

[New chat](#)

My Stuff



CSV to Protobu...

JSON to CSV...

Simple CSV to...



Gems



Chats

Kalman Filter for Bistatic Measureme...

in matlab the kalman filter predict eq...

Cache-Friendly C++ Sets and Maps

Connect ESP32 to Windows Laptop

Cache-Friendly C++ Map Implementa...

AC-3's Role in CSPs

using a unscented kalman filter can I update a ecef ecefdot state with a measurement of bistatic range and angle of transmission and bistatic range rate coordinate



Yes, you absolutely can. Using an **Unscented Kalman Filter (UKF)** is actually a very smart choice for this specific problem because the mapping from ECEF coordinates to bistatic measurements is highly non-linear.

While a standard Extended Kalman Filter (EKF) would require you to calculate complex Jacobian matrices (which are a nightmare for bistatic geometry), the UKF handles this by propagating "sigma points" through your non-linear equations.

## 1. Defining Your State and Measurement Vectors

To make this work, we define the relationship between what you have (the state) and what you see (the measurements).

### The State Vector ( $x$ )

In ECEF (Earth-Centered, Earth-Fixed), your state typically includes position and velocity:

$$x = [r_x \ r_y \ r_z \ v_x \ v_y \ v_z]^T$$

### The Measurement Vector ( $z$ )

Your bistatic setup provides three distinct pieces of data:

1. **Bistatic Range ( $\rho$ )**: The total path length from transmitter to target to receiver, minus the baseline distance.
2. **Angle of Transmission ( $\theta$ )**: The direction (azimuth/elevation) from the transmitter to the target.
3. **Bistatic Range Rate ( $\dot{\rho}$ )**: The time rate of change of the bistatic range (Doppler shift).

## 2. The Measurement Model ( $h(x)$ )

"If the target is at this ECEF position, what would the sensors see?"

### Bistatic Range

The bistatic range is the distance from the transmitter ( $T$ ) to the target ( $X$ ) plus the distance from the target to the receiver ( $R$ ):

$$\rho = (\|r_X - r_T\| + \|r_X - r_R\|) - \|r_R - r_T\|$$

### Bistatic Range Rate

This is the projection of the target's ECEF velocity onto the two unit vectors pointing toward the transmitter and receiver:

$$\dot{\rho} = (v_X \cdot \hat{u}_{TX}) + (v_X \cdot \hat{u}_{RX})$$

(Note: This assumes the transmitter and receiver are stationary in ECEF. If they are moving, like satellites, you'd subtract their velocity vectors as well.)  
Bistatic measurements have a "banana-shaped" uncertainty profile.

- **Linearization Errors**: An EKF tries to approximate that curve with a straight line, which often leads to the filter diverging if the target is close to the baseline.
- **The Unscented Transform**: The UKF picks a few points (sigma points) around your ECEF estimate, passes them through the range/angle equations, and calculates the new mean and covariance. This captures the **second-order** physics of the geometry, making it much more stable.

### Potential Pitfalls

- **Coordinate Singularity**: Ensure your ECEF-to-Angle conversion handles the poles correctly.
- **Baseline Issues**: If the target is directly on the line between the transmitter and receiver (the baseline), the geometry becomes "singular" and your range-rate measurements provide very little information about lateral movement.

In this example, we assume a stationary transmitter and receiver (like ground-based passive radar). The "Update" step takes your **predicted** state (where you think the object is) and corrects it using the **actual** measurement.

#### MATLAB Implementation: UKF Measurement Update

This script defines the state, the non-linear measurement function  $h(x)$ , and performs one iteration of the UKF update.

Matlab

```
% --- UKF Bistatic Update Example ---
% State vector x = [px; py; pz; vx; vy; vz] (ECEF)

% 1. Setup Scenario
tx_pos = [6378137; 0; 0]; % Transmitter ECEF (on equator)
rx_pos = [6378000; 1000; 0]; % Receiver ECEF (~1km away)
L = norm(tx_pos - rx_pos); % Baseline distance
```

To implement this, we will use the **Unscented Transform** to handle the non-linear mapping from ECEF space to bistatic measurement space.

In this example, we assume a stationary transmitter and receiver (like ground-based passive radar). The "Update" step takes your **predicted** state (where you think the object is) and corrects it using the **actual** measurement.

