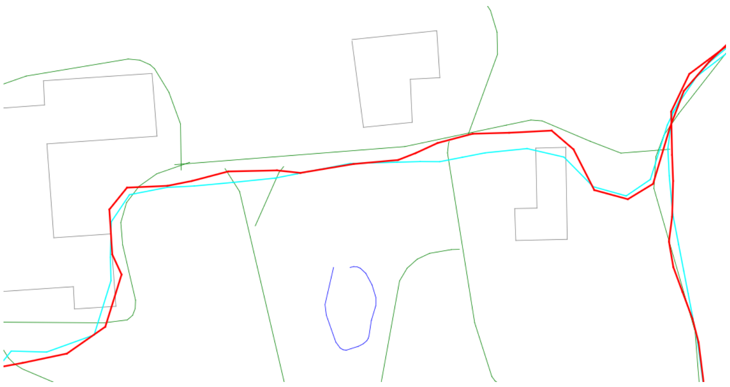


Figure 6: An example Kalman corrector run. Blue denotes the recorded track; Red – corrected track

3.4.5 Augmenting

A Kalman corrector modification. In the process of using previous correctors, it was found that GPS trackers sometimes do not send a signal for a long time. Because of this, the Kalman filter has problems with prediction. We attempted to solve this problem by adding fictitious points with a fixed time period between the known positions of the athlete.

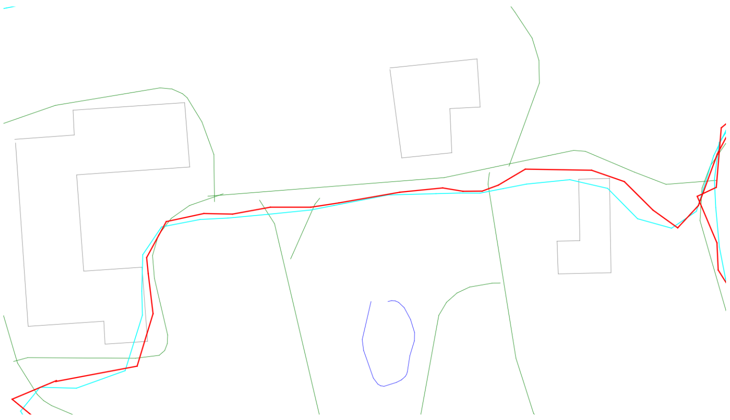


Figure 7: An example Augmenting corrector run. Blue denotes the recorded track; Red – corrected track

3.4.6 Hybrid

Combination of Kalman and Basic. First, the Kalman coordinate is calculated, and then it is "pulled" from the buildings (pushed to the wall, if inside) and "pulled" to the road. This option works better in situations where the athlete was not moving along the ski tracks.

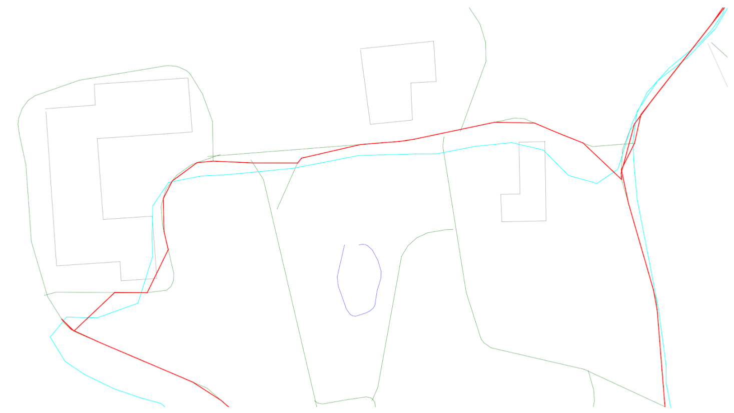


Figure 8: An example Hybrid corrector run. Blue denotes the recorded track; Red – corrected track

3.4.7 Reverse Hybrid

Combination of Basic and Kalman. First, “pulling” from buildings is performed (pushed to the wall, if inside) and “pulling” to the road, and only then is the Kalman coordinate calculated. This option works best in situations where the athlete moves strictly along the track.

