Application of the Paige Saunders Kalman Filter (PSKF) to ATLAS data in R

Dear Atlas Users,

Here is an in-depth look into the application of the Paige-Saunders Kalman filter to ATLAS data in R.

Materials:

You can find the base code on our Github page:
https://github.com/ATLAS-HUJI/R/tree/master/PSKF (R version)
https://github.com/ATLAS-HUJI/Matlab/tree/master/PSKF (Matlab version)

Both include the algorithm, a basic example implementation and some sample data to test the example implementation.

The R version also, includes a simple wrapper function **PSKFForATLAS** to which you pass your data, an example code **PSKF_SQLite_Example** that explains how to implement the wrapper function, and 6 examples of tracks showing a variety of behaviors and errors. These examples will be highlighted further down in the Result section to show you what you can expect from the filter in terms of accuracy and also show were some of the problems may lie.

Introduction:

For the sake of convenience I'll repeat here the basic introduction for the Kalman filter that I had already provided in my other filtering tutorial ***[please be aware that the following explanations are very much simplified]***

The Kalman filter [1] is the **gold standard in engineering** and produces vastly superior results to any other filtering approach. An implementation of a Kalman filter in position estimation is technically straight forward, provided we know how the system in question operates. Given a position estimate and a prior movement or so called action command we can make predictions about the likely position (given the known movement) and compare this with the current position.

This of course complicates things, as the systems, we as biologists research, are not fully understood and are also not controlled systems. In a implementation for localization of a plane for example we know:

- the position provided by a positioning system
- the thrust and heading direction of the plane.

While the ATLAS system provides us with the former the later is not known to us. Thus inherently any prediction has to be based solely on past movement data, introducing artificial correlations and linearities into the filtered data.

The current implementation is an **adaptation by Paige and Saunders [1977] utilizing known co-variance matrices** (thanks to Sivan for finding this one, and also writing the Matlab code) allowing us to circumvent some of these issues, by essentially filtering based on the ATLAS system and not only the unknown system i.e. the animal and provides smoothing and basic interpolation of missing data. Still issues remain and anyone using the filter should be aware of these issues:

- the Kalman filter assumes linearity (and almost nothing in the real world is linear)
- the Kalman filter assumes a Gaussian error distribution (the error in the ATLAS system does not follow a strict Gaussian error distribution)

Results:

All 6 tracks that are going to be discussed are taken from our bat data set of over 1000 individual track segments. Each original ATLAS track has been segmented carefully by hand (thanks to David Shoami) and represent the flight of a single Bat from a source location to a goal location. From these 1000 segments I manually selected a subset of 6 examples, because the bat displayed intricate flight behaviors, or because there were apparent problems in the recording. Given the present segmentation there are no problems (in theory) to pass on the track in its entirety to the wrapper function and filter all localizations in one go. Furthermore it should be noted that in our experience the bat data are generally among the best data we collect and thus may represent an somewhat ideal case. Yet, by purposely selecting faulty data and complex flight behaviors I hope to be able to show you, just how powerful the PSKF can be. I plan on looking at a more difficult example, without inherent data segmentation in the next tutorial.

The 6 track segments can be roughly separated into **3 main categories**:

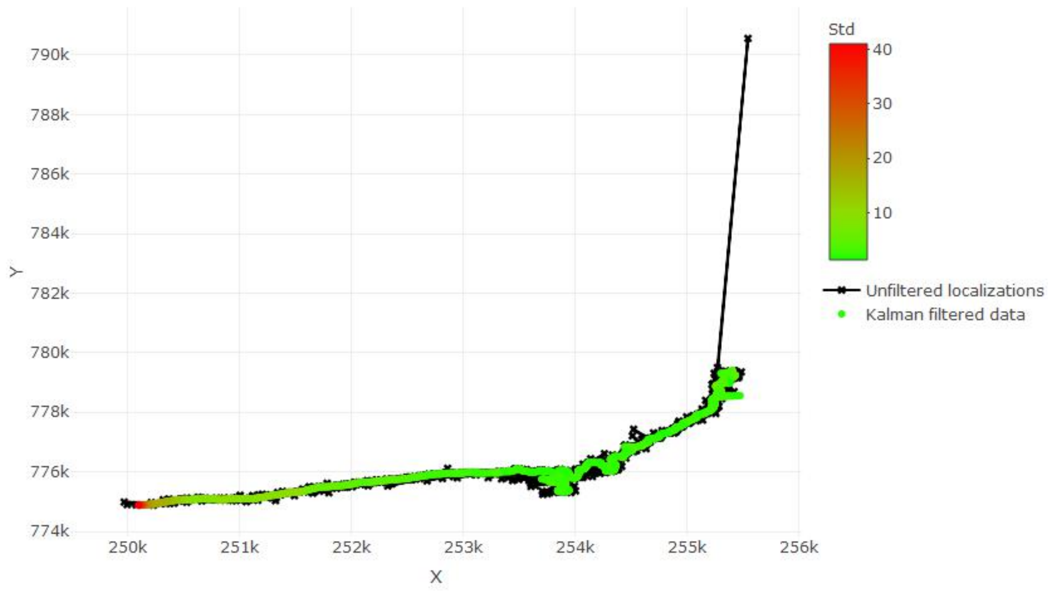
- Complex - tracks displaying complex flight behaviour
- Error - tracks including large out of place positions
- Jitter- tracks including smaller but repeatedly occurring errors

And **2 sub categories**:

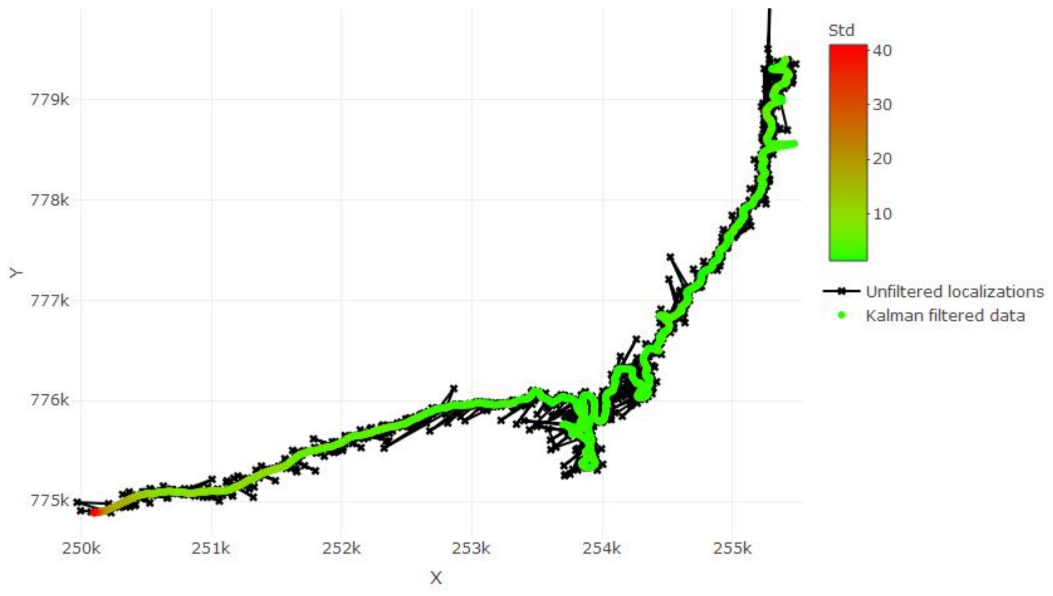
- Full - mostly complete tracks with only small or no gaps in the recording
- Sparse - tracks with gaps in the recording

Case: Complex- Full

A



B



C

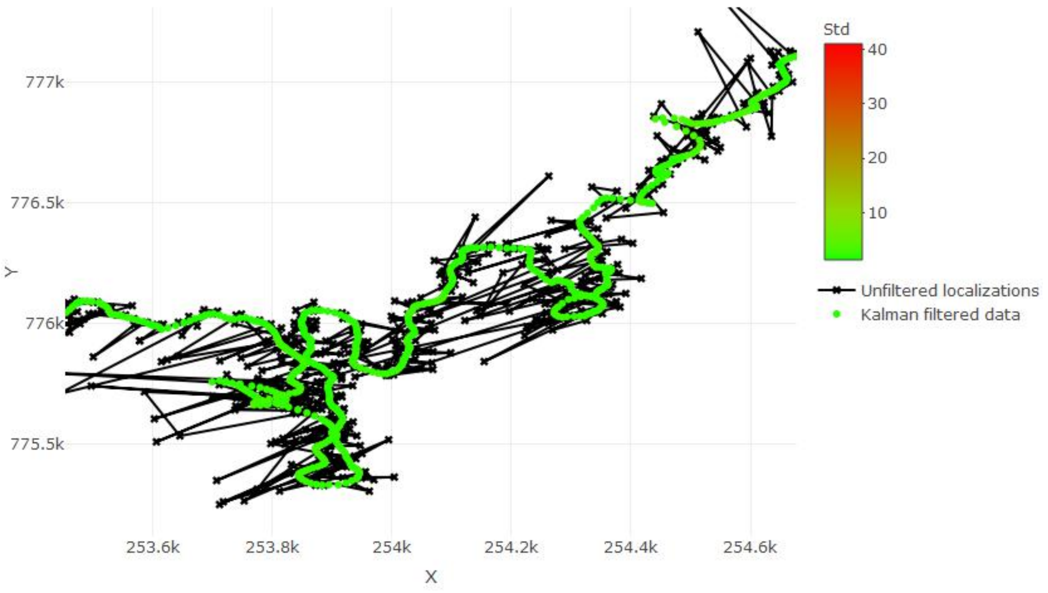
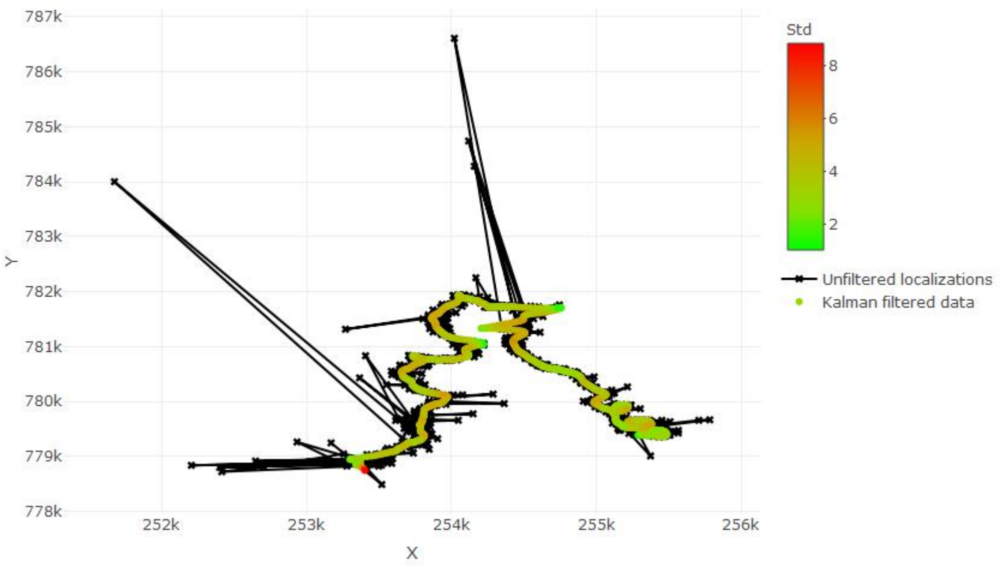


Fig 1A-C. Showing the original track (black lines with crosses marking the actual localizations) and the result of the PSKF (circles), with the colors ranging from green to red indicating the standard deviation, as estimated from the covariance matrices returned by the PSKF. A: Full track, B: track excluding the final outlier position, C: Detailed view of the complex flight pattern produced by the Bat in the middle section of the track.

As can be seen from Figure 1 there is quite high standard deviation in the filtered track at the beginning. This is to be expected because the filter needs some time to initialize and at that stage does not have any movement history to rely upon. Also obvious is that the Kalman filter is completely ignoring the far outlier at the end of the track and as shown in the detailed vied (Figure 1C) faithfully reconstructs the complex path of the Bat producing results with very low error.

