Topic 5: Kalman filtering & GNSS-INS integration

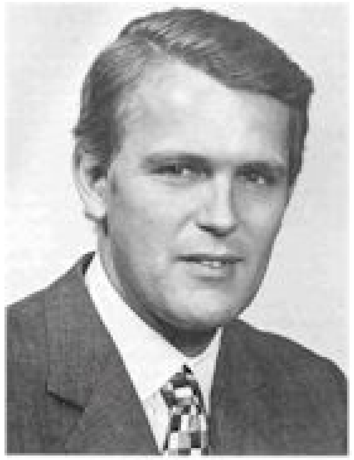
Christopher Parrish

Oregon State University

CE 661: Kinematic Positioning & Navigation

Rudolph (Rudi) E. Kalman

- Hungarian-born American engineer
- IEEE Medal of Honor (1974), the IEEE Centennial Medal (1984), the Kyoto Prize in High Technology from the Inamori foundation, Japan (1985), the Steele Prize of the American Mathematical Society (1987), and the Bellman Prize (1997). He is a member of the National Academy of Sciences (USA), the National Academy of Engineering (USA), and the American Academy of Arts and Sciences (USA)
- Famous paper:
  - Kalman, R. E. 1960. "A New Approach to Linear Filtering and Prediction Problems," *Transaction of the ASME—Journal of Basic Engineering*, pp. 35-45.
- Conceived of idea for Kalman filter on train from Princeton to Baltimore in 1958 while working for the Research Institute for Advanced Studies (RIAS)



Sources: http://www.cs.unc.edu/~welch/kalman/kalmanBio.html
http://www.ieeeghn.org/wiki/index.php/Rudolf_E._Kalman

Synergy between GNSS and INS - They have complementary error characteristics :

- **INS**
  - Short-term position errors are small, but they grow with time
  - High measurement rates (e.g., 200 Hz)
- **GNSS**
  - Short-term positioning accuracies are not as good, but they do not degrade with time
  - Typically lower measurement rates (e.g., 1-2 Hz)
- **Kalman filter provides an integrated navigation implementation that is better than either one alone**

What is a Kalman filter?

- A linear quadratic estimator (LQE)
  - Produces statistically optimal (in the MMSE sense) estimates of the **state** of a dynamic system from a series of noisy measurements
- “Navigation’s integration workhorse”
- A recursive estimator
  - Uses only the estimated state from the previous time step and the current measurement to compute the estimate for the current state
- A mechanism for creating an accurate sensor from multiple, mediocre ones
- A tool (and only a tool)
- A computer algorithm
- *Something people are intimidated by!*

  …but, not you!

What is a Kalman filter (continued)

- Is it a *filter*?

Ex: highpass filter

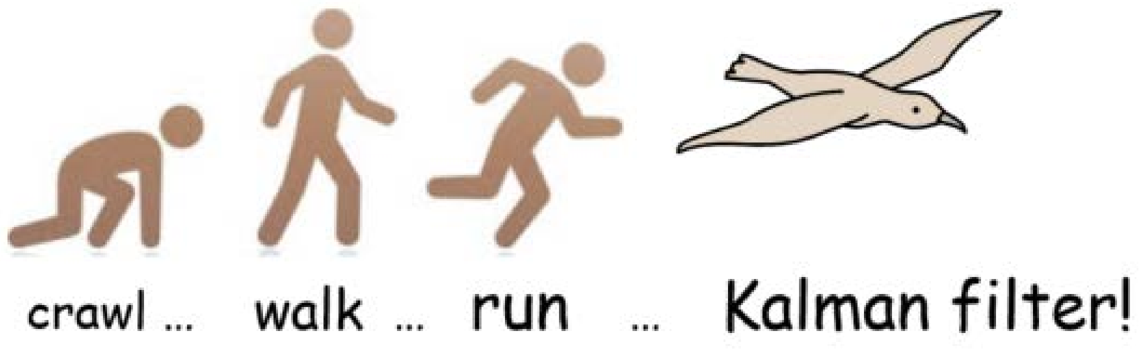
Signal comprised of a range of frequency components in Fourier domain

High frequency components

  - Yes, in the broad sense in which the word is used in the field of statistical estimation theory
- But…this term can be unnecessarily confusing to students in understanding what the Kalman filter is/does
- So, just think of the Kalman filter as a **computer algorithm** (which it is!)

Background: a few things we'll cover before getting to the KF

- Recursive filter
- Moving average filter
- Lowpass filter
- Terminology
- Notation



Recursive filter

Average of a time series of *k* data points

$$\langle x_k \rangle = \frac{\sum_{i=1}^{k} x_i}{k} = \frac{x_1 + x_2 + \cdots + x_k}{k}$$

Problems with this:

- If data are being added sequentially, this is very inefficient: need to re-compute the whole thing each time
  - Computationally expensive
  - Memory intensive (need to store entire time series)

(Recursive) average filter

$$\langle x_k \rangle = a\langle x_{k-1} \rangle + (1-a)x_k$$

$$\text{where } \; a = \frac{k-1}{k}$$

Re-uses previous output as an input: $\langle x_{k-1} \rangle$

- Overcomes issues listed above
- Just need to store previous average and *k* at each step

Moving Average filter

- Moving average:

$$\bar{x}_k = \frac{\sum_{i=k-n+1}^{k} x_i}{n} = \frac{x_{k-n+1} + x_{k-n+2} + \cdots + x_k}{n}$$

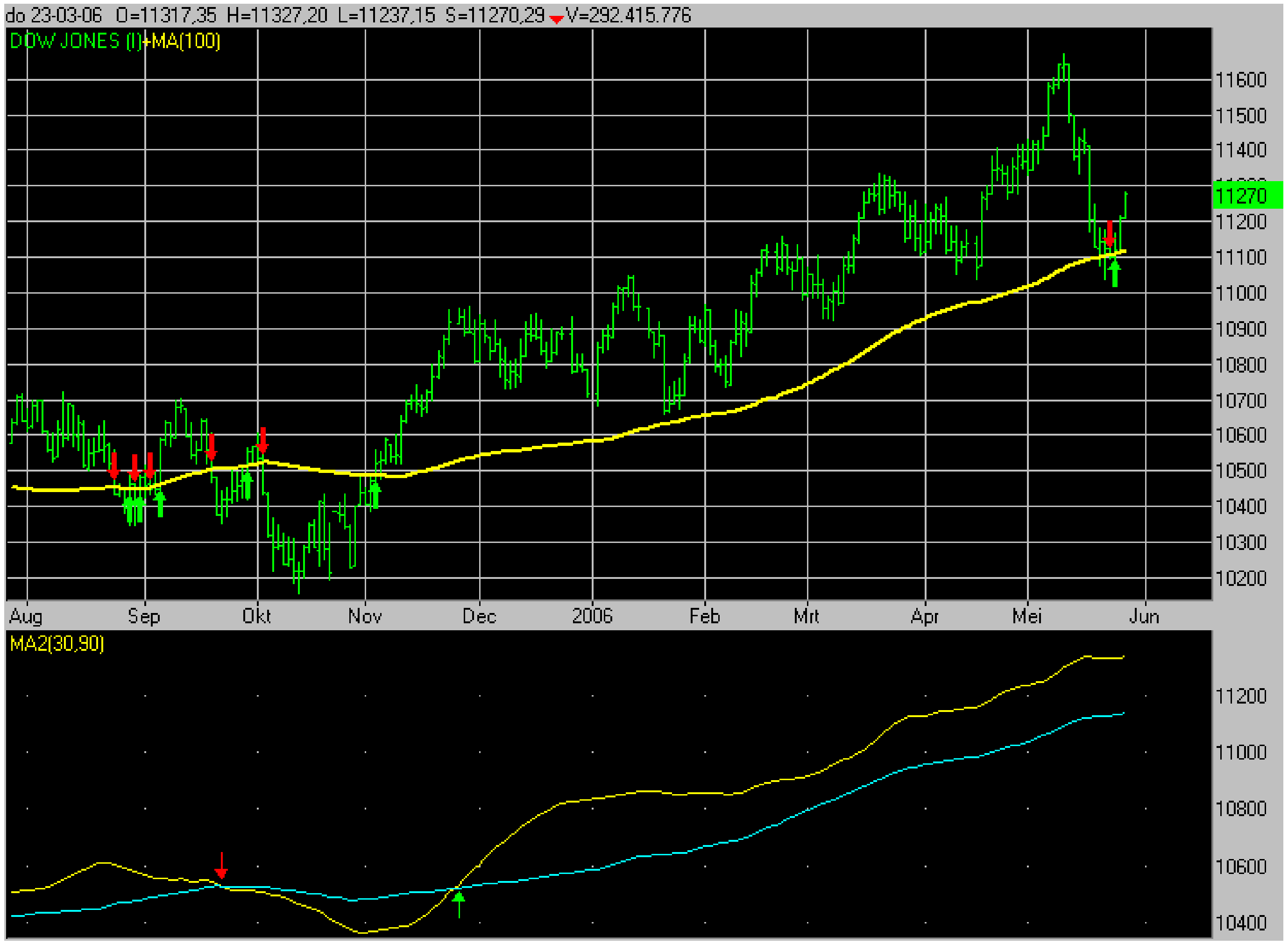
- Recursive form of moving average filter:

$$\bar{x}_k = \bar{x}_{k-1} + \frac{x_k - x_{k-n}}{n}$$

As in the previous example, this is recursive, in that it reuses its previous output, $\bar{x}_{k-1}$, as input for the current iteration

Tradeoff in Moving Average filter

- Larger *N* (number of samples being averaged)
  - Smoother trajectory (less noise), but with more of a time delay
- Smaller *N*
  - Less time delay, but noisier



Stock market:
Moving averages of different length (100 day, 30 day, 90 day)

So, how do you select *N*?

A: carefully, considering noise characteristics of your data and how rapidly they are varying in time!

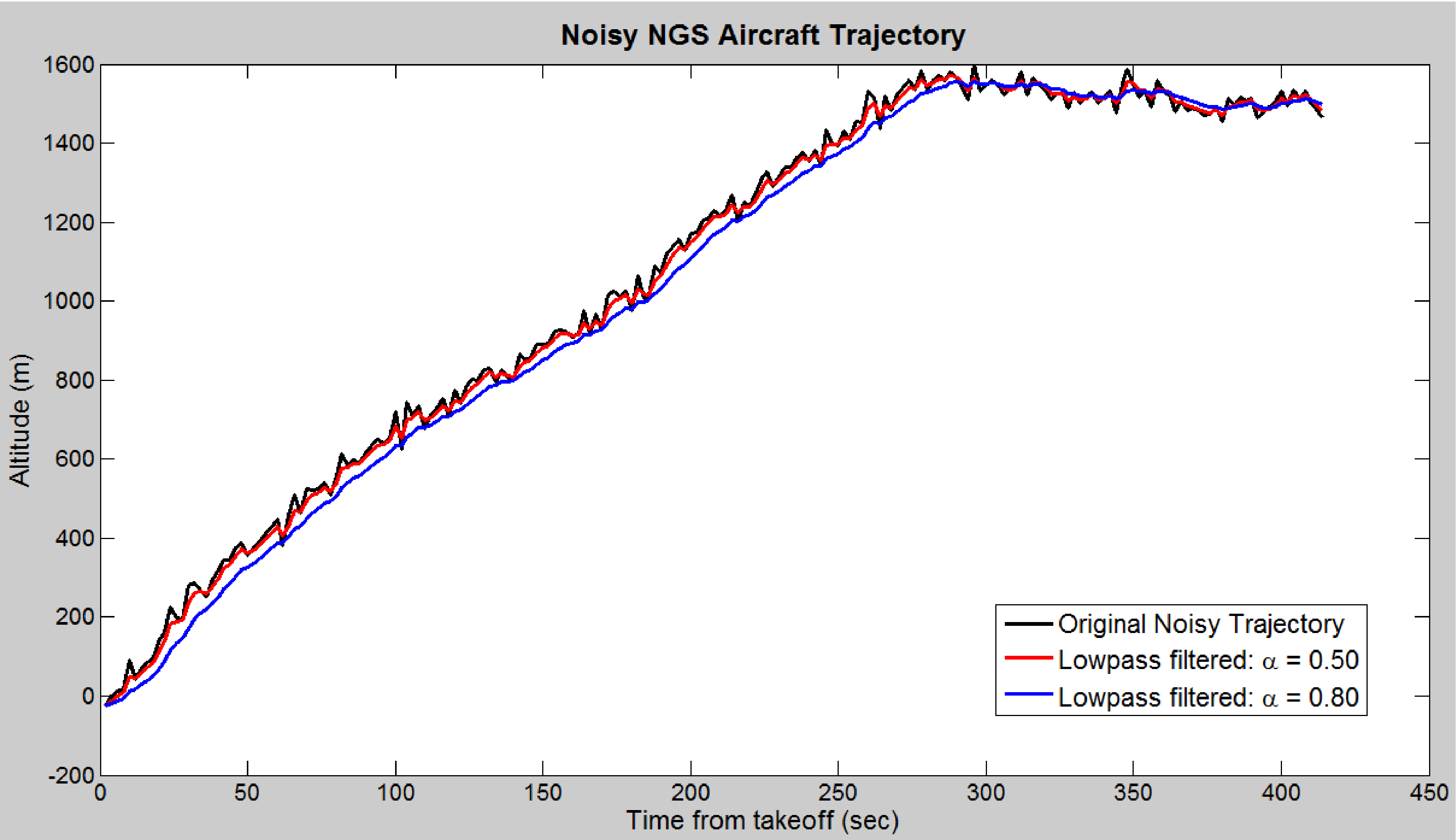
Recursive Lowpass filter

$$\bar{x}_k = \alpha \bar{x}_{k-1} + (1 - \alpha) x_k$$

- α is a constant between 0 and 1
- This is the only parameter to be set, but it has big effects on the filter performance!
  - Large α : weights previous average ($\bar{x}_{k-1}$) more
  - Small α : weights current measurement (x_k) more
- As with the moving average filter we saw previously, there is a tradeoff involved…

Tradeoff in lowpass filter design

Increasing value of α reduces noise (provides a smoother trajectory) but reduces responsiveness (note: blue curve is smoother, but exhibits a time lag)



Terminology: what is a *state*?

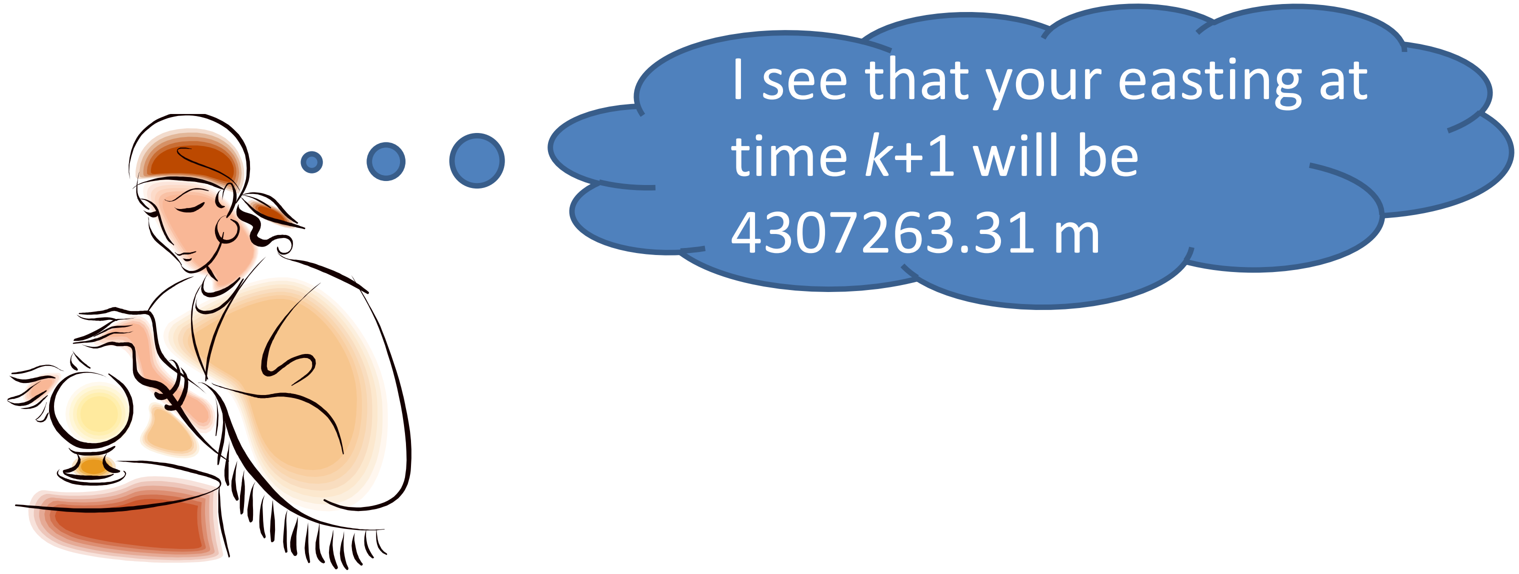
- The state of a dynamic system
  = The values of its attributes of interest (e.g., position, velocity, attitude) at any given instant
- State variables
  – Comprise the state vector
- Simple example:

$$\mathbf{x}_k = \begin{bmatrix} E \\ N \\ D \\ \dot{E} \\ \dot{N} \\ \dot{D} \end{bmatrix} = \begin{bmatrix} 203.7 \\ 117.8 \\ 14.5 \\ 1.2 \\ 0.8 \\ 0.4 \end{bmatrix}$$

Terminology: *predicted* vs. *corrected* values

- Predicted
    - *"a priori"*
    - Prior to using measurements
- Corrected
    - *"a posteriori"*
    - After using measurements
- Can also define predictors as **time updates** (i.e., modeling what happens from time k-1 to time k) and correctors as **measurement updates** (all variables are at time k)
- We will see that:
    - Only the $\mathbf{\Phi}$ matrix is used in going from the state estimate at time k-1 to the prediction at time k
    - On the other hand, the measurement, $\mathbf{z}$, is used in getting the corrected value

How is prediction done in the Kalman filter?



