Detection of Biases and Faults in Navigation Sensor Measurements

Sasha Draganov
Expedition Technology, Inc.
45195 Business Court, Suite 450
Dulles, VA 20166

Abstract- A modern navigation system integrates data from different sensors that have different error statistics, including biases. For example, inertial measurement units (IMUs) are known to have biases and other errors, and it is common to allocate states in the filter for estimating such errors. However, models for such biases are typically based on some assumptions about error statistics, such as a random walk for the bias magnitude. When these assumptions are incorrect, a new approach is required. Measurements from a barometric altitude sensor present an example: a bias may change if a user steps in or out of a building, or if a forced-air heating system is turned on/off while the user is in a building. In cases like this, a bias may exhibit a jump. In another example, one may need to watch for faults in the data that may occur due to equipment malfunction.

If the magnitude of a measurement error is large enough, a measurement fault can be detected in a single epoch by using measurement redundancy. An example of such algorithm is the Receiver Autonomous Integrity Monitor (RAIM), which is widely used in GPS measurement processing. However, biases in the measurements can be small enough to make their detection in a single epoch difficult. In this paper, we present an algorithm for detecting hidden biases in the measurements. Unlike the RAIM algorithm, we use redundancy not just across measurements in a single epoch, but also over time. By leveraging time history, we are able to detect small, otherwise hidden biases and faults in sensor measurements.

This approach requires that the navigation system employs an IMU or some other time update mechanism, which "stitches together" the user state estimates over time. The algorithm runs several independent processing threads, comparing user states with different sensors (or channels) excluded. The key challenge in this approach is to minimize the computational load, so that processing requirements scale less than the number of threads. We achieve this goal by observing that most computations in the filter are common across all threads and can be recycled.

As the result of the computation, we arrive at bias/fault estimates for each sensor (or a measurement channel). Obviously, the desirable value for a measurement error is close to zero, and the decision on the existence of a bias or a fault in a sensor must be made based on two pieces of data: (1) a bias/fault estimate and (2) an uncertainty of this estimate. The latter requires a separate computation; again, the key for its use in real applications is to minimize the processing requirements. We derive an analytical recursive formulation, which updates the uncertainty for fault estimate sequentially, using minimum resources. With the fault estimate and its uncertainty in hand, a decision on fault existence can be made using a chi-square test or a similar approach.

These algorithms have been implemented and tested on simulated and on real data. We present results that illustrate detection of small, hidden faults in the data.

This research was sponsored by DARPA (contract number HR0011-15-C-0045). The views, opinions, and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for public release. Distribution unlimited.

I. Introduction

To borrow a phrase, a data fusion engine is able to account for measurement errors if these errors are "known knowns", but remains vulnerable to "unknown unknowns". Specifically, hidden biases in the data are dangerous because they are exacerbated, and not mitigated by processing more data.

The challenge in detecting a bias is that even if a comparison between sensor measurements and the filter solution shows a persistent, biased residual, there is no way to know if the problem resides in the sensor or in the solution (or both). For example, a fault in one sensor may skew the solution, producing biases in measurement residuals for other sensors.

The only way to attribute a bias to a particular sensor is to compare that sensor's measurements to a filter solution that was obtained without using that sensor. Essentially, this is the path taken by the well-known RAIM algorithm for GPS: it computes a position using all but one channel and repeats this computation in a round robin sequence for all channels [1, 2].

We extend this basic idea by: (1) generalizing it to nearly any sensor and (2) applying it over time to detect biases that may be too small to be detected in a single epoch. (In fact, the very notion of a "bias" is statistical; a single deviation of a measurement from the expected value is classified as an outlier or as occurring due to measurement noise.) This approach requires that the navigation system employs an IMU or some other time update mechanism, which "stitches together" user state estimates over time. The general idea of leveraging propagation to detect biases was also used in [3]. Our algorithm runs several independent processing threads, comparing user states with different sensors (or channels) excluded. The key challenge in this approach is to minimize the computational load, so that processing requirements scale less than the number of threads. We achieve this goal by observing that most computations in the filter are common across all threads and can be recycled.