#### MATLAB Implementation: UKF Measurement Update

This script defines the state, the non-linear measurement function  $h(x)$ , and performs one iteration of the UKF update.

Matlab

```
% --- UKF Bistatic Update Example ---
% State vector x = [px; py; pz; vx; vy; vz] (ECEF)

% 1. Setup Scenario

% 4. Pass Sigma Points through Measurement Function h(x)
Z_sig = zeros(3, 2*n+1);
for i = 1:size(sigmas, 2)
    Z_sig(:,i) = bistatic_h(sigmas(:,i), tx_pos, rx_pos, L);
end
```

```
% 5. Calculate Mean and Covariance of Measurement
Wm = [Lambda/(n+lambda), repmat(1/(2*(n+lambda)), 1, 2*n)];
Wc = Wm; Wc(1) = Wc(1) + (1 - alpha^2 + beta);
```

```
z_hat = sum(Wm .* Z_sig, 2); % Predicted measurement
Pzz = R;
Pxz = zeros(n, 3);

for i = 1:2*n+1
    res_z = Z_sig(:,i) - z_hat;
    res_x = sigmas(:,i) - x_pred;
    Pzz = Pzz + Wc(i) * (res_z * res_z');
    Pxz = Pxz + Wc(i) * (res_x * res_z');
end
```

```
% 6. Compute Kalman Gain and Update
K = Pxz / Pzz;
x_updated = x_pred + K * (z_meas - z_hat);
P_updated = P_pred - K * Pzz * K';
```

```
% --- Display Results ---
disp('Updated ECEF State:');
disp(x_updated);
```

```
function z = bistatic_h(x, tx, rx, L)
    % Extract position and velocity
    pos = x(1:3); vel = x(4:6);

    r_tx = norm(pos - tx);
    r_rx = norm(pos - rx);

    % 1. Bistatic Range
    rho = (r_tx + r_rx) - L;

    % 2. Angle of Transmission (Azimuth in local TX frame)
    % Simplified: Using ECEF-relative angle for example
    pos = x(1:3); vel = x(4:6);

    r_tx = norm(pos - tx);
    r_rx = norm(pos - rx);

    % 1. Bistatic Range
    rho = (r_tx + r_rx) - L;

    % 2. Angle of Transmission (Azimuth in local TX frame)
    % Simplified: Using ECEF-relative angle for example
```

```

d = pos - tx;
az = atan2(d(2), d(1));

% 3. Bistatic Range Rate
u_tx = (pos - tx) / r_tx;
u_rx = (pos - rx) / r_rx;
rho_dot = dot(vel, u_tx) + dot(vel, u_rx);

z = [rho; az; rho_dot];
end

```

```

% Simplified: Using ECEF-relative angle for example
d = pos - tx;
az = atan2(d(2), d(1));

% 3. Bistatic Range Rate
u_tx = (pos - tx) / r_tx;
u_rx = (pos - rx) / r_rx;
rho_dot = dot(vel, u_tx) + dot(vel, u_rx);

z = [rho; az; rho_dot];
end

```

### Numeric Example Walkthrough

Let's look at how the filter processes the **Bistatic Range Rate**:

- Prior Estimate:** Your model thinks the object is moving at 100 m/s away from the transmitter.
- Sigma Points:** The UKF creates "test states" slightly faster and slower than 100 m/s.
- Observation:** It calculates the expected Doppler shift for every one of those points.
- Correction:** If the actual sensor measurement  $z_{meas}$  shows a range rate of  $-10$  m/s (moving toward the baseline), the Kalman Gain ( $K$ ) will weight the difference.
- Result:** The state vector  $x$  will be "pulled" from its predicted 100 m/s toward the measured velocity, while simultaneously adjusting the position ( $p_x, p_y, p_z$ ) because position and velocity are correlated through the geometry.

### Key Considerations for your Implementation

- The Angle Model:** In the code above, I used a simple `atan2` for the angle. In a real ECEF application, you must rotate the ECEF vector into the **Transmitter's Local Tangent Plane** (ENU or NED) before calculating Azimuth and Elevation.
- Measurement Noise ( $R$ ):** If your range-rate measurement is extremely accurate (e.g., from a high-stability oscillator), decrease the third value in the `R` matrix. This forces the filter to trust the velocity measurement more than the initial position estimate.

Would you like me to expand the `bistatic_h` function to include a full ECEF-to- $K$  by the residual, the measurement units cancel out, leaving you with units that match your **State Space** (ECEF).

