Pages

Generated by doxygen 1.8.19 and m.css.

IMU Propagation Derivations

Contents

IMU Measurements

导航系G 为东北天，g G = (0 ;0 ; 9 :81) ;
前面的G系即此处的w系，即n系;
在一个小范围内，认为重力
方向不变，可以计算p;
见course2-hw的imu.cpp；见《手写vio:2》

We use a 6-axis inertial measurement unit (IMU) to propagate the inertial navigation system (INS), which provides measurements of the local rotational velocity (angular rate) $\boldsymbol{\omega}_m$ and local translational acceleration $\mathbf{a}_m$:

$$\overset{\text{measure}}{\boldsymbol{\omega}_m(t)} = \overset{\text{true}}{\boldsymbol{\omega}}(t) + \mathbf{b}_g(t) + \mathbf{n}_g(t)$$

$$\mathbf{a}_m(t) = \mathbf{a}(t) + {}^I_G\mathbf{R}(t)\,{}^G\mathbf{g} + \mathbf{b}_a(t) + \mathbf{n}_a(t)$$

G:n系?? b系

where $\boldsymbol{\omega}$ and $\mathbf{a}$ are the true rotational velocity and translational acceleration in the IMU local frame $\{I\}$, $\mathbf{b}_g$ and $\mathbf{b}_a$ are the gyroscope and accelerometer biases, and $\mathbf{n}_g$ $\mathbf{n}_a$ are white Gaussian noise, ${}^G\mathbf{g} = [0\ \ 0\ \ 9.81]^\top$ n系中?? is the gravity expressed in the n系? global frame $\{G\}$ (noting that the gravity is slightly different on different locations of the globe), and ${}^I_G\mathbf{R}$ is the rotation matrix from global to IMU local frame.

State Vector

We define our INS state vector $\mathbf{x}_I$ at time t as:

$$\mathbf{x}_I(t) = \begin{bmatrix} {}^I_G\bar{q}(t) \\ {}^G\mathbf{p}_I(t) \\ {}^G\mathbf{v}_I(t) \\ \mathbf{b}_\mathbf{g}(t) \\ \mathbf{b}_\mathbf{a}(t) \end{bmatrix}$$

where ${}^I_G\bar{q}$ is the unit quaternion representing the rotation global to IMU frame, ${}^G\mathbf{p}_I$ is the position of IMU in global frame, and ${}^G\mathbf{v}_I$ is the velocity of IMU in global frame. We will often write time as a subscript of I describing the state of IMU at the time for notation clarity (e.g., ${}^{I_t}_G\bar{q} = {}^I_G\bar{q}(t)$). In order to define the IMU error state, the standard additive error definition is employed for the position, velocity, and biases, while we use the quaternion error state $\delta\bar{q}$ with a left quaternion multiplicative error $\otimes$:

$${}^I_G\bar{q} = \delta\bar{q} \otimes {}^I_G\hat{\bar{q}}$$

$$\delta\bar{q} = \begin{bmatrix} \hat{\mathbf{k}}\sin(\frac{1}{2}\tilde{\theta}) \\ \cos(\frac{1}{2}\tilde{\theta}) \end{bmatrix} \simeq \begin{bmatrix} \frac{1}{2}\tilde{\boldsymbol{\theta}} \\ 1 \end{bmatrix}$$

where $\hat{\mathbf{k}}$ is the rotation axis and $\tilde{\theta}$ is the rotation angle. For small rotation, the error angle vector is approximated by $\tilde{\boldsymbol{\theta}} = \tilde{\theta}\,\hat{\mathbf{k}}$ as the error vector about the three orientation axes. The total IMU error state thus is defined as the following 15x1 (not 16x1) vector:

$$\tilde{\mathbf{x}}_I(t) = \begin{bmatrix} {}^I_G\tilde{\boldsymbol{\theta}}(t) \\ {}^G\tilde{\mathbf{p}}_I(t) \\ {}^G\tilde{\mathbf{v}}_I(t) \\ \tilde{\mathbf{b}}_g(t) \\ \tilde{\mathbf{b}}_a(t) \end{bmatrix}$$

IMU Kinematics

The IMU state evolves over time as follows (see Indirect Kalman Filter for 3D Attitude Estimation [18]).

$$\begin{aligned} {}^I_G\dot{\bar{q}}(t) &= \frac{1}{2}\begin{bmatrix} -\lfloor \boldsymbol{\omega}(t)\times\rfloor & \boldsymbol{\omega}(t) \\ -\underbrace{\boldsymbol{\omega}}_{\text{true val}}^\top(t) & 0 \end{bmatrix} {}^{I_t}_G\bar{q} \\ &=: \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}(t)){}^{I_t}_G\bar{q} \\ {}^G\dot{\mathbf{p}}_I(t) &= {}^G\mathbf{v}_I(t) \\ {}^G\dot{\mathbf{v}}_I(t) &= {}^{I_t}_G\mathbf{R}^\top \underbrace{\mathbf{a}}_{\text{true val}}(t) \\ \dot{\mathbf{b}}_{\mathbf{g}}(t) &= \mathbf{n}_{wg} \\ \dot{\mathbf{b}}_{\mathbf{a}}(t) &= \mathbf{n}_{wa} \end{aligned}$$

where we have modeled the gyroscope and accelerometer biases as random walk and thus their time derivatives are white Gaussian. Note that the above kinematics have been defined in terms of the *true* acceleration and angular velocities.

Continuous-time IMU Propagation

Given the continuous-time measurements $\boldsymbol{\omega}_m(t)$ and $\mathbf{a}_m(t)$ in the time interval $t \in [t_k, t_{k+1}]$, and their estimates, i.e. after taking the expectation, $\hat{\boldsymbol{\omega}}(t) = \boldsymbol{\omega}_m(t) - \hat{\boldsymbol{b}}_g(t)$ and $\hat{\boldsymbol{a}}(t) = \boldsymbol{a}_m(t) - \hat{\boldsymbol{b}}_a(t) - {}^I_G\hat{\mathbf{R}}(t){}^G\mathbf{g}$, we can define the solutions to the above IMU kinematics differential equation. The solution to the quaternion evolution has the following general form:

$${}^{I_t}_G\bar{q} = \boldsymbol{\Theta}(t, t_k){}^{I_k}_G\bar{q}$$

Differentiating and reordering the terms yields the governing equation for $\boldsymbol{\Theta}(t, t_k)$ as

$$\begin{aligned} \boldsymbol{\Theta}(t, t_k) &= {}^{I_t}_G\bar{q}\;{}^{I_k}_G\bar{q}^{-1} \\ \Rightarrow \dot{\boldsymbol{\Theta}}(\underbrace{t}_{\substack{\text{t:自变量;}\\ \text{tk:固定}}}, t_k) &= {}^{I_t}_G\dot{\bar{q}}\;\underbrace{{}^{I_k}_G\bar{q}}_{\text{固定,不求导}}{}^{-1} \\ &= \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}(t))\;{}^{I_t}_G\bar{q}\;{}^{I_k}_G\bar{q}^{-1} \\ &= \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}(t))\boldsymbol{\Theta}(t, t_k) \end{aligned}$$

with $\boldsymbol{\Theta}(t_k, t_k) = \mathbf{I}_4$. If we take $\boldsymbol{\omega}(t) = \boldsymbol{\omega}$ to be constant over the the period $\Delta t = t_{k+1} - t_k$, then the above system is linear time-invarying (LTI), and $\boldsymbol{\Theta}$ can be solved as (see [Stochastic Models, Estimation, and Control] [11]):

$$
\begin{aligned}
\boldsymbol{\Theta}(t_{k+1}, t_k) &= \exp\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\Delta t\right) \\
&= \cos\left(\frac{|\boldsymbol{\omega}|}{2}\Delta t\right)\cdot \mathbf{I}_4 + \frac{1}{|\boldsymbol{\omega}|}\sin\left(\frac{|\boldsymbol{\omega}|}{2}\Delta t\right)\cdot \boldsymbol{\Omega}(\boldsymbol{\omega}) \\
&\simeq \mathbf{I}_4 + \frac{\Delta t}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})
\end{aligned}
$$

where the approximation assumes small $|\boldsymbol{\omega}|$. We can formulate the quaternion propagation from t_k to t_{k+1} using the estimated rotational velocity $\hat{\boldsymbol{\omega}}(t) = \hat{\boldsymbol{\omega}}$ as:

$$
{}^{I_{k+1}}_{G}\hat{\bar{q}} = \exp\left(\frac{1}{2}\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}})\Delta t\right){}^{I_k}_{G}\hat{\bar{q}}
$$

Having defined the integration of the orientation, we can integrate the velocity and position over the measurement interval:

$$
\begin{aligned}
{}^{G}\hat{\mathbf{v}}_{k+1} &= {}^{G}\hat{\mathbf{v}}_{I_k} + \int_{t_k}^{t_{k+1}} {}^{G}\hat{\mathbf{a}}(\tau)d\tau \\
&= {}^{G}\hat{\mathbf{v}}_{I_k} - {}^{G}\mathbf{g}\Delta t + \int_{t_k}^{t_{k+1}} {}^{G}_{I_\tau}\hat{\mathbf{R}}(\mathbf{a}_m(\tau) - \hat{\mathbf{b}}_{\mathbf{a}}(\tau))d\tau
\end{aligned}
$$

$$
\begin{aligned}
{}^{G}\hat{\mathbf{p}}_{I_{k+1}} &= {}^{G}\hat{\mathbf{p}}_{I_k} + \int_{t_k}^{t_{k+1}} {}^{G}\hat{\mathbf{v}}_{I}(\tau)d\tau \\
&= {}^{G}\hat{\mathbf{p}}_{I_k} + {}^{G}\hat{\mathbf{v}}_{I_k}\Delta t - \frac{1}{2}{}^{G}\mathbf{g}\Delta t^2 + \int_{t_k}^{t_{k+1}}\int_{t_k}^{s} {}^{G}_{I_\tau}\hat{\mathbf{R}}(\mathbf{a}_m(\tau) - \hat{\mathbf{b}}_{\mathbf{a}}(\tau))d\tau ds
\end{aligned}
$$

Propagation of each bias $\hat{\mathbf{b}}_{\mathbf{g}}$ and $\hat{\mathbf{b}}_{\mathbf{a}}$ is given by:

$$
\begin{aligned}
\hat{\mathbf{b}}_{\mathbf{g},k+1} &= \hat{\mathbf{b}}_{\mathbf{g},k} + \int_{t_{k+1}}^{t_k} \hat{\mathbf{n}}_{wg}(\tau)d\tau \\
&= \hat{\mathbf{b}}_{\mathbf{g},k} \\
\hat{\mathbf{b}}_{\mathbf{a},k+1} &= \hat{\mathbf{b}}_{\mathbf{a},k} + \int_{t_{k+1}}^{t_k} \hat{\mathbf{n}}_{wa}(\tau)d\tau \\
&= \hat{\mathbf{b}}_{\mathbf{a},k}
\end{aligned}
$$

The biases will not evolve since our random walk noises $\hat{\mathbf{n}}_{wg}$ and $\hat{\mathbf{n}}_{wa}$ are zero-mean white Gaussian. All of the above integrals could be analytically or numerically solved if one wishes to use the continuous-time measurement evolution model.

Discrete-time IMU Propagation

A simpler method is to model the measurements as discrete-time over the integration period. To do this, the measurements can be assumed to be constant during the sampling period. We employ this assumption and approximate that the measurement at time t_k remains the same until we get the next measurement at t_{k+1}. For the quaternion propagation, it is the same as continuous-time propagation with constant measurement assumption $\boldsymbol{\omega}_m(t_k) = \boldsymbol{\omega}_{m,k}$. We use subscript k to denote it is the measurement we get at time t_k. Therefore the propagation of quaternion can be written as:

$$ {}^{I_{k+1}}_{G}\hat{\bar{q}} = \exp\left(\frac{1}{2}\boldsymbol{\Omega}\big(\underset{\text{measure NOT true}}{\underline{\boldsymbol{\omega}_{m,k}}} - \hat{\mathbf{b}}_{g,k}\big)\Delta t\right) {}^{I_k}_{G}\hat{\bar{q}} $$

For the velocity and position propagation we have constant $\mathbf{a}_m(t_k) = \mathbf{a}_{m,k}$ over $t \in [t_k, t_{k+1}]$. We can therefore directly solve for the new states as:

$$ \begin{aligned} {}^G\hat{\mathbf{v}}_{k+1} &= {}^G\hat{\mathbf{v}}_{I_k} - {}^G\mathbf{g}\Delta t + {}^{I_k}_G\hat{\mathbf{R}}^\top(\mathbf{a}_{m,k} - \hat{\mathbf{b}}_{\mathbf{a},k})\Delta t \\ {}^G\hat{\mathbf{p}}_{I_{k+1}} &= {}^G\hat{\mathbf{p}}_{I_k} + {}^G\hat{\mathbf{v}}_{I_k}\Delta t - \frac{1}{2}{}^G\mathbf{g}\Delta t^2 + \frac{1}{2}{}^{I_k}_G\hat{\mathbf{R}}^\top(\mathbf{a}_{m,k} - \hat{\mathbf{b}}_{\mathbf{a},k})\Delta t^2 \end{aligned} $$

The propagation of each bias is likewise the continuous system:

$$ \begin{aligned} \hat{\mathbf{b}}_{\mathbf{g},k+1} &= \hat{\mathbf{b}}_{\mathbf{g},k} \\ \hat{\mathbf{b}}_{\mathbf{a},k+1} &= \hat{\mathbf{b}}_{\mathbf{a},k} \end{aligned} $$

Discrete-time Error-state Propagation

In order to propagate the covariance matrix, we should derive the error-state propagation, i.e., computing the system Jacobian $\boldsymbol{\Phi}(t_{k+1}, t_k)$ and noise Jacobian $\mathbf{G}_k$. In particular, when the covariance matrix of the continuous-time measurement noises is given by $\mathbf{Q}_c$, then the discrete-time noise covariance $\mathbf{Q}_d$ can be computed as (see [Indirect Kalman Filter for 3D Attitude Estimation] [18]):

$$ \mathbf{Q}_d = \frac{1}{\Delta t}\mathbf{Q}_c $$

The method of computing Jacobians is to "perturb" each variable in the system and see how the old error "perturbation" relates to the new error state. That is, $\boldsymbol{\Phi}(t_{k+1}, t_k)$ and $\mathbf{G}_k$ can be found by perturbing each variable as:

$$ \tilde{\mathbf{x}}_I(t_{k+1}) = \boldsymbol{\Phi}(t_{k+1}, t_k)\tilde{\mathbf{x}}_I(t_k) + \mathbf{G}_k\mathbf{n} $$

where $\mathbf{n} = [\mathbf{n}_g\ \mathbf{n}_a\ \mathbf{n}_{bg}\ \mathbf{n}_{ba}]^\top$ are the IMU sensor noises.