As the result of the computation, we arrive at bias/fault estimates for each sensor (or a measurement channel). Obviously, the desirable value for a measurement error is close to zero, and the decision on the existence of a bias or a fault in a sensor must be made based on two pieces of data: (1) a bias/fault estimate and (2) an uncertainty of this estimate. The latter requires a separate computation; again, the key for its use in real applications is to minimize the processing requirements. Below, we derive an analytical recursive formulation, which updates the uncertainty for fault estimate sequentially, using minimum resources. With the fault estimate and its uncertainty in hand, a decision on fault existence can be made using a chi-square test or a similar approach.

Before proceeding further, we need to differentiate between biases of various types. A modern navigation system integrates data from different sensors that have different error statistics, including biases. For example, inertial measurement units (IMUs) are known to have biases and other errors, and it is common to allocate states in the filter for estimating such errors. However, models for such biases are typically based on some assumptions about their statistics, such as a random walk for the bias magnitude. On the contrary, measurements from a barometric altitude sensor often contain a bias, which may change if a user steps in or out of a building, or if a forced-air heating system is turned on/off while the user is in a building. In cases like this, a bias may exhibit a jump. It is difficult to account for such bias by adding a state in the filter, which is the motivation for developing a new method for detecting it.

An example of detrimental bias effects in navigation is described in Jules Verne's novel "Dick Sand, A Captain at Fifteen". A plot is hinged on an episode when at night a villain has put a piece of iron under a ship's compass to change the course. This created a bias in the heading measurements, and the ship ended up in a wrong place, with many adventures to follow. Fig. 1 shows a quote from this novel and an etching by Henri Meyer for this episode.

Figure 1. "In fact, it was a piece of iron, whose influence had just altered the indications of the compass. The magnetic needle had been deviated, and instead of marking the magnetic north, which differs a little from the north of the world, it marked the northeast. It was then, a deviation of four points; in other words, of half a right angle." Jules Verne, Dick Sand, A Captain at Fifteen.

II. Measurement Exclusion Algorithm

Our bias detection algorithm compares results from multiple threads: a default thread that uses all sensors and other threads, each with data from one sensor excluded. This exclusion is defined in a particular way: it is if a measurement is processed with a zero residual. In this case, the state vector is not updated, but the covariance matrix is. This approach allows us to compare data from different threads in the "apples to apples" way.

If no bias is detected, the algorithm defaults to the filter output that uses all available sensors; if a bias is present, data from the offending sensor are excluded from processing.

The standard Kalman filter measurement update for state x is as follows:

$$\hat{x} = \tilde{x} + K\Delta z \tag{1}$$

where K is the Kalman gain, and $\Delta z = z - H \cdot \tilde{x}$ is a measurement residual.

If a measurement is skipped from processing in one thread, but is processed in another, the difference in the state between these threads is given by $\Delta x = \hat{x} - \tilde{x} = K\Delta z$. For every subsequent measurement update (whether from the same or a different sensor), this difference is scaled by the following multiplier:

$$\Delta\vec{x}_n = \left(I - K_n H_n^T\right)\Delta\vec{x}_{n-1} \tag{2}$$

(Compare this to the standard Kalman filter equation for the covariance update: $\tilde{P}_n = \left(I - K_n H_n^T\right)\hat{P}_{n-1}$).

If multiple measurements from a particular sensor are skipped from processing over time, the combined effect is as follows:

$$\begin{aligned}\Delta\vec{x}_n &= \vec{K}_n \Delta z_n \\ &+\Phi_n \prod_m \left(I - K_{n,m}H_{n,m}^T\right) \cdot K_{n,m}\Delta z_{n-1} + \\ &\Phi_n \prod_m \left(I - K_{n,m}H_{n,m}^T\right) \cdot \\ &\Phi_n \prod_m \left(I - K_{n,m}H_{n,m}^T\right) \cdot K_{n,m}\Delta z_{n-1} \\ &+ \cdots = \vec{K}_n \Delta z_n + \alpha_n \vec{K}_{n-1}\Delta z_{n-1} \\ &\quad +\alpha_n \alpha_{n-1}\vec{K}_{n-2}\Delta z_{n-2} + \cdots\end{aligned} \tag{3}$$