## 2. Visualizing the Update

Think of it like a GPS navigator:

- Prediction ( $x_{pred}$ ):** Your car's internal model thinks you are at a specific ECEF coordinate.
- Innovation ( $z_{meas} - z_{hat}$ ):** The satellite says, "Your range is 100 meters longer than you thought."
- Correction:** The Kalman Gain  $K$  knows that a 100m error in *range* (bistatic space) translates to, say, a 15-meter error in *X* and a 20-meter error in *Y* (ECEF space).

## 3. Why the UKF handles this better than the EKF

In a standard EKF, you would use a Jacobian matrix ( $H$ ) to do this translation. The Jacobian is a matrix of partial derivatives:

$$\frac{\partial \rho}{\partial r_x}, \frac{\partial \rho}{\partial r_y}, \dots$$

These derivatives are the "slope" that tells you how much the ECEF state should change for a tiny change in bistatic range.

**In your UKF:** Instead of calculus, you used **Sigma Points**. By pushing those ECEF points through your `bistatic_h` function, the filter "discovered" the relationship between ECEF and bistatic units numerically. The cross-covariance matrix  $P_{xz}$  you calculated in the code is what builds that unit-bridge for  $K$ .

## 4. One Critical Warning: Angles

While you can use Bistatic ECEF angles, Angles of Arrival/Emission are not recommended.

While you can mix Range and ECEF easily, Angles (Azimuth/Elevation) require care because they are periodic (they wrap around at  $2\pi$ ).

When you calculate  $z_{meas} - z_{hat}$  for an angle:

- If  $z_{meas}$  is  $359^\circ$  and  $z_{hat}$  is  $1^\circ$ , the raw math says the error is  $358^\circ$ .
- In reality, the error is only  $-2^\circ$ .

**Pro Tip:** Always use a "residual wrap" function for your angle measurements to ensure the difference stays within  $[-\pi, \pi]$ .

Would you like me to show you how to add the angle-wrapping logic to the MATLAB code to prevent the filter from "jumping" at the  $2\pi$  boundary?

my measurement sometimes does not have a range rate, can i still update my state position and velocity?

handle partial updates.

If your sensor only returns range and angle (but no range rate), you can still update the entire state—including velocity. Here is how that works mechanically and why it's possible.

### 1. How the Filter Handles "Missing" Data

In the Kalman Update, you don't have to change your state vector; you simply change the size of your **Measurement Vector ( $z$ )** and your **Measurement Noise Matrix ( $R$ )**.

If the range rate is missing for a specific time step:

- Your  $z_{meas}$  becomes a  $2 \times 1$  vector (Range, Angle).
- Your  $h(x)$  function only outputs those two values.
- Your  $R$  matrix becomes a  $2 \times 2$  matrix.

The math still works perfectly. The Kalman Gain ( $K$ ) will now be a  $6 \times 2$  matrix instead of  $6 \times 3$ . It will still distribute the "correction" across all 6 state variables (Position and Velocity).

ensure the difference stays within  $[-\pi, \pi]$ .

Would you like me to show you how to add the angle-wrapping logic to the MATLAB code to prevent the filter from "jumping" at the  $2\pi$  boundary?

my measurement sometimes does not have a range rate, can i still update my state position and velocity?

Yes, you absolutely can. One of the best features of the Kalman Filter family is its ability to handle partial updates.

If your sensor only returns range and angle (but no range rate), you can still update the entire state—including velocity. Here is how that works mechanically and why it's possible.

### 1. How the Filter Handles "Missing" Data

In the Kalman Update, you don't have to change your state vector; you simply change the size of your **Measurement Vector ( $z$ )** and your **Measurement Noise Matrix ( $R$ )**.

If the range rate is missing for a specific time step:

- Your  $z_{meas}$  becomes a  $2 \times 1$  vector (Range, Angle).
- Your  $h(x)$  function only outputs those two values.
- Your  $R$  matrix becomes a  $2 \times 2$  matrix.

The math still works perfectly. The Kalman Gain ( $K$ ) will now be a  $6 \times 2$  matrix instead of  $6 \times 3$ . It will still distribute the "correction" across all 6 state variables (Position and Velocity).

### 2. Can you update Velocity without a Velocity Measurement?

Yes. This is due to the **Cross-Covariance ( $P$ )**.