Case: Complex-Sparse

A



B

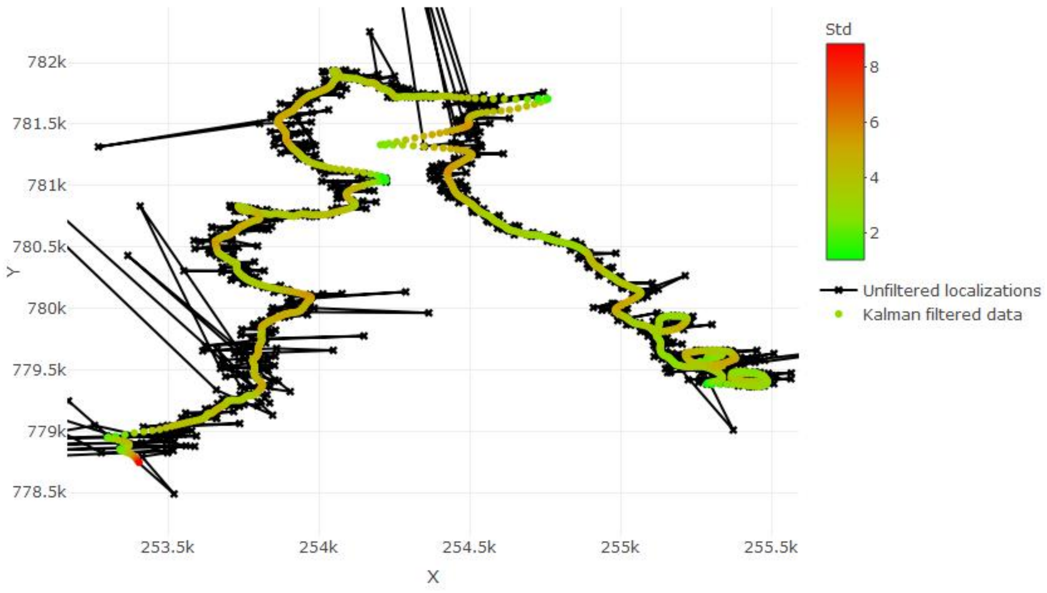


Fig 2A-B. Showing the original track (black lines with crosses marking the actual localizations) and the result of the PSKF (circles), with the colors ranging from green to red indicating the standard deviation, as estimated from the covariance matrices returned by the PSKF. A: Full track, B: Zoomed in view of the track.

Figure 2(A and B) provide a look at a similarly complex track with gaps in the recording. Despite the flawless performance throughout the majority of the track, rather large gaps in the middle section of the track produce a somewhat strange flight pattern, with large deviations from the the straight path. Although the PSKF produces a track with extremely low standard deviation in that section this part looks artificial (and is most likely based solely on past movement data).

Case: Error-Full

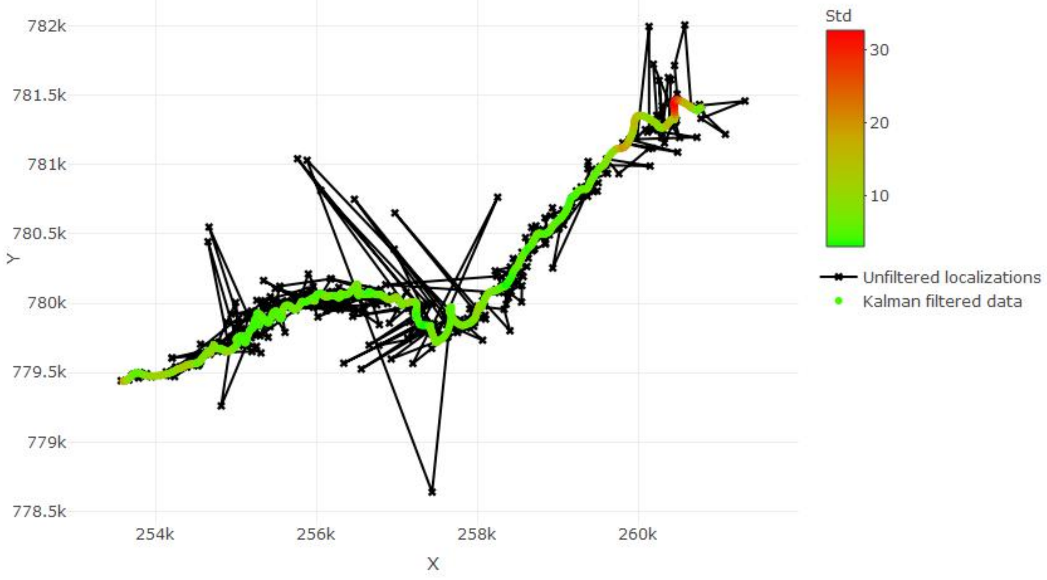
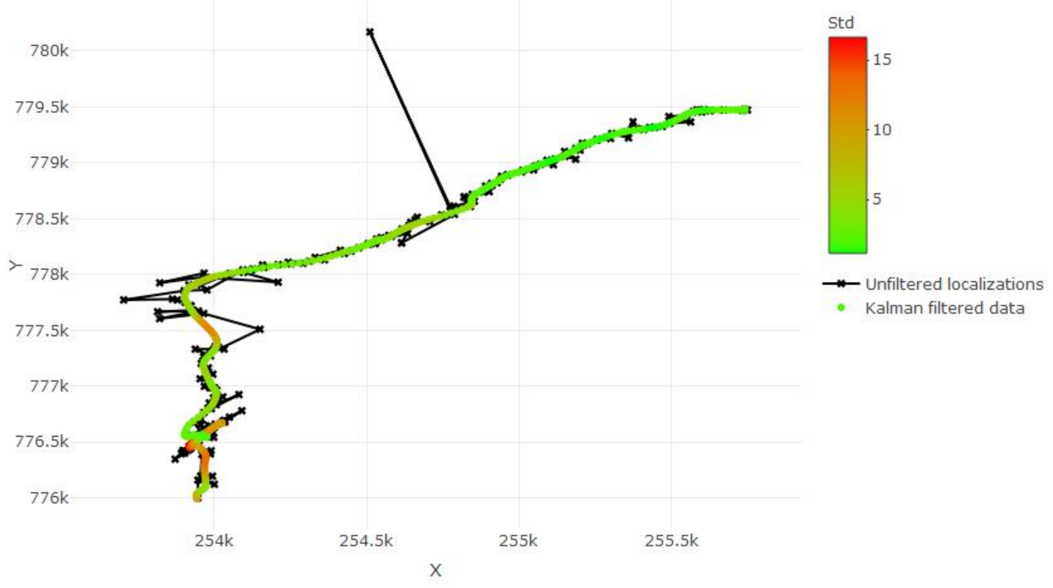


Fig 3. Showing the original track (black lines with crosses marking the actual localizations) and the result of the PSKF (circles), with the colors ranging from green to red indicating the standard deviation, as estimated from the covariance matrices returned by the PSKF.