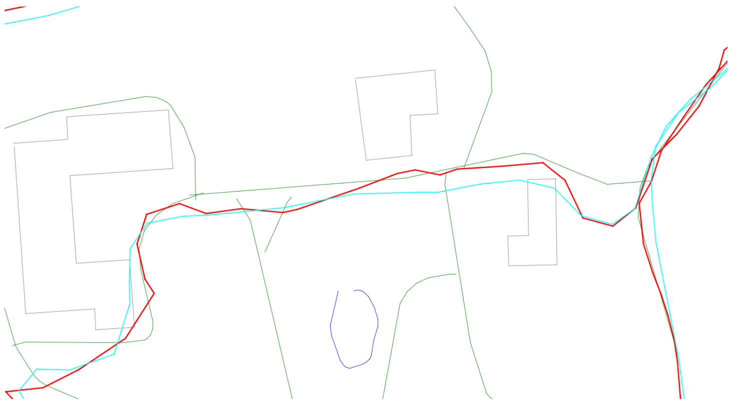


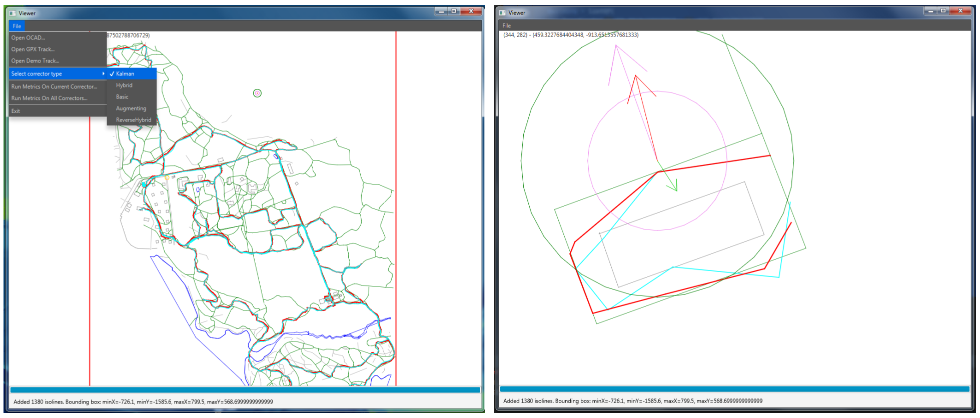
Figure 9: An example Reverse Hybrid corrector run. Blue denotes the recorded track; Red – corrected track

3.5 viewer

This module is a simple GUI that uses the popular JavaFX library[9]. Because of this, this application is cross-platform.

Its features include:

- Viewing OCAD maps
- Displaying the influence of the environment in the form of a directed segment
- Choice of corrector used
- Correcting a track from a GPX[10] file and displaying the result on the screen
- Demo tracks that are set on the small map for demonstration and debugging purposes during development
- Launching the calculation of the metric (see the 3.7) for the currently used or all correctors.

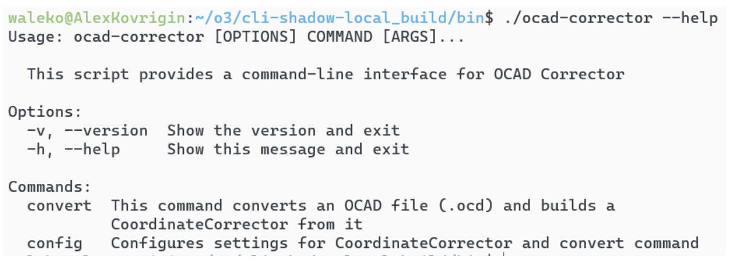


```
(344, 282) - (459.3227684404348, -913.6513557681333)
Added 1380 isolines. Bounding box: minX=-726.1, minY=-1585.6, maxX=799.5, maxY=568.6999999999999
Added 1380 isolines. Bounding box: minX=-726.1, minY=-1585.6, maxX=799.5, maxY=568.6999999999999
```

Figure 10: General application functionalities

3.6 cli

This module is the console interface for our program. The clikt library [2] was used for it. Its main purpose is to generate a corrector according to the given settings. Each parameter of any corrector can be configured through the cli, thanks to working with kotlin-reflection[5]: all parameters of the correctors are automatically parsed across all classes and they are assigned the appropriate names for setting in the cli. Its main purpose is to load parameters from another program (for example, from a web server).



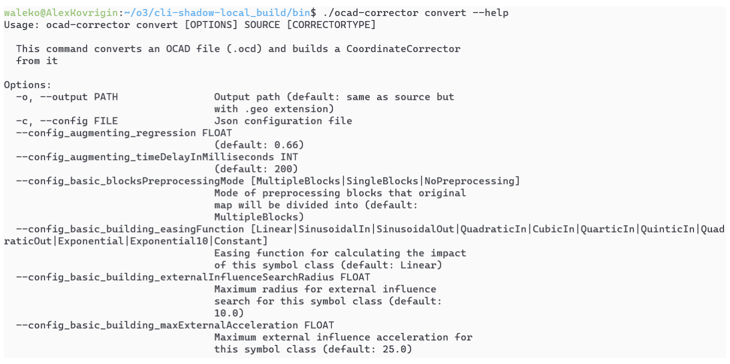
```
waleko@AlexKovrigin:~/o3/cli-shadow-local_build/bin$ ./ocad-corrector --help
Usage: ocad-corrector [OPTIONS] COMMAND [ARGS]...

  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
```

Figure 11: General help message



```
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)
```

Figure 12: Full configuration help message

3.7 Metrics

When we developed a large number of correctors, our goal was to determine the highest quality of them.