#### 4. How to implement this in MATLAB

You don't need two separate scripts. You can simply use an `if` statement to check your data:

```
Matlab
```

```
% Logic for handling missing Range Rate
if is_range_rate_available
    z = [rho; az; rho_dot];
    R_current = diag([R_range, R_az, R_dot]);
else
    z = [rho; az]; % Drop the third element
    R_current = diag([R_range, R_az]); % Drop the third row/column
end
```

**High Maneuvers** If the target turns sharply while you have no range rate, the filter will lag significantly behind the real velocity.

[Export to Sheets](#)

#### 4. How to implement this in MATLAB

You don't need two separate scripts. You can simply use an `if` statement to check your data:

```
Matlab
```

```
% Logic for handling missing Range Rate
if is_range_rate_available
    z = [rho; az; rho_dot];
    R_current = diag([R_range, R_az, R_dot]);
else
    z = [rho; az]; % Drop the third element
    z = [rho; az]; % Drop the third element
    R_current = diag([R_range, R_az]); % Drop the third row/column
end
```

```
% The rest of the UKF math (sigma points, K-gain)
% remains exactly the same!
```

##### Recommendation

If your system frequently loses the range rate (Doppler), ensure your **Process Noise ( $Q$ )** is tuned correctly. If  $Q$  is too high, the filter will "forget" the previous velocity too quickly, making it harder to estimate movement from position alone.

Would you like me to show you how to tune the Process Noise ( $Q$ ) to keep your velocity estimates stable when range rate data is missing?

yes



Tuning the **Process Noise covariance matrix ( $Q$ )** is the "secret sauce" for keeping a filter stable when sensor data gets spotty.

Think of  $Q$  as your "**Uncertainty Growth**" rate. It tells the filter how much you expect the target's velocity to change (due to maneuvering or wind) between measurements.

#### 1. The Discretized Constant Velocity $Q$

For an ECEF state  $[p_x, p_y, p_z, v_x, v_y, v_z]$ , we typically assume the "noise" enters the system as random accelerations. We use a parameter called **Power Spectral Density ( $S_q$ )**, which represents how "jerky" the target's movement is in  $(m/s^2)^2/Hz$ .

For a time step  $\Delta t$ , the  $Q$  matrix for one dimension (e.g.,  $x$ ) looks like this:

##### If $Q$ is too LOW:

The filter thinks the target is moving in a perfectly straight line (like a satellite).

- **The Result:** The filter becomes "sluggish." If the target actually turns, the filter will keep sailing straight for several seconds before the position error finally drags the velocity estimate around.

#### 3. MATLAB Code: Dynamic $Q$ and Logic

Here is how you implement the logic to handle missing data while keeping the velocity stable.

### Matlab

```
% --- UKF Tuning for Missing Range Rate ---

dt = 1.0;           % Time step (seconds)
Sq = 0.5;           % Tuning parameter (lower = smoother velocity)
```

#### If $Q$ is too LOW:

The filter thinks the target is moving in a perfectly straight line (like a satellite).

- **The Result:** The filter becomes "sluggish." If the target actually turns, the filter will keep sailing straight for several seconds before the position error finally drags the velocity estimate around.

### 3. MATLAB Code: Dynamic Q and Logic

Here is how you implement the logic to handle missing data while keeping the velocity stable.

### Matlab

```
% --- UKF Tuning for Missing Range Rate ---

dt = 1.0;           % Time step (seconds)
Sq = 0.5;           % Tuning parameter (lower = smoother velocity)

% 1. Build the Process Noise Matrix Q
q_block = Sq * [dt^3/3, dt^2/2; dt^2/2, dt];
Q = blkdiag(q_block, q_block, q_block); % For X, Y, and Z

% 2. Prediction Step (Standard)
x_pred = F * x_old;           % F is your State Transition Matrix
P_pred = F * P_old * F' + Q;   % Velocity uncertainty grows here

% 3. Update Step with Conditional Logic
if range_rate_valid
    % Full 3-measurement update
    z_meas = [rho; az; rho_dot];
    R = diag([10^2, 0.001^2, 0.5^2]);
else
    % Partial 2-measurement update (Velocity is updated via P-matrix correlation)
    z_meas = [rho; az];
    R = diag([10^2, 0.001^2]);
end

% ... Proceed with Sigma Points and K-Gain as normal ...
```

### Summary Checklist for Stability

1. **Lower  $S_q$ :** If your velocity is "noisy" when Range Rate is missing, decrease  $S_q$  (e.g., from 1.0 to 0.1).
2. **Increase  $\Delta t$ :** If your sensor frequency is low, you need a slightly larger  $Q$  to account for what the target might have done while you weren't looking.
3. **Cross-Check ECEF:** Ensure your  $Q$  is defined in ECEF. If the target is moving in a specific local direction (like North), a 3D block-diagonal  $Q$  treats all directions as equally likely to maneuver.