Figure 3 shows an example track with several large errors in localization, which marginally affect the standard deviation at the beginning of the track. Especially the middle section of the track sees a lot of errors ,but for the most part the resulting filtered track remains relatively unaffected by these errors..

Case: Error-Sparse

A



B

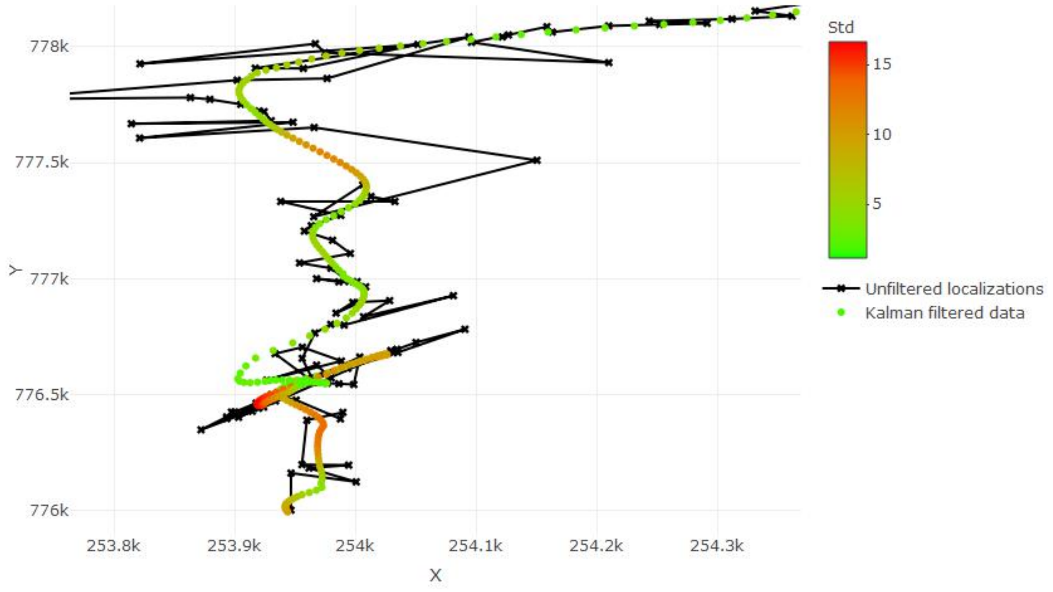
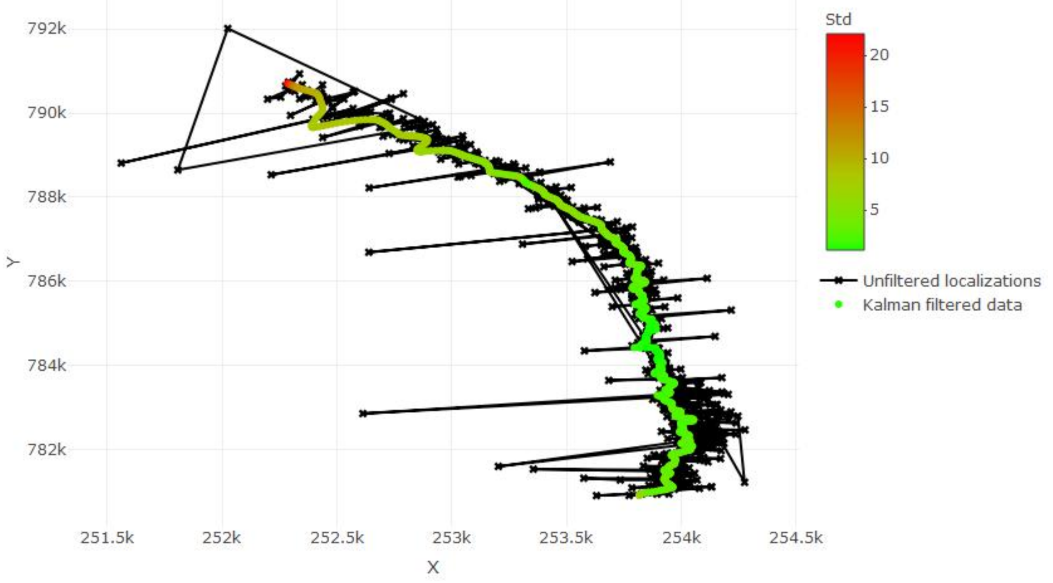


Fig 4A-B. Showing the original track (black lines with crosses marking the actual localizations) and the result of the PSKF (circles), with the colors ranging from green to red indicating the standard deviation, as estimated from the covariance matrices returned by the PSKF. A: Full track, B: Detailed view of the complex flight pattern produced by the Bat in the beginning section of the track.

Figure 4A shows a sparse example of a track with errors. As can bee seen especially in the beginning section (Figure 4B) of the track where there are several larger gaps, the track appears somewhat artificial, because the filtered data primarily consist of predicted positions by the PSKF.