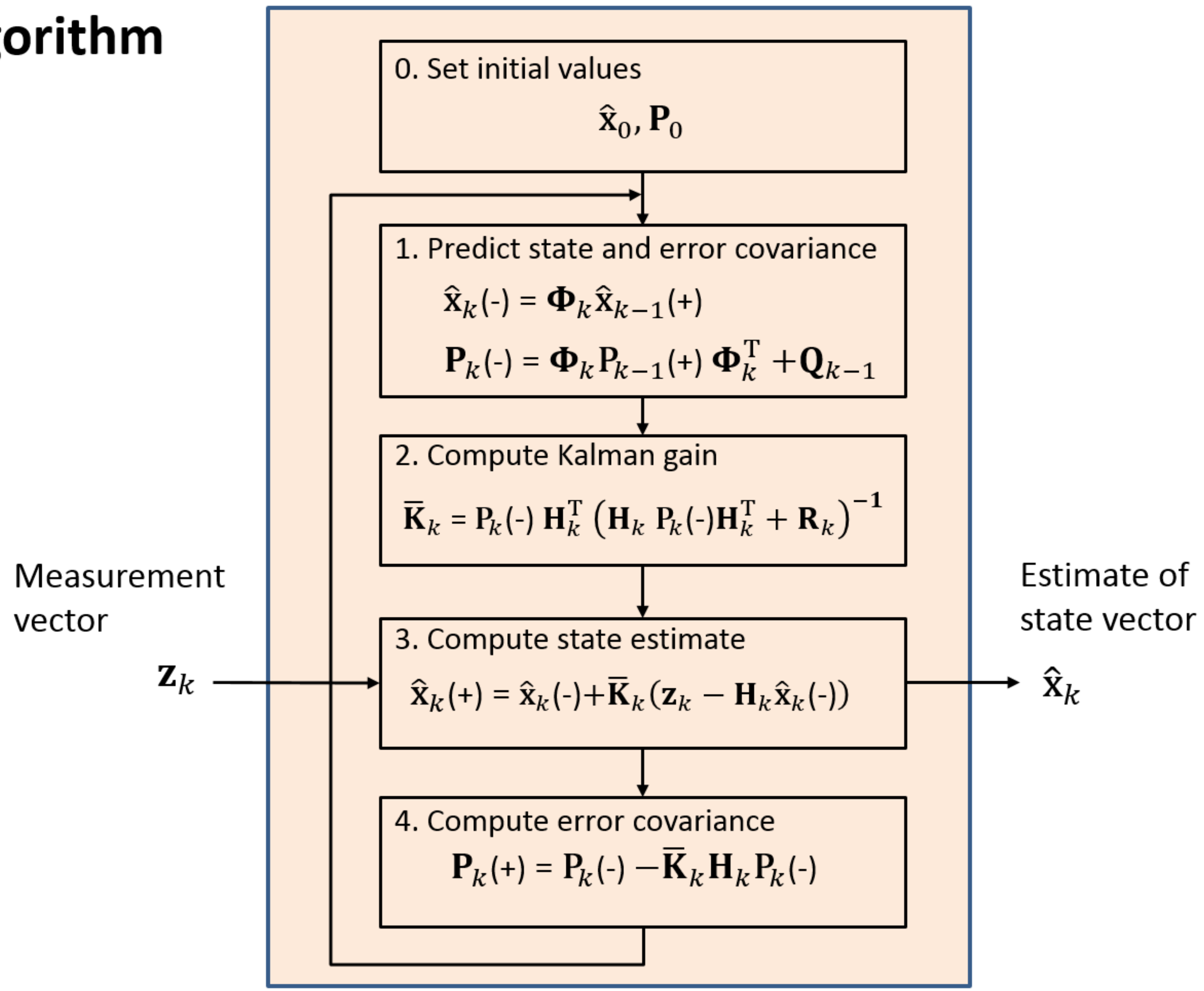
Not exactly! The state transition matrix, $\mathbf{\Phi}$, which models how the system changes with time, is used to predict where you are going to be and what your error is going to be at time k+1

Notation

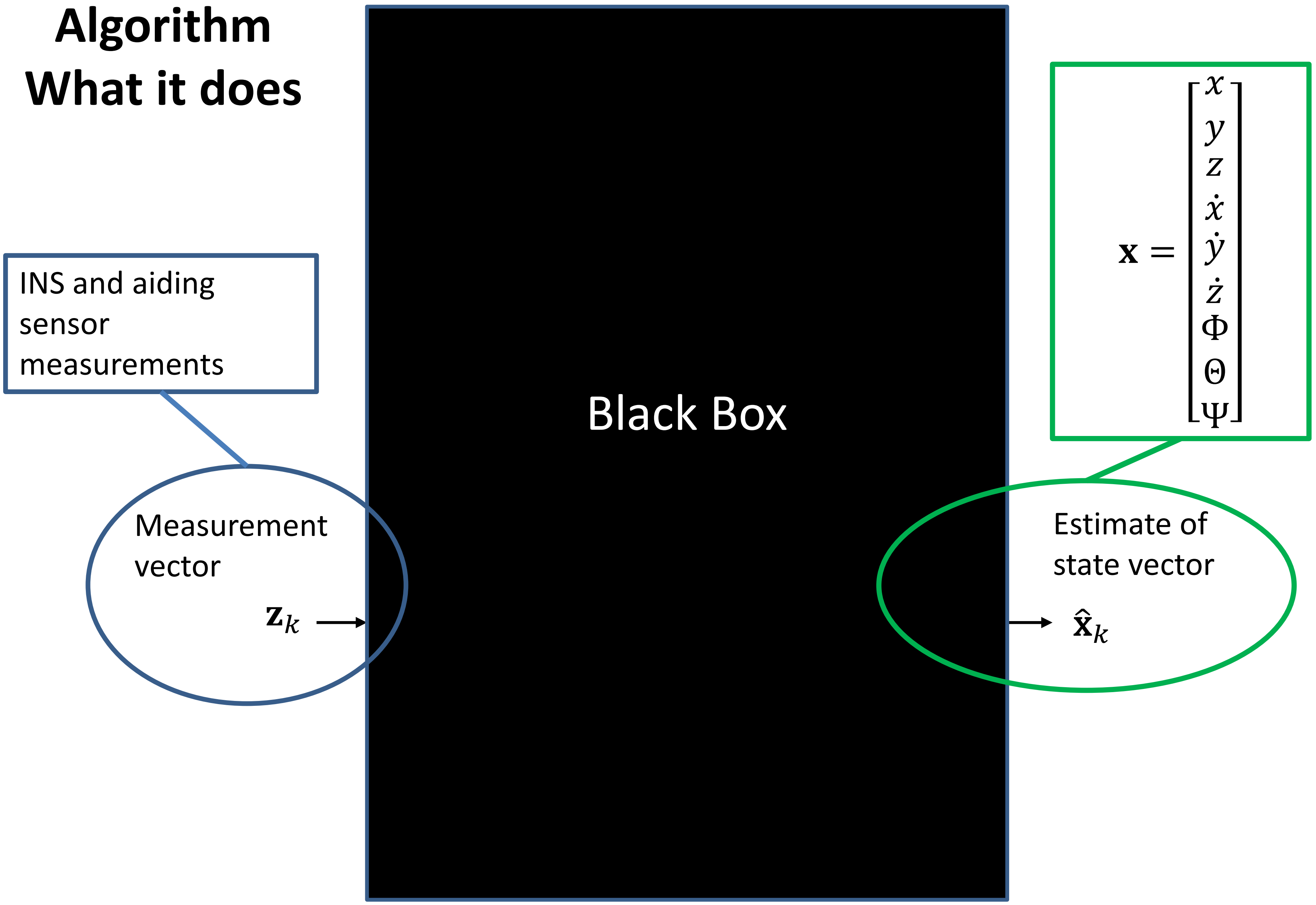
- Bold capital letter = matrix
  - Example: $\mathbf{P} = \begin{bmatrix} 2.8 & 0 \\ 0 & 5.1 \end{bmatrix}$
- Bold lower case letter = vector
  - Example: $\mathbf{z} = [204.5 \quad 100.6 \quad 1.7 \quad 0.3]^{\mathrm{T}}$
- Variable with a "hat" = estimate
  - Example: $\hat{\mathbf{x}}$
    - The state is the estimated value (note that it's also a vector)
- Negative sign (-) indicates an *a priori* (predicted) value
- Positive sign (+) indicated an *a posteriori* (corrected) value)
- Dot over variable = derivative wrt time (d/dt)
  - Example: $\dot{E}$

Kalman Filter Algorithm

Pre-determined system model: $\mathbf{\Phi}, \mathbf{H}, \mathbf{Q}, \mathbf{R}$

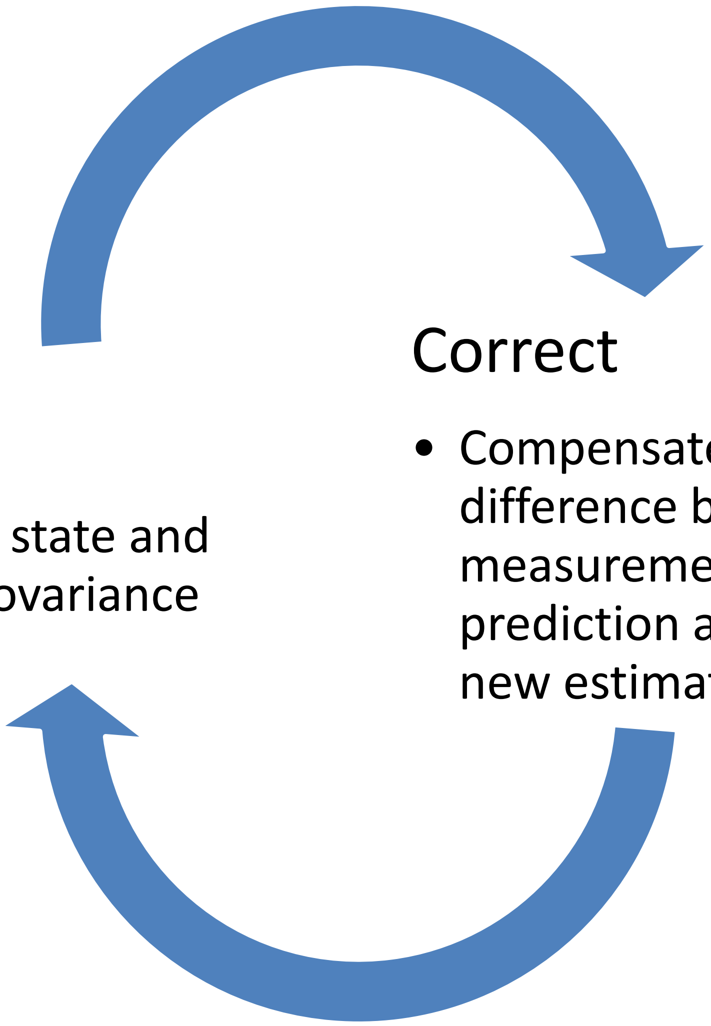

0. Set initial values
$\hat{\mathbf{x}}_0, \mathbf{P}_0$
1. Predict state and error covariance
$\hat{\mathbf{x}}_k(-) = \mathbf{\Phi}_k \hat{\mathbf{x}}_{k-1}(+)$
$\mathbf{P}_k(-) = \mathbf{\Phi}_k \mathrm{P}_{k-1}(+)\, \mathbf{\Phi}_k^{\mathrm{T}} + \mathbf{Q}_{k-1}$
2. Compute Kalman gain
$\overline{\mathbf{K}}_k = \mathrm{P}_k(-)\, \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{H}_k\, \mathrm{P}_k(-)\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k\right)^{-1}$
Measurement vector
$\mathbf{z}_k$
3. Compute state estimate
$\hat{\mathbf{x}}_k(+) = \hat{\mathbf{x}}_k(-) + \overline{\mathbf{K}}_k(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k(-))$
Estimate of state vector
$\hat{\mathbf{x}}_k$
4. Compute error covariance
$\mathbf{P}_k(+) = \mathrm{P}_k(-) - \overline{\mathbf{K}}_k \mathbf{H}_k \mathrm{P}_k(-)$

Kalman Filter Algorithm What it does


INS and aiding sensor measurements
Measurement vector
$\mathbf{z}_k$
Black Box
Estimate of state vector
$\hat{\mathbf{x}}_k$
$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \Phi \\ \Theta \\ \Psi \end{bmatrix}$

Kalman filter loop

Initial estimates for state vector and error covariance matrix



Step 1 of algorithm

Predict

- Predict state and error covariance

Correct

- Compensate for difference between measurement and prediction and compute new estimate

Steps 2-4 of algorithm

Notation

Symbol	Meaning
$\mathbf{P}_k(-)$	Predicted or *a priori* value of the error covariance (Note: the error covariance expresses the degree of accuracy of the estimate)
$\mathbf{P}_k(+)$	Corrected or *a posteriori* value of the error covariance
$\hat{\mathbf{x}}_k(-)$	Predicted or *a priori* value of the estimated state vector
$\hat{\mathbf{x}}_k(+)$	Corrected or *a posteriori* value of the estimated state vector
$\mathbf{z}$	Measurement vector (or observation vector)
$\bar{\mathbf{K}}$	Kalman gain
$\mathbf{R}$	Covariance of sensor noise (or measurement uncertainty) (part of the system model)
$\mathbf{Q}$	Covariance of state transition noise (part of the system model)
$\mathbf{\Phi}$	State transition matrix (part of the system model) (Note: models how the system changes with time; i.e., contains the equations of motion of the system)
$\mathbf{H}$	State-to-measurement matrix (part of the system model) (Note: models the relationship between the measurement and the state vector)

Step 3: Compute state estimate

$$\hat{x}_k(+) = \hat{x}_k(-) + \bar{K}_k(z_k - H_k\hat{x}_k(-))$$

$$\begin{pmatrix}\text{Corrected value}\\ \text{of state}\end{pmatrix} = \begin{pmatrix}\text{Predicted value}\\ \text{of state}\end{pmatrix} + (\text{Kalman gain})\begin{pmatrix}\text{Measurement}\\ \text{prediction error}\end{pmatrix}$$

- Has similar form to 1st order lowpass filter
  - But, note that $\bar{K}$ (the weight) changes (is recomputed) at each iteration of the algorithm
- The Kalman gain determines how much to weight the measurements vs. predictions
- Large $\bar{K}$ => filter places more weight on measurements
- Small $\bar{K}$ => filter weights model predicti
  - This tends to smooth out noise, but make the filter less responsive

Similar to the tradeoff we saw with the parameter a in the lowpass filter!