where Φ_n is the matrix for the propagation update, and where $\alpha_n = \Phi_n \prod_m \left(I - K_{n,m}H_{n,m}^T\right)$. It is clear from the last line in Equation (3), that this equation can be updated sequentially, without tracing back all previous time steps and without doing all the bookkeeping for it. Indeed, two successive estimates for $\Delta\vec{x}_n$ are connected as follows:

$$\begin{aligned}\Delta\vec{x}_n &= \vec{K}_n \Delta z_n + \\ &\Phi_n \prod_m \left(I - K_{n,m}H_{n,m}^T\right) \cdot \Delta\vec{x}_{n-1}\end{aligned} \tag{4}$$

Equation (4) shows that we do not need to do all of the cumbersome computations in Equation (3); rather we can update $\Delta\vec{x}$ sequentially. Moreover, if the fusion engine uses a Kalman filter, all quantities in Equation (4) are already computed in the course of the normal Kalman filter processing, and can be applied to update $\Delta\vec{x}$. At each step, we only need to apply a previously computed matrix for the covariance update, and add a new term from the new measurement. This greatly reduces the computation load and makes this fault detection algorithm economical.

III. Small Bias Detection

Our goal is to detect hidden biases, which are not necessarily evident from a single measurement residual. We achieve this goal by processing the residuals in question over time. The estimate for the measurement bias b_q^n is as follows:

$$b_q^n = \alpha b_q^{n-1} + (1-\alpha)\Delta z_q^n \tag{5}$$

where q is the sensor id; Δz_q^n is the *a priori* residual for measurement from this sensor with respect to the thread, which excludes data from sensor q; n is the epoch; $0 < \alpha < 1$ is a parameter. In practice, values $1 - \alpha << 1$ will smooth results over longer time, enabling us to detect hidden biases. If the value of estimated b_q^n is non-zero in the statistical sense for measurements from sensor q only, then measurements from this sensor contain a bias.

The last statement requires us to define a criterion for b_q^n being statistically non-zero. We do this by computing the variance for b_q^n. This computation is complicated by the fact that b_q^n and b_q^{n-1} are correlated. The rest of this section provides a derivation for the variance of b_q^n.

We square (5) and compute the mathematical expectation of the result to get:

$$\begin{aligned}\overline{\left(\delta b_q^n\right)^2} &= \alpha^2 \overline{\left(\delta b_q^{n-1}\right)^2} + \\ &(1-\alpha)^2 \overline{\left(\Delta z_q^n\right)^2} + 2\alpha(1-\alpha)\lambda_{q,1}^n\end{aligned} \tag{6}$$

where we denote $\lambda_{q,m}^{n} = \overline{\delta b_q^{n-m} \Delta z_q^{n}}$; $\delta b_q^n = b_q^n - \overline{b_q^n}$; and $\overline{\ldots}$ is the mathematical expectation.

This equation can be used iteratively for updating the value of bias variance $\overline{\left(b_q^n\right)^2}$. The only difficulty is to compute $\lambda_{q,1}^{n}$,

$$\lambda_{q,1}^{n} = \overline{\delta b_q^{n-1} \Delta z_q^{n}} = \overline{\left(\alpha b_q^{n-1} + (1-\alpha)\Delta z_q^{n-1}\right)\Delta z_q^{n}} = \alpha\lambda_{q,1}^{n} + (1-\alpha)\kappa_{q,1}^{n} \tag{7}$$

which describes correlations between the previous bias estimate and the new residual. Next, we proceed to computing $\lambda_{q,1}^{n}$. The key idea is to do it recursively:

where we denote the term describing successive correlations in the residuals as $\kappa_{q,m}^{n} = \overline{\Delta z_q^{n} \Delta z_q^{n-m}}$. Residuals are correlated, because they are computed with respect to the user state, which is propagated from one epoch to another. This process propagates any errors in the state as well, producing correlations between epochs. We can represent these correlations by:

$$\Delta z_q^n = \rho_q^n \Delta z_q^{n-1} + \nu_q^n \tag{8}$$

where $\rho_q^n \Delta z_q^{n-1}$ is the scaled residual from the same sensor for the previous epoch, and ν_q^n is the uncorrelated part of the new residual (due to the measurement noise at the current epoch and all measurements from other sensors that have been processed between the measurements from sensor q at the previous and the current epoch). Then:

$$\begin{aligned} \kappa_{q,1}^{n} &= \rho_q^n \overline{\left(\Delta z_q^{n-1}\right)^2} \\ \kappa_{q,2}^{n} &= \rho_q^n \rho_q^{n-1} \overline{\left(\Delta z_q^{n-2}\right)^2} \\ &\ldots \end{aligned} \tag{9}$$

Substitution of (9) in (7) and applying (7) recursively yields:

$$\begin{aligned} \lambda_{q,1}^{n} &= (1-\alpha)\rho_q^n \overline{\left(\Delta z_q^{n-1}\right)^2} \\ &+\alpha(1-\alpha)\rho_q^n \rho_q^{n-1} \overline{\left(\Delta z_q^{n-2}\right)^2} + \\ &+\alpha^2(1-\alpha)\rho_q^n \rho_q^{n-1} \rho_q^{n-2} \overline{\left(\Delta z_q^{n-3}\right)^2} + \cdots \end{aligned} \tag{10}$$

By replacing indices $n \rightarrow n+1$, we get:

$$\begin{aligned} \lambda_{q,1}^{n+1} &= (1-\alpha)\rho_q^{n+1} \overline{\left(\Delta z_q^{n}\right)^2} \\ &+\alpha(1-\alpha)\rho_q^{n+1} \rho_q^{n} \overline{\left(\Delta z_q^{n-1}\right)^2} + \\ &+\alpha^2(1-\alpha)\rho_q^{n+1} \rho_q^{n} \rho_q^{n-1} \overline{\left(\Delta z_q^{n-2}\right)^2} + \cdots \end{aligned} \tag{11}$$

From comparing the last two equations, we can see that

$$\lambda_{q,1}^{n+1} = (1-\alpha)\rho_q^{n+1} \overline{\left(\Delta z_q^{n}\right)^2} + \alpha\rho_q^{n+1}\lambda_{q,1}^{n} \tag{12}$$

This equation allows us to update $\lambda_{q,1}^{n}$ sequentially. All quantities in the right-hand side are relatively easy to compute. From Kalman filter update equations, we get:

$$\overline{\left(\Delta z_q^n\right)^2} = \sigma_z^2 + \left(H_q^n\right)^T \tilde{P}^n H_q^n \tag{13}$$

To compute ρ_q^n, we use

$$\overline{\Delta z_q^n \Delta z_q^{n-1}} = \rho_n^q \overline{\left(\Delta z_q^n\right)^2} = \rho_n^q \left(\sigma_z^2 + \left(H_q^{n-1}\right)^T \tilde{P}^{n-1} H_q^{n-1}\right) \tag{14}$$

Then

$$\rho_q^n = \frac{\overline{\Delta z_q^n \Delta z_q^{n-1}}}{\sigma_z^2 + \left(H_q^{n-1}\right)^T \tilde{P}^{n-1} H_q^{n-1}} \tag{15}$$

We note that any error in the state scales with processing every measurement, whether from the sensor in question or from a different sensor. In addition, it scales during the state propagation (the time update step of the Kalman filter processing. Mathematically, this is expressed as follows:

$$\overline{\Delta z_q^n \Delta z_q^{n-1}} = \left(H_q^n\right)^T \prod_j \left(I - K_j^n H_j^n\right) \cdot \Phi_n \cdot \prod_l \left(I - K_l^{n-1} H_l^{n-1}\right) \tilde{P}^{n-1} H_q^{n-1} \tag{16}$$

where Φ_n is the time update transition matrix; the product $\prod_l \left(I - K_l^{n-1} H_l^{n-1}\right)$ is computed over all measurements, starting from (and including) measurement q during epoch $n-1$; and the product $\prod_j \left(I - K_j^n H_j^n\right)$ is computed over all measurements during epoch n, which are preceding measurement q). Even though it may seem that this is a cumbersome computation, it is easily implemented and is not computationally intensive, as all scaling multipliers are computed in the course of regular Kalman filter processing. Scaling is applied successively, as measurements are being processed.