Case: Jitter-Full

A



B

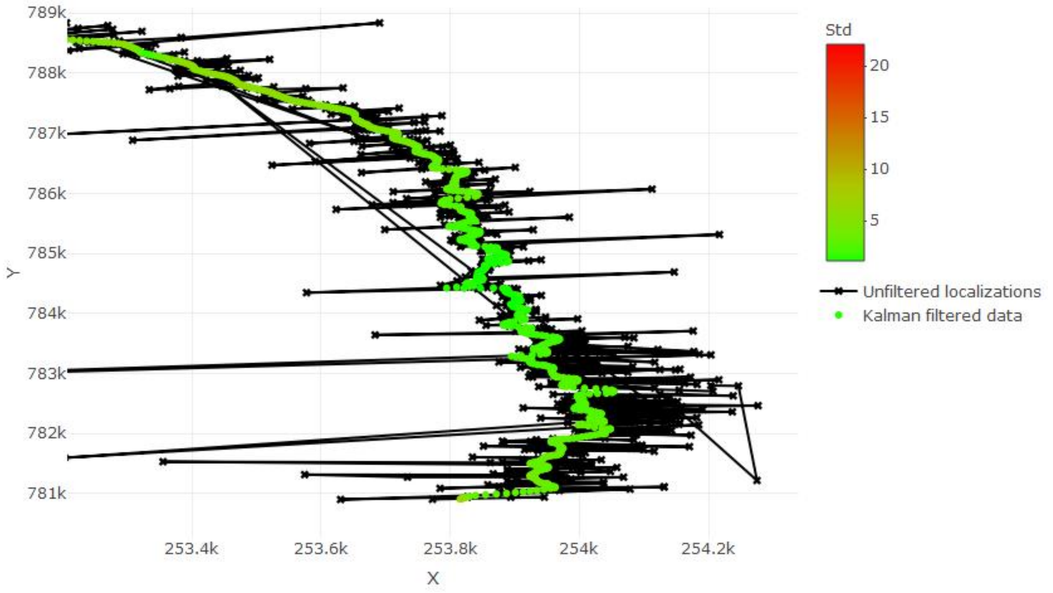
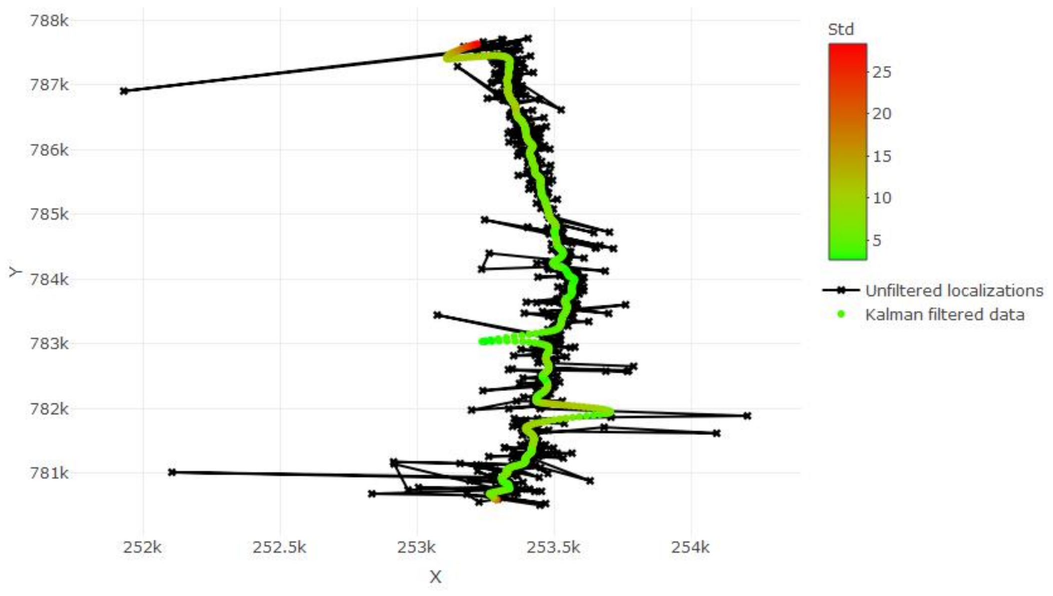


Fig 5A-B. Showing the original track (black lines with crosses marking the actual localizations) and the result of the PSKF (circles), with the colors ranging from green to red indicating the standard deviation, as estimated from the covariance matrices returned by the PSKF. A: Full track, B: Zoomed in view of the track.

Figure 5A shows a full example of data with repeated smaller errors, while the majority of the resulting track from the PSKF appears fine, there are several minor features in the filtered version that appear artificial (see Figure 5B).

Case: Jitter-Sparse

A



B

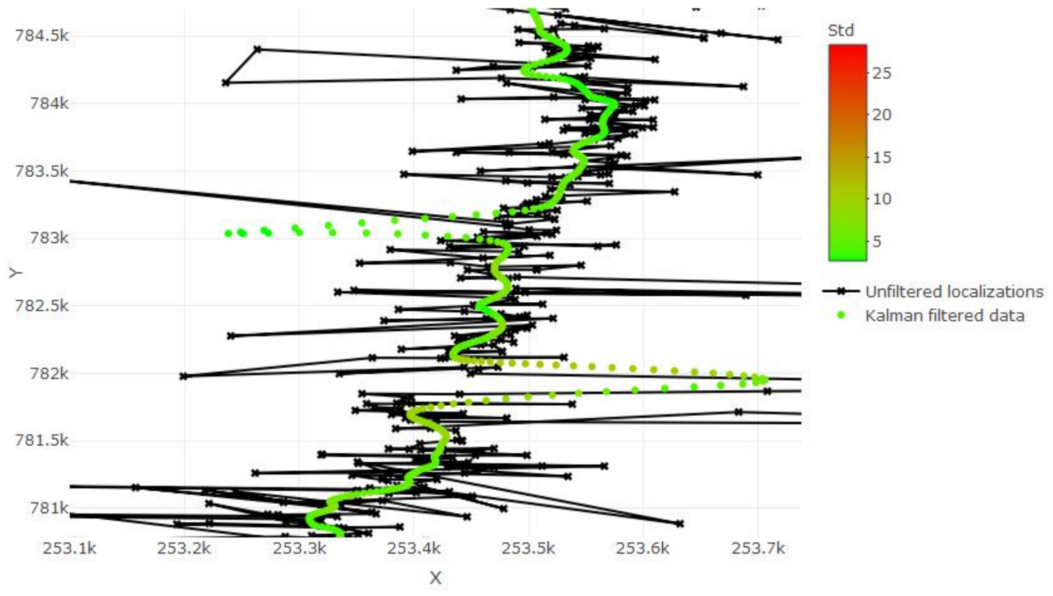


Fig 6A-B. Showing the original track (black lines with crosses marking the actual localizations) and the result of the PSKF (circles), with the colors ranging from green to red indicating the standard deviation, as estimated from the covariance matrices returned by the PSKF. A: Full track, B: Detailed view of the complex flight pattern produced by the Bat in the final section of the track.