Step 2: Compute Kalman Gain

$$\overline{\mathbf{K}}_k = \mathrm{P}_k(\text{-})\,\mathbf{H}_k^{\mathrm{T}}\left(\mathbf{H}_k\,\mathrm{P}_k(\text{-})\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k\right)^{-1}$$

- The Kalman gain is used in weighting in Step 3
- We will return to what the Kalman gain does and how it can be varied (through selection of the System Model)
- For now, let's just look at the variables from which the Kalman gain is computed:

$\mathbf{P}_k$(-)	Predicted *or a priori* value of the error covariance (Note: the error covariance expresses the degree of accuracy of the estimate)
$\mathbf{H}_k$	State-to-measurement matrix (part of the system model) (Note: models the relationship between the measurement and the state vector)
$\mathbf{R}_k$	Covariance of sensor noise (or measurement uncertainty) (part of the system model)

Step 1: Predict state and error covariance

$$\hat{\mathbf{x}}_k(\text{-}) = \mathbf{\Phi}_k \hat{\mathbf{x}}_{k-1}(+)$$

$$\mathbf{P}_k(\text{-}) = \mathbf{\Phi}_k \mathbf{P}_{k-1}(+)\, \mathbf{\Phi}_k^{\mathrm{T}} + \mathbf{Q}_{k-1}$$

- Predict how state and error covariance will change from time *k* to time *k*+1
  - *Where are we going to be?*
  - *What is your error going to be?*
- How do you do this? Using System Model!
  - Recall that $\mathbf{\Phi}$ is the state transition matrix (part of the System Model), which models how the system changes with time
- Note that predicted (or *a priori*) values at time *k* are on the LHS, while corrected (or *a posteriori*) values at the previous time, *k*-1, are on the RHS

Step 4: Compute Error Covariance

$$\mathbf{P}_k(+) = \mathbf{P}_k(\text{-}) - \overline{\mathbf{K}}_k \mathbf{H}_k \mathbf{P}_k(\text{-})$$

$$\begin{pmatrix}\text{Corrected}\\ \text{value of error}\\ \text{covariance}\end{pmatrix} = \begin{pmatrix}\text{Predicted}\\ \text{valueof error}\\ \text{covariance}\end{pmatrix} - \begin{pmatrix}\text{Kalman}\\ \text{gain}\end{pmatrix}\begin{pmatrix}\text{State−to−}\\ \text{Measurement}\\ \text{matirx}\end{pmatrix}\begin{pmatrix}\text{Predicted}\\ \text{value of error}\\ \text{covariance}\end{pmatrix}$$

- This is how you compute the corrected value of the error covariance at time *k* that then gets fed back into the next iteration of the algorithm as the corrected value of the error covariance matrix at time *k*-1
- Important: how do we interpret the error covariance?
  - It expresses the difference between the estimate of the states and their “true” values

$$\mathbf{P}_k \equiv E\{(\hat{\mathbf{x}}_k - \mathbf{x}_k)(\hat{\mathbf{x}}_k - \mathbf{x}_k)^{\mathrm{T}}\}$$

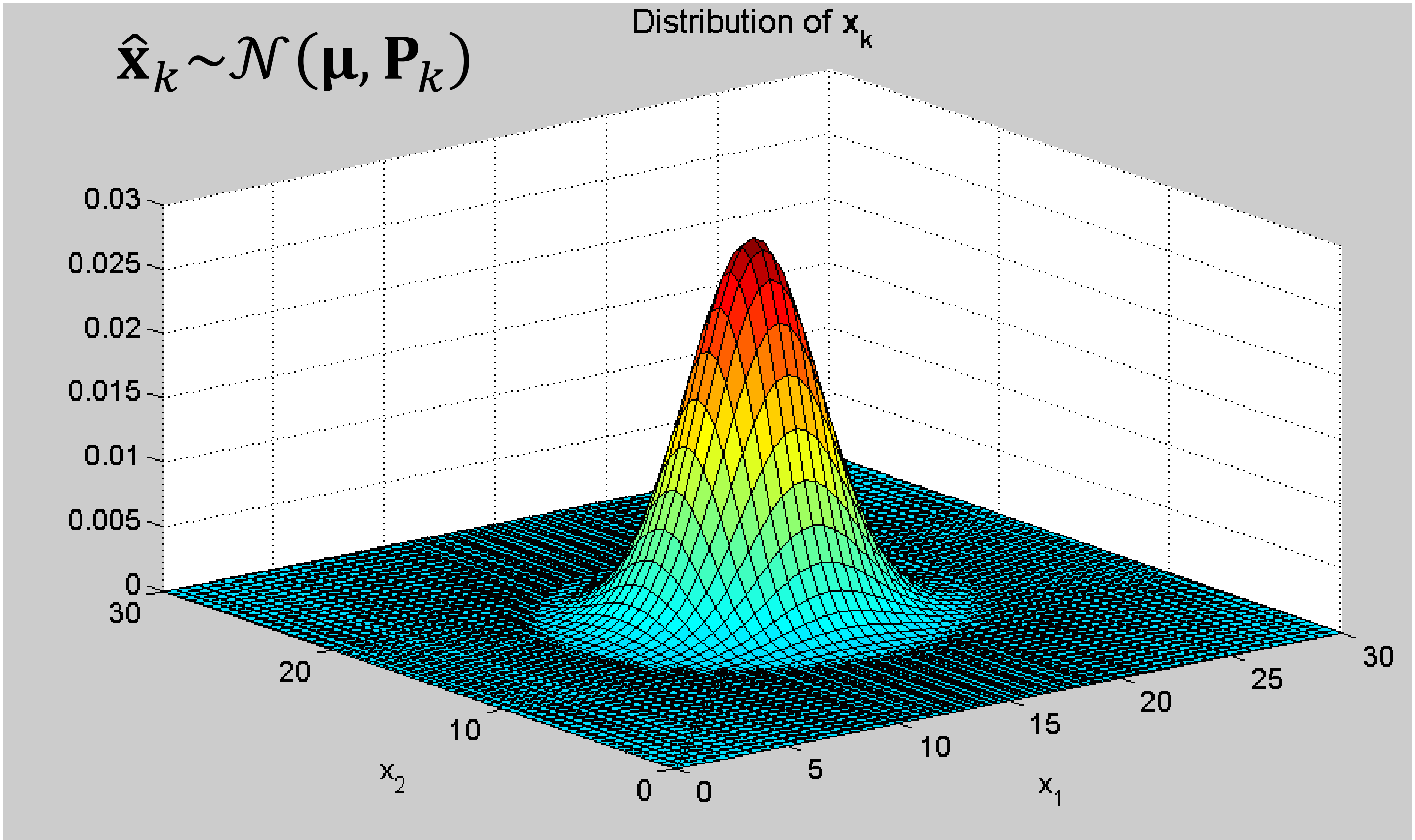
  - How good are our estimates?

More on Interpreting the Error Covariance

- Say you have 2 states in your state vector and:

$$\mathbf{P}_k = \begin{bmatrix} 6.1 & 0 \\ 0 & 4.9 \end{bmatrix}$$

- Interpretation:



The mean (here [15 15]) is the estimate of $\mathbf{x}_k$, and the spread provides an indication of the uncertainty (or error) about this value

How about the System Model?

- This is the key to the whole thing!
- The algorithm was handed down to you (from Dr. Kalman)
- So, designing and implementing a Kalman filter comes down to establishing the system model
- How do you get the system model
  A. By mathematically modeling the real-world dynamics of the intended application
     - Generally have systems of linear differential equations, based on laws of physics

  B. Find an existing one for your intended application
     - Modify as needed, based on, e.g., the quality of your sensors

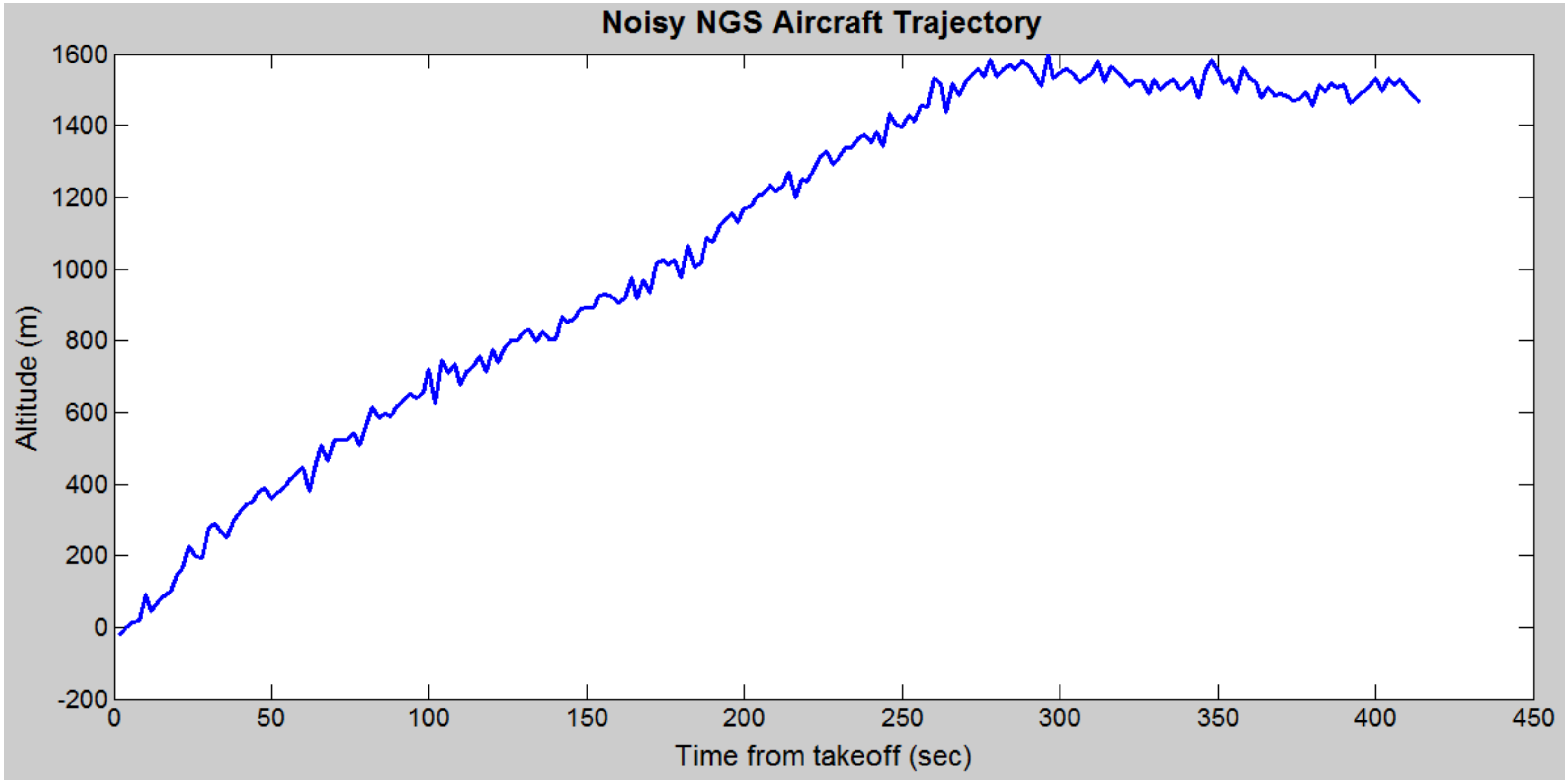
So, what's in the System Model?

Variable	Name/description	Equations
$\boldsymbol{\Phi}$	State transition matrix (n×n matrix). Represents the known dynamic behavior of the system (how the system changes with time).	$\mathbf{x}_k = \boldsymbol{\Phi}_{k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1}$ Where **w** is the state transition noise (n×1 column vector)
H	State-to-measurement matrix (m×n matrix): defines the (linear) relationship between the measurement, **z**, and state vector, **x**	$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$ Where **v** = measurement noise (m×1 column vector)
Q	Covariance matrix of state transition noise (or process noise in the system state dynamics) (n×n matrix)	$\mathbf{Q} = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \cdots, \sigma_n^2)$ Note: **w** = state transition noise (n×1 column vector)
R	Covariance matrix of sensor noise (or "measurement noise") (m×m matrix)	$\mathbf{R} = E\{\mathbf{v}\mathbf{v}^{\mathrm{T}}\}$ Where **v** = measurement noise (m×1 column vector)

m = number of measurements; *n* = number of state variables

OK, how about an example?

- Let’s return to the application we looked at earlier
  - 1D aircraft trajectory: altitude (specifically, NAD83 ellipsoid height) as a function of time
  - Data collected by NOAA NGS on flight mission out of Portsmouth, NH
  - Data = time series of ellipsoid heights (plotted below)
  - We want our Kalman filter to estimate the ellipsoid height and vertical velocity (climb rate)



State vector for this application

$$\mathbf{x} = \begin{bmatrix} h_{el} \\ \dot{h}_{el} \end{bmatrix} = \begin{bmatrix} \text{ellipsoid height} \\ \text{vertical velocity} \end{bmatrix}$$

System Model for this Application

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

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$\mathbf{R} = 625$$

Wait a second here…*where did this come from??*

System Model

How we got $\mathbf{H}$

- Recall our linear measurement model

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

- Also recall from the problem statement that we are only measuring the ellipsoid height at each time (we're not directly measuring velocity).
- So, we only have one measurement: ellipsoid height at time k

$$\begin{aligned}\mathbf{z}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \\ &= \boxed{\begin{bmatrix}1 & 0\end{bmatrix}} \begin{bmatrix} h_{el,k} \\ \dot{h}_{el},k \end{bmatrix} + \mathrm{v}_k \\ &= h_{el,k} + \mathrm{v}_k\end{aligned}$$

Again, this correctly models what we know about how our measurements relate to the state

How we got $\mathbf{\Phi}$

- $\mathbf{\Phi}$ comes from understanding and modeling the physics of the system (specifically, how parameters of interest change with time)
- In our case, we have:

$$\text{vertical velocity} = \dot{h}_{el} = \frac{\mathrm{d}h_{el}}{\mathrm{d}t} \approx \frac{\Delta h_{el}}{\Delta t} = \frac{h_{el,k} - h_{el,k-1}}{\Delta t}$$

- Which can be rearranged, as follows

$$h_{el,k} = h_{el,k-1} + \dot{h}_{el}\Delta t$$

  – This just says that the ellipsoid height at time *k* is equal to the ellipsoid height at *k*-1 plus the product of the vertical velocity and the time interval over which it is measured

How we got $\mathbf{\Phi}$ (continued)

- Next, how does the vertical velocity change with time?
  - As a reasonable, first-order approximation, let's assume that the aircraft climb rate is nearly constant at the aircraft ascends to its working altitude
  - But, we will assume the climb rate is not exactly constant, due to noise, *w*
  - The following equation expresses these conditions:

$$\dot{h}_{el,k} = \dot{h}_{el,k-1} + \text{noise}$$

How we got $\mathbf{\Phi}$ (continued)

- Recall:

$$\mathbf{x}_k = \mathbf{\Phi}_{k-1}\mathbf{x}_{k-1} + \text{noise}$$

- Expanding our state vector, plugging in our $\mathbf{\Phi}$ and momentarily ignoring the noise term, we have

$$\begin{bmatrix} h_{el,k} \\ \dot{h}_{el},k \end{bmatrix} = \boxed{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}} \begin{bmatrix} h_{el,k-1} \\ \dot{h}_{el,k-1} \end{bmatrix}$$

So, this did what we wanted it to, in terms of modeling the known dynamics of the system

- And, if we multiply this out, we have:

$$h_{el,k} = h_{el,k-1} + \dot{h}_{el,k-1}\Delta t$$

$$\dot{h}_{el,k} = \dot{h}_{el,k-1}$$

- Exactly what we had on previous slides (again ignoring noise terms here)!

How we got $\mathbf{R}$

- Recall that **R** models our sensor noise (or "measurement noise")
- Typically, you might get this from the sensor manufacturer's specs for their sensor noise
- Or, you could estimate it from data
- In this case, I had an (unfair?) advantage, in that I artificially added noise to the NGS trajectory for demonstration purposes
  - And, I know the characteristics of the noise I added!

$$\sigma_n = 25$$

$$\sigma_n^2 = 625$$ Noise variance

How we got $\mathbf{Q}$

- Truth in advertising:
  - I set **Q** empirically by trial and error to get performance of the Kalman filter how I wanted it
- Which brings us to an important point
  - The initial values you assign to **Q** and **R** should be based on theory, system analysis and published sensor specs, to the extent possible (or practical)
  - But, at the end of the day, you should realize that you’re probably going to end up having to tweak them empirically

How to empirically adjust **R** and **Q**

- If you want a smoother, less noisy trajectory (albeit, one that may not track variations as well), do one or both of the following:
  - Increase **R**
  - Decrease **Q**
- These settings adjustments result in a lower value of the Kalman gain, **K**, which has the effect of deemphasizing the current measurement and following the prediction more closely

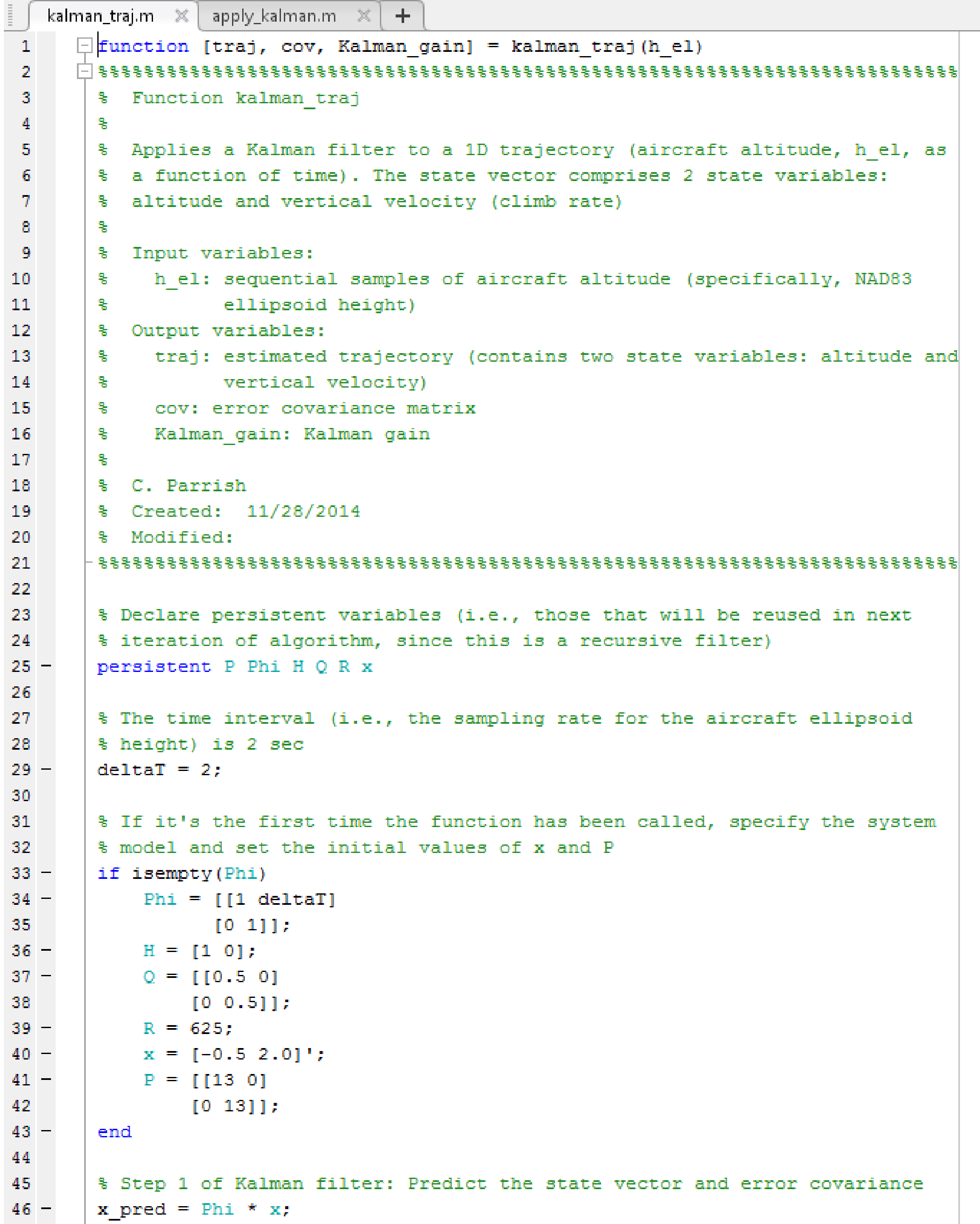
Summary of how to empirically tune **R** and **Q**

How you want to change the performance of the KF	What you need K to do	How you should change R	How you should change Q
You want the filter to place more weight on the current measurement and less on the prediction to track the data better	$\uparrow$	$\downarrow$	$\uparrow$
You want to place more weight on the prediction and less on the current measurement to get a smoother trajectory	$\downarrow$	$\uparrow$	$\downarrow$

What if you spend a lot of time adjusting **R** and $\mathbf{Q}$, but still can't get good performance from your Kalman filter?