Would you like me to help you define the State Transition Matrix ( $F$ ) used in the prediction step to ensure it matches this ECEF/Velocity model?

show derivation of discretized constant velocity Q with  
parameter power spectral density



To derive the discretized  $Q$  matrix, we start with the continuous-time physics of the system and "integrate" the noise over a single time step  $\Delta t$ .

#### 1. The Continuous Model

We assume the target moves at a constant velocity, but is subject to a random, zero-mean "acceleration noise"  $w(t)$ . For a single dimension (e.g., the X-axis), the state equations are:

$$\dot{x}(t) = v(t)$$

$$\dot{v}(t) = w(t)$$

In matrix form ( $\dot{x} = Ax + Gw$ ):

$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t)$$

The noise  $w(t)$  has a **Power Spectral Density (PSD)**, denoted as  $S_q$ . This represents the intensity of the random white noise.

## 2. The Stochastic Integral

The discrete-time process noise covariance  $Q_k$  is the result of the continuous noise being propagated through the system dynamics over the interval  $[0, \Delta t]$ :

$$Q_k = \int_0^{\Delta t} \Phi(\tau) G S_q G^T \Phi(\tau)^T d\tau$$

Where:

- $G = [0; 1]$ : The noise effect matrix (noise enters through acceleration).
- $\Phi(\tau) = e^{A\tau}$ : The State Transition Matrix. For a constant velocity model, this is:

$$\Phi(\tau) = \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix}$$

## 3. The Integration Step

Now, we substitute these into the integral:

$$Q_k = \int_0^{\Delta t} \left( \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} S_q [0 \ 1] \begin{bmatrix} 1 & 0 \\ \tau & 1 \end{bmatrix} \right) d\tau$$

Multiply the inner terms first:

$$\begin{aligned} 1. \quad GS_q G^T &= \begin{bmatrix} 0 & 0 \\ 0 & S_q \end{bmatrix} \\ 2. \quad \Phi(\tau)(GS_q G^T)\Phi(\tau)^T &= \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & S_q \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tau & 1 \end{bmatrix} = \begin{bmatrix} S_q \tau^2 & S_q \tau \\ S_q \tau & S_q \end{bmatrix} \end{aligned}$$

Now, integrate each element with respect to  $\tau$  from 0 to  $\Delta t$ :

- **Top-Left:**  $\int_0^{\Delta t} S_q \tau^2 d\tau = S_q \frac{\Delta t^3}{3}$
- **Off-Diagonal:**  $\int_0^{\Delta t} S_q \tau d\tau = S_q \frac{\Delta t^2}{2}$
- **Bottom-Right:**  $\int_0^{\Delta t} S_q d\tau = S_q \Delta t$

## 4. The Final Discretized $Q$

For a single dimension, the matrix is:

$$Q_i = S_q \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}$$

In your **ECEF** model, since you have three independent axes ( $x, y, z$ ), your full  $6 \times 6$  matrix is the block diagonal:

$$Q = \begin{bmatrix} Q_x & 0 & 0 \\ 0 & Q_y & 0 \\ 0 & 0 & Q_z \end{bmatrix}$$

### Why this matters for your "Missing Data" problem

The off-diagonal term ( $S_q \frac{\Delta t^2}{2}$ ) is the **coupling** between position and velocity. When you lose range-rate measurements, the filter uses this specific term to decide how much a "surprise" in position should change the estimate of the velocity.

**Would you like me to show you how to calculate the State Transition Matrix ( $F$ ) that pairs with this  $Q$  matrix for your prediction step?**

Sources

↶ ↷ ⌂ ☰ :

Permutations With and Without Repe...

Solving Task Assignment with Bipartit...

⚙️ Settings & help

Ask Gemini 3

+

✖ Tools

Fast ▾



Gemini can make mistakes, so double-check it