To do this, we created a loss function that allows us to numerically evaluate the effectiveness of each corrector.

First, we took the true tracks - how the athlete actually ran. We add a random error to the coordinates and give this track to the corrector. The more accurately the corrector restores the route, the better it is, and, accordingly, the lower its metric value.

We decided to introduce a numerical characteristic of such a metric using the following formula:

$$score = \frac{1}{N}\sum_{i=1}^{N} \text{dist}^2(\text{perfect}_i, \text{corrected}_i) + \text{buildingInfluence}^2(\text{corrected}_i) + \text{fenceInfluence}^2(\text{corrected}_i) \quad (2)$$

Having calculated such a characteristic for all correctors, we obtained the following result:

Corrector name	Average loss function value
Kalman	349.78165476372806
Hybrid	180.38854570850103
Basic	281.866790514779
Augmenting	628.8243648778987
Reverse Hybrid	230.28450833775446

Thus, the best results are achieved with the **Hybrid** corrector. This is logical, as it combines the simple idea of attraction to the road (since skiers are more likely to ski) and the Kalman filter, which allows smoothing that removes random errors.

However, it is worth noting that another hybrid method (**ReverseHybrid**) also got a good result. From this we can conclude that hybrid algorithms are indeed the best solution to our problem.

4 Conclusion

As a result, four modules were written for our project. A platform was made in the form of BaseCoordinateCorrector for creating correctors, to which many unit tests were written, covering most of the code, allowing you to protect the codebase from random errors. This platform has a highly configurable external influence accounting system. On this platform, 5 correctors were created, 4 of which use the Kalman filter. They were compared, and the best ones were identified - **Hybrid** and **ReverseHybrid**.

4.1 Future work

An obvious continuation is the creation of new correctors on our platform using other modifications of the Kalman filter or other methods to reduce track errors[6][8]. In addition, 3D terrain maps can also be used to predict the change in speed due to the change in the athlete's potential energy.

It is also possible to further optimize the parameters with more advanced global optimization methods[11].

In addition, further improvement of metrics depending on the specific sport is possible.

Moreover, one of the options for further development is to add processing and receive not only readings from the gps satellite, but also from the accelerometer and gyroscope, which are in most smartphones and, probably, in some transmitters. Using the readings from them to more accurately obtain the current acceleration vector, which, in our opinion, will significantly increase the accuracy of the prediction and model for the Kalman filter.

References

[1] OCAD AG. Ocad 11 wiki, 2018. URL `http://www.ocad.com/wiki/ocad11/en/index.php?title=Main_Page`.

[2] AJ Alt. clikt, 2020. URL `https://ajalt.github.io/clikt/`.

[3] Ramsey Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal processing magazine*, 29(5):128–132, 2012.

[4] Kotlin Foundation. Kotlin documentation reference, 2020. URL `https://kotlinlang.org/docs/reference`.

[5] Kotlin Foundation. Kotlin reflection, 2020. URL `https://kotlinlang.org/docs/reference/reflection.html`.

[6] Norman F Krasner. Method for open loop tracking gps signals, October 14 2003. US Patent 6,633,255.

[7] Locationtech. Jts documentation, 2020. URL `https://locationtech.github.io/jts/`.

[8] Shunsuke Miura, Li-Ta Hsu, Feiyu Chen, and Shunsuke Kamijo. Gps error correction with pseudorange evaluation using three-dimensional maps. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3104–3115, 2015.

[9] OpenJFX. Javafx, 2020. URL `https://openjfx.io/`.

[10] TopoGrafix. Gpx: the gps exchange format, 2004. URL `https://www.topografix.com/gpx.asp`.

[11] Aimo Törn and Antanas Žilinskas. *Global optimization*. Springer, 1989.

Acknowledgements

We are grateful to:

- Academic lyceum "Physical-Technical High School" named after Zh. I. Alfyorov for the opportunity to conduct research activities included in the school curriculum
- O-GPS Center for methodological support
- JetBrains for the Kotlin programming language and the IntelliJ IDEA development environment