For the orientation error propagation, we start with the $\mathbf{SO}(3)$ perturbation using ${}^I_G\mathbf{R} \approx (\mathbf{I}_3 - \lfloor {}^I_G\tilde{\boldsymbol{\theta}}\times\rfloor){}^I_G\hat{\mathbf{R}}$:

$$ \begin{aligned} {}^{I_{k+1}}_G\mathbf{R} &= {}^{I_{k+1}}_{I_k}\mathbf{R}\,{}^{I_k}_G\mathbf{R} \\ (\mathbf{I}_3 - \lfloor {}^{I_{k+1}}_G\tilde{\boldsymbol{\theta}}\times\rfloor){}^{I_{k+1}}_G\hat{\mathbf{R}} &\approx \exp(-{}^{I_k}\hat{\boldsymbol{\omega}}\Delta t - {}^{I_k}\tilde{\boldsymbol{\omega}}\Delta t)(\mathbf{I}_3 - \lfloor {}^{I_k}_G\tilde{\boldsymbol{\theta}}\times\rfloor){}^{I_k}_G\hat{\mathbf{R}} \\ &= \exp(-{}^{I_k}\hat{\boldsymbol{\omega}}\Delta t)\exp(-\mathbf{J}_r(-{}^{I_k}\hat{\boldsymbol{\omega}}\Delta t){}^{I_k}\tilde{\boldsymbol{\omega}}\Delta t)(\mathbf{I}_3 - \lfloor {}^{I_k}_G\tilde{\boldsymbol{\theta}}\times\rfloor){}^{I_k}_G\hat{\mathbf{R}} \\ &= {}^{I_{k+1}}_{I_k}\hat{\mathbf{R}}(\mathbf{I}_3 - \lfloor \mathbf{J}_r(-{}^{I_k}\hat{\boldsymbol{\omega}}\Delta t)\tilde{\boldsymbol{\omega}}_k\Delta t\times\rfloor)(\mathbf{I}_3 - \lfloor {}^{I_k}_G\tilde{\boldsymbol{\theta}}\times\rfloor){}^{I_k}_G\hat{\mathbf{R}} \end{aligned} $$

where $\overset{\text{扰动量}}{\tilde{\boldsymbol{\omega}}} = \overset{\text{true}}{\boldsymbol{\omega}} - \overset{\text{估计值}}{\hat{\boldsymbol{\omega}}} = -(\tilde{\mathbf{b}}_{\mathbf{g}} + \mathbf{n}_g)$ handles both the perturbation to the bias and measurement noise. $\mathbf{J}_r(\boldsymbol{\theta})$ is the right Jacobian of $\mathbf{SO}(3)$ that maps the variation of rotation angle in the parameter vector space into the variation in the tangent vector space to the manifold [see ov_core::Jr_so3()]. By neglecting the second order terms from above, we obtain the following orientation error propagation:

$$ {}^{I_{k+1}}_G\tilde{\boldsymbol{\theta}} \approx {}^{I_{k+1}}_{I_k}\hat{\mathbf{R}}\,{}^{I_k}_G\tilde{\boldsymbol{\theta}} - {}^{I_{k+1}}_{I_k}\hat{\mathbf{R}}\mathbf{J}_r({}^{I_{k+1}}_{I_k}\hat{\boldsymbol{\theta}})\Delta t(\tilde{\mathbf{b}}_{\mathbf{g},k} + \mathbf{n}_{\mathbf{g},k}) $$

Now we can do error propagation of position and velocity using the same scheme:

$$
\begin{aligned}
{}^{G}\mathbf{p}_{I_{k+1}} &= {}^{G}\mathbf{p}_{I_k} + {}^{G}\mathbf{v}_{I_k}\Delta t - \frac{1}{2}{}^{G}\mathbf{g}\Delta t^2 + \frac{1}{2}{}^{I_k}_{G}\mathbf{R}^\top \mathbf{a}_k \Delta t^2 \\
{}^{G}\hat{\mathbf{p}}_{I_{k+1}} + {}^{G}\tilde{\mathbf{p}}_{I_{k+1}} &\approx {}^{G}\hat{\mathbf{p}}_{I_k} + {}^{G}\tilde{\mathbf{p}}_{I_k} + {}^{G}\hat{\mathbf{v}}_{I_k}\Delta t + {}^{G}\tilde{\mathbf{v}}_{I_k}\Delta t - \frac{1}{2}{}^{G}\mathbf{g}\Delta t^2 \\
&\quad + \frac{1}{2}{}^{I_k}_{G}\hat{\mathbf{R}}^\top (\mathbf{I}_3 + \lfloor {}^{I_k}_{G}\tilde{\boldsymbol{\theta}}\times\rfloor)(\hat{\mathbf{a}}_k + \tilde{\mathbf{a}}_k)\Delta t^2
\end{aligned}
$$

$$
\begin{aligned}
{}^{G}\mathbf{v}_{k+1} &= {}^{G}\mathbf{v}_{I_k} - {}^{G}\mathbf{g}\Delta t + {}^{I_k}_{G}\mathbf{R}^\top \mathbf{a}_k\Delta t \\
{}^{G}\hat{\mathbf{v}}_{k+1} +^{G} \tilde{\mathbf{v}}_{k+1} &\approx {}^{G}\hat{\mathbf{v}}_{I_k} + {}^{G}\tilde{\mathbf{v}}_{I_k} - {}^{G}\mathbf{g}\Delta t + {}^{I_k}_{G}\hat{\mathbf{R}}^\top (\mathbf{I}_3 + \lfloor {}^{I_k}_{G}\tilde{\boldsymbol{\theta}}\times\rfloor)(\hat{\mathbf{a}}_k + \tilde{\mathbf{a}}_k)\Delta t
\end{aligned}
$$

where $\tilde{\mathbf{a}} = \mathbf{a} - \hat{\mathbf{a}} = -(\tilde{\mathbf{b}}_{\mathbf{a}} + \mathbf{n}_{\mathbf{a}})$. By neglecting the second order error terms, we obtain the following position and velocity error propagation:

$$
\begin{aligned}
{}^{G}\tilde{\mathbf{p}}_{I_{k+1}} &= {}^{G}\tilde{\mathbf{p}}_{I_k} + \Delta t^{G}\tilde{\mathbf{v}}_{I_k} - \frac{1}{2}{}^{I_k}_{G}\hat{\mathbf{R}}^\top \lfloor \hat{\mathbf{a}}_k\Delta t^2\times\rfloor {}^{I_k}_{G}\tilde{\boldsymbol{\theta}} - \frac{1}{2}{}^{I_k}_{G}\hat{\mathbf{R}}^\top \Delta t^2(\tilde{\mathbf{b}}_{\mathbf{a},k} + \mathbf{n}_{\mathbf{a},k}) \\
{}^{G}\tilde{\mathbf{v}}_{k+1} &= {}^{G}\tilde{\mathbf{v}}_{I_k} - {}^{I_k}_{G}\hat{\mathbf{R}}^\top \lfloor \hat{\mathbf{a}}_k\Delta t\times\rfloor {}^{I_k}_{G}\tilde{\boldsymbol{\theta}} - {}^{I_k}_{G}\hat{\mathbf{R}}^\top \Delta t(\tilde{\mathbf{b}}_{\mathbf{a},k} + \mathbf{n}_{\mathbf{a},k})
\end{aligned}
$$

The propagation of two biases can be derived in the same way:

$$
\begin{aligned}
\mathbf{b}_{\mathbf{g},k+1} &= \mathbf{b}_{\mathbf{g},k} + \int_{t_k}^{t_{k+1}} \mathbf{n}_{wg}(\tau)d\tau \\
\hat{\mathbf{b}}_{\mathbf{g},k+1} + \tilde{\mathbf{b}}_{\mathbf{g},k+1} &= \hat{\mathbf{b}}_{\mathbf{g},k} + \tilde{\mathbf{b}}_{\mathbf{g},k} + \int_{t_k}^{t_{k+1}} \hat{\mathbf{n}}_{wg}(\tau)d\tau \\
\tilde{\mathbf{b}}_{\mathbf{g},k+1} &= \tilde{\mathbf{b}}_{\mathbf{g},k} + \mathbf{n}_{wg}\Delta t
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{b}_{\mathbf{a},k+1} &= \mathbf{b}_{\mathbf{a},k} + \int_{t_k}^{t_{k+1}} \mathbf{n}_{wa}(\tau)d\tau \\
\hat{\mathbf{b}}_{\mathbf{a},k+1} + \tilde{\mathbf{b}}_{\mathbf{a},k+1} &= \hat{\mathbf{b}}_{\mathbf{a},k} + \tilde{\mathbf{b}}_{\mathbf{a},k} + \int_{t_k}^{t_{k+1}} \hat{\mathbf{n}}_{wa}(\tau)d\tau \\
\tilde{\mathbf{b}}_{\mathbf{a},k+1} &= \tilde{\mathbf{b}}_{\mathbf{a},k} + \mathbf{n}_{wa}\Delta t
\end{aligned}
$$

By collecting all the perturbation results, we can build $\boldsymbol{\Phi}(t_{k+1}, t_k)$ and $\mathbf{G}_k$ matrices as:

$$
\boldsymbol{\Phi}(t_{k+1}, t_k) = \begin{bmatrix}
{}^{I_{k+1}}_{I_k}\hat{\mathbf{R}} & \mathbf{0}_3 & \mathbf{0}_3 & -{}^{I_{k+1}}_{I_k}\hat{\mathbf{R}}\mathbf{J}_r({}^{I_{k+1}}_{I_k}\hat{\boldsymbol{\theta}})\Delta t & \mathbf{0}_3 \\
-\frac{1}{2}{}^{I_k}_{G}\hat{\mathbf{R}}^\top \lfloor \hat{\mathbf{a}}_k\Delta t^2\times\rfloor & \mathbf{I}_3 & \Delta t\mathbf{I}_3 & \mathbf{0}_3 & -\frac{1}{2}{}^{I_k}_{G}\hat{\mathbf{R}}^\top\Delta t^2 \\
-{}^{I_k}_{G}\hat{\mathbf{R}}^\top \lfloor \hat{\mathbf{a}}_k\Delta t\times\rfloor & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & -{}^{I_k}_{G}\hat{\mathbf{R}}^\top\Delta t \\
\mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \\
\mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3
\end{bmatrix}
$$

$$
\mathbf{G}_k = \begin{bmatrix}
-{}^{I_{k+1}}_{I_k}\hat{\mathbf{R}}\mathbf{J}_r({}^{I_{k+1}}_{I_k}\hat{\boldsymbol{\theta}})\Delta t & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\
\mathbf{0}_3 & -\frac{1}{2}{}^{I_k}_{G}\hat{\mathbf{R}}^\top\Delta t^2 & \mathbf{0}_3 & \mathbf{0}_3 \\
\mathbf{0}_3 & -{}^{I_k}_{G}\hat{\mathbf{R}}^\top\Delta t & \mathbf{0}_3 & \mathbf{0}_3 \\
\mathbf{0}_3 & \mathbf{0}_3 & \Delta t\mathbf{I}_3 & \mathbf{0}_3 \\
\mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \Delta t\mathbf{I}_3
\end{bmatrix}
$$

Now, with the computed $\boldsymbol{\Phi}(t_{k+1}, t_k)$ and $\mathbf{G}_k$ matrices, we can propagate the covariance from t_k to t_{k+1}:

$$\mathbf{P}_{k+1|k} = \mathbf{\Phi}(t_{k+1}, t_k)\mathbf{P}_{k|k}\mathbf{\Phi}(t_{k+1}, t_k)^\top + \mathbf{G}_k\mathbf{Q}_d\mathbf{G}_k^\top$$

Generated by doxygen 1.8.19 and m.css.

First-Estimate Jacobian Estimators

EKF Linearized Error-State System

When developing an extended Kalman filter (EKF), one needs to linearize the nonlinear motion and measurement models about some linearization point. This linearization is one of the sources of error causing inaccuracies in the estimates (in addition to, for exmaple, model errors and measurement noise). Let us consider the following linearized error-state visual-inertial system:

$$\begin{aligned}\tilde{\mathbf{x}}_{k|k-1} &= \mathbf{\Phi}_{(k,k-1)}\,\tilde{\mathbf{x}}_{k-1|k-1} + \mathbf{G}_k\mathbf{w}_k \\ \tilde{\mathbf{z}}_k &= \mathbf{H}_k\,\tilde{\mathbf{x}}_{k|k-1} + \mathbf{n}_k\end{aligned}$$

where the state contains the inertial navigation state and a single environmental feature (noting that we do not include biases to simplify the derivations):

$$\mathbf{x}_k = \begin{bmatrix} {}^{I_k}_{G}\bar{q}^\top & {}^{G}\mathbf{p}_{I_k}^\top & {}^{G}\mathbf{v}_{I_k}^\top & {}^{G}\mathbf{p}_{f}^\top \end{bmatrix}^\top$$

Note that we use the left quaternion error state (see [Indirect Kalman Filter for 3D Attitude Estimation] **[18]** for details). For simplicity we assume that the camera and IMU frame have an identity transform. We can compute the measurement Jacobian of a given feature based on the perspective projection camera model at the *k*-th timestep as follows:

$$\begin{aligned}\mathbf{H}_k &= \mathbf{H}_{proj,k}\,\mathbf{H}_{state,k} \\ &= \begin{bmatrix} \frac{1}{{}^Iz} & 0 & \frac{-{}^Ix}{({}^Iz)^2} \\ 0 & \frac{1}{{}^Iz} & \frac{-{}^Iy}{({}^Iz)^2} \end{bmatrix} \begin{bmatrix} \lfloor {}^{I_k}_{G}\mathbf{R}({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_k})\times\rfloor & -{}^{I_k}_{G}\mathbf{R} & \mathbf{0}_{3\times3} & {}^{I_k}_{G}\mathbf{R} \end{bmatrix} \\ &= \mathbf{H}_{proj,k}\,{}^{I_k}_{G}\mathbf{R}\begin{bmatrix} \lfloor({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_k})\times\rfloor {}^{I_k}_{G}\mathbf{R}^\top & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix}\end{aligned}$$

The state-transition (or system Jacobian) matrix from timestep *k-1* to *k* as (see [**IMU Propagation Derivations**] for more details):

$$\mathbf{\Phi}_{(k,k-1)} = \begin{bmatrix} {}^{I_k}_{I_{k-1}}\mathbf{R} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -{}^{I_{k-1}}_{G}\mathbf{R}^\top\lfloor\boldsymbol{\alpha}(k,k-1)\times\rfloor & \mathbf{I}_{3\times3} & (t_k - t_{k-1})\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ -{}^{I_{k-1}}_{G}\mathbf{R}^\top\lfloor\boldsymbol{\beta}(k,k-1)\times\rfloor & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix}$$

$$\begin{aligned}\boldsymbol{\alpha}(k,k-1) &= \int_{t_{k-1}}^{k}\int_{t_{k-1}}^{s} {}^{I_{k-1}}_{\tau}\mathbf{R}(\mathbf{a}(\tau) - \mathbf{b}_a - \mathbf{w}_a)d\tau ds \\ \boldsymbol{\beta}(k,k-1) &= \int_{t_{k-1}}^{t_k} {}^{I_{k-1}}_{\tau}\mathbf{R}(\mathbf{a}(\tau) - \mathbf{b}_a - \mathbf{w}_a)d\tau\end{aligned}$$

where $\mathbf{a}(\tau)$ is the true acceleration at time τ, ${}^{I_k}_{I_{k-1}}\mathbf{R}$ is computed using the gyroscope angular velocity measurements, and ${}^G\mathbf{g} = [0\;0\;9.81]^\top$ is gravity in the global frame of reference. During propagation one would need to solve these integrals using either analytical or numerical integration, while we here are interested in how the state evolves in order to examine its observability.

