`motion-analysis`: **Progress update**

Andrew Lock *The University of Southern Queensland*

November 16, 2022

Overarching objective

Motion is governed by the forces acting on a body:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{r}\\ \mathbf{v}\\ \mathbf{a}\\ \mathbf{q}\\ \boldsymbol{\omega}\end{bmatrix} = \begin{bmatrix}\mathbf{v}\\ \mathbf{a}\\ \mathbf{F}/\mathbf{m}\\ [\Omega]\mathbf{q}\\ [I]^{-1}\left\{-[\omega]\times([I]\boldsymbol{\omega})+\mathbf{M}\right\}\end{bmatrix} \tag{1}$$

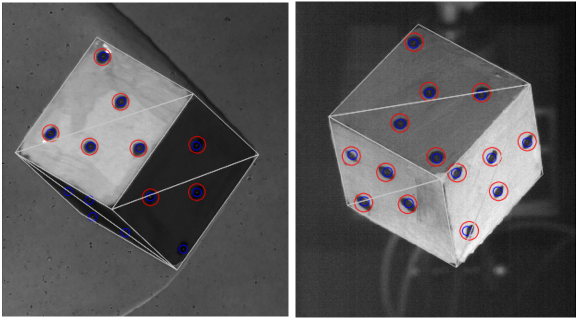
where

$$[\Omega] = \begin{bmatrix}0 & -p & -q & -r\\ p & 0 & r & -q\\ q & -r & 0 & p\\ r & q & -p & 0\end{bmatrix} \qquad [\omega] = \begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix} \tag{2}$$

By optically tracking position $(\mathbf{r})$ *and attitude* $(\mathbf{q})$, *can we derive forces* $(\mathbf{F})$ *and moments* $(\mathbf{M})$ *accurately?*

Approach

- We create a digital replica of the test model, including 'trackable features' (currently blobs)
- We define camera view functions $v_i(\mathbf{r}, \mathbf{q})$ that show the camera model view for any position/attitude
- By comparing the camera image to the camera projection at each frame, we can track the body motion
    - An interesting question: what is the best way to do this?



(a) East view (Schlieren camera) (b) Top view

Figure 1: High-speed images overlaid with digital model projection

Dealing with measurement error

- Kalman filters use Bayesian probably to find the most *likely* state at each point in time k given a measurement $\mathbf{z}_k$, and a process model $\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$.

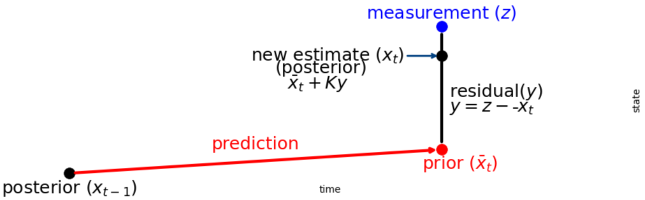


Figure 2: Kalman filter diagram

- Even better, Kalman filter measurements do not have to be system states directly - they can be any system observable $\mathbf{z} = \mathbf{h}(\mathbf{x})$
  - E.g. 2D blob pixel locations on a camera image
- For nonlinear systems we use either Extended Kalman filter (first-order linearisation), or Unscented Kalman filter (linearises the Gaussian tranform).
  - EKF better for mildly nonlinear systems and small timesteps.
- For a great resource on Kalman and Bayesian filters, see the online book Kalman and Bayesian Filters in Python [1]

Kalman filter dynamic system

We don't know the dynamic system (aerodynamics) - so we account for it with process noise matrix $[\mathrm{Q}]$

- constant-velocity model

$$\frac{d}{dt}\begin{bmatrix}\mathbf{r}\\ \mathbf{v}\\ \mathbf{a}\\ \mathbf{q}\\ \boldsymbol{\omega}\end{bmatrix} = \begin{bmatrix}\mathbf{v}\\ \mathbf{a}\\ \mathbf{0}\\ [\Omega]\mathbf{q}\\ [I]^{-1}\left\{-[\omega]\times([I]\boldsymbol{\omega})\right\}\end{bmatrix} \tag{3}$$