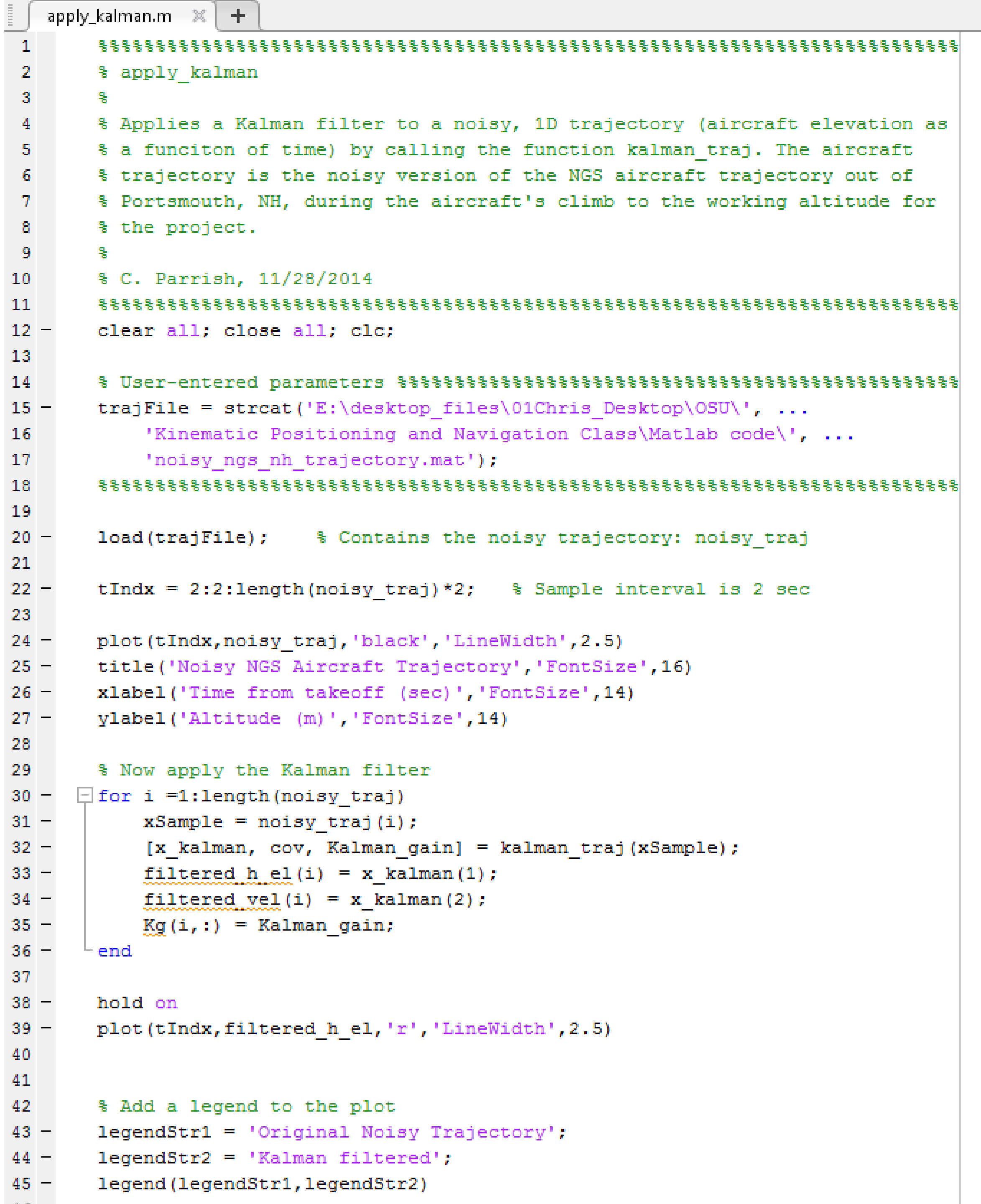
- This typically means that your system model is not doing a good job of modeling the physics of your system
- The performance of the Kalman filter depends critically on how well you model the real world through your system model
- In this example, I was able to adjust **R** and **Q** and get the filter to work well. But, if I hadn't, I would have gone back and reconsidered whether $\mathbf{\Phi}$ was adequately modeling how parameters change with time
  – For example, maybe it was a poor assumption to take the vertical velocity to be constant from *k* to *k*+1

Having designed our Kalman filter, the only remaining step is to implement it in MATLAB



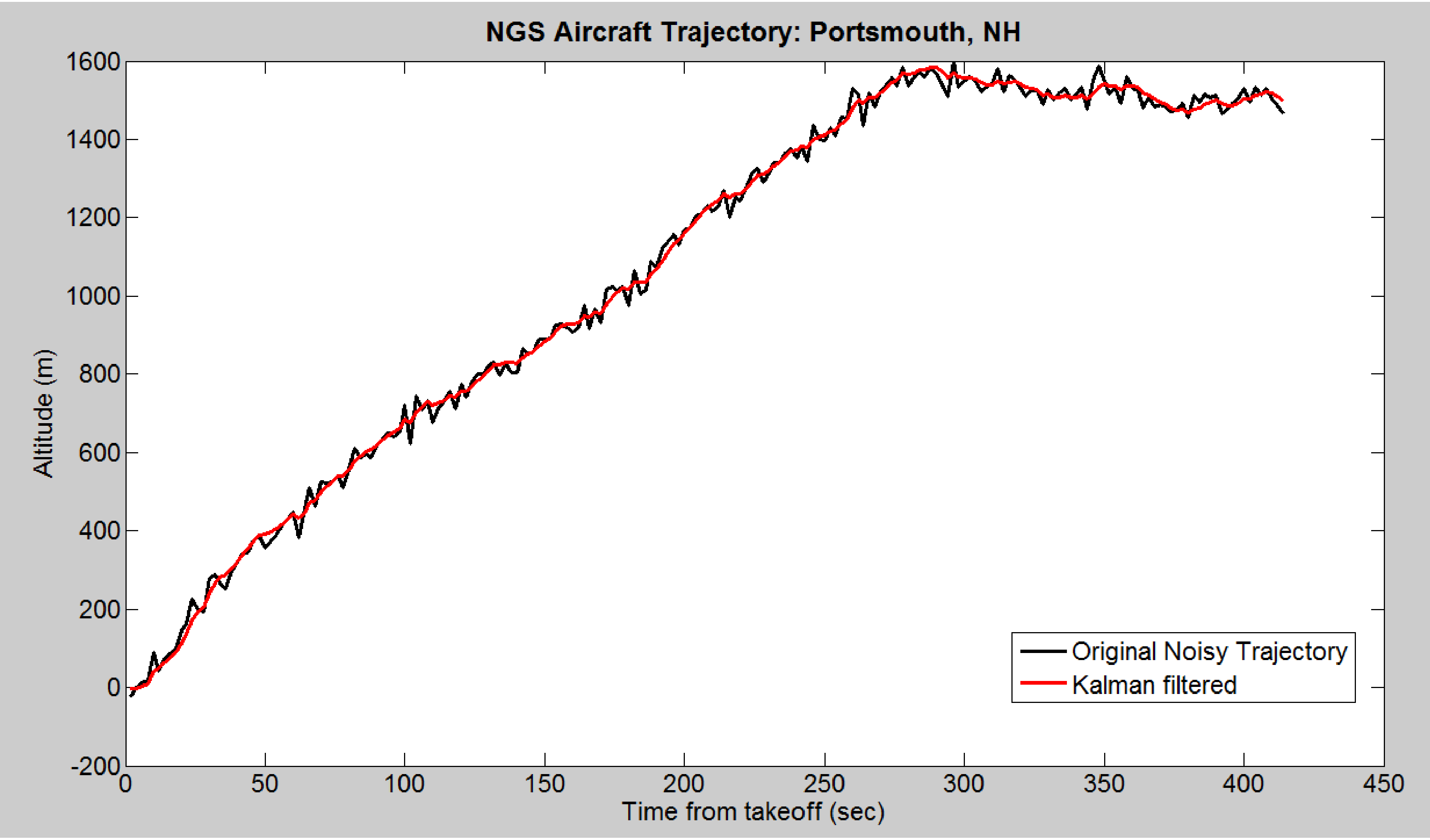
```
function [traj, cov, Kalman_gain] = kalman_traj(h_el)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Function kalman_traj
%
%  Applies a Kalman filter to a 1D trajectory (aircraft altitude, h_el, as
%  a function of time). The state vector comprises 2 state variables:
%  altitude and vertical velocity (climb rate)
%
%  Input variables:
%    h_el: sequential samples of aircraft altitude (specifically, NAD83
%          ellipsoid height)
%  Output variables:
%    traj: estimated trajectory (contains two state variables: altitude and
%          vertical velocity)
%    cov: error covariance matrix
%    Kalman_gain: Kalman gain
%
%  C. Parrish
%  Created:  11/28/2014
%  Modified:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Declare persistent variables (i.e., those that will be reused in next
% iteration of algorithm, since this is a recursive filter)
persistent P Phi H Q R x

% The time interval (i.e., the sampling rate for the aircraft ellipsoid
% height) is 2 sec
deltaT = 2;

% If it's the first time the function has been called, specify the system
% model and set the initial values of x and P
if isempty(Phi)
    Phi = [[1 deltaT]
           [0 1]];
    H = [1 0];
    Q = [[0.5 0]
         [0 0.5]];
    R = 625;
    x = [-0.5 2.0]';
    P = [[13 0]
         [0 13]];
end

% Step 1 of Kalman filter: Predict the state vector and error covariance
x_pred = Phi * x;
```

Script to call the Matlab Kalman filter function



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply_kalman
%
% Applies a Kalman filter to a noisy, 1D trajectory (aircraft elevation as
% a funciton of time) by calling the function kalman_traj. The aircraft
% trajectory is the noisy version of the NGS aircraft trajectory out of
% Portsmouth, NH, during the aircraft's climb to the working altitude for
% the project.
%
% C. Parrish, 11/28/2014
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all; clc;

% User-entered parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
trajFile = strcat('E:\desktop_files\01Chris_Desktop\OSU\', ...
    'Kinematic Positioning and Navigation Class\Matlab code\', ...
    'noisy_ngs_nh_trajectory.mat');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load(trajFile);    % Contains the noisy trajectory: noisy_traj

tIndx = 2:2:length(noisy_traj)*2;   % Sample interval is 2 sec

plot(tIndx,noisy_traj,'black','LineWidth',2.5)
title('Noisy NGS Aircraft Trajectory','FontSize',16)
xlabel('Time from takeoff (sec)','FontSize',14)
ylabel('Altitude (m)','FontSize',14)

% Now apply the Kalman filter
for i =1:length(noisy_traj)
    xSample = noisy_traj(i);
    [x_kalman, cov, Kalman_gain] = kalman_traj(xSample);
    filtered_h_el(i) = x_kalman(1);
    filtered_vel(i) = x_kalman(2);
    Kg(i,:) = Kalman_gain;
end

hold on
plot(tIndx,filtered_h_el,'r','LineWidth',2.5)