Figure 6A shows a sparse example of data with repeated smaller errors in combination with several larger gaps in the recording. Again the majority of the resulting track from the PSKF appears fine, but there are several larger artifacts in the track resulting from the consistent and repeated errors in the localizations in combination with missing data (see Figure 5B).

Running Window Implementation

Although preferable it may sometimes not be easy to segment the data before filtering. In such cases a running window implementation can help (see

PSKF_SQLite_RW_Example.R). In this section I will take brief look at the differences and problems that may arise from a running window implementation comparing the data of full implementation to a running window implementation. For this purpose I am going to look again at the first case (complex-full) and compare results from both implementations. The running window was defined as a 60 second interval and each filtering step was re-initialized 15 seconds prior to the next step (the data from which was discarded later). The first difference is the number of resulting samples, this is of course because the the Kalman filter may get initialized during a gap. The second difference is a slight decrease in standard deviation from an average standard deviation of 3.88 to 3.76. This is of course because of the shorter interval and less history the Kalman filter can better fit the data, even though the fit may actually be slightly inferior. On average the results appear visually very similar (see Figure 7)

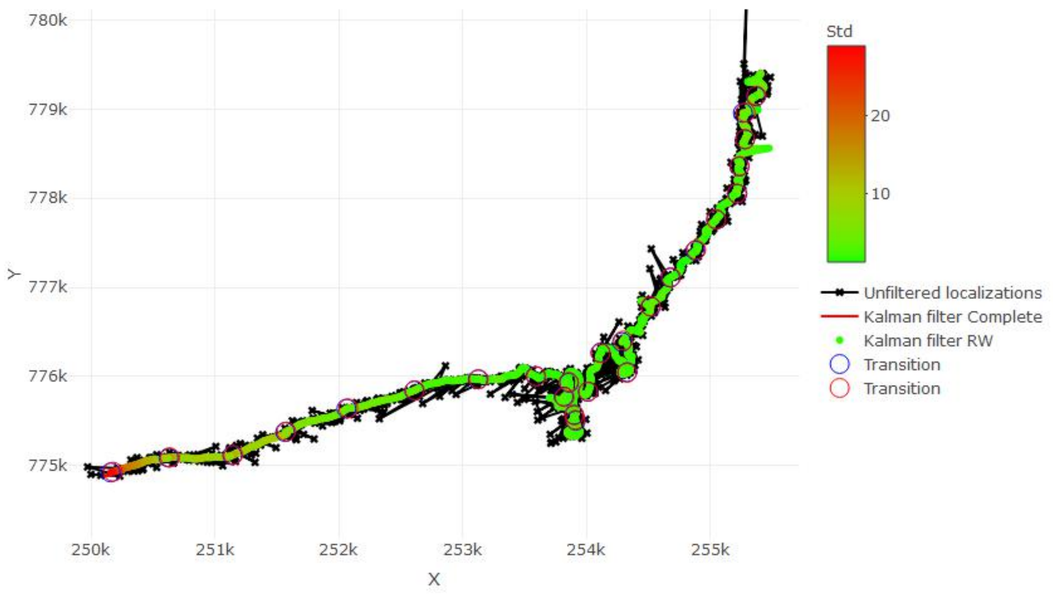
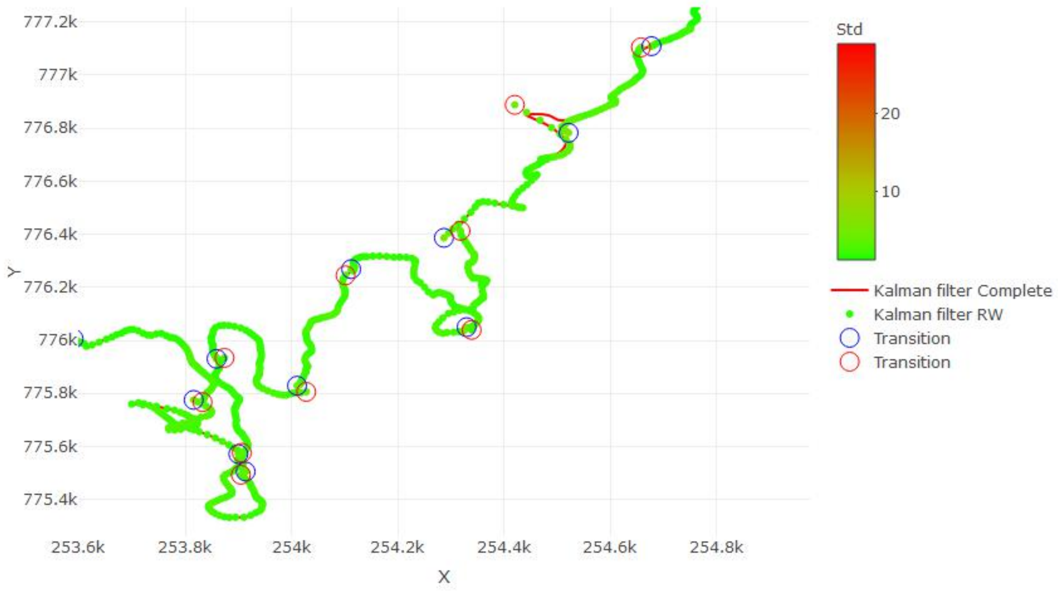


Fig 7. Direct comparison between running window and complete implementation. The Complete implementation (red line) is not visible because on this scale both implementations are virtually identical. The red and blue open circles mark the transition between consecutive filtering steps.