- Continuous-time process noise

$$[\mathrm{Q_c}] = \begin{bmatrix}[\mathbf{0}]_3 & [\mathbf{0}]_4 & [\mathbf{0}]_3 & [\mathbf{0}]_3\\ [\mathbf{0}]_3 & [\mathbf{0}]_4 & [\mathbf{0}]_3 & [\mathbf{0}]_3\\ [\mathbf{0}]_3 & [\mathbf{0}]_4 & c_1[\boldsymbol{I}]_3 & [\mathbf{0}]_3\\ [\mathbf{0}]_3 & [\mathbf{0}]_4 & [\mathbf{0}]_3 & [\mathbf{0}]_3\\ [\mathbf{0}]_3 & [\mathbf{0}]_4 & [\mathbf{0}]_3 & c_2[\boldsymbol{I}]_3\end{bmatrix} \tag{4}$$

An alternative is constant-velocity model, but this has steady tracking offset for body under constant acceleration

Kalman filter implementation

Current approach

- Use Runge-Kutta to provide discrete-time process function $\mathbf{f}_d(\mathbf{x}) = \int_0^{\Delta t} \mathbf{f}(\mathbf{x})\ dt$
- Linearise dynamic system at each timestep from Jacobian $[\mathrm{F}]_k = \left.\frac{\partial \mathbf{f}_d}{\partial \mathbf{x}}\right|_{\mathbf{x}_{k|k-1}}$
- Observation function constructed from camera frame x and y blob coordinates

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} v_{\mathrm{top},x}(\mathbf{x},\mathbf{q}) \\ v_{\mathrm{top},y}(\mathbf{x},\mathbf{q}) \\ v_{\mathrm{east},x}(\mathbf{x},\mathbf{q}) \\ v_{\mathrm{east},y}(\mathbf{x},\mathbf{q}) \end{bmatrix} \tag{5}$$

  and linearised observation function $[\mathrm{H}]_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{k|k-1}}$

- Discrete-time process noise

$$[Q]_k = \int_0^{\Delta t} [\mathrm{F}]_k [\mathrm{Q_C}] [\mathrm{F}]_k^{\mathrm{T}}\ dt \tag{6}$$

- Measurement uncertainty matrix $[\mathrm{R}] = u_m[\mathrm{I}]$ represents pixel-accuracy of blob detection

Extended Kalman filter algorithm

The EKF then follows the standard predict step using the Runge-Kutta integrator,

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}_d(\mathbf{x}_{k-1|k-1}) \tag{7}$$

$$[\mathrm{P}]_{k|k-1} = [\mathrm{F}]_k[\mathrm{P}]_{k-1|k-1}[\mathrm{F}]_k^{\mathrm{T}} + [\mathrm{Q}]_k \tag{8}$$

which is proceeded by the update step

$$[\mathrm{K}]_k = [\mathrm{P}]_{k|k-1}[\mathrm{H}]_k^{\mathrm{T}}\Big([\mathrm{H}]_k[\mathrm{P}]_{k|k-1}[\mathrm{H}]_k + [\mathrm{R}]\Big)^{-1} \tag{9}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + [\mathrm{K}]_k\Big(z_k - h(\hat{\mathbf{x}}_{k|k-1}\Big) \tag{10}$$

$$[\mathrm{P}]_{k|k} = \Big([\mathrm{I}] - [\mathrm{K}]_k[\mathrm{H}]_k\Big)[\mathrm{P}]_{k|k-1} \tag{11}$$

After tracking through the desired set of frames, Rauch–Tung–Striebel smoothing [2] is used on the data to find the highest likelihood state at each timestep.

How does it all fit together?