Linearized System Observability

The observability matrix of this linearized system is defined by:

$$\mathcal{O} = \begin{bmatrix} \mathbf{H}_0 \mathbf{\Phi}_{(0,0)} \\ \mathbf{H}_1 \mathbf{\Phi}_{(1,0)} \\ \mathbf{H}_2 \mathbf{\Phi}_{(2,0)} \\ \vdots \\ \mathbf{H}_k \mathbf{\Phi}_{(k,0)} \end{bmatrix}$$

where $\mathbf{H}_k$ is the measurement Jacobian at timestep *k* and $\mathbf{\Phi}_{(k,0)}$ is the compounded state transition (system Jacobian) matrix from timestep 0 to k. For a given block row of this matrix, we have:

$$\mathbf{H}_k \mathbf{\Phi}_{(k,0)} = \mathbf{H}_{proj,k} \, {}^{I_k}_{G}\mathbf{R} \begin{bmatrix} \mathbf{\Gamma}_1 & \mathbf{\Gamma}_2 & \mathbf{\Gamma}_3 & \mathbf{\Gamma}_4 \end{bmatrix}$$

$$\begin{aligned} \mathbf{\Gamma}_1 &= \left\lfloor \left({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_k} + {}^{I_0}_{G}\mathbf{R}^\top \boldsymbol{\alpha}(k,0) \right) \times \right\rfloor {}^{I_0}_{G}\mathbf{R}^\top \\ &= \left\lfloor \left({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_0} - {}^G\mathbf{v}_{I_0}(t_k - t_0) - \frac{1}{2} {}^G\mathbf{g}(t_k - t_0)^2 \right) \times \right\rfloor {}^{I_0}_{G}\mathbf{R}^\top \\ \mathbf{\Gamma}_2 &= -\mathbf{I}_{3\times3} \\ \mathbf{\Gamma}_3 &= -(t_k - t_0)\mathbf{I}_{3\times3} \\ \mathbf{\Gamma}_4 &= \mathbf{I}_{3\times3} \end{aligned}$$

We now verify the following nullspace which corresponds to the global yaw about gravity and global IMU and feature positions:

$$\mathcal{N}_{vins} = \begin{bmatrix} {}^{I_0}_{G}\mathbf{R} \, {}^G\mathbf{g} & \mathbf{0}_{3\times3} \\ -\lfloor {}^G\mathbf{p}_{I_0} \times \rfloor {}^G\mathbf{g} & \mathbf{I}_{3\times3} \\ -\lfloor {}^G\mathbf{v}_{I_0} \times \rfloor {}^G\mathbf{g} & \mathbf{0}_{3\times3} \\ -\lfloor {}^G\mathbf{p}_f \times \rfloor {}^G\mathbf{g} & \mathbf{I}_{3\times3} \end{bmatrix}$$

It is not difficult to verify that $\mathbf{H}_k \mathbf{\Phi}_{(k,0)} \mathcal{N}_{vio} = \mathbf{0}$. Thus this is a nullspace of the system, which clearly shows that there are the four unobserable directions (global yaw and position) of visual-inertial systems.

First Estimate Jacobians

The main idea of First-Estimate Jacobains (FEJ) approaches is to ensure that the state transition and Jacobian matrices are evaluated at correct linearization points such that the above observability analysis will hold true. For those interested in the technical details please take a look at: [6] and [9]. Let us first consider a small thought experiment of how the standard Kalman filter computes its state transition matrix. From a timestep zero to one it will use the current estimates from state zero forward in time. At the next timestep after it updates the state with measurements from other sensors, it will compute the state transition with the updated values to evolve the state to timestep two. This causes a miss-match in the "continuity" of the state transition matrix which when multiply sequentially should represent the evolution from time zero to time two.

k-1=>k+1是一个计算和update单元(1个整体)

$$\mathbf{\Phi}_{(k+1,k-1)}(\mathbf{x}_{k+1|k}, \mathbf{x}_{k-1|k-1}) \neq \mathbf{\Phi}_{(k+1,k)}(\mathbf{x}_{k+1|k}, \mathbf{x}_{k|k}) \, \mathbf{\Phi}_{(k,k-1)}(\mathbf{x}_{k|k-1}, \mathbf{x}_{k-1|k-1})$$

As shown above, we wish to compute the state transition matrix from the *k-1* timestep given all *k-1* measurements up until the current propagated timestep *k+1* given all *k* measurements. The right side of the above equation is how one would normally perform this in a Kalman filter framework. $\mathbf{\Phi}_{(k,k-1)}(\mathbf{x}_{k|k-1}, \mathbf{x}_{k-1|k-1})$ corresponds to propagating from the *k-1* update time to the *k* timestep. One would then normally perform the *k*'th update to the state and then propagate from this **updated** state to the newest timestep (i.e. the $\mathbf{\Phi}_{(k+1,k)}(\mathbf{x}_{k+1|k}, \mathbf{x}_{k|k})$ state transition matrix). This clearly is different then if one was to compute the state transition from time *k-1* to the *k+1* timestep as the second state transition is evaluated at the different $\mathbf{x}_{k|k}$ linearization point! To fix this, we can change the linearization point we evaluate these at:

$$\mathbf{\Phi}_{(k+1,k-1)}(\mathbf{x}_{k+1|k}, \mathbf{x}_{k-1|k-1}) = \mathbf{\Phi}_{(k+1,k)}(\mathbf{x}_{k+1|k}, \underline{\mathbf{x}_{k|k-1}})\ \mathbf{\Phi}_{(k,k-1)}(\mathbf{x}_{k|k-1}, \mathbf{x}_{k-1|k-1})$$

We also need to ensure that our measurement Jacobians match the linearization point of the state transition matrix. Thus they also need to be evaluated at the $\mathbf{x}_{k|k-1}$ linearization point instead of the $\mathbf{x}_{k|k}$ that one would normally use. This gives way to the name FEJ since we will evaluate the Jacobians at the same linearization point to ensure that the nullspace remains valid. For example if we evaluated the $\mathbf{H}_k$ Jacobian with a different ${}^G\mathbf{p}_f$ at each timestep then the nullspace would not hold past the first time instance.

Generated by doxygen 1.8.19 and m.css.

Measurement Update Derivations

Minimum Mean Square Error (MMSE) Estimation

Consider the following static state estimation problem: Given a prior distribution (probability density function or pdf) for a Gaussian random vector $\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{x}}^{\ominus}, \mathbf{P}_{xx}^{\ominus})$ with dimension of n and a new m dimentional measurement $\mathbf{z}_m = \mathbf{z} + \mathbf{n} = \mathbf{h}(\mathbf{x}) + \mathbf{n}$ corrupted by zero-mean white Gaussian noise independent of state, $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, we want to compute the first two (central) moments of the posterior pdf $p(\mathbf{x}|\mathbf{z}_m)$. Generally (given a nonlinear measurement model), we approximate the posterior pdf as: $p(\mathbf{x}|\mathbf{z}_m) \simeq \mathcal{N}(\hat{\mathbf{x}}^{\oplus}, \mathbf{P}_{xx}^{\oplus})$. By design, this is the (approximate) solution to the MMSE estimation problem [Kay 1993] [8].

Conditional Probability Distribution

To this end, we employ the Bayes Rule:

$$p(\mathbf{x}|\mathbf{z}_m) = \frac{p(\mathbf{x}, \mathbf{z}_m)}{p(\mathbf{z}_m)}$$

In general, this conditional pdf cannot be computed analytically without imposing simplifying assumptions. For the problem at hand, we first approximate (if indeed) $p(\mathbf{z}_m) \simeq \mathcal{N}(\hat{\mathbf{z}}, \mathbf{P}_{zz})$, and then have the following joint Gaussian pdf (noting that joint of Gaussian pdfs is Gaussian):

$$p(\mathbf{x}, \mathbf{z}_m) = \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{x}}^{\ominus} \\ \mathbf{z} \end{bmatrix}, \begin{bmatrix} \mathbf{P}_{xx} & \mathbf{P}_{xz} \\ \mathbf{P}_{zx} & \mathbf{P}_{zz} \end{bmatrix}\right) =: \mathcal{N}(\hat{\mathbf{y}}, \mathbf{P}_{yy})$$

Substitution of these two Gaussians into the first equation yields the following conditional Gaussian pdf:

$$\begin{aligned}
p(\mathbf{x}|\mathbf{z}_m) &\simeq \frac{\mathcal{N}(\hat{\mathbf{y}},\mathbf{P}_{yy})}{\mathcal{N}(\hat{\mathbf{z}},\mathbf{P}_{zz})} \\
&= \frac{\frac{1}{\sqrt{(2\pi)^{n+m}|\mathbf{P}_{yy}|}}e^{-\frac{1}{2}(\mathbf{y}-\hat{\mathbf{y}})^\top\mathbf{P}_{yy}^{-1}(\mathbf{y}-\hat{\mathbf{y}})}}{\frac{1}{\sqrt{(2\pi)^{m}|\mathbf{P}_{zz}|}}e^{-\frac{1}{2}(\mathbf{z}_m-\hat{\mathbf{z}})^\top\mathbf{P}_{zz}^{-1}(\mathbf{z}_m-\hat{\mathbf{z}})}} \\
&= \frac{1}{\sqrt{(2\pi)^{n}|\mathbf{P}_{yy}|/|\mathbf{P}_{zz}|}}e^{-\frac{1}{2}\left[(\mathbf{y}-\hat{\mathbf{y}})^\top\mathbf{P}_{yy}^{-1}(\mathbf{y}-\hat{\mathbf{y}})-(\mathbf{z}_m-\hat{\mathbf{z}})^\top\mathbf{P}_{zz}^{-1}(\mathbf{z}_m-\hat{\mathbf{z}})\right]} \\
&=: \mathcal{N}(\hat{\mathbf{x}}^\oplus,\mathbf{P}_{xx}^\oplus)
\end{aligned}$$

We now derive the conditional mean and covariance can be computed as follows: First we simplify the denominator term $|\mathbf{P}_{yy}|/|\mathbf{P}_{zz}|$ in order to find the conditional covariance.

$$|\mathbf{P}_{yy}| = \left|\begin{bmatrix}\mathbf{P}_{xx} & \mathbf{P}_{xz}\\ \mathbf{P}_{zx} & \mathbf{P}_{zz}\end{bmatrix}\right| = \left|\mathbf{P}_{xx}-\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\right|\;\left|\mathbf{P}_{zz}\right|$$

where we assumed $\mathbf{P}_{zz}$ is invertible and employed the determinant property of Schur complement. Thus, we have:

$$\frac{|\mathbf{P}_{yy}|}{|\mathbf{P}_{zz}|} = \frac{\left|\mathbf{P}_{xx}-\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\right|\left|\mathbf{P}_{zz}\right|}{|\mathbf{P}_{zz}|} = \left|\mathbf{P}_{xx}-\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\right|$$

Next, by defining the error states $\mathbf{r}_x = \mathbf{x}-\hat{\mathbf{x}}^\ominus, \mathbf{r}_z = \mathbf{z}_m - \hat{\mathbf{z}}, \mathbf{r}_y = \mathbf{y}-\hat{\mathbf{y}}$, and using the matrix inersion lemma, we rewrite the exponential term as follows:

$$
\begin{aligned}
&(\mathbf{y}-\hat{\mathbf{y}})^\top \mathbf{P}_{yy}^{-1}(\mathbf{y}-\hat{\mathbf{y}}) - (\mathbf{z}_m-\hat{\mathbf{z}})^\top \mathbf{P}_{zz}^{-1}(\mathbf{z}_m-\hat{\mathbf{z}}) \\
&= \mathbf{r}_y^\top \mathbf{P}_{yy}^{-1}\mathbf{r}_y - \mathbf{r}_z^\top \mathbf{P}_{zz}^{-1}\mathbf{r}_z \\
&= \begin{bmatrix}\mathbf{r}_x \\ \mathbf{r}_z\end{bmatrix}^\top \begin{bmatrix}\mathbf{P}_{xx} & \mathbf{P}_{xz} \\ \mathbf{P}_{zx} & \mathbf{P}_{zz}\end{bmatrix}^{-1} \begin{bmatrix}\mathbf{r}_x \\ \mathbf{r}_z\end{bmatrix} - \mathbf{r}_z^\top \mathbf{P}_{zz}^{-1}\mathbf{r}_z \\
&= \begin{bmatrix}\mathbf{r}_x \\ \mathbf{r}_z\end{bmatrix}^\top \begin{bmatrix}\mathbf{Q} & -\mathbf{Q}\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1} \\ -\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\mathbf{Q} & \mathbf{P}_{zz}^{-1}+\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\mathbf{Q}\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\end{bmatrix} \begin{bmatrix}\mathbf{r}_x \\ \mathbf{r}_z\end{bmatrix} - \mathbf{r}_z^\top \mathbf{P}_{zz}^{-1}\mathbf{r}_z \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{where } \mathbf{Q} = (\mathbf{P}_{xx} - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx})^{-1} \\
&= \mathbf{r}_x^\top \mathbf{Q}\mathbf{r}_x - \mathbf{r}_x^\top \mathbf{Q}\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z - \mathbf{r}_z^\top \mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\mathbf{Q}\mathbf{r}_x \\
&\qquad\qquad + \mathbf{r}_z^\top({\color{red}\mathbf{P}_{zz}^{-1}} + \mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}\mathbf{Q}\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1})\mathbf{r}_z - {\color{red}\mathbf{r}_z^\top \mathbf{P}_{zz}^{-1}\mathbf{r}_z} \\
&= \mathbf{r}_x^\top \mathbf{Q}\mathbf{r}_x - \mathbf{r}_x^\top \mathbf{Q}[\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_x] - [\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z]^\top \mathbf{Q}\mathbf{r}_x + [\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z]^\top \mathbf{Q}[\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z] \\
&= (\mathbf{r}_x - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z)^\top \mathbf{Q}(\mathbf{r}_x - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z) \\
&= (\mathbf{r}_x - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z)^\top (\mathbf{P}_{xx} - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx})^{-1}(\mathbf{r}_x - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z)
\end{aligned}
$$

where $(\mathbf{P}_{zz}^{-1})^\top = \mathbf{P}_{zz}^{-1}$ since covariance matrices are symmetric. Up to this point, we can now construct the conditional Gaussian pdf as follows:

$$
\begin{aligned}
p(\mathbf{x}_k|\mathbf{z}_m) = &\frac{1}{\sqrt{(2\pi)^n|\mathbf{P}_{xx} - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}|}} \times \\
&\exp\left(-\frac{1}{2}\left[(\mathbf{r}_x - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z)^\top \underbrace{(\mathbf{P}_{xx} - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx})}_{\text{covariance}}{}^{-1}(\mathbf{r}_x - \underbrace{\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{r}_z)}_{\text{mean}}\right]\right)
\end{aligned}
$$

which results in the following conditional mean and covariance we were seeking:

$$
\begin{aligned}
\hat{\mathbf{x}}^{\oplus} &= \hat{\mathbf{x}}^{\ominus} + \underline{\mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}(\mathbf{z}_m - \hat{\mathbf{z}})} \\
\mathbf{P}_{xx}^{\oplus} &= \underline{\mathbf{P}_{xx}^{\ominus} - \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}\mathbf{P}_{zx}}
\end{aligned}
$$

These are the fundamental equations for (linear) state estimation.