% Add a legend to the plot
legendStr1 = 'Original Noisy Trajectory';
legendStr2 = 'Kalman filtered';
legend(legendStr1,legendStr2)
```

Output:



NGS Aircraft Trajectory: Portsmouth, NH
Altitude (m)
Time from takeoff (sec)
1600
1400
1200
1000
800
600
400
200
0
-200
0
50
100
150
200
250
300
350
400
450
Original Noisy Trajectory
Kalman filtered

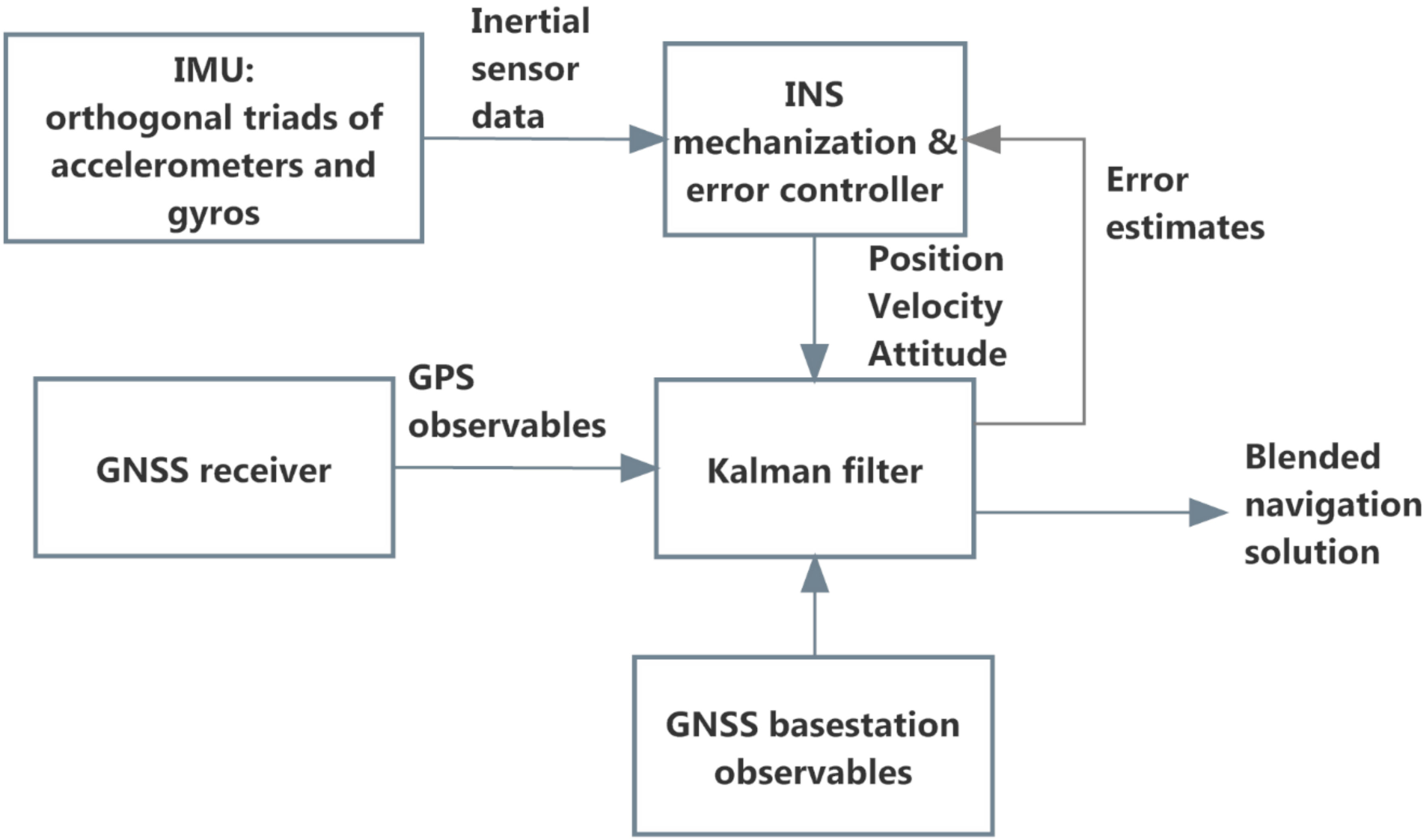
This week's HW

- You will create your own Kalman filter
- Will do essentially the same thing, except sort of the reverse
  - Given measurements of vertical velocity, output estimate of state vector that includes both aircraft altitude and vertical velocity
- Q: What do you need to change from what we just did?

Indirect (Error State) Kalman Filter

- One of the most common implementations of Kalman filter (or EKF) for the navigation problem
- The vector being estimated is actually the trajectory error
- The errors are then subtracted off at each iteration of the algorithm

$$\mathbf{x} = \begin{bmatrix} \delta\mathbf{P} \\ \delta\mathbf{v} \\ \delta\boldsymbol{\Phi} \\ \delta c_b \\ \delta\mathbf{f} \\ \delta\boldsymbol{\omega} \end{bmatrix} \begin{array}{l} \longleftarrow \text{position errors (3x1)} \\ \longleftarrow \text{velocity errors (3x1)} \\ \longleftarrow \text{misalignment errors (3x1)} \\ \longleftarrow \text{User clock bias error} \\ \longleftarrow \text{Error in accelerometer biases (3x1)} \\ \longleftarrow \text{Error in gyro biases (3x1)} \end{array}$$

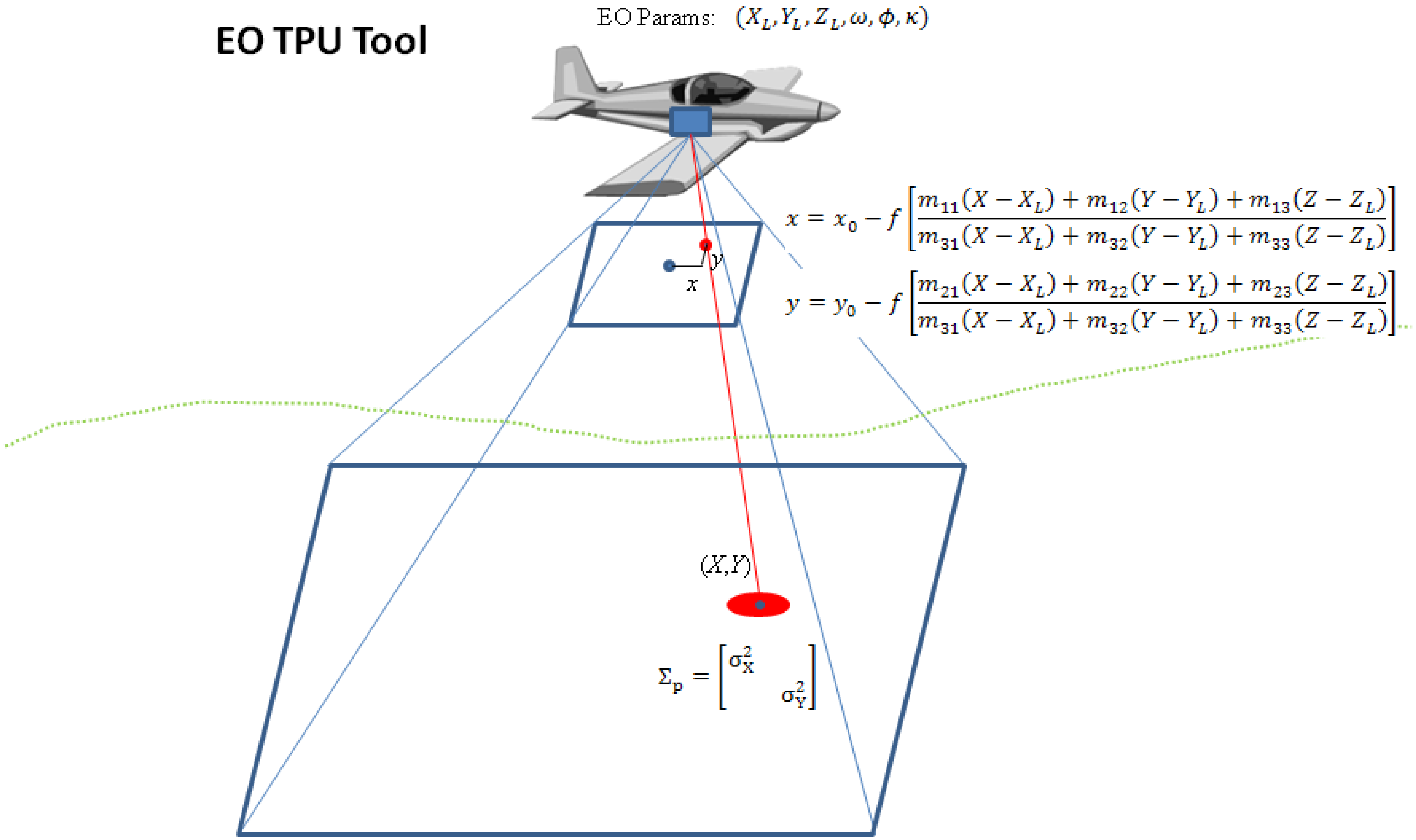

IMU:
orthogonal triads of accelerometers and gyros
Inertial sensor data
INS mechanization & error controller
Error estimates
Position
Velocity
Attitude
GPS observables
GNSS receiver
Kalman filter
Blended navigation solution
GNSS basestation observables

Key feature of Kalman filter

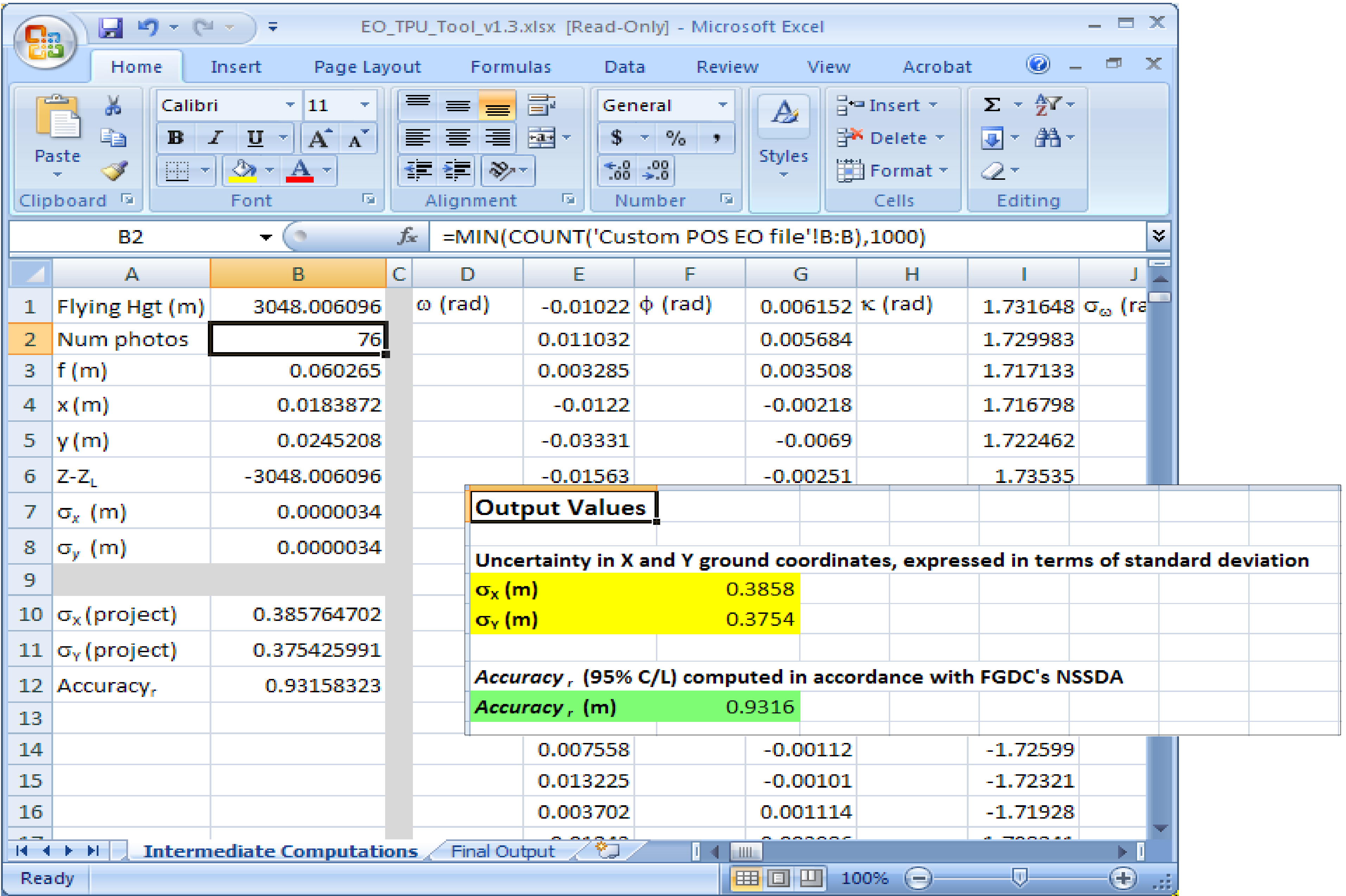
- It provides an estimate of its own accuracy
  - Through the error covariance matrix, $\mathbf{P}_k$
- This is extremely important in geomatics
  - We typically want to use the navigation solution to produce georeferenced data products
  - We'd like to be able to estimate the spatial accuracy (or coordinate uncertainty, to use the term I prefer) of these products
  - Can model the total propagated uncertainty (TPU)

What if you need uncertainties of ground points measured photogrammetricially and the imagery was directly georeferenced using GNSS-aided INS?

EO TPU Tool


EO Params: $(X_L, Y_L, Z_L, \omega, \phi, \kappa)$
x
y
$x = x_0 - f\left[\frac{m_{11}(X - X_L) + m_{12}(Y - Y_L) + m_{13}(Z - Z_L)}{m_{31}(X - X_L) + m_{32}(Y - Y_L) + m_{33}(Z - Z_L)}\right]$
$y = y_0 - f\left[\frac{m_{21}(X - X_L) + m_{22}(Y - Y_L) + m_{23}(Z - Z_L)}{m_{31}(X - X_L) + m_{32}(Y - Y_L) + m_{33}(Z - Z_L)}\right]$
(X,Y)
$\Sigma_p = \begin{bmatrix} \sigma_X^2 & \\ & \sigma_Y^2 \end{bmatrix}$

Excel version of EO TPU Tool



B2 =MIN(COUNT('Custom POS EO file'!B:B),1000)

	A	B	C	D	E	F	G	H	I	J
1	Flying Hgt (m)	3048.006096		ω (rad)	-0.01022	ϕ (rad)	0.006152	κ (rad)	1.731648	σ_ω (ra
2	Num photos	76			0.011032		0.005684		1.729983	
3	f (m)	0.060265			0.003285		0.003508		1.717133	
4	x (m)	0.0183872			-0.0122		-0.00218		1.716798	
5	y (m)	0.0245208			-0.03331		-0.0069		1.722462	
6	$Z-Z_L$	-3048.006096			-0.01563		-0.00251		1.73535	
7	σ_x (m)	0.0000034								
8	σ_y (m)	0.0000034								
9										
10	σ_X (project)	0.385764702								
11	σ_Y (project)	0.375425991								
12	Accuracy_r	0.93158323								
13										
14					0.007558		-0.00112		-1.72599	
15					0.013225		-0.00101		-1.72321	
16					0.003702		0.001114		-1.71928	

Output Values

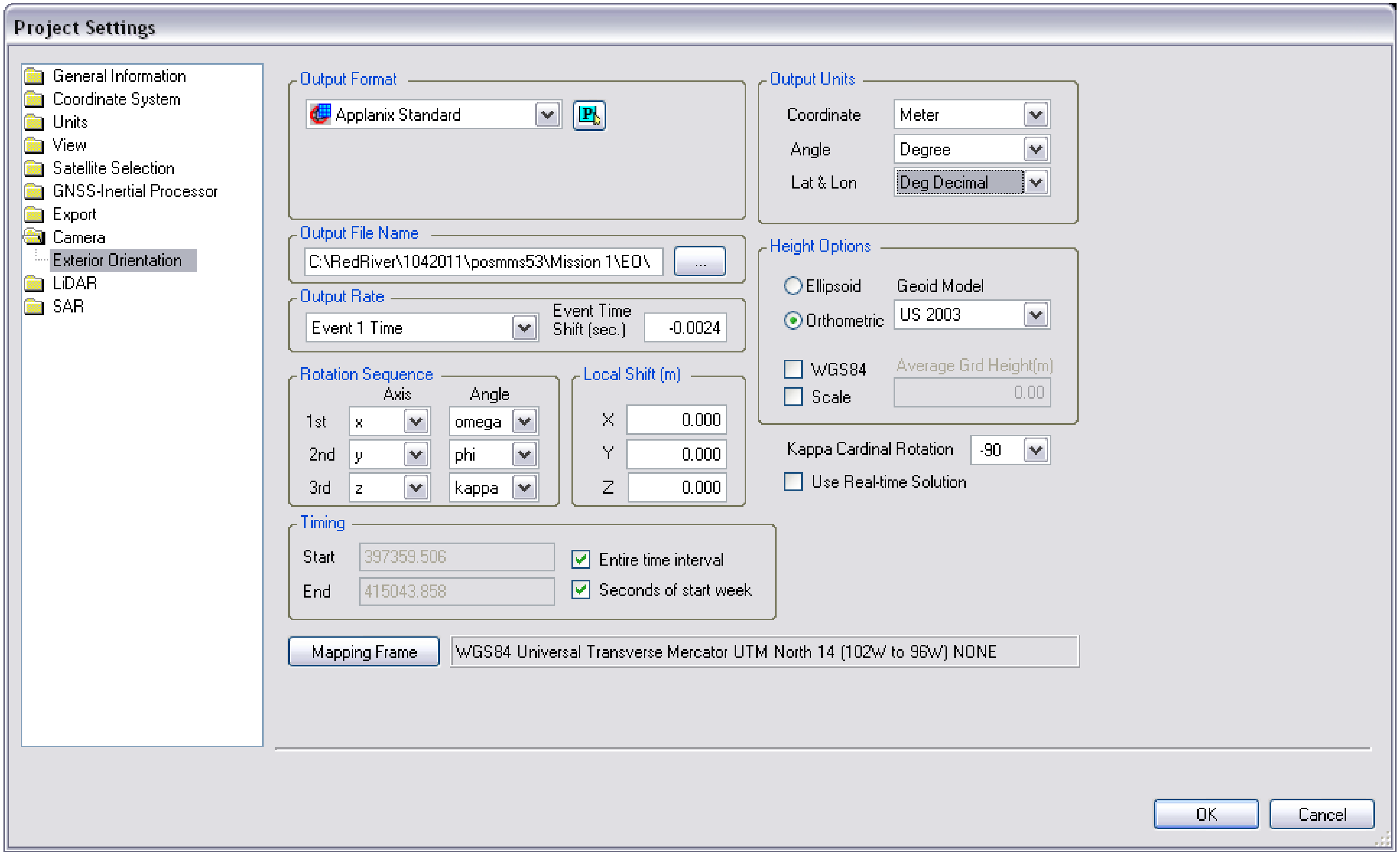
Uncertainty in X and Y ground coordinates, expressed in terms of standard deviation

σ_X (m)	0.3858
σ_Y (m)	0.3754

***Accuracy**$_r$* **(95% C/L) computed in accordance with FGDC's NSSDA**

Accuracy$_r$ (m)	0.9316