IV. Simulation Results

With the smoothed value and an estimated variance for the bias computed, it is straightforward to implement a criterion to detect a bias. Simply speaking, we declare that a measurement is biased if the absolute value of the estimated bias exceeds a threshold, e.g. $M\sigma_b$, where M is a configurable multiplier (e.g., $M = 3$) and σ_b is the standard deviation for the bias (the square root of its estimated variance, as derived in the previous section). Since we detect a bias over multiple epochs, smoothing leads to a smaller variance, and even a small bias (in comparison to the measurement variance) can be detected. This section presents simulated results that illustrate this idea.

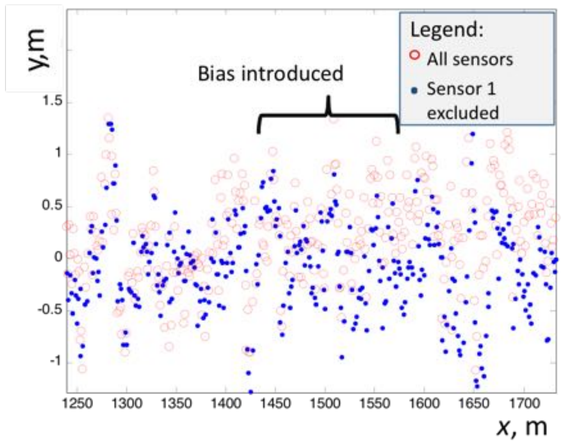

Figure 2. A segment of user trajectory as computed by two threads

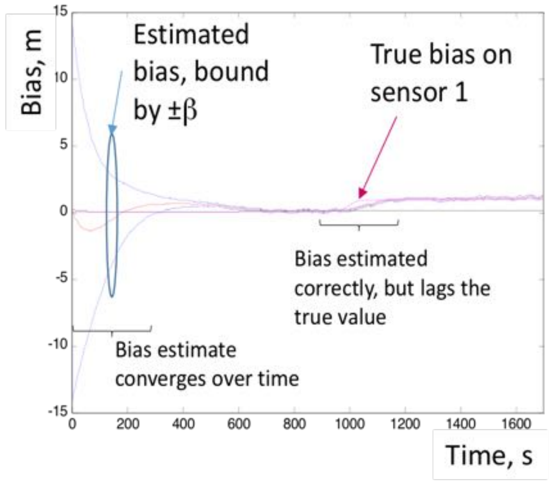

Figure 3. Simulated (red) and estimated bias (magenta). The ± standard deviation (blue) shows the convergence of the estimate.

The simulation scenario is deliberately simple: a user measures ranges to four distant RF sources (e.g., GPS satellites), randomly positioned in the upper hemisphere. The user trajectory is a straight line. The state is the user position and user velocity (6 states), and the propagation algorithm is an extrapolation of the current state using the velocity estimate. IMU is not modeled in this scenario. We set the user velocity to 1.5 m/s (a walking speed), the process noise variance for velocity to 10^{-2} m^2/s^2, and the measurement variance to 1 m^2. The smoothing parameter is set to $\alpha = 0.99$. Approximately at *t*=1000 s, measurements from Beacon 1 gradually acquire a bias of 1 m. This bias is on the order of the measurement noise, and would be difficult to detect using conventional methods.

Fig. 2 shows a segment of the trajectory approximately at the time when Sensor 1 acquired a bias. Positions in the (*x, y*) plane are shown as computed from two threads, one using all sensors, and another with Sensor 1 data excluded. Since the bias is small, there is no obvious sign of a bias.