Linear Measurement Update

As a special case, we consider a simple linear measurement model to illustrate the linear MMSE estimator:

$$\begin{aligned}
\mathbf{z}_{m,k} &= \mathbf{H}_k\mathbf{x}_k + \mathbf{n}_k \\
\hat{\mathbf{z}}_k &:= \mathbb{E}[\mathbf{z}_{m,k}] = \mathbb{E}[\mathbf{H}_k\mathbf{x}_k + \mathbf{n}_k] = \mathbf{H}_k\hat{\mathbf{x}}_k^{\ominus}
\end{aligned}$$

With this, we can derive the covariance and cross-correlation matrices as follows:

$$\begin{aligned}
\mathbf{P}_{zz} &= \mathbb{E}\left[(\mathbf{z}_{m,k} - \hat{\mathbf{z}}_k)(\mathbf{z}_{m,k} - \hat{\mathbf{z}}_k)^\top\right] \\
&= \mathbb{E}\left[(\mathbf{H}_k\mathbf{x}_k + \mathbf{n}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^{\ominus})(\mathbf{H}_k\mathbf{x}_k + \mathbf{n}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^{\ominus})^\top\right] \\
&= \mathbb{E}\left[(\mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus}) + \mathbf{n}_k)(\mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus}) + \mathbf{n}_k)^\top\right] \\
&= \mathbb{E}\Big[\mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})^\top\mathbf{H}_k^\top + \textcolor{red}{\mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})\mathbf{n}_k^\top} \\
&\qquad\qquad + \textcolor{red}{\mathbf{n}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})^\top\mathbf{H}_k^\top} + \mathbf{n}_k\mathbf{n}_k^\top\Big] \\
&= \mathbb{E}\left[\mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})^\top\mathbf{H}_k^\top + \mathbf{n}_k\mathbf{n}_k^\top\right] \\
&= \mathbf{H}_k\ \mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})^\top\right]\mathbf{H}_k^\top +\ \mathbb{E}\left[\mathbf{n}_k\mathbf{n}_k^\top\right] \\
&= \mathbf{H}_k\mathbf{P}_{xx}^{\ominus}\mathbf{H}_k^\top + \mathbf{R}_k
\end{aligned}$$

where $\mathbf{R}_k$ is the *discrete* measurement noise matrix, $\mathbf{H}_k$ is the measurement Jacobian mapping the state into the measurement domain, and $\mathbf{P}_{xx}^{\ominus}$ is the current state covariance.

$$
\begin{aligned}
\mathbf{P}_{xz} &= \mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{z}_{m,k} - \hat{\mathbf{z}}_k)^\top\right] \\
&= \mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{H}_k\mathbf{x}_k + \mathbf{n}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^{\ominus})^\top\right] \\
&= \mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus}) + \mathbf{n}_k)^\top\right] \\
&= \mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})^\top\mathbf{H}_k^\top + (\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})\mathbf{n}_k^\top\right] \\
&= \mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})^\top\right]\mathbf{H}_k^\top + {\color{red}\mathbb{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{\ominus})\mathbf{n}_k^\top\right]} \\
&= \mathbf{P}_{xx}^{\ominus}\mathbf{H}_k^\top
\end{aligned}
$$

where we have employed the fact that the noise is independent of the state. Substitution of these quantities into the fundamental equation leads to the following update equations:

$$
\begin{aligned}
\hat{\mathbf{x}}_k^{\oplus} &= \hat{\mathbf{x}}_k^{\ominus} + \mathbf{P}_k^{\ominus}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_k^{\ominus}\mathbf{H}_k^\top + \mathbf{R}_k)^{-1}(\mathbf{z}_{m,k} - \hat{\mathbf{z}}_k) \\
&= \hat{\mathbf{x}}_k^{\ominus} + \mathbf{P}_k^{\ominus}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_k^{\ominus}\mathbf{H}_k^\top + \mathbf{R}_k)^{-1}(\mathbf{z}_{m,k} - \mathbf{H}_k\hat{\mathbf{x}}_k^{\ominus}) \\
&= \hat{\mathbf{x}}_k^{\ominus} + \mathbf{K}\mathbf{r}_z \\
\mathbf{P}_{xx}^{\oplus} &= \mathbf{P}_k^{\ominus} - \mathbf{P}_k^{\ominus}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_k^{\ominus}\mathbf{H}_k^\top + \mathbf{R}_k)^{-1}(\mathbf{P}_k^{\ominus}\mathbf{H}_k^\top)^\top \\
&= \mathbf{P}_k^{\ominus} - \mathbf{P}_k^{\ominus}\mathbf{H}_k^\top(\mathbf{H}_k\mathbf{P}_k^{\ominus}\mathbf{H}_k^\top + \mathbf{R}_k)^{-1}\mathbf{H}_k\mathbf{P}_k^{\ominus}
\end{aligned}
$$

These are essentially the Kalman filter (or linear MMSE) update equations.

Update Equations and Derivations

- **3D Feature Triangulation** — 3D feature triangulation derivations for getting a feature linearization point
- **Camera Measurement Update** — Measurement equations and derivation for 3D feature point
- **Delayed Feature Initialization** — How to perform delayed initialization
- **MSCKF Nullspace Projection** — MSCKF nullspace projection
- **Measurement Compression** — MSCKF measurement compression
- **Zero Velocity Update** — Zero velocity stationary update

Generated by doxygen 1.8.19 and m.css.

Measurement Update Derivations » 3D Feature Triangulation

Linear Triangulation

We wish to create a solvable linear system that can give us an initial guess for the 3D position of our feature. To do this, we take all the poses that the feature is seen from to be of known quantity. This feature will be triangulated in some anchor camera frame $\{A\}$ which we can arbitrary pick. If the feature $\mathbf{p}_f$ is observed by pose $1\ldots m$, given the anchor pose A, we can have the following transformation from any camera pose $C_i, i=1\ldots m$:

$$\begin{aligned}{}^{C_i}\mathbf{p}_f &= {}^{C_i}_{A}\mathbf{R}\left({}^{A}\mathbf{p}_f - {}^{A}\mathbf{p}_{C_i}\right)\\ {}^{A}\mathbf{p}_f &= {}^{C_i}_{A}\mathbf{R}^{\top}\,{}^{C_i}\mathbf{p}_f + {}^{A}\mathbf{p}_{C_i}\end{aligned}$$

In the absents of noise, the measurement in the current frame is the unknown bearing ${}^{C_i}\mathbf{b}$ and its depth ${}^{C_i}z$. Thus we have the following mapping to a feature seen from the current frame:

$${}^{C_i}\mathbf{p}_f = {}^{C_i}z_f\,{}^{C_i}\mathbf{b}_f = {}^{C_i}z_f\begin{bmatrix}u_n\\ v_n\\ 1\end{bmatrix}$$

We note that u_n and v_n represent the undistorted normalized image coordinates. This bearing can be warped into the the anchor frame by substituting into the above equation:

$$\begin{aligned}{}^{A}\mathbf{p}_f &= {}^{C_i}_{A}\mathbf{R}^{\top}z_f\,{}^{C_i}\mathbf{b}_f + {}^{A}\mathbf{p}_{C_i}\\ &= z_f\,{}^{A}\mathbf{b}_{C_i\rightarrow f} + {}^{A}\mathbf{p}_{C_i}\end{aligned}$$

To remove the need to estimate the extra degree of freedom of depth z_f, we define the following two vectors:

$$\begin{aligned}{}^{A}\mathbf{n}_1 &= \begin{bmatrix}-{}^{A}b_{C_i\rightarrow f}(3) & 0 & {}^{A}b_{C_i\rightarrow f}(1)\end{bmatrix}^{\top}\\ {}^{A}\mathbf{n}_2 &= \begin{bmatrix}0 & {}^{A}b_{C_i\rightarrow f}(3) & -{}^{A}b_{C_i\rightarrow f}(2)\end{bmatrix}^{\top}\end{aligned}$$

These are perpendicular with the vector ${}^{A}\mathbf{b}_{C_i \to f}$ and thus ${}^{A}\mathbf{n}_1^{\top}{}^{A}\mathbf{b}_{C_i \to f} = 0$ and ${}^{A}\mathbf{n}_2^{\top}{}^{A}\mathbf{b}_{C_i \to f} = 0$ holds true. We can then multiple the transform equation/constraint to form two equation which only relates to the unknown 3 d.o.f ${}^{A}\mathbf{p}_f$:

$$\begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} {}^{A}\mathbf{p}_f = \begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} z_f {}^{A}\mathbf{b}_{C_i \to f} + \begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} {}^{A}\mathbf{p}_{C_i}$$
$$\begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} {}^{A}\mathbf{p}_f = \begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} {}^{A}\mathbf{p}_{C_i}$$

By stacking all the measurements, we can have:

$$\begin{bmatrix} \vdots \\ \begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} \\ \vdots \end{bmatrix} {}^{A}\mathbf{p}_f = \begin{bmatrix} \vdots \\ \begin{bmatrix} {}^{A}\mathbf{n}_1^{\top} \\ {}^{A}\mathbf{n}_2^{\top} \end{bmatrix} {}^{A}\mathbf{p}_{C_i} \\ \vdots \end{bmatrix}$$

Since each pixel measurement provides two constraints, as long as $m > 1$, we will have enough constraints to triangulate the feature. In practice, the more views of the feature the better the triangulation and thus normally want to have a feature seen from at least five views. We additionally check that the triangulated feature is "valid" and in front of the camera and not too far away. The condition number of the above linear system and reject systems that are "sensitive" to errors and have a large value.

Non-linear Feature Optimization

After we get the triangulated feature 3D position, a nonlinear least-squares will be performed to refine this estimate. In order to achieve good numerical stability, we use the inverse depth representation for point feature which helps with convergence. We find that in most cases this problem converges within 2-3 iterations in indoor environments. The feature transformation can be written as:

$$\begin{aligned} {}^{C_i}\mathbf{p}_f &= {}^{C_i}_{A}\mathbf{R}\left({}^{A}\mathbf{p}_f - {}^{A}\mathbf{p}_{C_i}\right) \\ &= {}^{A}z_f {}^{C_i}_{A}\mathbf{R}\left(\begin{bmatrix} {}^{A}x_f/{}^{A}z_f \\ {}^{A}y_f/{}^{A}z_f \\ 1 \end{bmatrix} - \frac{1}{{}^{A}z_f}{}^{A}\mathbf{p}_{C_i}\right) \\ \Rightarrow \frac{1}{{}^{A}z_f}{}^{C_i}\mathbf{p}_f &= {}^{C_i}_{A}\mathbf{R}\left(\begin{bmatrix} {}^{A}x_f/{}^{A}z_f \\ {}^{A}y_f/{}^{A}z_f \\ 1 \end{bmatrix} - \frac{1}{{}^{A}z_f}{}^{A}\mathbf{p}_{C_i}\right) \end{aligned}$$

We define $u_A = {}^{A}x_f/{}^{A}z_f, v_A = {}^{A}y_f/{}^{A}z_f$, and $\rho_A = 1/{}^{A}z_f$ to get the following measurement equation:

$$h(u_A, v_A, \rho_A) = {}^{C_i}_{A}\mathbf{R}\left(\begin{bmatrix} u_A \\ v_A \\ 1 \end{bmatrix} - \rho_A {}^{A}\mathbf{p}_{C_i}\right)$$

The feature measurement seen from the $\{C_i\}$ camera frame can be reformulated as:

$$\begin{aligned} \mathbf{z} &= \begin{bmatrix} u_i \\ v_i \end{bmatrix} \\ &= \begin{bmatrix} h(u_A, v_A, \rho_A)(1)/h(u_A, v_A, \rho_A)(3) \\ h(u_A, v_A, \rho_A)(2)/h(u_A, v_A, \rho_A)(3) \end{bmatrix} \\ &= \mathbf{h}(u_A, v_A, \rho_A) \end{aligned}$$

Therefore, we can have the least-squares formulated and Jacobians:

$$\underset{u_A, v_A, \rho_A}{\operatorname{argmin}} ||\mathbf{z} - \mathbf{h}(u_A, v_A, \rho_A)||^2$$

$$\frac{\partial \mathbf{h}(u_A, v_A, \rho_A)}{\partial h(u_A, v_A, \rho_A)} = \begin{bmatrix} 1/h(\cdots)(1) & 0 & -h(\cdots)(1)/h(\cdots)(3)^2 \\ 0 & 1/h(\cdots)(2) & -h(\cdots)(2)/h(\cdots)(3)^2 \end{bmatrix}$$

$$\frac{\partial h(u_A, v_A, \rho_A)}{\partial [u_A, v_A, \rho_A]} = {}^{C_i}_{A}\mathbf{R} \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} & -{}^{A}\mathbf{p}_{C_i} \end{bmatrix}$$

The least-squares problem can be solved with Gaussian-Newton or Levenberg-Marquart algorithm.

Generated by doxygen 1.8.19 and m.css.

OpenVINS

Measurement Update Derivations » Camera Measurement Update

Contents

(用罗盘测定的) 方位;

Perspective Projection (Bearing) Measurement Model

Consider a 3D feature is detected from the camera image at time k, whose uv measurement (i.e., the corresponding pixel coordinates) on the image plane is given by:

$$\begin{aligned}
\mathbf{z}_{m,k} &= \mathbf{h}(\mathbf{x}_k) + \mathbf{n}_k \\
&= \mathbf{h}_d(\mathbf{z}_{n,k},\ \boldsymbol{\zeta}) + \mathbf{n}_k \\
&= \mathbf{h}_d(\mathbf{h}_p({}^{C_k}\mathbf{p}_f),\ \boldsymbol{\zeta}) + \mathbf{n}_k \\
&= \mathbf{h}_d(\mathbf{h}_p(\mathbf{h}_t(\underline{{}^{G}\mathbf{p}_f,\ {}^{C_k}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{C_k})),\ \boldsymbol{\zeta})} + \mathbf{n}_k \\
&= \mathbf{h}_d(\mathbf{h}_p(\mathbf{h}_t(\mathbf{h}_r(\boldsymbol{\lambda},\cdots),\ {}^{C_k}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{C_k})),\ \boldsymbol{\zeta}) + \mathbf{n}_k
\end{aligned}$$

where $\mathbf{n}_k$ is the measurement noise and typically assumed to be zero-mean white Gaussian; $\mathbf{z}_{n,k}$ is the normalized undistorted uv measurement; $\boldsymbol{\zeta}$ is the camera intrinsic parameters such as focal length and distortion parameters; ${}^{C_k}\mathbf{p}_f$ is the feature position in the current camera frame $\{C_k\}$; ${}^{G}\mathbf{p}_f$ is the feature position in the global frame $\{G\}$; $\{{}^{C_k}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{C_k}\}$ denotes the current camera pose (position and orientation) in the global frame (or camera extrinsics); and $\boldsymbol{\lambda}$ is the feature's parameters of different representations (other than position) such as simply a xyz position or an inverse depth with bearing.