Outputting EO parameter uncertainties from Applanix POSPac



Output Format: Applanix Standard

Output File Name: C:\RedRiver\1042011\posmms53\Mission 1\EO\

Output Rate: Event 1 Time; Event Time Shift (sec.): -0.0024

Rotation Sequence

	Axis	Angle
1st	x	omega
2nd	y	phi
3rd	z	kappa

Local Shift (m): X 0.000, Y 0.000, Z 0.000

Timing: Start 397359.506; End 415043.858; [✓] Entire time interval; [✓] Seconds of start week

Output Units: Coordinate Meter; Angle Degree; Lat & Lon Deg Decimal

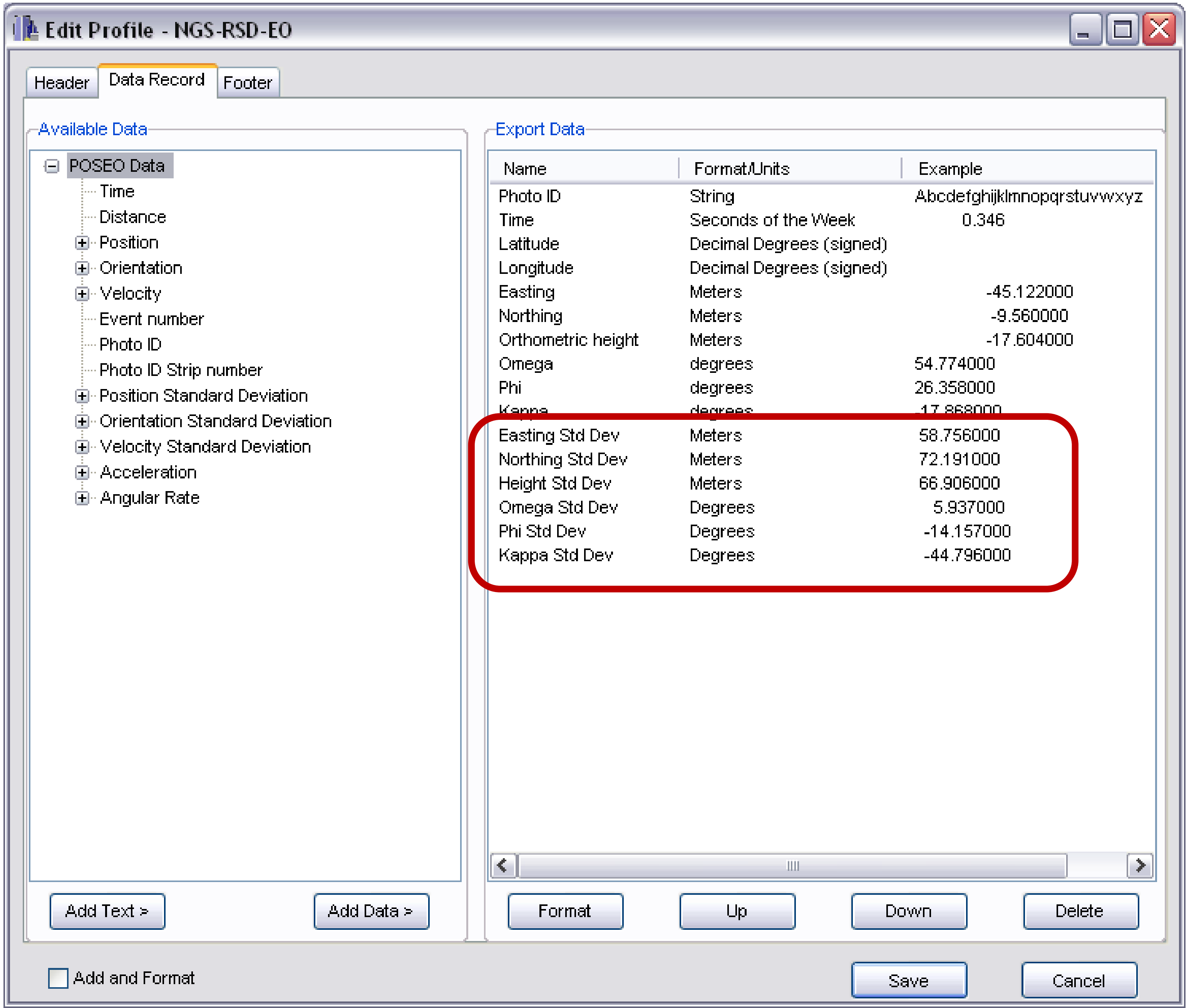
Height Options: Ellipsoid [✓ Orthometric]; Geoid Model US 2003; [ ] WGS84; [ ] Scale; Average Grd Height(m) 0.00

Kappa Cardinal Rotation: -90

[ ] Use Real-time Solution

Mapping Frame: WGS84 Universal Transverse Mercator UTM North 14 (102W to 96W) NONE

Outputting EO parameter uncertainties from Applanix POSPac



Edit Profile - NGS-RSD-EO

Available Data

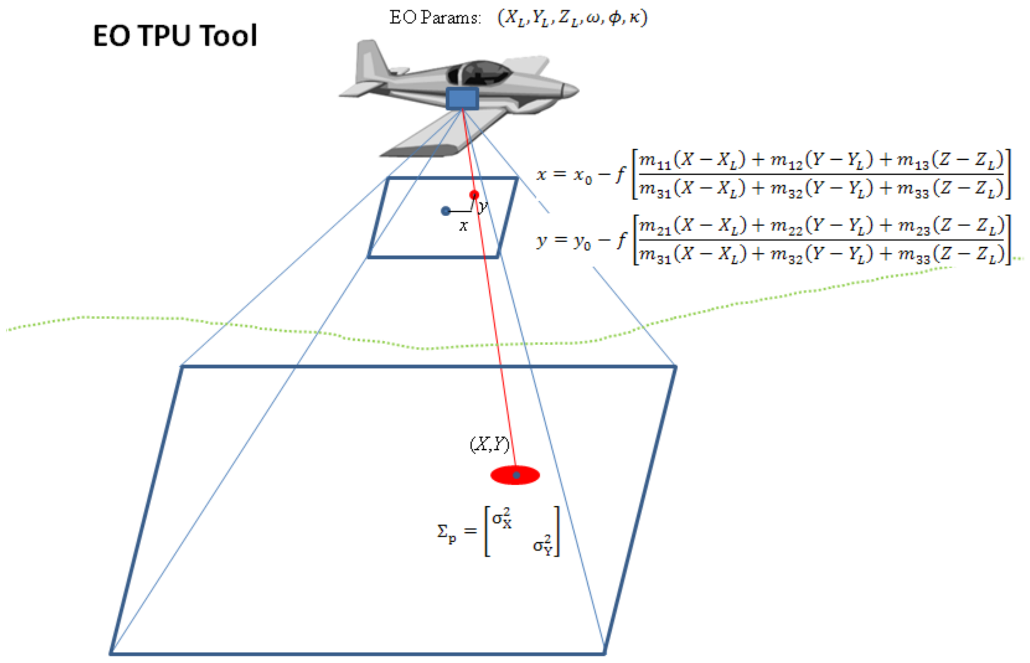
- POSEO Data
  - Time
  - Distance
  - Position
  - Orientation
  - Velocity
  - Event number
  - Photo ID
  - Photo ID Strip number
  - Position Standard Deviation
  - Orientation Standard Deviation
  - Velocity Standard Deviation
  - Acceleration
  - Angular Rate

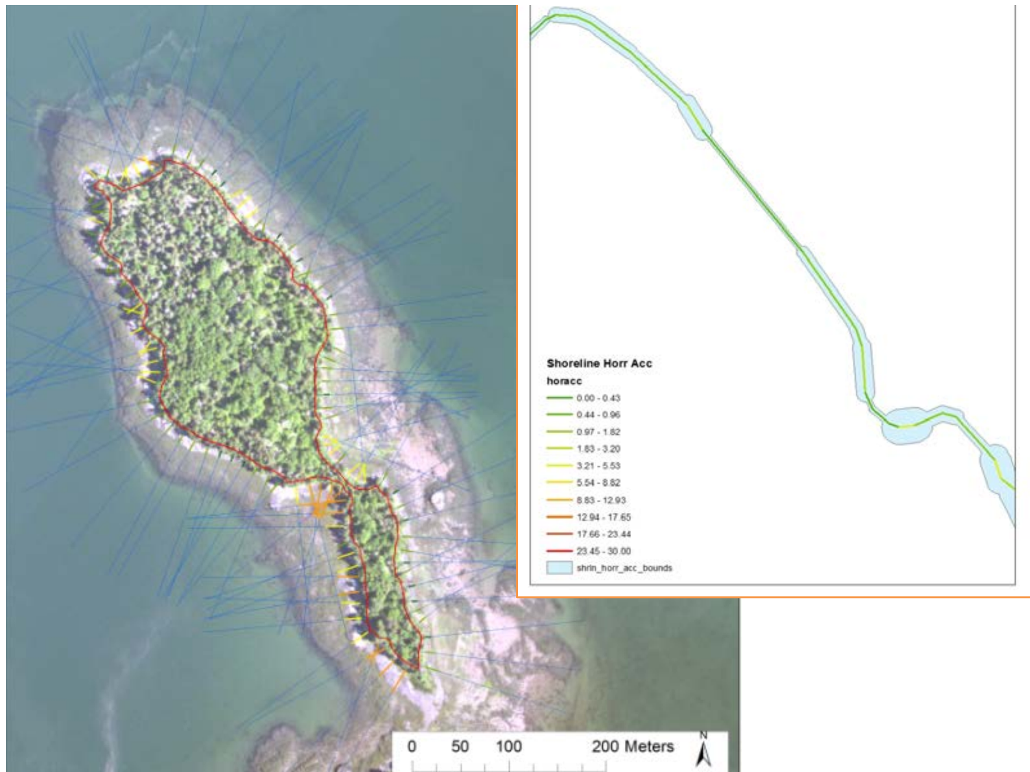
Export Data

Name	Format/Units	Example
Photo ID	String	Abcdefghijklmnopqrstuvwxyz
Time	Seconds of the Week	0.346
Latitude	Decimal Degrees (signed)	
Longitude	Decimal Degrees (signed)	
Easting	Meters	-45.122000
Northing	Meters	-9.560000
Orthometric height	Meters	-17.604000
Omega	degrees	54.774000
Phi	degrees	26.358000
Kappa	degrees	-17.868000
Easting Std Dev	Meters	58.756000
Northing Std Dev	Meters	72.191000
Height Std Dev	Meters	66.906000
Omega Std Dev	Degrees	5.937000
Phi Std Dev	Degrees	-14.157000
Kappa Std Dev	Degrees	-44.796000

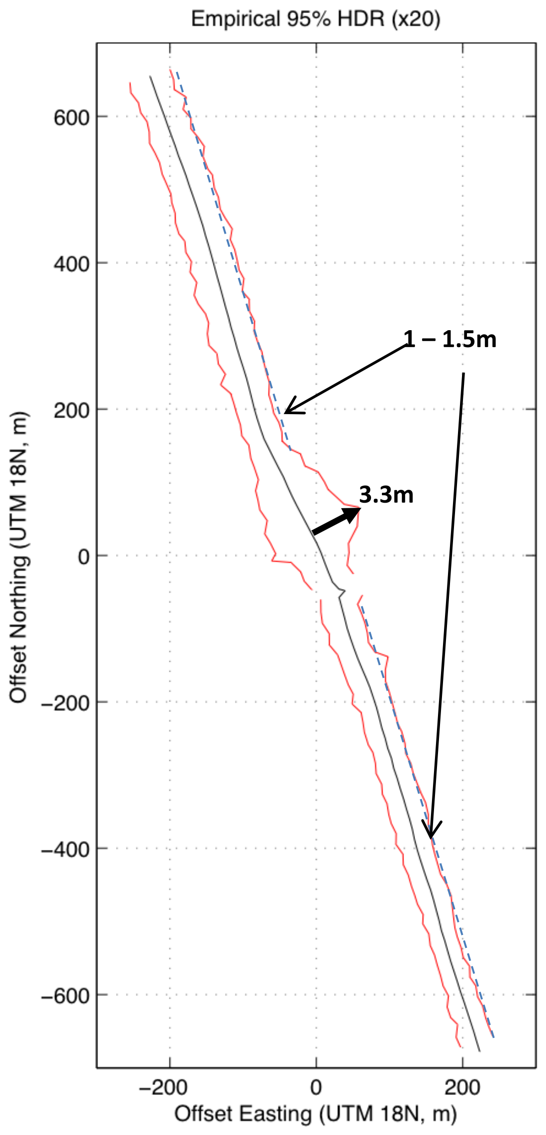
Shoreline TPU Modeling

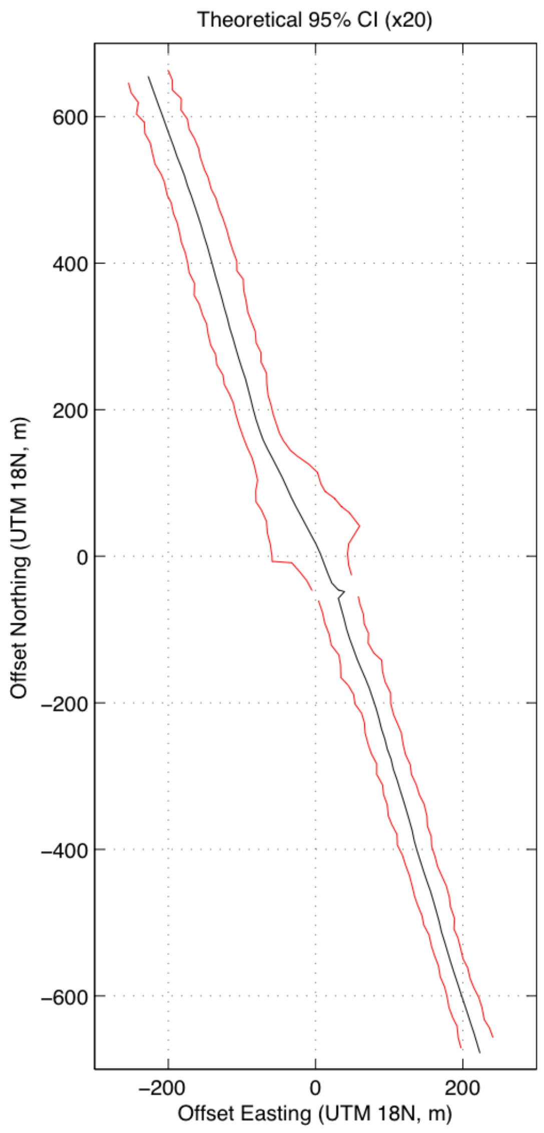
- Important for mapping /charting applications
  - Accuracy metadata
  - Verify satisfaction of IHO S-44 specs for coastline uncertainty
- Supports informed decision making, related to:
  - Ecosystem-based management
  - Zoning and permitting
  - Wetlands restoration
- Enables assessment of uncertainty in inundation models and other “downstream” analysis


EO TPU Tool
EO Params: $(X_L, Y_L, Z_L, \omega, \phi, \kappa)$
$x = x_0 - f\left[\frac{m_{11}(X-X_L)+m_{12}(Y-Y_L)+m_{13}(Z-Z_L)}{m_{31}(X-X_L)+m_{32}(Y-Y_L)+m_{33}(Z-Z_L)}\right]$
$y = y_0 - f\left[\frac{m_{21}(X-X_L)+m_{22}(Y-Y_L)+m_{23}(Z-Z_L)}{m_{31}(X-X_L)+m_{32}(Y-Y_L)+m_{33}(Z-Z_L)}\right]$
y
x
(X,Y)
$\Sigma_p = \begin{bmatrix} \sigma_X^2 & \\ & \sigma_Y^2 \end{bmatrix}$



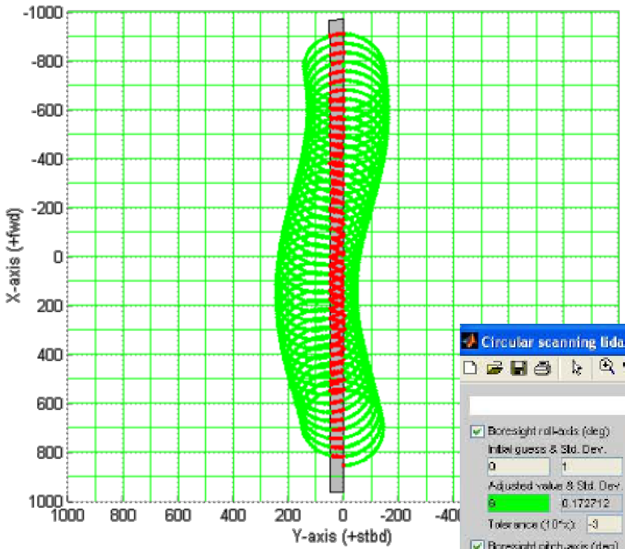
Shoreline TPU: validation using empirical accuracy assessment (field-surveyed data)



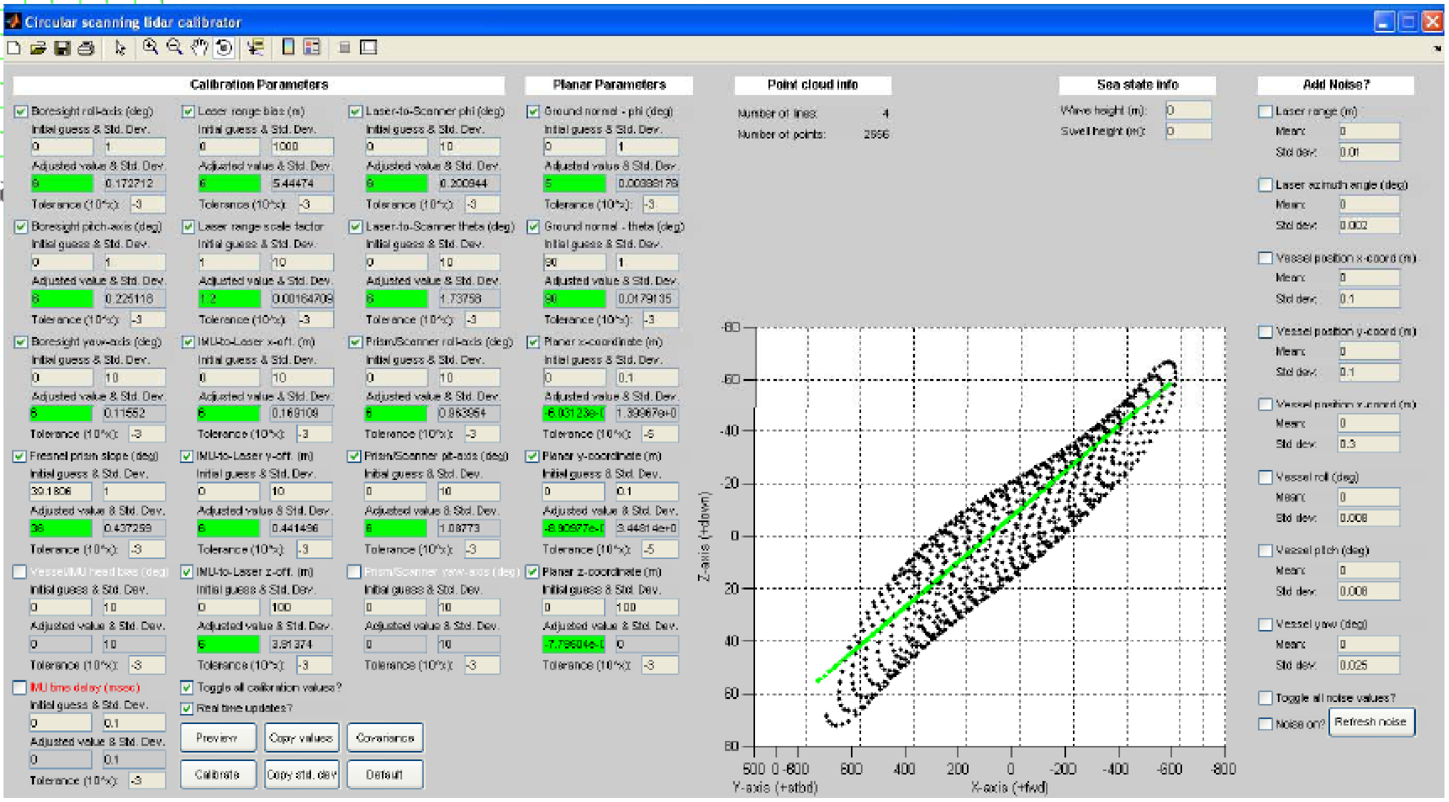


Empirical bounds computed from the data

Bathy Lidar: Geometric Calibration and TPU



Need estimates of positioning and orientation uncertainty



Calibration Parameters

	Parameter	Initial guess	Std. Dev.	Adjusted value	Std. Dev.	Tolerance (10^x)
✓	Boresight roll-axis (deg)	0	1	6	0.172712	-3
✓	Boresight pitch-axis (deg)	0	1	6	0.225118	-3
✓	Boresight yaw-axis (deg)	0	10	6	0.11552	-3
✓	Fresnel prism slope (deg)	39.1806	1	36	0.437259	-3
☐	Vessel/IMU head bias (deg)	0	10	0	10	-3
☐	IMU time delay (msec)	0	0.1	0	0.1	-3
✓	Laser range bias (m)	0	1000	6	5.44474	-3
✓	Laser range scale factor	1	10	1.2	0.00164709	-3
✓	IMU-to-Laser x-off. (m)	0	10	6	0.169109	-3
✓	IMU-to-Laser y-off. (m)	0	10	6	0.441496	-3
✓	IMU-to-Laser z-off. (m)	0	100	6	3.91374	-3
✓	Laser-to-Scanner phi (deg)	0	10	6	0.200944	-3
✓	Laser-to-Scanner theta (deg)	0	10	6	1.73758	-3
✓	Prism/Scanner roll-axis (deg)	0	10	6	0.963964	-3
✓	Prism/Scanner pit-axis (deg)	0	10	6	1.08773	-3