Fig. 3 shows the results of bias estimation along with its standard deviation. After the initial period of convergence, the

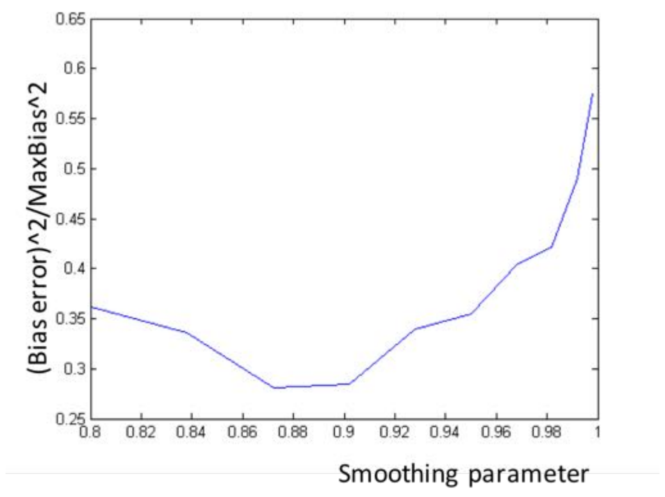

Figure 4. Bias estimation error as a function of the smoothing parameter

estimated bias (the red curve) follows the true value (magenta curve) with some lag.

The smoothing parameter α defines a trade-off between the robustness and the time lag for bias estimation. A larger value tallies data over longer time, hence a smaller bias can be detected, at the cost of taking it longer to detect. This is illustrated in Fig. 4, which shows the RMS bias estimation error as the function of the smoothing parameter. For smaller values (e.g., $\alpha = 0.8$), there is not enough data to estimate the bias accurately, and the bias estimation error is relatively large. For very large values (e.g., $\alpha = 0.99$), the lag in the bias estimate eats into the estimate, increasing the total RMS error. Yet, there is a sweet spot, where the bias is estimated relatively accurately and within an acceptable time.

V. Real Data Results

A performance for real data is always the key test of any new algorithm. We used the data, which were originally collected for the DARPA's ASPN program [4]. There are several datasets available, each with multiple sensors. Fortunately for other purposes and unfortunately for us, there were few occurrences of biases in the data. One exception is a particular GPS channel on a driving test, which we used for validation of the bias detection algorithm.

In the beginning of the test, the vehicle made two loops over several city blocks. Fig. 5 shows GPS residuals for one the satellites with respect to the true user position (GPS residuals are shown with magenta dots; please temporarily disregard other data shown in this figure, as they will be referenced later). We have removed the ionospheric delay, the mean clock error and the linear clock drift for this plot. Note that residuals exhibit several jumps, as if there is a bias that appears and

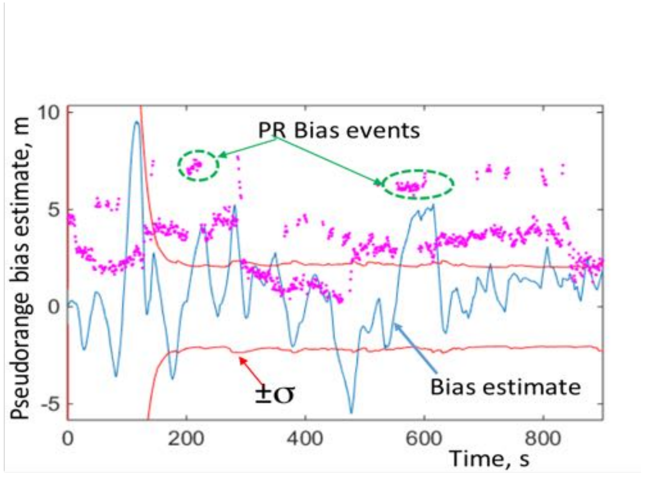

Figure 5. Pseudorange residuals (magenta), a bias estimate (blue) and ± standard deviation (red) in GPS data for a selected satellite..

disappears in the data. Below, we concentrate on two such events, which are circled in the figure.