In the above expression, we decompose the measurement function into multiple concatenated functions corresponding to different operations, which map the states into the raw uv measurement on the image plane. It should be noted that as we will perform intrinsic calibration along with extrinsic with different feature representations, the above camera measurement model is general. The high-level description of each function is given in the next section.

Measurement Function Overview

Function	Description
$\mathbf{z}_k = \mathbf{h}_d(\mathbf{z}_{n,k},\ \boldsymbol{\zeta})$	The distortion function that takes normalized coordinates and maps it into distorted uv coordinates
$\mathbf{z}_{n,k} = \mathbf{h}_p({}^{C_k}\mathbf{p}_f)$	The projection function that takes a 3D point in the image and converts it into the normalized uv coordinates
${}^{C_k}\mathbf{p}_f = \mathbf{h}_t({}^{G}\mathbf{p}_f,\ {}^{C_k}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{C_k})$	Transforming a feature's position in the global frame into the current camera frame
${}^{G}\mathbf{p}_f = \mathbf{h}_r(\boldsymbol{\lambda},\cdots)$	Converting from a feature representation to a 3D feature in the global frame

Jacobian Computation

Given the above nested functions, we can leverage the chainrule to find the total state Jacobian. Since our feature representation function $\mathbf{h}_r(\cdots)$ might also depend on the state, i.e. an anchoring pose, we need to carefully consider its additional derivatives. Consider the following example of our

+translation

measurement in respect to a state $\mathbf{x}$ Jacobian:

$$\frac{\partial \mathbf{z}_k}{\partial \mathbf{x}} = \frac{\partial \mathbf{h}_d(\cdot)}{\partial \mathbf{z}_{n,k}} \frac{\partial \mathbf{h}_p(\cdot)}{\partial {}^{C_k}\mathbf{p}_f} \frac{\partial \mathbf{h}_t(\cdot)}{\partial \mathbf{x}} + \frac{\partial \mathbf{h}_d(\cdot)}{\partial \mathbf{z}_{n,k}} \frac{\partial \mathbf{h}_p(\cdot)}{\partial {}^{C_k}\mathbf{p}_f} \frac{\partial \mathbf{h}_t(\cdot)}{\partial {}^{G}\mathbf{p}_f} \frac{\partial \mathbf{h}_r(\cdot)}{\partial \mathbf{x}}$$

In the global feature representations, see **Point Feature Representations** section, the second term will be zero while for the anchored representations it will need to be computed.

Distortion Function

Radial model

To calibrate camera intrinsics, we need to know how to map our normalized coordinates into the raw pixel coordinates on the image plane. We first employ the radial distortion as in OpenCV model:

$$\begin{bmatrix} u \\ v \end{bmatrix} := \mathbf{z}_k = \mathbf{h}_d(\mathbf{z}_{n,k},\ \boldsymbol{\zeta}) = \begin{bmatrix} f_x * x + c_x \\ f_y * y + c_y \end{bmatrix}$$

$$\begin{aligned} \text{where}\ \ x &= x_n(1 + k_1 r^2 + k_2 r^4) + 2p_1 x_n y_n + p_2(r^2 + 2x_n^2) \\ y &= y_n(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y_n^2) + 2p_2 x_n y_n \end{aligned}$$

$$r^2 = x_n^2 + y_n^2$$

where $\mathbf{z}_{n,k} = [x_n\ y_n]^\top$ are the normalized coordinates of the 3D feature and u and v are the distorted image coordinates on the image plane. The following distortion and camera intrinsic (focal length and image center) parameters are involved in the above distortion model, which can be estimated online:

$$\boldsymbol{\zeta} = \begin{bmatrix} f_x & f_y & c_x & c_y & k_1 & k_2 & p_1 & p_2 \end{bmatrix}^\top$$

Note that we do not estimate the higher order (i.e., higher than fourth order) terms as in most offline calibration methods such as Kalibr. To estimate these intrinsic parameters (including the distortation parameters), the following Jacobian for these parameters is needed:

$$\frac{\partial \mathbf{h}_d(\cdot)}{\partial \boldsymbol{\zeta}} = \begin{bmatrix} x & 0 & 1 & 0 & f_x * (x_n r^2) & f_x * (x_n r^4) & f_x * (2x_n y_n) & f_x * (r^2 + 2x_n^2) \\ 0 & y & 0 & 1 & f_y * (y_n r^2) & f_y * (y_n r^4) & f_y * (r^2 + 2y_n^2) & f_y * (2x_n y_n) \end{bmatrix}$$

Similarly, the Jacobian with respect to the normalized coordinates can be obtained as follows:

$$\frac{\partial \mathbf{h}_d(\cdot)}{\partial \mathbf{z}_{n,k}} = \begin{bmatrix} f_x * ((1 + k_1 r^2 + k_2 r^4) + (2k_1 x_n^2 + 4k_2 x_n^2(x_n^2 + y_n^2)) + 2p_1 y_n + (2p_2 x_n + 4p_2 x_n)) & f_x * (2k_1 x_n y_n + \\ f_y * (2k_1 x_n y_n + 4k_2 x_n y_n (x_n^2 + y_n^2) + 2p_1 x_n + 2p_2 y_n) & f_y * ((1 + k_1 r^2 + k_2 r^4) + (2k \end{bmatrix}$$

Fisheye model

As fisheye or wide-angle lenses are widely used in practice, we here provide mathematical derivations of such distortion model as in OpenCV fisheye.

$$\begin{bmatrix} u \\ v \end{bmatrix} := \mathbf{z}_k = \mathbf{h}_d(\mathbf{z}_{n,k},\ \boldsymbol{\zeta}) = \begin{bmatrix} f_x * x + c_x \\ f_y * y + c_y \end{bmatrix}$$

$$\begin{aligned} \text{where}\ \ x &= \frac{x_n}{r} * \theta_d \\ y &= \frac{y_n}{r} * \theta_d \\ \theta_d &= \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \\ r^2 &= x_n^2 + y_n^2 \\ \theta &= atan(r) \end{aligned}$$

where $\mathbf{z}_{n,k} = [x_n\ y_n]^\top$ are the normalized coordinates of the 3D feature and u and v are the distorted image coordinates on the image plane. Clearly, the following distortion intrinsic parameters are used in the above model:

$$\boldsymbol{\zeta} = \begin{bmatrix} f_x & f_y & c_x & c_y & k_1 & k_2 & k_3 & k_4 \end{bmatrix}^\top$$

In analogy to the previous radial distortion case, the following Jacobian for these parameters is needed for intrinsic calibration:

$$\frac{\partial \mathbf{h}_d(\cdot)}{\partial \boldsymbol{\zeta}} = \begin{bmatrix} x_n & 0 & 1 & 0 & f_x * (\frac{x_n}{r}\theta^3) & f_x * (\frac{x_n}{r}\theta^5) & f_x * (\frac{x_n}{r}\theta^7) & f_x * (\frac{x_n}{r}\theta^9) \\ 0 & y_n & 0 & 1 & f_y * (\frac{y_n}{r}\theta^3) & f_y * (\frac{y_n}{r}\theta^5) & f_y * (\frac{y_n}{r}\theta^7) & f_y * (\frac{y_n}{r}\theta^9) \end{bmatrix}$$

Similarly, with the chain rule of differentiation, we can compute the following Jacobian with respect to the normalized coordinates:

$$\frac{\partial \mathbf{h}_d(\cdot)}{\partial \mathbf{z}_{n,k}} = \frac{\partial uv}{\partial xy}\frac{\partial xy}{\partial x_n y_n} + \frac{\partial uv}{\partial xy}\frac{\partial xy}{\partial r}\frac{\partial r}{\partial x_n y_n} + \frac{\partial uv}{\partial xy}\frac{\partial xy}{\partial \theta_d}\frac{\partial \theta_d}{\partial \theta}\frac{\partial \theta}{\partial r}\frac{\partial r}{\partial x_n y_n}$$

$$\text{where} \quad \frac{\partial uv}{\partial xy} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}$$

$$\frac{\partial xy}{\partial x_n y_n} = \begin{bmatrix} \theta_d/r & 0 \\ 0 & \theta_d/r \end{bmatrix}$$

$$\frac{\partial xy}{\partial r} = \begin{bmatrix} -\frac{x_n}{r^2}\theta_d \\ -\frac{y_n}{r^2}\theta_d \end{bmatrix}$$

$$\frac{\partial r}{\partial x_n y_n} = \begin{bmatrix} \frac{x_n}{r} & \frac{y_n}{r} \end{bmatrix}$$

$$\frac{\partial xy}{\partial \theta_d} = \begin{bmatrix} \frac{x_n}{r} \\ \frac{y_n}{r} \end{bmatrix}$$

$$\frac{\partial \theta_d}{\partial \theta} = \begin{bmatrix} 1 + 3k_1\theta^2 + 5k_2\theta^4 + 7k_3\theta^6 + 9k_4\theta^8 \end{bmatrix}$$

$$\frac{\partial \theta}{\partial r} = \begin{bmatrix} \frac{1}{r^2+1} \end{bmatrix}$$

Perspective Projection Function

The standard pinhole camera model is used to project a 3D point in the *camera* frame into the normalized image plane (with unit depth):

$$\mathbf{z}_{n,k} = \mathbf{h}_p({}^{C_k}\mathbf{p}_f) = \begin{bmatrix} {}^Cx/{}^Cz \\ {}^Cy/{}^Cz \end{bmatrix}$$

$$\text{where} \quad {}^{C_k}\mathbf{p}_f = \begin{bmatrix} {}^Cx \\ {}^Cy \\ {}^Cz \end{bmatrix}$$

global不同的深度；image-plane上相同深度,unit-depth;

whose Jacobian matrix is computed as follows:

$$\frac{\partial \mathbf{h}_p(\cdot)}{\partial {}^{C_k}\mathbf{p}_f} = \begin{bmatrix} \frac{1}{{}^Cz} & 0 & \frac{-{}^Cx}{({}^Cz)^2} \\ 0 & \frac{1}{{}^Cz} & \frac{-{}^Cy}{({}^Cz)^2} \end{bmatrix}$$

Euclidean Transformation

We employ the 6DOF rigid-body Euclidean transformation to transform the 3D feature position in the global frame $\{G\}$ to the current camera frame $\{C_k\}$ based on the current global camera pose:

向量的起点在Ck

$${}^{C_k}\mathbf{p}_f = \mathbf{h}_t({}^G\mathbf{p}_f,\ {}^{C_k}_G\mathbf{R},\ {}^G\mathbf{p}_{C_k}) = {}^{C_k}_G\mathbf{R}(\underline{{}^G\mathbf{p}_f - {}^G\mathbf{p}_{C_k}})$$

Note that in visual-inertial navigation systems, we often keep the IMU, instead of camera, state in the state vector. So, we need to further transform the above geometry using the time-invariant IMU-camera extrinsic parameters $\{{}^C_I\mathbf{R},\ {}^C\mathbf{p}_I\}$ as follows:

$$\begin{aligned} {}^G\mathbf{p}_{C_k} &= {}^G\mathbf{p}_{I_k} + {}^G_I\mathbf{R}\,{}^I\mathbf{p}_{C_k} = {}^G\mathbf{p}_{I_k} + {}^G_I\mathbf{R}\,{}^I\mathbf{p}_C \\ {}^{C_k}_G\mathbf{R} &= {}^{C_k}_I\mathbf{R}\,{}^{I_k}_G\mathbf{R} = {}^C_I\mathbf{R}\,{}^{I_k}_G\mathbf{R} \end{aligned}$$

Substituting these quantities into the equation of ${}^{C_k}\mathbf{p}_f$ yields:

$${}^{C_k}\mathbf{p}_f = {}^C_I\mathbf{R}\,{}^{I_k}_G\mathbf{R}({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_k}) + {}^C\mathbf{p}_I$$

We now can compute the following Jacobian with respect to the pertinent states:

$$\begin{aligned} \text{feature pt in global;}\quad \frac{\partial \mathbf{h}_t(\cdot)}{\partial {}^G\mathbf{p}_f} &= {}^C_I\mathbf{R}\,{}^{I_k}_G\mathbf{R} \\ \text{imu和global旋转矩阵}\quad \frac{\partial \mathbf{h}_t(\cdot)}{\partial {}^{I_k}_G\mathbf{R}} &= {}^C_I\mathbf{R}\left\lfloor {}^{I_k}_G\mathbf{R}({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_k})\times \right\rfloor \\ \text{imu在global中的位置}\quad \frac{\partial \mathbf{h}_t(\cdot)}{\partial {}^G\mathbf{p}_{I_k}} &= -{}^C_I\mathbf{R}\,{}^{I_k}_G\mathbf{R} \end{aligned}$$

where $\lfloor \mathbf{a}\times \rfloor$ denotes the skew symmetric matrix of a vector $\mathbf{a}$ (see Quaternion TR [18]). Note also that in above expression (as well as in ensuing derivations), there is a little abuse of notation; that is, the Jacobian with respect to the rotation matrix is not the direct differentiation with respect to the 3x3 rotation matrix, instead with respect to the corresponding 3x1 rotation angle vector. Moreover, if performing online extrinsic calibration, the Jacobian with respect to the IMU-camera extrinsics is needed:

$$\frac{\partial \mathbf{h}_t(\cdot)}{\partial {}_I^C\mathbf{R}} = \left\lfloor {}_I^C\mathbf{R}\, {}_G^{I_k}\mathbf{R}({}^G\mathbf{p}_f - {}^G\mathbf{p}_{I_k})\times \right\rfloor$$
$$\frac{\partial \mathbf{h}_t(\cdot)}{\partial {}^C\mathbf{p}_I} = \mathbf{I}_{3\times 3}$$

Point Feature Representations

There are two main parameterizations of a 3D point feature: 3D position (xyz) and inverse depth with bearing. Both of these can either be represented in the global frame or in an anchor frame of reference which adds a dependency on having an "anchor" pose where the feature is observed. To allow for a unified treatment of different feature parameterizations $\boldsymbol{\lambda}$ in our codebase, we derive in detail the generic function ${}^G\mathbf{p}_f = \mathbf{f}(\cdot)$ that maps different representations into global position.