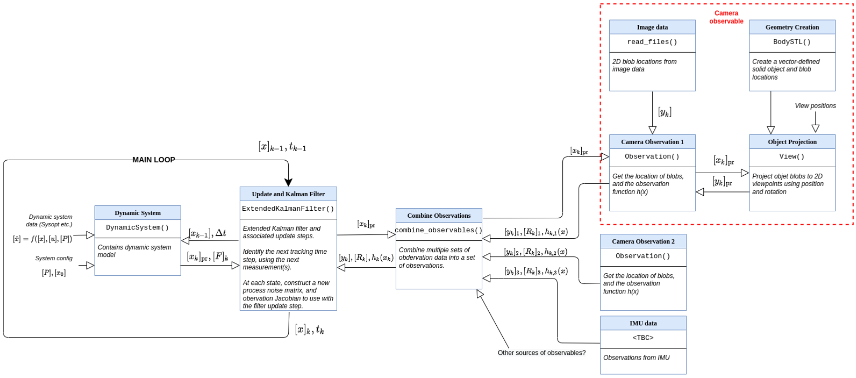


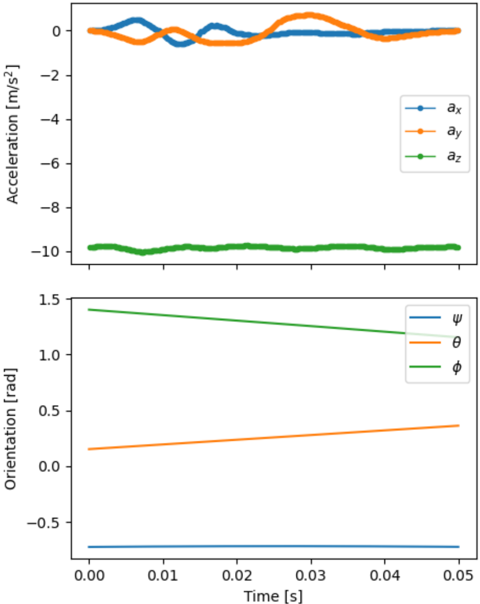
Figure 3: `motion-analysis` framework

Demonstration

Demonstration of tracking

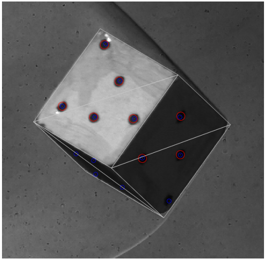
Preliminary results; Free-fall

- Can we measure gravity in a vacuum free-fall?
- Yes (...roughly)
- The cube has some rotation - not as simple as tracking individual blob acceleration
- Acceleration noise around 0 is predictable - known feature of constant-acceleration model with very low constant velocity (small perturbations show as large acceleration).



Preliminary results: no-spin aerodynamics

- What about aerodynamics in flow (with slow rotation)
- Results look sensible (yet to be verified)
- Next step: compare to CFD (Hello, Flynn)
- Working on tracking spinning cube*



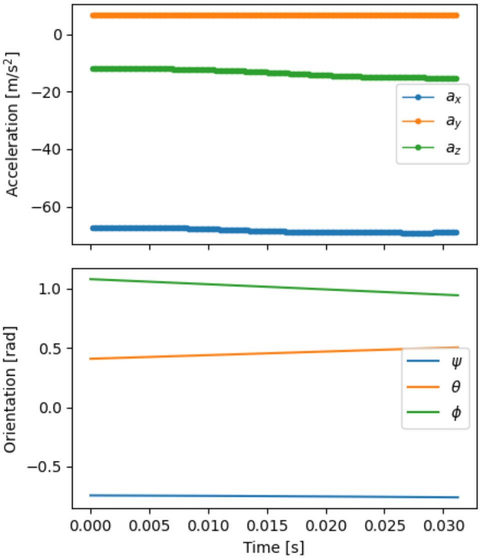


Figure 5: Measured acceleration in Mach 6 flow

Challenges

- We need an accurate camera transfer function $v(\mathbf{x}, \mathbf{q})$ which transforms body features in local coordinates to 2D camera pixel coordinates:
  - Scale, offset, direction
  - Perspective
  - Lens distortion
  - Schlieren misalignment
- Creating digital model of features (blobs) could be challenging for more complex shapes
  - Code currently handles STL file input, but no blob location information
  - Streamlined process for detecting blob locations on model?
  - Detecting other features (edges, vertices etc.)?
- Need accurate MOI tensor to derive moment forces
  - For vehicles, could be complex. May need to measure physically?
- Kalman filters won't handle large temporal changes in process model well (like transition from vacuum to flow). Need to isolate region of interest.

Status and future work

We currently have a proof-of-concept. The two future areas to develop:

Increasing accuracy:

- Accurate camera calibration
- Accurate MOI measurement

Increased usability:

- Streamlined camera calibration process
- Automated model blob location detection
- Robust image feature detection settings (`OpenCV`)

Note: All codes are stored in the USQ repositories:
https://github.com/tusq-at-usq/motion-analysis
https://github.com/tusq-at-usq/tracking-projects

References

[1] Roger Labbe. Kalman-and-bayesian-filters-in-python. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python, 2022.

[2] H E Raunch, F Tung, and C T Striebel. Maximum likelihood estimates of linear dynamic systems. AIAA Journal, 3(8):1445–1450, aug 1965.