☐	Prism/Scanner yaw-axis (deg)	0	10	0	10	-3

✓ Toggle all calibration values?
✓ Real time updates?

Preview | Copy values | Covariance | Calibrate | Copy std. dev | Default

Planar Parameters

	Parameter	Initial guess	Std. Dev.	Adjusted value	Std. Dev.	Tolerance (10^x)
✓	Ground normal - phi (deg)	0	1	5	0.00388178	-3
✓	Ground normal - theta (deg)	90	1	90	0.0179135	-3
✓	Planar x-coordinate (m)	0	0.1	-6.03123e-(	1.39967e+0	-5
✓	Planar y-coordinate (m)	0	0.1	-8.90977e-(	3.44814e+0	-5
✓	Planar z-coordinate (m)	0	100	-7.79604e-(	0	-3

Point cloud info

Number of lines:	4
Number of points:	2556

Sea state info

Wave height (m):	0
Swell height (m):	0

Add Noise?

	Parameter	Mean	Std dev
☐	Laser range (m)	0	0.01
☐	Laser azimuth angle (deg)	0	0.002
☐	Vessel position x-coord (m)	0	0.1
☐	Vessel position y-coord (m)	0	0.1
☐	Vessel position z-coord (m)	0	0.3
☐	Vessel roll (deg)	0	0.008
☐	Vessel pitch (deg)	0	0.008
☐	Vessel yaw (deg)	0	0.025

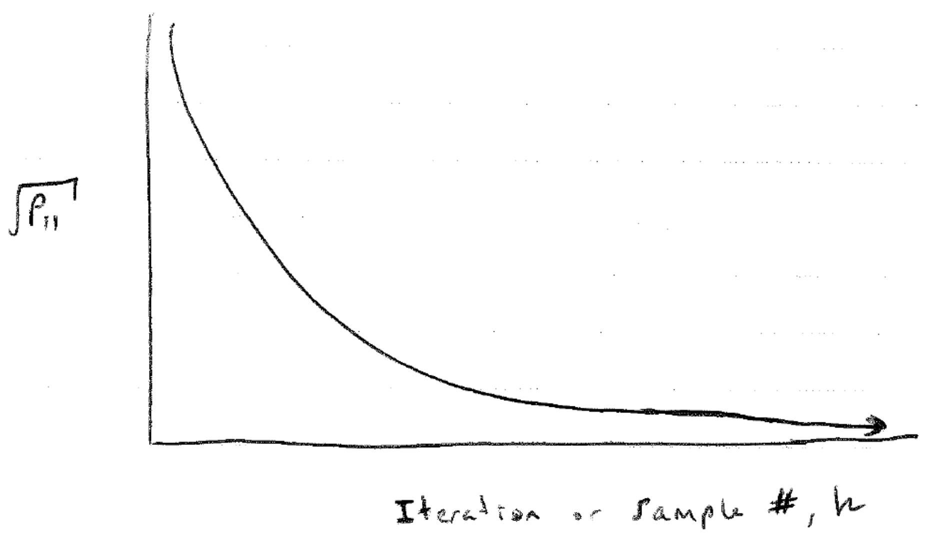
☐ Toggle all noise values?
☐ Noise on? Refresh noise

Images courtesy of Dr. Michael Gonsalves, LT/NOAA (NOS/NGS/RSD)

Something interesting

- What happens to the Kalman gain and Error Covariance over time as the algorithm iterates?

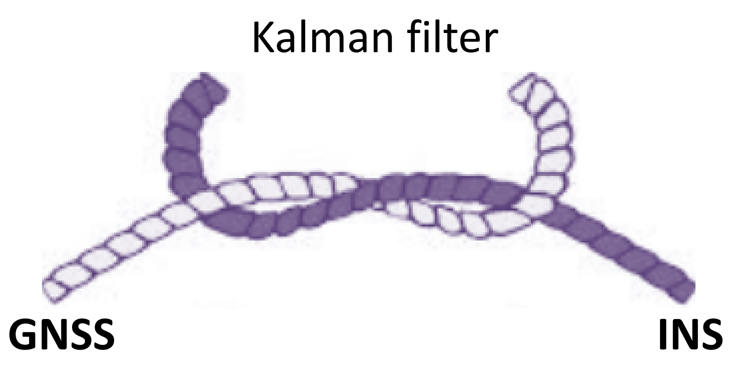
A: they initially change rapidly with time, but tend to converge to some steady state values after a certain number of iterations (at least for the types of state variable models we will consider in this class)



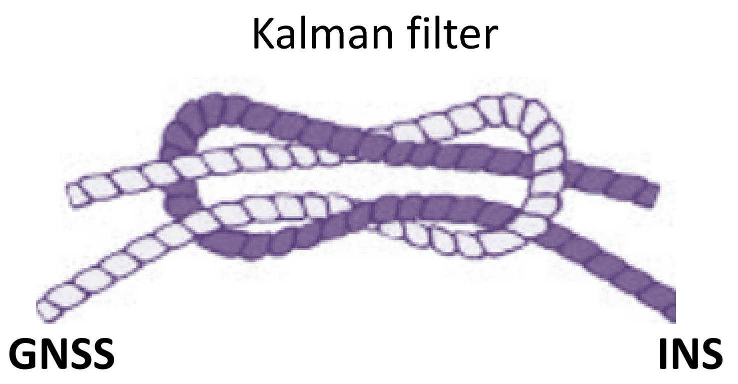
Uncertainty in Easting (Taken as square root of 1st element of main diagonal of $\underline{P}$)

Loose vs. tight coupling

Loosely coupled



Tightly coupled



Loosely-coupled vs. tightly-coupled

- Loosely coupled: GNSS processing is performed separately and the used in the Kalman filter to correct the inertial solution
- Tightly coupled: single Kalman filter is used to estimate both the inertial errors and GNSS integer ambiguities
  - Especially useful when the number of GNSS satellites drops below the minimum number
  - Urban canyons are a frequent issue in mobile lidar

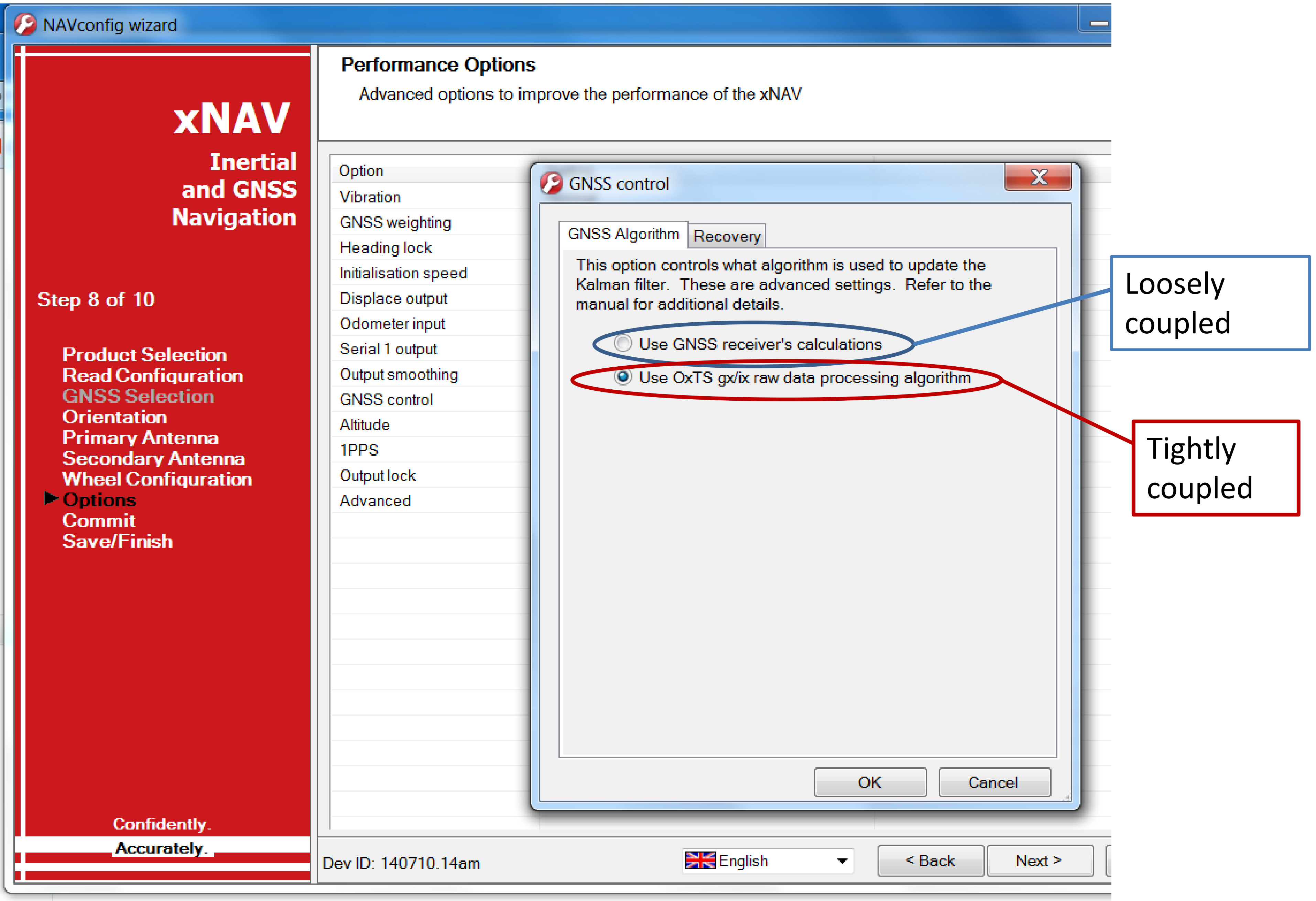
Benefit of tightly-coupled solutions

- Continues to produce good navigation solution, even when GNSS signals are lost
- Very important for working in urban canyons!

Additional Benefits of Tight Coupling

- INS sensors can be continuously calibrated whenever GNSS is available
  - Recall lecture on inertial navigation and the need for good calibration…
- System health can be monitored via rapid changes in calibration parameters

NAVconfig: GNSS Control Window



NAVconfig wizard

xNAV

Inertial and GNSS Navigation

Step 8 of 10

- Product Selection
- Read Configuration
- GNSS Selection
- Orientation
- Primary Antenna
- Secondary Antenna
- Wheel Configuration
- ▶ Options
- Commit
- Save/Finish

Confidently. Accurately.

Performance Options

Advanced options to improve the performance of the xNAV

Option
Vibration
GNSS weighting
Heading lock
Initialisation speed
Displace output
Odometer input
Serial 1 output
Output smoothing
GNSS control
Altitude
1PPS
Output lock
Advanced

GNSS control

GNSS Algorithm | Recovery

This option controls what algorithm is used to update the Kalman filter. These are advanced settings. Refer to the manual for additional details.

- ○ Use GNSS receiver's calculations
- ● Use OxTS gx/ix raw data processing algorithm

OK | Cancel

Dev ID: 140710.14am

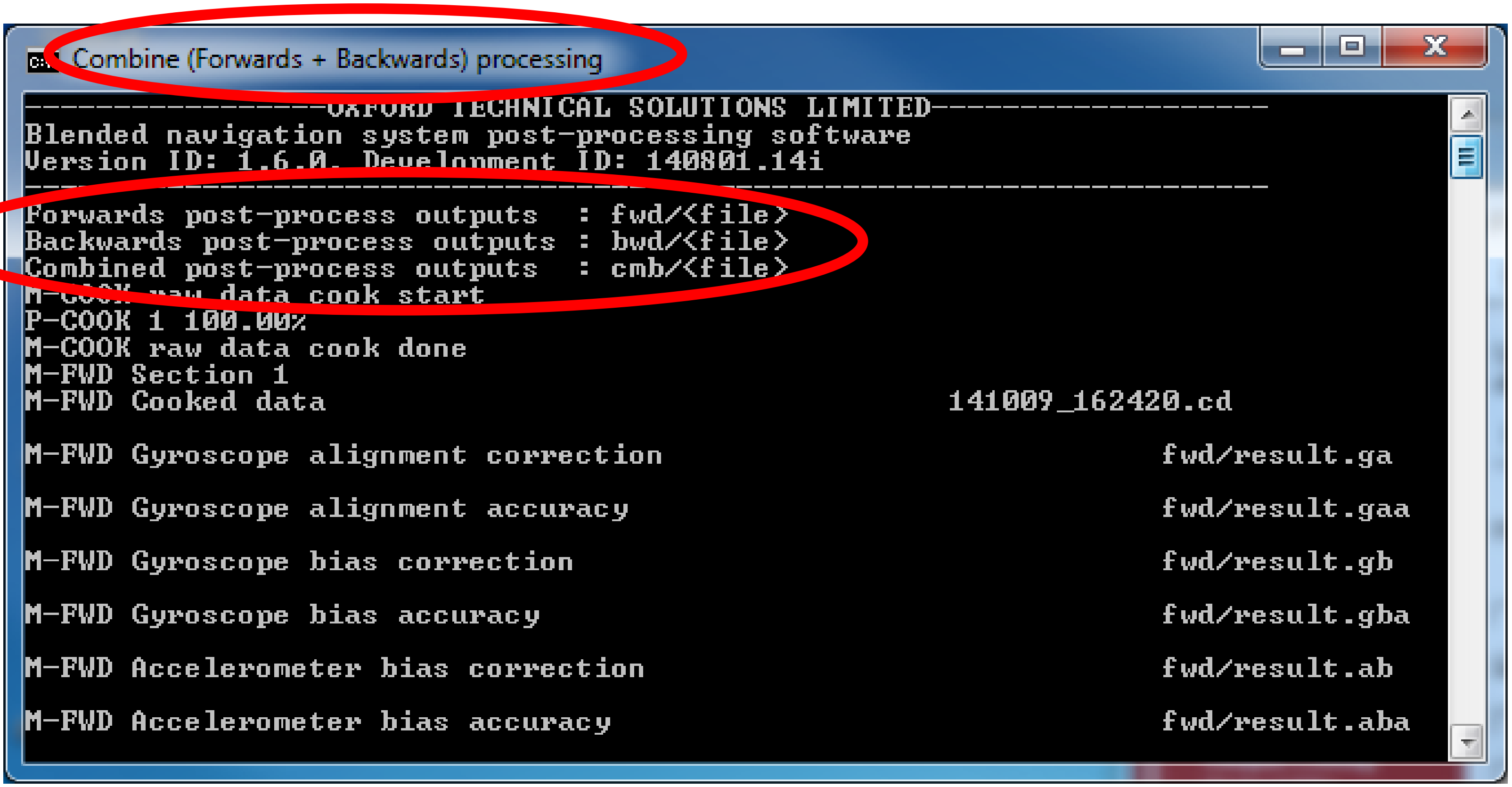
English | < Back | Next >

Loosely coupled

Tightly coupled

Other miscellaneous things to know:

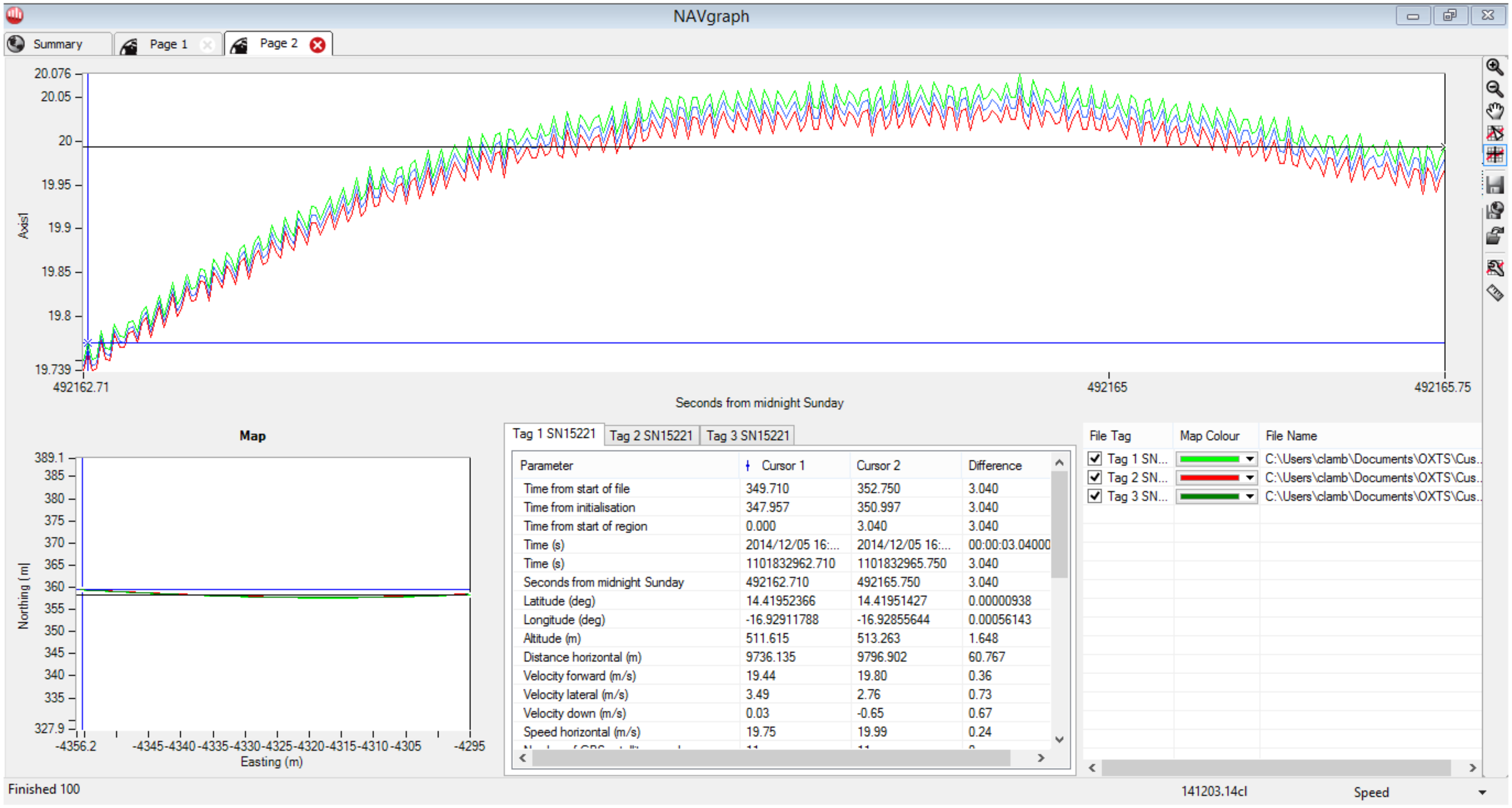
- Forward and reverse time processing helps make Kalman filter invariant to changes in DOP (satellites coming into and out of view)



Combine (Forwards + Backwards) processing