Global XYZ

As the canonical parameterization, the global position of a 3D point feature is simply given by its xyz coordinates in the global frame of reference:

$$\begin{aligned} {}^G\mathbf{p}_f &= \mathbf{f}(\boldsymbol{\lambda}) \\ &= \begin{bmatrix} {}^Gx \\ {}^Gy \\ {}^Gz \end{bmatrix} \\ \text{where} \quad \boldsymbol{\lambda} &= {}^G\mathbf{p}_f = \begin{bmatrix} {}^Gx & {}^Gy & {}^Gz \end{bmatrix}^\top \end{aligned}$$

It is clear that the Jacobian with respect to the feature parameters is:

$$\frac{\partial \mathbf{f}(\cdot)}{\partial \boldsymbol{\lambda}} = \mathbf{I}_{3\times 3}$$

Global Inverse Depth

The global inverse-depth representation of a 3D point feature is given by (akin to spherical coordinates):

$$
\begin{aligned}
{}^G\mathbf{p}_f &= \mathbf{f}(\boldsymbol{\lambda}) \\
&= \frac{1}{\rho}\begin{bmatrix} \cos(\theta)\sin(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\phi) \end{bmatrix} \\
\text{where} \quad \boldsymbol{\lambda} &= \begin{bmatrix} \theta & \phi & \rho \end{bmatrix}^\top
\end{aligned}
$$

The Jacobian with respect to the feature parameters can be computed as:

$$
\frac{\partial \mathbf{f}(\cdot)}{\partial \boldsymbol{\lambda}} = \begin{bmatrix} -\frac{1}{\rho}\sin(\theta)\sin(\phi) & \frac{1}{\rho}\cos(\theta)\cos(\phi) & -\frac{1}{\rho^2}\cos(\theta)\sin(\phi) \\ \frac{1}{\rho}\cos(\theta)\sin(\phi) & \frac{1}{\rho}\sin(\theta)\cos(\phi) & -\frac{1}{\rho^2}\sin(\theta)\sin(\phi) \\ 0 & -\frac{1}{\rho}\sin(\phi) & -\frac{1}{\rho^2}\cos(\phi) \end{bmatrix}
$$

Global Inverse Depth (MSCKF VERSION)

Note that as this representation has a singularity when the z-distance goes to zero, it is not recommended to use in practice. Instead, one should use the Anchored Inverse Depth (MSCKF Version) representation. The anchored version doesn't have this issue if features are represented in a camera frame that they where seen from (in which features should never have a non-positive z-direction).

Anchored XYZ

We can represent a 3D point feature in some "anchor" frame (say some IMU local frame, $\{{}^{I_a}_G\mathbf{R},\ {}^G\mathbf{p}_{I_a}\}$), which would normally be the IMU pose corresponding to the first camera frame where the feature was detected.

$$
\begin{aligned}
{}^G\mathbf{p}_f &= \mathbf{f}(\boldsymbol{\lambda},\ {}^{I_a}_G\mathbf{R},\ {}^G\mathbf{p}_{I_a},\ {}^C_I\mathbf{R},\ {}^C\mathbf{p}_I) \\
&= {}^{I_a}_G\mathbf{R}^\top {}^C_I\mathbf{R}^\top(\boldsymbol{\lambda} - {}^C\mathbf{p}_I) + {}^G\mathbf{p}_{I_a} \\
\text{where} \quad \boldsymbol{\lambda} &= {}^{C_a}\mathbf{p}_f = \begin{bmatrix} {}^{C_a}x & {}^{C_a}y & {}^{C_a}z \end{bmatrix}^\top
\end{aligned}
$$

The Jacobian with respect to the feature state is given by:

$$
\frac{\partial \mathbf{f}(\cdot)}{\partial \boldsymbol{\lambda}} = {}^{I_a}_G\mathbf{R}^\top {}^C_I\mathbf{R}^\top
$$

As the anchor pose is involved in this representation, its Jacobians are computed as:

$$
\begin{aligned}
\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{I_a}_{G}\mathbf{R}} &= -{}^{I_a}_{G}\mathbf{R}^\top \left\lfloor {}^{C}_{I}\mathbf{R}^\top ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I)\times \right\rfloor \\
\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{G}\mathbf{p}_{I_a}} &= \mathbf{I}_{3\times 3}
\end{aligned}
$$

Moreover, if performing extrinsic calibration, the following Jacobians with respect to the IMU-camera extrinsics are also needed:

$$
\begin{aligned}
\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{C}_{I}\mathbf{R}} &= -{}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \left\lfloor ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I)\times \right\rfloor \\
\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{C}\mathbf{p}_{I}} &= -{}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top
\end{aligned}
$$

Anchored Inverse Depth

In analogy to the global inverse depth case, we can employ the inverse-depth with bearing (akin to spherical coordinates) in the anchor frame, $\{{}^{I_a}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{I_a}\}$, to represent a 3D point feature:

$$
\begin{aligned}
{}^{G}\mathbf{p}_f &= \mathbf{f}(\boldsymbol{\lambda},\ {}^{I_a}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{I_a},\ {}^{C}_{I}\mathbf{R},\ {}^{C}\mathbf{p}_I) \\
&= {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I) + {}^{G}\mathbf{p}_{I_a} \\
&= {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \left(\frac{1}{\rho} \begin{bmatrix} \cos(\theta)\sin(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\phi) \end{bmatrix} - {}^{C}\mathbf{p}_I \right) + {}^{G}\mathbf{p}_{I_a} \\
\text{where} \quad \boldsymbol{\lambda} &= \begin{bmatrix} \theta & \phi & \rho \end{bmatrix}^\top
\end{aligned}
$$

The Jacobian with respect to the feature state is given by:

$$
\frac{\partial \mathbf{f}(\cdot)}{\partial \boldsymbol{\lambda}} = {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \begin{bmatrix} -\frac{1}{\rho}\sin(\theta)\sin(\phi) & \frac{1}{\rho}\cos(\theta)\cos(\phi) & -\frac{1}{\rho^2}\cos(\theta)\sin(\phi) \\ \frac{1}{\rho}\cos(\theta)\sin(\phi) & \frac{1}{\rho}\sin(\theta)\cos(\phi) & -\frac{1}{\rho^2}\sin(\theta)\sin(\phi) \\ 0 & -\frac{1}{\rho}\sin(\phi) & -\frac{1}{\rho^2}\cos(\phi) \end{bmatrix}
$$

The Jacobians with respect to the anchor pose are:

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{I_a}_{G}\mathbf{R}} = -{}^{I_a}_{G}\mathbf{R}^\top \lfloor {}^{C}_{I}\mathbf{R}^\top ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I)\times \rfloor$$

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{G}\mathbf{p}_{I_a}} = \mathbf{I}_{3\times 3}$$

The Jacobians with respect to the IMU-camera extrinsics are:

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{C}_{I}\mathbf{R}} = -{}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \lfloor ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I)\times \rfloor$$

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{C}\mathbf{p}_{I}} = -{}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top$$

Anchored Inverse Depth (MSCKF Version)

Note that a simpler version of inverse depth was used in the original MSCKF paper [12]. This representation does not have the singularity if it is represented in a camera frame the feature was measured from.

$$\begin{aligned} {}^{G}\mathbf{p}_f &= \mathbf{f}(\boldsymbol{\lambda},\ {}^{I_a}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{I_a},\ {}^{C}_{I}\mathbf{R},\ {}^{C}\mathbf{p}_I) \\ &= {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I) + {}^{G}\mathbf{p}_{I_a} \\ &= {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \left(\frac{1}{\rho} \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} - {}^{C}\mathbf{p}_I \right) + {}^{G}\mathbf{p}_{I_a} \\ \text{where} \quad \boldsymbol{\lambda} &= \begin{bmatrix} \alpha & \beta & \rho \end{bmatrix}^\top \end{aligned}$$

The Jacobian with respect to the feature state is:

$$\frac{\partial \mathbf{f}(\cdot)}{\partial \boldsymbol{\lambda}} = {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \begin{bmatrix} \frac{1}{\rho} & 0 & -\frac{1}{\rho^2}\alpha \\ 0 & \frac{1}{\rho} & -\frac{1}{\rho^2}\beta \\ 0 & 0 & -\frac{1}{\rho^2} \end{bmatrix}$$

The Jacobians with respect to the anchor state are:

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{I_a}_{G}\mathbf{R}} = -{}^{I_a}_{G}\mathbf{R}^\top \lfloor {}^{C}_{I}\mathbf{R}^\top ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I)\times \rfloor$$

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{G}\mathbf{p}_{I_a}} = \mathbf{I}_{3\times 3}$$

The Jacobians with respect to the IMU-camera extrinsics are:

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{C}_{I}\mathbf{R}} = -{}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \lfloor ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I)\times \rfloor$$

$$\frac{\partial \mathbf{f}(\cdot)}{\partial {}^{C}\mathbf{p}_{I}} = -{}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top$$

Anchored Inverse Depth (MSCKF Single Depth Version)

This feature representation is based on the MSCKF representation [12], and the the single depth from VINS-Mono [15]. As compared to the implementation in [15], we are careful about how we handle treating of the bearing of the feature. During initialization we initialize a full 3D feature and then follow that by marginalize the bearing portion of it leaving the depth in the state vector. The marginalized bearing is then fixed for all future linearizations.

Then during update, we perform nullspace projection at every timestep to remove the feature dependence on this bearing. To do so, we need at least *two* sets of UV measurements to perform this bearing nullspace operation since we loose two dimensions of the feature in the process. We can define the feature measurement function as follows:

$$\begin{aligned}
{}^{G}\mathbf{p}_f &= \mathbf{f}(\boldsymbol{\lambda},\ {}^{I_a}_{G}\mathbf{R},\ {}^{G}\mathbf{p}_{I_a},\ {}^{C}_{I}\mathbf{R},\ {}^{C}\mathbf{p}_I) \\
&= {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top ({}^{C_a}\mathbf{p}_f - {}^{C}\mathbf{p}_I) + {}^{G}\mathbf{p}_{I_a} \\
&= {}^{I_a}_{G}\mathbf{R}^\top {}^{C}_{I}\mathbf{R}^\top \left(\frac{1}{\rho}\hat{\mathbf{b}} - {}^{C}\mathbf{p}_I\right) + {}^{G}\mathbf{p}_{I_a} \\
\text{where} \quad \boldsymbol{\lambda} &= [\rho]
\end{aligned}$$

In the above case we have defined a bearing $\hat{\mathbf{b}}$ which is the marginalized bearing of the feature after initialization. After collecting two measurement, we can nullspace project to remove the Jacobian in respect to this bearing variable.

Generated by doxygen 1.8.19 and m.css.

Measurement Update Derivations » Delayed Feature Initialization

We describe a method of delayed initialization of a 3D point feature as in Visual-Inertial Odometry on Resource-Constrained Systems [10]. Specifically, given a set of measurements involving the state $\mathbf{x}$ and a new feature $\mathbf{f}$, we want to optimally and efficiently initialize the feature.

$$\mathbf{z}_i = \mathbf{h}_i\left(\mathbf{x}, \mathbf{f}\right) + \mathbf{n}_i$$

In general, we collect more than the minimum number of measurements at different times needed for initialization (i.e. delayed). For example, although in principle we need two monocular images to initialize a 3D point feature, we often collect more than two images in order to obtain better initialization. To process all collected measurements, we stack them and perform linearization around some linearization points (estimates) denoted by $\hat{\mathbf{x}}$ and $\hat{\mathbf{f}}$:

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_m \end{bmatrix} = \mathbf{h}\left(\mathbf{x}, \mathbf{f}\right) + \mathbf{n}$$

$$\Rightarrow \quad \mathbf{r} = \mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}, \hat{f}) = \mathbf{H}_x\tilde{\mathbf{x}} + \mathbf{H}_f\tilde{\mathbf{f}} + \mathbf{n}$$

To efficiently compute the resulting augmented covariance matrix, we perform Givens rotations to zero-out rows in $\mathbf{H}_f$ with indices larger than the dimension of $\tilde{\mathbf{f}}$, and apply the same Givens rotations to $\mathbf{H}_x$ and $\mathbf{r}$. As a result of this operation, we have the following linear system:

$$\begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{x1} \\ \mathbf{H}_{x2} \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} \mathbf{H}_{f1} \\ \mathbf{0} \end{bmatrix} \tilde{\mathbf{f}} + \begin{bmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \end{bmatrix}$$

Note that the bottom system essentially is corresponding to the nullspace projection as in the MSCKF update and $\mathbf{H}_{f1}$ is generally invertible. Note also that we assume the measurement noise is isotropic; otherwise, we should first perform whitening to make it isotropic, which would save significant computations. So, if the original measurement noise covariance $\mathbf{R} = \sigma^2\mathbf{I}_m$ and the dimension of $\tilde{\mathbf{f}}$ is n, then the inferred measurement noise covariance will be $\mathbf{R}_1 = \sigma^2\mathbf{I}_n$ and $\mathbf{R}_2 = \sigma^2\mathbf{I}_{m-n}$.

Now we can directly solve for the error of the new feature based on the first subsystem:

$$\tilde{\mathbf{f}} = \mathbf{H}_{f1}^{-1}(\mathbf{r}_1 - \mathbf{n}_1 - \mathbf{H}_x\tilde{\mathbf{x}})$$
$$\Rightarrow \mathbb{E}[\tilde{\mathbf{f}}] = \mathbf{H}_{f1}^{-1}(\mathbf{r}_1)$$

where we assumed noise and state error are zero mean. We can update $\hat{\mathbf{f}}$ with this correction by $\hat{\mathbf{f}} + \mathbb{E}[\tilde{\mathbf{f}}]$. Note that this is equivalent to a Gauss Newton step for solving the corresponding maximum likelihood estimation (MLE) formed by fixing the estimate of $\mathbf{x}$ and optimizing over the value of $\hat{\mathbf{f}}$, and should therefore be zero if we used such an optimization to come up with our initial estimate for the new variable.

We now can compute the covariance of the new feature as follows:

$$\begin{aligned}\mathbf{P}_{ff} &= \mathbb{E}\Big[(\tilde{\mathbf{f}} - \mathbb{E}[\tilde{\mathbf{f}}])(\tilde{\mathbf{f}} - \mathbb{E}[\tilde{\mathbf{f}}])^\top\Big] \\ &= \mathbb{E}\Big[(\mathbf{H}_{f1}^{-1}(-\mathbf{n}_1 - \mathbf{H}_{x1}\tilde{\mathbf{x}}))(\mathbf{H}_{f1}^{-1}(-\mathbf{n}_1 - \mathbf{H}_{x1}\tilde{\mathbf{x}}))^\top\Big] \\ &= \mathbf{H}_{f1}^{-1}(\mathbf{H}_{x1}\mathbf{P}_{xx}\mathbf{H}_{x1}^\top + \mathbf{R}_1)\mathbf{H}_{f1}^{-\top}\end{aligned}$$

and the cross correlation can be computed as:

$$\begin{aligned}\mathbf{P}_{xf} &= \mathbb{E}\Big[(\tilde{\mathbf{x}})(\tilde{\mathbf{f}} - \mathbb{E}[\tilde{\mathbf{f}}])^\top\Big] \\ &= \mathbb{E}\Big[(\tilde{\mathbf{x}})(\mathbf{H}_{f1}^{-1}(-\mathbf{n}_1 - \mathbf{H}_{x1}\tilde{\mathbf{x}}))^\top\Big] \\ &= -\mathbf{P}_{xx}\mathbf{H}_{x1}^\top\mathbf{H}_{f1}^{-\top}\end{aligned}$$

These entries can then be placed in the correct location for the covariance. For example when initializing a new feature to the end of the state, the augmented covariance would be:

$$\mathbf{P}_{aug} = \begin{bmatrix}\mathbf{P}_{xx} & \mathbf{P}_{xf} \\ \mathbf{P}_{xf}^\top & \mathbf{P}_{ff}\end{bmatrix}$$

Note that this process does not update the estimate for $\mathbf{x}$. However, after initialization, we can then use the second system, $\mathbf{r}_2$, $\mathbf{H}_{x2}$, and $\mathbf{n}_2$ to update our new state through a standard EKF update (see Linear Measurement Update section).