In both cases, a bias of about 4 meters with respect to the ambient value appears and then stays for few tens of seconds. These events start when the vehicle stops at the same intersection and almost exactly at the same location. This may point to some local conditions at this particular location, which are the cause of the bias. Fig. 6 shows a segment of the vehicle trajectory (white line). The time periods when the pseudorange from a particular satellite was affected by bias are shown in green and in blue lines that overlap the white trajectory line. The azimuth towards the satellite in question is shown by a short red line. One can clearly see that the line of sight to the satellite is in the direction of a nearby building at both locations where the bias events have started. The satellite elevation here was 1.10 rad. It is not clear if the building has obstructed the line of sight to the satellite when the vehicle was stopped approximately at the viewpoint of this picture.

While the above argument on the bias origins is not definitive, one thing is clear: pseudorange measurements were affected by biases, and these biases switched on and off, likely due to effects of the local environment. This is exactly the type of faults, which our bias detection algorithm is designed to mitigate.

We ran the bias detection algorithm for these pseudorange GPS data. The results are shown in Fig. 5 along with the pseudorange measurement residuals. We can see that the bias estimate (the blue line) has spiked above 1σ threshold during both bias events.

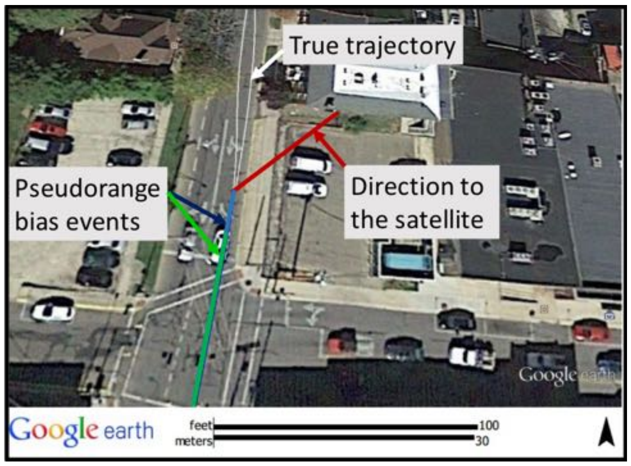

Figure 6. True trajectory (white), trajectory segments with bias events (blue and green) and the direction to the satellite, which is associated with a pseudorange bias (red).

The correspondence between the "detected" bias events and "true" ones is not one-to-one. There are several possible reasons for that:

1. "True" bias events are defined in the eyes of a beholder
2. Criteria for a detected bias should include a sigma multiplier threshold and the duration of the event
3. Bias in the measurements, if any, is in the single digit meters – quite small for pseudorange.

The last point speaks to the good performance of this algorithm: even relatively small biases can be detected.

VI. Conclusion

Since a Kalman filter is often defined as a minimum variance, unbiased estimate, it is clear that biases in the measurements may have a detrimental effect on filter output. Moreover, in a contrast to zero-mean noise, a bias is not averaged out by a filter. This creates a motivation for developing an algorithm that detects possible hidden biases in the measurements. Our algorithm reuses computations from, and is designed to be integrated with, a Kalman filter. This makes it computationally efficient. The idea can be traced to the RAIM algorithm; however, we exploit data over time, which makes it possible to detect relatively small biases.

VII. References

1. R. G. Brown et al., *GPS RAIM: Calculation of threshold and protection radius using chi-square methods - a geometric approach*, in Global Positioning System: Inst. Navigat., 1997, vol. V, pp. 155-179.
2. Parkinson, B. and Axelrad, P., "Autonomous GPS Integrity Monitoring Using the Pseudorange Residual," NAVIGATION, Vol. 35, No. 2, Summer 1988, pp. 255−74.
3. Cong, L., et al., A Fault Detection Method for GNSS/INS Integrated Navigation System Based on GARCH Model, Proceedings of the ION 2015 Pacific PNT Meeting, Honolulu, Hawaii, April 2015, pp. 713-718.
4. http://www.wired.com/2012/06/darpa-gps/