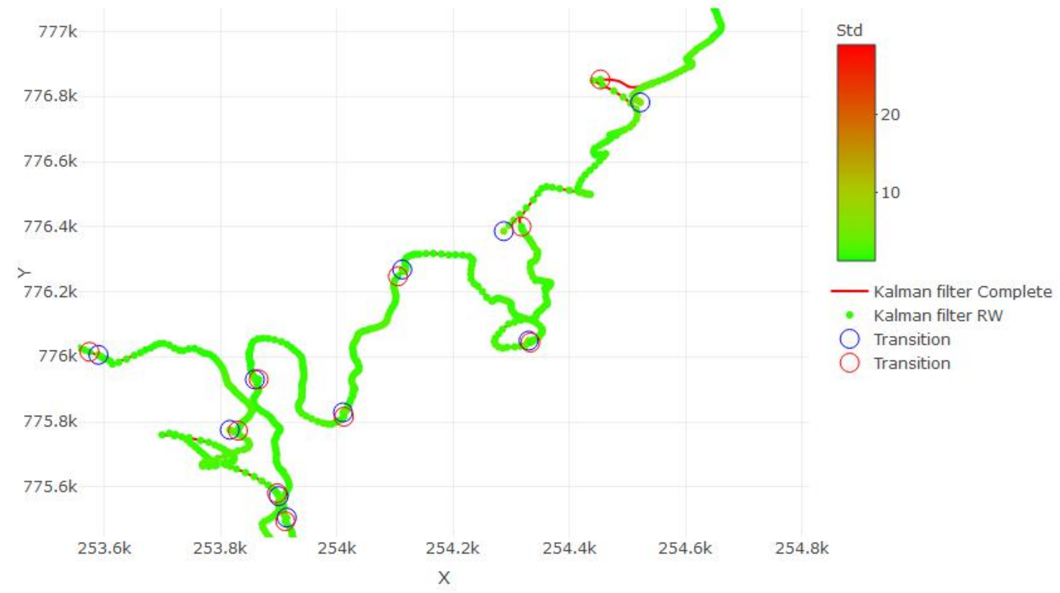
The actual problem with this particular implementation only becomes visible when we look at the points of transition. When one filtering step ends and a new one begins inconsistencies start to appear (See Figure 8 A-C). While the running filter implementation quickly assumes the same shape of the complete implementation after roughly 2-3 steps, the start and end points are clearly of the path prescribed by the full implementation and the majority of the points generated by the running window implementation. This can of course be alleviated to a degree by increasing the overhead in the calculation and allowing for longer initialization, but comes at the cost of longer run times and as can bee seen from figure 8C even when setting the initialization window to the full length of the behavioral time window the improvements compared to a short initialization window are negligible and some transitions remain problematic.

A



777.2k
777k
776.8k
776.6k
776.4k
776.2k
776k
775.8k
775.6k
775.4k
Y
253.6k
253.8k
254k
254.2k
254.4k
254.6k
254.8k
X
Std
20
10
Kalman filter Complete
Kalman filter RW
Transition
Transition

B



777k
776.8k
776.6k
776.4k
776.2k
776k
775.8k
775.6k
Y
253.6k
253.8k
254k
254.2k
254.4k
254.6k
254.8k
X
Std
20
10
Kalman filter Complete
Kalman filter RW
Transition
Transition

C

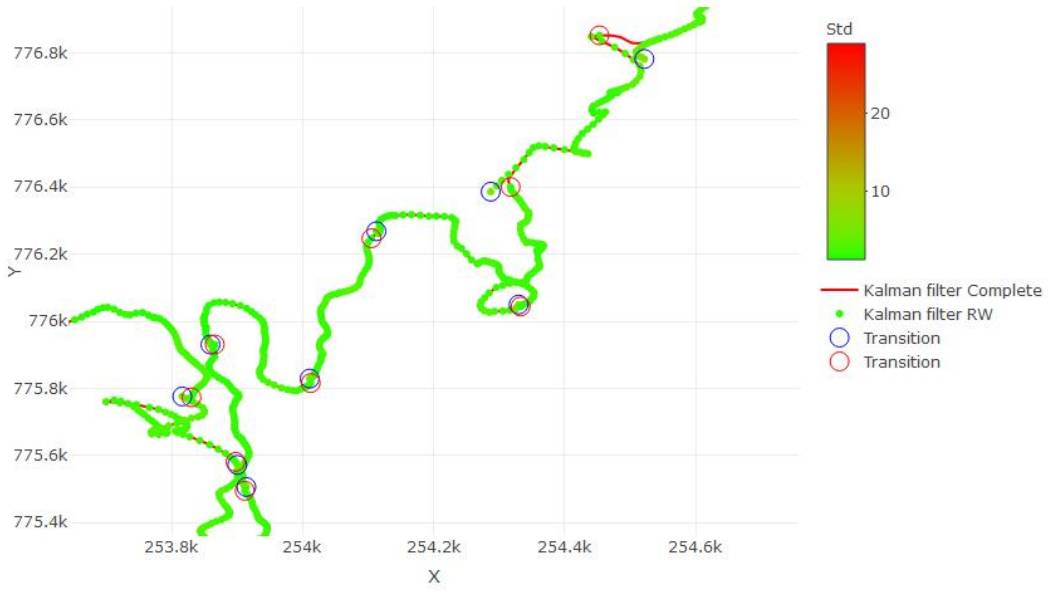


Fig 8 A-B. Running window implementation produces minor inconsistencies at the points of transition. A: No initialization, B 15 second initialization and C 60 second initialization.

Two stage Kalman Filter:

This may appear strange at first glance, but there is theoretically nothing wrong with filtering your data twice. Theoretically both filtering stages are methodically the same, but because the covariance matrix provided by the ATLAS system is intrinsically dimensionless and the covariance matrix returned by the PSKF is in meters,both stages are actually quite different.

Also by applying the filter twice we can theoretically remove uncertain positions from the second filtering step and potentially gain more accurate position estimates, The fist step is then to remove the predicted positions (we shouldn't necessarily trust them) and the second step would be to remove any position that exceed the threshold of the desired standard deviation. It should be noted that the setting a threshold will most likely force discard some of the positions at the beginning and end of the track and finding a good threshold value may practically be quite difficult (it is certainly impossible for the examples presented here, as they are actually quite good). High thresholds may have no effect at all and Low thresholds may cause more portions of the track to be based solely on predicted movement introducing even larger artifacts, but even without the filtering by standard deviation some improvements can be seen (See Figure 9).