Generated by doxygen 1.8.19 and m.css.

Measurement Update Derivations » MSCKF Nullspace Projection

In the standard EKF update, given a linearized measurement error (or residual) equation:

$$\tilde{\mathbf{z}}_{m,k} \simeq \mathbf{H}_x \tilde{\mathbf{x}}_k + \mathbf{H}_f {}^G\tilde{\mathbf{p}}_f + \mathbf{n}_k$$

we naively need to compute the residual covariance matrix $\mathbf{P}_{zz}$ as follows:

$$\begin{aligned}
\mathbf{P}_{zz} &= \mathbb{E}\left[\tilde{\mathbf{z}}_{m,k}m, k\tilde{\mathbf{z}}_{m,k}^\top\right] \\
&= \mathbb{E}\left[(\mathbf{H}_x \tilde{\mathbf{x}}_k + \mathbf{H}_f {}^G\tilde{\mathbf{p}}_f + \mathbf{n}_k)(\mathbf{H}_x \tilde{\mathbf{x}}_k + \mathbf{H}_f {}^G\tilde{\mathbf{p}}_f + \mathbf{n}_k)^\top\right] \\
&= \mathbb{E}\Big[\mathbf{H}_x \tilde{\mathbf{x}}_k \tilde{\mathbf{x}}_k^\top \mathbf{H}_x^\top + \mathbf{H}_x \tilde{\mathbf{x}}_k {}^G\tilde{\mathbf{p}}_f^\top \mathbf{H}_f^\top + {\color{red}\mathbf{H}_x \tilde{\mathbf{x}}_k \mathbf{n}_k^\top} \\
&\qquad + \mathbf{H}_f {}^G\tilde{\mathbf{p}}_f \tilde{\mathbf{x}}_k^\top \mathbf{H}_x^\top + \mathbf{H}_f {}^G\tilde{\mathbf{p}}_f {}^G\tilde{\mathbf{p}}_f^\top \mathbf{H}_f^\top + \mathbf{H}_f {}^G\tilde{\mathbf{p}}_f \mathbf{n}_k^\top \\
&\qquad + {\color{red}\mathbf{n}_k \tilde{\mathbf{x}}_k^\top \mathbf{H}_x^\top} + \mathbf{n}_k {}^G\tilde{\mathbf{p}}_f^\top \mathbf{H}_f^\top + \mathbf{n}_k \mathbf{n}_k^\top\Big] \\
&= \mathbf{H}_x \mathbb{E}\left[\tilde{\mathbf{x}}_k \tilde{\mathbf{x}}_k^\top\right] \mathbf{H}_x^\top + \mathbf{H}_x \mathbb{E}\left[\tilde{\mathbf{x}}_k {}^G\tilde{\mathbf{p}}_f^\top\right] \mathbf{H}_f^\top + \mathbf{H}_f \mathbb{E}\left[{}^G\tilde{\mathbf{p}}_f \tilde{\mathbf{x}}_k^\top\right] \mathbf{H}_x^\top + \mathbf{H}_f \mathbb{E}\left[{}^G\tilde{\mathbf{p}}_f {}^G\tilde{\mathbf{p}}_f^\top\right] \mathbf{H}_f^\top \\
&\qquad + \mathbf{H}_f \mathbb{E}\left[{}^G\tilde{\mathbf{p}}_f \mathbf{n}_k^\top\right] + \mathbb{E}\left[\mathbf{n}_k {}^G\tilde{\mathbf{p}}_f^\top\right] \mathbf{H}_f^\top + \mathbb{E}\left[\mathbf{n}_k \mathbf{n}_k^\top\right] \\
&= \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^\top + \mathbf{H}_x \mathbf{P}_{xf} \mathbf{H}_f^\top + \mathbf{H}_f \mathbf{P}_{fx} \mathbf{H}_x^\top + \mathbf{H}_f \mathbf{P}_{ff} \mathbf{H}_f^\top \\
&\qquad + \mathbf{H}_f \mathbf{P}_{fn} + \mathbf{P}_{nf} \mathbf{H}_f^\top + \mathbf{R}_d
\end{aligned}$$

However, there would be a big problem in visual-inertial odometry (VIO); that is, we do not know what the prior feature covariance and it is coupled with both the state, itself, and the noise (i.e., $\mathbf{P}_{xf}$, $\mathbf{P}_{ff}$, and $\mathbf{P}_{nf}$). This motivates the need for a method to remove the feature ${}^G\tilde{\mathbf{p}}_f$ from the linearized measurement equation (thus removing the correlation between the measurement and its error).

To this end, we start with the measurement residual function by removing the "sensitivity" to feature error we compute and apply the left nullspace of the Jacobian $\mathbf{H}_f$. We can compute it using QR decomposition as follows:

$$\mathbf{H}_f = \begin{bmatrix} \mathbf{Q_1} & \mathbf{Q_2} \end{bmatrix} \begin{bmatrix} \mathbf{R_1} \\ \mathbf{0} \end{bmatrix} = \mathbf{Q_1}\mathbf{R_1}$$

Multiplying the linearized measurement equation by the nullspace of the feature Jacobian from the left yields:

$$\begin{aligned} \tilde{\mathbf{z}}_{m,k} &\simeq \mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q_1}\mathbf{R_1}{}^G\tilde{\mathbf{p}}_f + \mathbf{n}_k \\ \Rightarrow \mathbf{Q_2}^\top\tilde{\mathbf{z}}_m &\simeq \mathbf{Q_2}^\top\mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q_2}^\top\mathbf{Q_1}\mathbf{R_1}{}^G\tilde{\mathbf{p}}_f + \mathbf{Q_2}^\top\mathbf{n}_k \\ \Rightarrow \mathbf{Q_2}^\top\tilde{\mathbf{z}}_m &\simeq \mathbf{Q_2}^\top\mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q_2}^\top\mathbf{n}_k \\ \Rightarrow \tilde{\mathbf{z}}_{o,k} &\simeq \mathbf{H}_{o,k}\tilde{\mathbf{x}}_k + \mathbf{n}_{o,k} \end{aligned}$$

where we have employed the fact that $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are orthonormal.

We now examine the dimensions of the involved matrices to appreciate the computation saving gained from this nullspace projection.

$$\begin{aligned} \text{size}(\mathbf{H}_f) &= 2n \times 3 \ \ \text{where } n \text{ is the number of uv measurements of this feature} \\ \text{size}({}^G\tilde{\mathbf{p}}_f) &= 3 \times 1 \\ \text{size}(\mathbf{H}_x) &= 2n \times 15 + 6c \ \ \text{where } c \text{ is the number of clones} \\ \text{size}(\tilde{\mathbf{x}}_k) &= 15 + 6c \times 1 \ \ \text{where } c \text{ is the number of clones} \\ \text{rank}(\mathbf{H}_f) &\leq \min(2n, 3) = 3 \ \ \text{where equality holds in most cases} \\ \text{nullity}(\mathbf{H}_f) &= \text{size}(\mathbf{x}) - \text{rank}(\mathbf{H}_f) = 2n - 3 \ \ \text{assuming full rank} \end{aligned}$$

With that, we can have the following conclusion about the sizes when the nullspace is applied:

$$
\begin{aligned}
\mathbf{Q_2}^\top\tilde{\mathbf{z}}_{m,k} &\simeq \mathbf{Q_2}^\top\mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q_2}^\top\mathbf{n}_k \\
\Rightarrow (2n-3\times 2n)(2n\times 1) &= (2n-3\times 2n)(2n\times 15+6c)(15+6c\times 1) \\
&\quad + (2n-3\times 2n)(2n\times 1)
\end{aligned}
$$

$$
\begin{aligned}
\tilde{\mathbf{z}}_{o,k} &\simeq \mathbf{H}_{o,k}\tilde{\mathbf{x}}_k + \mathbf{n}_o \\
\Rightarrow (2n-3\times 1) &= (2n-3\times 15+6c)(15+6c\times 1) + (2n-3\times 1)
\end{aligned}
$$

Finally, we perform the EKF update using the inferred measurement $\mathbf{z}_{o,k}$:

$$
\begin{aligned}
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{P}_{k|k-1}\mathbf{H}_{o,k}^\top(\mathbf{H}_{o,k}\mathbf{P}_{k|k-1}\mathbf{H}_{o,k}^\top + \mathbf{R}_o)^{-1}\tilde{\mathbf{z}}_{o,k} \\
\mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_{o,k}^\top(\mathbf{H}_{o,k}\mathbf{P}_{k|k-1}\mathbf{H}_{o,k}^\top + \mathbf{R}_o)^{-1}\mathbf{H}_{o,k}\mathbf{P}_{k|k-1}^\top
\end{aligned}
$$

where the time index (subscript) $k|k-1$ refers to the prior estimate which was denoted before by symbol $\ominus$ and $k|k$ corresponds to the posterior (or updated) estimate indicated before by $\oplus$.

Implementation

Using Eigen 3 library, we perform QR decomposition to get the nullspace. Here we know that the size of $\mathbf{Q}_1$ is a 3x3, which corresponds to the size of the 3D point feature state.

```
Eigen::ColPivHouseholderQR<Eigen::MatrixXd> qr(H_f.rows(), H_f.cols());
qr.compute(H_f);
Eigen::MatrixXd Q = qr.householderQ();
Eigen::MatrixXd Q1 = Q.block(0,0,3,3);
Eigen::MatrixXd Q2 = Q.block(0,3,Q.rows(),Q.cols()-3);
```

Generated by doxygen 1.8.19 and m.css.

Measurement Update Derivations » Measurement Compression

One of the most costly opeerations in the EKF update is the matrix multiplication. To mitigate this issue, we perform the thin QR decomposition of the measurement Jacobian after nullspace projection:

$$\mathbf{H}_{o,k} = \begin{bmatrix} \mathbf{Q_1} & \mathbf{Q_2} \end{bmatrix} \begin{bmatrix} \mathbf{R_1} \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_1 \mathbf{R}_1$$

This QR decomposition can be performed again using Givens rotations (note that this operation in general is not cheap though). We apply this QR to the linearized measurement residuals to compress measurements:

$$\begin{aligned} \tilde{\mathbf{z}}_{o,k} &\simeq \mathbf{H}_{o,k}\tilde{\mathbf{x}}_k + \mathbf{n}_o \\ \tilde{\mathbf{z}}_{o,k} &\simeq \mathbf{Q}_1\mathbf{R}_1\tilde{\mathbf{x}}_k + \mathbf{n}_o \\ \mathbf{Q_1}^\top\tilde{\mathbf{z}}_{o,k} &\simeq \mathbf{Q_1}^\top\mathbf{Q_1}\mathbf{R_1}\tilde{\mathbf{x}}_k + \mathbf{Q_1}^\top\mathbf{n}_o \\ \mathbf{Q_1}^\top\tilde{\mathbf{z}}_{o,k} &\simeq \mathbf{R_1}\tilde{\mathbf{x}}_k + \mathbf{Q_1}^\top\mathbf{n}_o \\ \Rightarrow~ \tilde{\mathbf{z}}_{n,k} &\simeq \mathbf{H}_{n,k}\tilde{\mathbf{x}}_k + \mathbf{n}_n \end{aligned}$$

As a result, the compressed measurement Jacobian will be of the size of the state, which will signficantly reduce the EKF update cost:

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{P}_{k|k-1}\mathbf{H}_{n,k}^\top(\mathbf{H}_{n,k}\mathbf{P}_{k|k-1}\mathbf{H}_{n,k}^\top + \mathbf{R}_n)^{-1}\tilde{\mathbf{z}}_{n,k} \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_{n,k}^\top(\mathbf{H}_{n,k}\mathbf{P}_{k|k-1}\mathbf{H}_{n,k}^\top + \mathbf{R}_n)^{-1}\mathbf{H}_{n,k}\mathbf{P}_{k|k-1}^\top \end{aligned}$$

Generated by doxygen 1.8.19 and m.css.

Measurement Update Derivations » Zero Velocity Update

The key idea of the zero velocity update (ZUPT) is to allow for the system to reduce its uncertainty leveraging motion knowledge (i.e. leverage the fact that the system is stationary). This is of particular importance in cases where we have a monocular system without any temporal SLAM features. In this case, if we are stationary we will be unable to triangulate features and thus will be unable to update the system. This can be avoided by either using a stereo system or temporal SLAM features. One problem that both of these don't solve is the issue of dynamic environmental objects. In a typical autonomous car scenario the sensor system will become stationary at stop lights in which dynamic objects, such as other cars crossing the intersection, can quickly corrupt the system. A zero velocity update and skipping feature tracking can address these issues if we are able to classify the cases where the sensor system is at rest.

Constant Velocity Synthetic Measurement

To perform update, we create a synthetic "measurement" which says that the current **true** acceleration and angular velocity is zero. As compared to saying the velocity is zero, we can model the uncertainty of these measurements based on the readings from our inertial measurement unit.

$$\begin{aligned}\mathbf{a} &= \mathbf{0}\\ \boldsymbol{\omega} &= \mathbf{0}\end{aligned}$$

It is important to realize this is not strictly enforcing zero velocity, but really a constant velocity. This means we can have a false detection at constant velocity times (zero acceleration), but this can be easily addressed by a velocity magnitude check. We have the following measurement equation relating this above synthetic "measurement" to the currently recorded inertial readings:

$$\begin{aligned}\mathbf{a} &= \mathbf{a}_m - \mathbf{b}_a - {}^{I_k}_{G}\mathbf{R}^{G}\mathbf{g} - \mathbf{n}_a\\ \boldsymbol{\omega} &= \boldsymbol{\omega}_m - \mathbf{b}_g - \mathbf{n}_g\end{aligned}$$

It is important to note that here our actual measurement is the true $\mathbf{a}$ and $\boldsymbol{\omega}$ and thus we will have the following residual where we will subtract the synthetic "measurement" and our measurement function:

$$\tilde{\mathbf{z}} = \begin{bmatrix}\mathbf{a} - \left(\mathbf{a}_m - \mathbf{b}_a - {}^{I_k}_{G}\mathbf{R}^{G}\mathbf{g} - \mathbf{n}_a\right)\\ \boldsymbol{\omega} - \left(\boldsymbol{\omega}_m - \mathbf{b}_g - \mathbf{n}_g\right)\end{bmatrix} = \begin{bmatrix}-\left(\mathbf{a}_m - \mathbf{b}_a - {}^{I_k}_{G}\mathbf{R}^{G}\mathbf{g} - \mathbf{n}_a\right)\\ -\left(\boldsymbol{\omega}_m - \mathbf{b}_g - \mathbf{n}_g\right)\end{bmatrix}$$

Where we have the following Jacobians in respect to our state:

$$\frac{\partial \tilde{\mathbf{z}}}{\partial {}^{I_k}_{G}\mathbf{R}} = -\left\lfloor {}^{I_k}_{G}\mathbf{R}^{G}\mathbf{g}\times \right\rfloor$$

$$\frac{\partial \tilde{\mathbf{z}}}{\partial \mathbf{b}_a} = \frac{\partial \tilde{\mathbf{z}}}{\partial \mathbf{b}_g} = -\mathbf{I}_{3\times 3}$$

Zero Velocity Detection

Zero velocity detection in itself is a challenging problem which has seen many different works tried to address this issue [19], [16], [2]. Most works boil down to simple thresholding and the approach is to try to determine the optimal threshold which allows for the best classifications of zero velocity update (ZUPT) portion of the trajectories. There have been other works, [19] and [16], which have looked at more complicated methods and try to address the issue that this threshold can be dependent on the type of different motions (such as running vs walking) and characteristics of the platform which the sensor is mounted on (we want to ignore vehicle engine vibrations and other non-essential observed vibrations).