```
-----------------OXFORD TECHNICAL SOLUTIONS LIMITED-------------------
Blended navigation system post-processing software
Version ID: 1.6.0. Development ID: 140801.14i
----------------------------------------------------------------------
Forwards post-process outputs  : fwd/<file>
Backwards post-process outputs : bwd/<file>
Combined post-process outputs  : cmb/<file>
M-COOK raw data cook start
P-COOK 1 100.00%
M-COOK raw data cook done
M-FWD Section 1
M-FWD Cooked data                                    141009_162420.cd

M-FWD Gyroscope alignment correction                             fwd/result.ga

M-FWD Gyroscope alignment accuracy                               fwd/result.gaa

M-FWD Gyroscope bias correction                                  fwd/result.gb

M-FWD Gyroscope bias accuracy                                    fwd/result.gba

M-FWD Accelerometer bias correction                              fwd/result.ab

M-FWD Accelerometer bias accuracy                                fwd/result.aba
```

- Green = forward solution
- Red = reverse solution
- Blue = combined solution



Parameter	Cursor 1	Cursor 2	Difference
Time from start of file	349.710	352.750	3.040
Time from initialisation	347.957	350.997	3.040
Time from start of region	0.000	3.040	3.040
Time (s)	2014/12/05 16:...	2014/12/05 16:...	00:00:03.04000
Time (s)	1101832962.710	1101832965.750	3.040
Seconds from midnight Sunday	492162.710	492165.750	3.040
Latitude (deg)	14.41952366	14.41951427	0.00000938
Longitude (deg)	-16.92911788	-16.92855644	0.00056143
Altitude (m)	511.615	513.263	1.648
Distance horizontal (m)	9736.135	9796.902	60.767
Velocity forward (m/s)	19.44	19.80	0.36
Velocity lateral (m/s)	3.49	2.76	0.73
Velocity down (m/s)	0.03	-0.65	0.67
Speed horizontal (m/s)	19.75	19.99	0.24

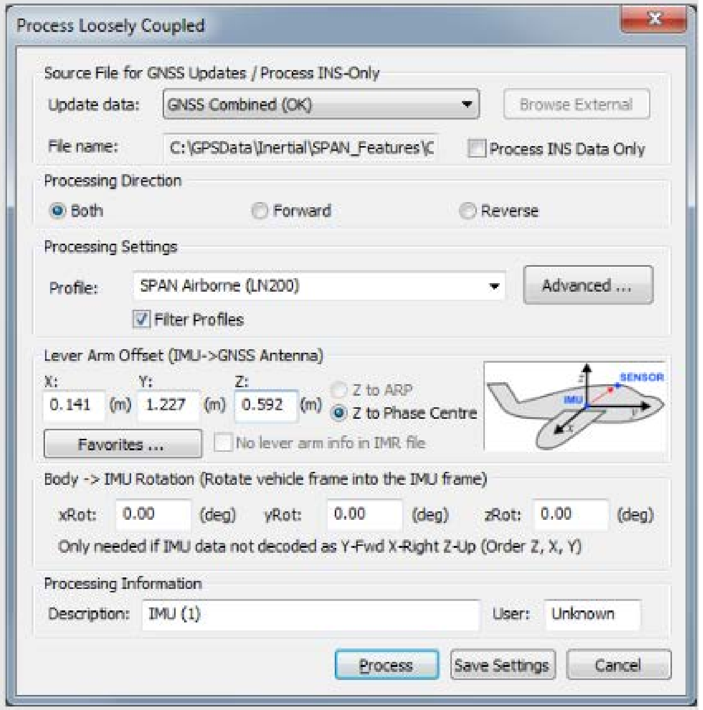
File Tag	Map Colour	File Name
✓ Tag 1 SN...	🟩	C:\Users\clamb\Documents\OXTS\Cus...
✓ Tag 2 SN...	🟥	C:\Users\clamb\Documents\OXTS\Cus...
✓ Tag 3 SN...	🟩	C:\Users\clamb\Documents\OXTS\Cus...

Finished 100

141203.14cl

Speed

Inertial Explorer settings



Process Loosely Coupled

Source File for GNSS Updates / Process INS-Only

Update data: GNSS Combined (OK) | Browse External

File name: C:\GPSData\Inertial\SPAN_Features\C | Process INS Data Only

Processing Direction

[✓ Both] Forward Reverse

Processing Settings

Profile: SPAN Airborne (LN200) | Advanced ...

[✓] Filter Profiles

Lever Arm Offset (IMU->GNSS Antenna)

X: 0.141 (m) Y: 1.227 (m) Z: 0.592 (m)

Z to ARP [✓ Z to Phase Centre]

Favorites ... | No lever arm info in IMR file

Body -> IMU Rotation (Rotate vehicle frame into the IMU frame)

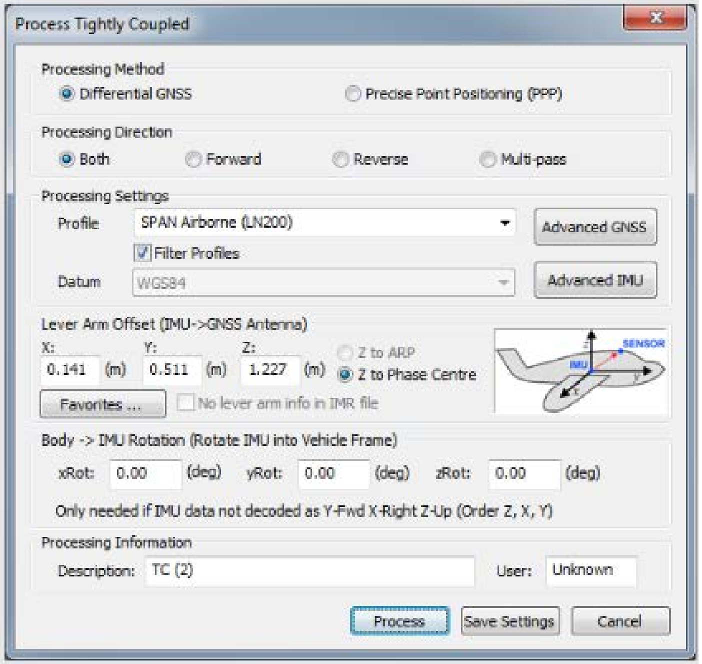
xRot: 0.00 (deg) yRot: 0.00 (deg) zRot: 0.00 (deg)

Only needed if IMU data not decoded as Y-Fwd X-Right Z-Up (Order Z, X, Y)

Processing Information

Description: IMU (1) User: Unknown

Process | Save Settings | Cancel



Process Tightly Coupled

Processing Method

[✓ Differential GNSS] Precise Point Positioning (PPP)

Processing Direction

[✓ Both] Forward Reverse Multi-pass

Processing Settings

Profile SPAN Airborne (LN200) | Advanced GNSS

[✓] Filter Profiles

Datum WGS84 | Advanced IMU

Lever Arm Offset (IMU->GNSS Antenna)

X: 0.141 (m) Y: 0.511 (m) Z: 1.227 (m)

Z to ARP [✓ Z to Phase Centre]

Favorites ... | No lever arm info in IMR file

Body -> IMU Rotation (Rotate IMU into Vehicle Frame)

xRot: 0.00 (deg) yRot: 0.00 (deg) zRot: 0.00 (deg)

Only needed if IMU data not decoded as Y-Fwd X-Right Z-Up (Order Z, X, Y)

Processing Information

Description: TC (2) User: Unknown

Process | Save Settings | Cancel

http://www.novatel.com/assets/Documents/Waypoint/Downloads/InertialExplorer850_Manual.pdf

Extended Kalman Filter (EKF)

- The Kalman filter is designed for linear systems
  - The Kalman Filter is the optimal estimate for *linear* system models with additive independent white noise in the transition and measurement systems
  - The system model is assumed to be linear
- But…most real world systems are nonlinear!
- So, what do we do??
  - One idea: we can linearize the system model and then just proceed to use the regular Kalman filter
  - But, this is risky
    - The linearized model only does a good job of approximating the real system near the linearization point
    - Could give poor performance or diverge

EKF (continued)

- Solution: The Extended Kalman Filter (EKF)
  - Version of Kalman filter for nonlinear systems
- In the EKF, the state transition and observation models are no longer assumed to be linear functions of the state

Linear Kalman filter	Extended Kalman filter
$\mathbf{x}_k = \mathbf{\Phi}_{k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1}$	$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$
$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$	$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$

where *f* and *h* are differentiable, but possibly nonlinear, functions

So, steps 1 and 3 of the Kalman filter algorithm change to use the nonlinear model equations, and in the other steps, **Φ** and **H** become the matrices of partial derivatives of the nonlinear equations (i.e., Jacobians). Otherwise the EKF algorithm is the same as that of the linear Kalman filter.

EKF (continued)

- But, one important caveat you should know about…

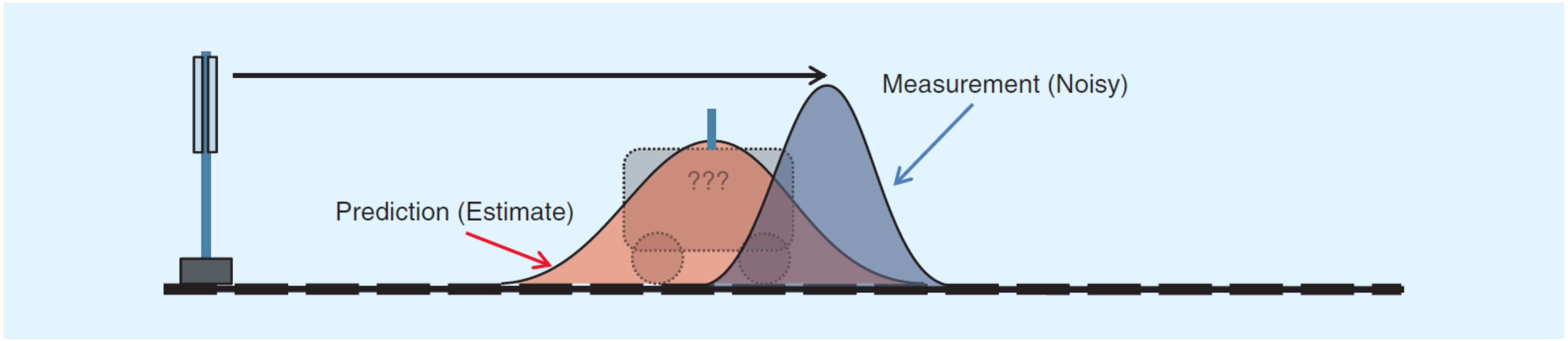
The EKF can diverge!

- Application of the Kalman filter to nonlinear systems is still an active area of research with new algorithms being developed

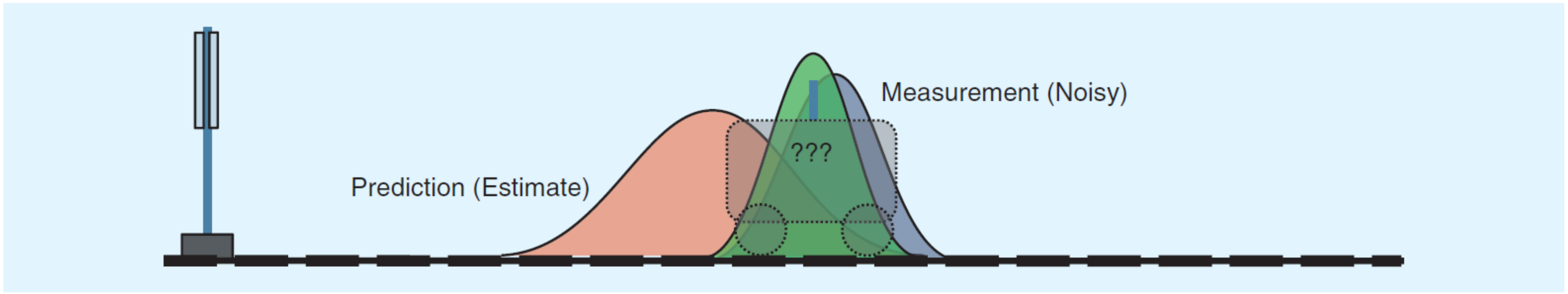
Unscented Kalman filter (UKF)

- Alternative to Extended Kalman Filter (EKF) for dealing with the problem of nonlinearity
  - Works well for highly-nonlinear systems
  - No need to explicitly calculate Jacobians
  - Doesn’t have the divergence problem of EKf
- Has rapidly gained popularity in last decade
- Developed at University of Oxford (UK) by Julier and Uhlman, along with Durrant-Whyte at University of Sydney (AUS)
- Based on the “unscented transformation”

Another intuitive view of what Kalman filter does



[FIG4] **Shows the measurement of the location of the train at time *t* = 1 and the level of uncertainty in that noisy measurement, represented by the blue Gaussian pdf. The combined knowledge of this system is provided by multiplying these two pdfs together.**



[FIG5] **Shows the new pdf (green) generated by multiplying the pdfs associated with the prediction and measurement of the train's location at time *t* = 1. This new pdf provides the best estimate of the location of the train, by fusing the data from the prediction and the measurement.**

Faragher, R., 2012. Understanding the Basics of the Kalman Filter Via a Simple and Intuitive Derivation. *IEEE Signal Processing Magazine*, pp. 128-132.

References

∗ Kim, P., 2011. *Kalman Filter for Beginners with MATLAB Examples*. Createspace. ISBN: 1463648359, 233 pp.

Grewal, M.S., and A.P. Andrews, 2011. *Kalman filtering: theory and practice using MATLAB*. John Wiley & Sons, 575 pp.

Riisgaard, S. and M.R. Blas, 2005. SLAM for Dummies, MIT OpenCourseWare: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

Faragher, R., 2012. Understanding the Basics of the Kalman Filter Via a Simple and Intuitive Derivation. *IEEE Signal Processing Magazine*, pp. 128-132.