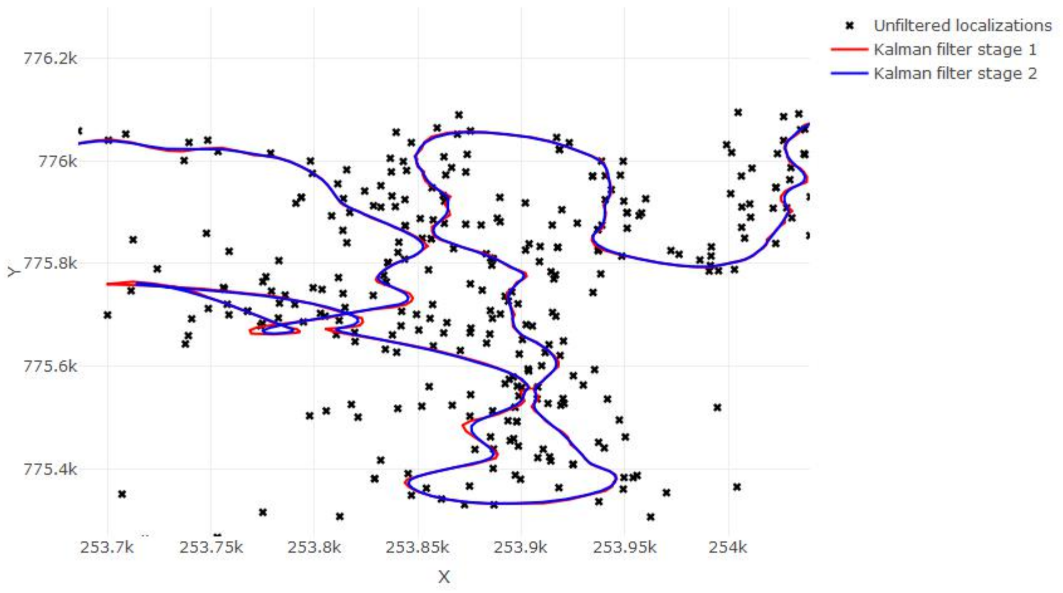


Fig. 9. Differences between single stage (red) and two stage(blue) PSKF.

Another use of a second filtering stage is to use it to smooth over the rough transitions introduced by a running window implementation by offsetting the transitions relative to the first filtering stage (see Figure 10).

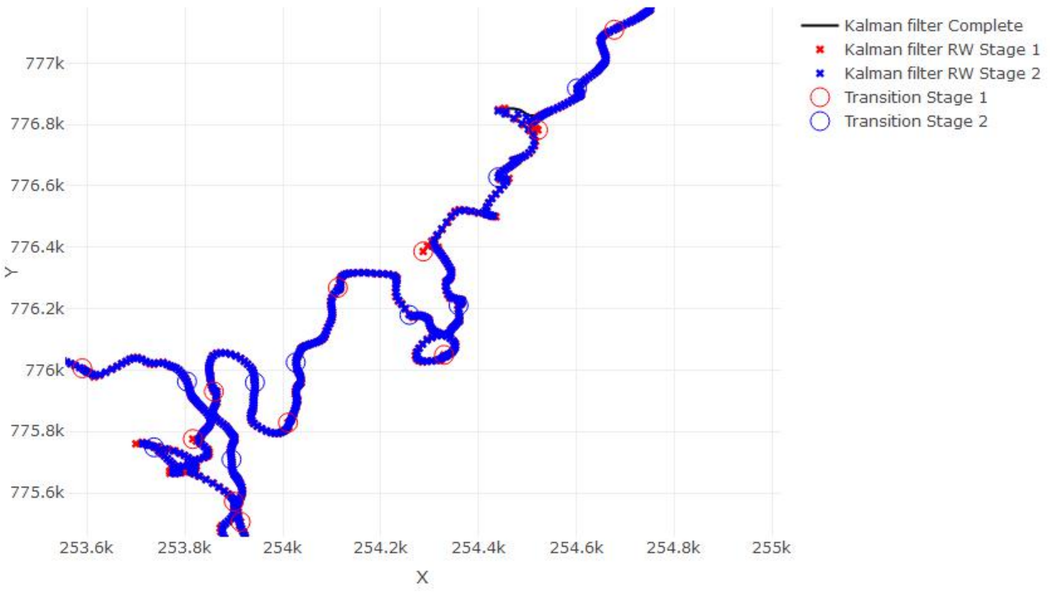


Fig.10. Comparison between single stage running window implementation and two stage running window implementation. The black line shows the result of the complete pass, the red crosses the single stage implementation and the blue crosses the two stage implementation the red and blue open circles show the transitions of the respective stages.

Summary:

As shown by the 6 examples the PSKF can deal with large localization errors and reconstruct very intricate flight patterns, even if there are some relatively large errors. All the while producing new localizations with average precision of around +/- 5m. It can also relatively well deal with repeated errors in consecutive positions, although this may lead occasionally to the introduction of minor artifacts into the track. Problems primarily arise when data are missing. Assuming these missing segments are short and their presumed track remains simple, then the algorithm can fill these gaps based on past movement data. Large errors with large gaps or repeated small errors in combination with smaller (or larger) gaps can, however, lead to the introduction very large artifacts.

I Additionally explored an implementation using a running window with variable initialization periods. While the data used here was actually already segmented I used it to highlight the differences in the two approaches and the problems that can arise from a running window implementation, namely the introduction of further artifacts. But the approach also shows how important proper segmentation or the implementation of a behavioral time window is in the first place, as the PSKF retains quite a lot of past data that would negatively affect filtering of data over several behavioral time scales.

The data presented here are obviously not ground truth-ed, but the errors in the original data are, I think, blatantly obvious, as are the artifacts, as they are not backed up by any original positions. Aside from that the artificial nature of these segments has been verified (not shown here) by systems analysis, with results clearly indicating that these portions are governed not by the behaviour of the animal, but by the PSKF. Finally, as mentioned in the beginning of the result section the data here, while taken from our bat data, represent exceptionally extreme accumulation of errors, showing just how robust the PSKF is.