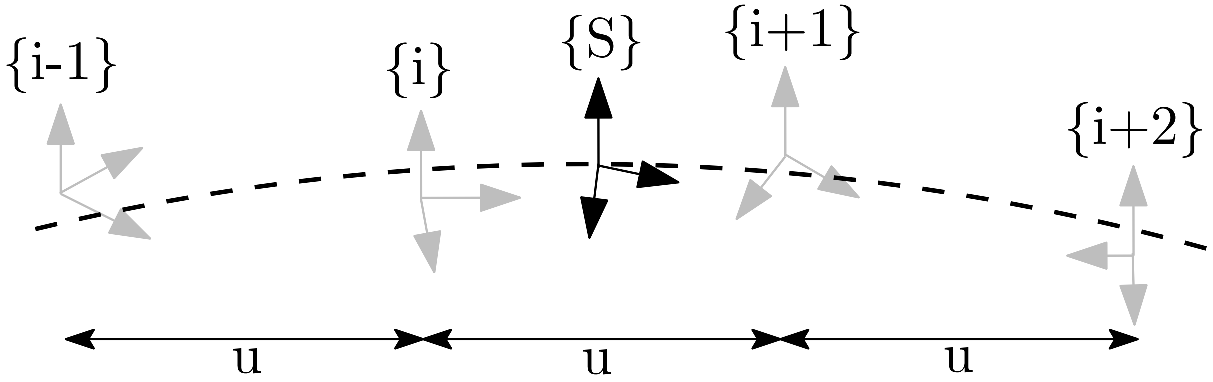
We approach this detection problem based on tuning of a χ^2, chi-squared, thresholding based on the measurement model above. It is important to note that we also have a velocity magnitude check which is aimed at preventing constant velocity cases which have non-zero magnitude. More specifically, we perform the following threshold check to see if we are current at zero velocity:

$$\tilde{\mathbf{z}}^\top(\mathbf{H}\mathbf{P}\mathbf{H}^\top + \mathbf{R})^{-1}\tilde{\mathbf{z}} < \chi^2$$

We found that in the real world experiments, typically the inertial measurement noise $\mathbf{R}$ needs to be inflated by 50-100 times to allow for proper detection. This can hint that we are using overconfident inertial noises, or that there are additional frequencies (such as the vibration of motors) which inject additional noises.

Visual-Inertial Simulator

B-Spline Interpolation



At the center of the simulator is an $\mathbb{SE}(3)$ b-spline which allows for the calculation of the pose, velocity, and accelerations at any given timestep along a given trajectory. We follow the work of Mueggler et al. [13] and Patron et al. [14] in which given a series of uniformly distributed "control points" poses the pose $\{S\}$ at a given timestep t_s can be interpolated by:

$$ {}^G_S\mathbf{T}(u(t_s)) = {}^G_{i-1}\mathbf{T}\ \mathbf{A}_0\ \mathbf{A}_1\ \mathbf{A}_2 $$

$$ \mathbf{A}_j = \exp\Big(B_j(u(t))\ {}^{i-1+j}_{i+j}\mathbf{\Omega}\Big) $$
$$ {}^{i-1}_{i}\mathbf{\Omega} = \log\Big({}^G_{i-1}\mathbf{T}^{-1}\ {}^G_i\mathbf{T}\Big) $$

$$ B_0(u(t)) = \frac{1}{3!}\ (5 + 3u - 3u^2 + u^3) $$
$$ B_1(u(t)) = \frac{1}{3!}\ (1 + 3u + 3u^2 - 2u^3) $$
$$ B_2(u(t)) = \frac{1}{3!}\ (u^3) $$

where $u(t_s) = (t_s - t_i)/(t_{i+1} - t_i)$, $\exp(\cdot)$, $\log(\cdot)$ are the $\mathbb{SE}(3)$ matrix exponential **ov_core::exp_se3** and logarithm **ov_core::log_se3**. The frame notations can be seen in the above figure and we refer the reader to the **ov_core::BsplineSE3** class for more details. The above equation can be interpretative as compounding the fractions portions of the bounding poses to the first pose ${}^G_{i-1}\mathbf{T}$. From this above equation, it is simple to take the derivative in respect to time, thus allowing the computation of the velocity and acceleration at any point.

The only needed input into the simulator is a pose trajectory which we will then uniformly sample to construct control points for this spline. This spline is then used to both generate the inertial measurements while also providing the pose information needed to generate visual-bearing measurements.

Inertial Measurements

To incorporate inertial measurements from a IMU sensor, we can leverage the continuous nature and C^2-continuity of our cubic B-spline. We can define the sensor measurement from a IMU as follows:

$$ {}^I\boldsymbol{\omega}_m(t) = {}^I\boldsymbol{\omega}(t) + \mathbf{b}_\omega + \mathbf{n}_\omega $$
$$ {}^I\mathbf{a}_m(t) = {}^I\mathbf{a}(t) + {}^{I(t)}_G\mathbf{R}\,{}^G\mathbf{g} + \mathbf{b}_a + \mathbf{n}_a $$
$$ \dot{\mathbf{b}}_\omega = \mathbf{n}_{wg} $$
$$ \dot{\mathbf{b}}_a = \mathbf{n}_{wa} $$

where each measurement is corrupted with some white noise and random-walk bias. To obtain the true measurements from our $\mathbb{SE}(3)$ b-spline we can do the following:

$$
\begin{aligned}
{}^I\boldsymbol{\omega}(t) &= \mathrm{vee}\Big({}^G_I\mathbf{R}(u(t))^\top {}^G_I\dot{\mathbf{R}}(u(t))\Big) \\
{}^I\mathbf{a}(t) &= {}^G_I\mathbf{R}(u(t))^\top {}^G\ddot{\mathbf{p}}_I(u(t))
\end{aligned}
$$

where $\mathrm{vee}(\cdot)$ returns the vector portion of the skew-symmetric matrix (see **ov_core::vee**). These are then corrupted using the random walk biases and corresponding white noises. For example we have the following:

$$
\begin{aligned}
\omega_m(t) &= \omega(t) + b_\omega(t) + \sigma_w \frac{1}{\sqrt{\Delta t}}\mathrm{gennoise}(0,1) \\
b_\omega(t+\Delta t) &= b_\omega(t) + \sigma_{wg}\sqrt{\Delta t}\ \mathrm{gennoise}(0,1) \\
t &= t + \Delta t
\end{aligned}
$$

Note that this is repeated per-scalar value as compared to the vector and identically for the accelerometer readings. The $\mathrm{gennoise}(m,v)$ function generates a random scalar float with mean *m* and variance *v*. The Δt is our sensor sampling rate that we advance time forward with.

Visual-Bearing Measurement

The first step that we perform after creating the b-spline trajectory is the generation of a map of point features. To generate these features, we increment along the spline at a fixed interval and ensure that all cameras see enough features in the map. If there are not enough features in the given frame, we generate new features by sending random rays from the camera out and assigning a random depth. This feature is then added to the map so that it can be projected into future frames.

After the map generation phase, we generate feature measurements by projecting them into the current frame. Projected features are limited to being with-in the field of view of the camera, in front of the camera, and close in distance. Pixel noise can be directly added to the raw pixel values.

Generated by doxygen 1.8.19 and m.css.

System Evaluation

The goal of our evaluation is to ensure fair comparison to other methods and our own. The actual metrics we use can be found on the **Filter Evaluation Metrics** page. Using our metrics we wish to provide insight into *why* our method does better and in what ways (as no method will outperform in all aspects). Since we are also interested in applying the systems to real robotic applications, the realtime performance is also a key metric we need to investigate. Timing of different system components is also key to removing bottlenecks and seeing where performance improvements or estimator approximations might help reduce complexity.

The key metrics we are interested in evaluating are the following:

- Absolute Trajectory Error (ATE)
- Relative Pose Error (RPE)
- Root Mean Squared Error (RMSE)
- Normalized Estimation Error Squared (NEES)
- Estimator Component Timing
- System Hardware Usage (memory and computation)

System Evaluation Guides

- **Filter Evaluation Metrics** — Definitions of different metrics for estimator accuracy.
- **Filter Error Evaluation Methods** — Error evaluation methods for evaluating system performance.
- **Filter Timing Analysis** — Timing of estimator components and complexity.

Generated by doxygen 1.8.19 and m.css.

System Evaluation » Filter Evaluation Metrics

Absolute Trajectory Error (ATE)

The Absolute Trajectory Error (ATE) is given by the simple difference between the estimated trajectory and groundtruth after it has been aligned so that it has minimal error. First the "best" transform between the groundtruth and estimate is computed, afterwhich the error is computed at every timestep and then averaged. We recommend reading Zhang and Scaramuzza [20] paper for details. For a given dataset with N runs of the same algorithm with K pose measurements, we can compute the following for an aligned estimated trajectory $\hat{\mathbf{x}}^+$:

$$e_{ATE} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{K}\sum_{k=1}^{K}||\mathbf{x}_{k,i} \boxminus \hat{\mathbf{x}}^{+}_{k,i}||_2^2}$$

Relative Pose Error (RPE)

The Relative Pose Error (RPE) is calculated for segments of the dataset and allows for introspection of how localization solutions drift as the length of the trajectory increases. The other key advantage over ATE error is that it is less sensitive to jumps in estimation error due to sampling the trajectory over many smaller segments. This allows for a much fairer comparision of methods and is what we recommend all authors publish results for. We recommend reading Zhang and Scaramuzza [20] paper for details. We first define a set of segment lengths $\mathcal{D} = [d_1,\ d_2, \cdots,\ d_V]$ which we compute the relative error for. We can define the relative error for a trajectory split into D_i segments of d_i length as follows:

$$\tilde{\mathbf{x}}_r = \mathbf{x}_k \boxminus \mathbf{x}_{k+d_i}$$

$$e_{rpe,d_i} = \frac{1}{D_i}\sum_{k=1}^{D_i}||\tilde{\mathbf{x}}_r \boxminus \hat{\tilde{\mathbf{x}}}_r||_2^2$$

Root Mean Squared Error (RMSE)

When evaluating a system on a *single* dataset is the Root Mean Squared Error (RMSE) plots. This plots the RMSE at every timestep of the trajectory and thus can provide insight into timesteps where the estimation performance suffers. For a given dataset with N runs of the same algorithm we can compute the following at each timestep k:

$$e_{rmse,k} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x}_{k,i} \boxminus \hat{\mathbf{x}}_{k,i}||_2^2}$$

Normalized Estimation Error Squared (NEES)

Normalized Estimation Error Squared (NEES) is a standard way to characterize if the estimator is being consistent or not. In general NEES is just the normalized error which should be the degrees of freedoms of the state variables. Thus in the case of position and orientation we should get a NEES of three at every timestep. To compute the average NEES for a dataset with N runs of the same algorithm we can compute the following at each timestep k :

$$e_{nees,k} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_{k,i} \boxminus \hat{\mathbf{x}}_{k,i})^\top \mathbf{P}_{k,i}^{-1}(\mathbf{x}_{k,i} \boxminus \hat{\mathbf{x}}_{k,i})$$

Single Run Consistency

When looking at a *single run* and wish to see if the system is consistent it is interesting to look a its error in respect to its estimated uncertainty. Specifically we plot the error and the estimator 3σ bound. This provides insight into if the estimator is becoming over confident at certain timesteps. Note this is for each component of the state, thus we need to plot x,y,z and orientation independently. We can directly compute the error at timestep k:

$$\begin{gathered}\mathbf{e}_k = \mathbf{x}_k \boxminus \hat{\mathbf{x}}_k \\ \text{where } \ \mathbf{e}_k \sim \mathcal{N}(0, \mathbf{P})\end{gathered}$$

Generated by doxygen 1.8.19 and m.css.

System Evaluation » Filter Error Evaluation Methods

Installation Warning

If you plan to use the included plotting from the cpp code, you will need to make sure that you have matplotlib and python 2.7 installed. We use the to matplotlib-cpp to call this external library and generate the desired figures. Please see **Additional Evaluation Requirements** for more details on the exact install.

Collection

The first step in any evaluation is to first collect the estimated trajectory of the proposed systems. Since we are interested in robotic application of our estimators we want to record the estimate at the current timestep (as compared to a "smoothed" output or one that includes loop-closures from future timesteps). Within the ROS framework, this means that we just need to publish the current estimate at the current timestep. We recommend using the following **ov_eval::Recorder** utility for recording the estimator output directly into a text file. Works with topics of type `PoseWithCovarianceStamped`, `PoseStamped`, `TransformStamped`, and `Odometry`.

```
<node name="recorder_estimate" pkg="ov_eval" type="pose_to_file" output="screen">
    <param name="topic"      type="str" value="/ov_msckf/poseimu" />
    <param name="topic_type" type="str" value="PoseWithCovarianceStamped" />
    <param name="output"     type="str" value="/home/user/data/traj_log.txt" />
</node>
```

Transformation

We now need to ensure both our estimated trajectory and groundtruth are in the correct formats for us to read in. We need things to be in the RPE text file format (i.e. time(s), px, py, pz, qx, qy, qz, qw). We have a nice helper script that will transform ASL / EuRoC groundtruth files to the correct format. By default the EuRoC groundtruth has the timestamp in nanoseconds and the quaternion is in an incorrect order (i.e. time(ns), px, py, pz, qw, qx, qy, qz). A user can either process all CSV files in a given folder, or just a specific one.

```
rosrun ov_eval format_convert folder/path/
rosrun ov_eval format_convert file.csv
```

In addition we have a specific folder structure that is assumed. We store trajectories by first their algorithm name and then a folder for each dataset this algorithm was run on. The folder names of the datasets need to match the groundtruth trajectory files which should be in their own separate folder. Please see the example recorded datasets for how to structure your folders.

```
truth/
    dateset_name_1.txt
    dateset_name_2.txt
algorithms/
    open_vins/
        dataset_name_1/
            run1.txt
            run2.txt
            run3.txt
        dataset_name_2/
            run1.txt
            run2.txt
            run3.txt
    okvis_stereo/
        dataset_name_1/
            run1.txt
            run2.txt
            run3.txt
        dataset_name_2/
            run1.txt
            run2.txt
            run3.txt
    vins_mono/
        dataset_name_1/
            run1.txt
            run2.txt
            run3.txt
        dataset_name_2/
            run1.txt
            run2.txt
            run3.txt
```

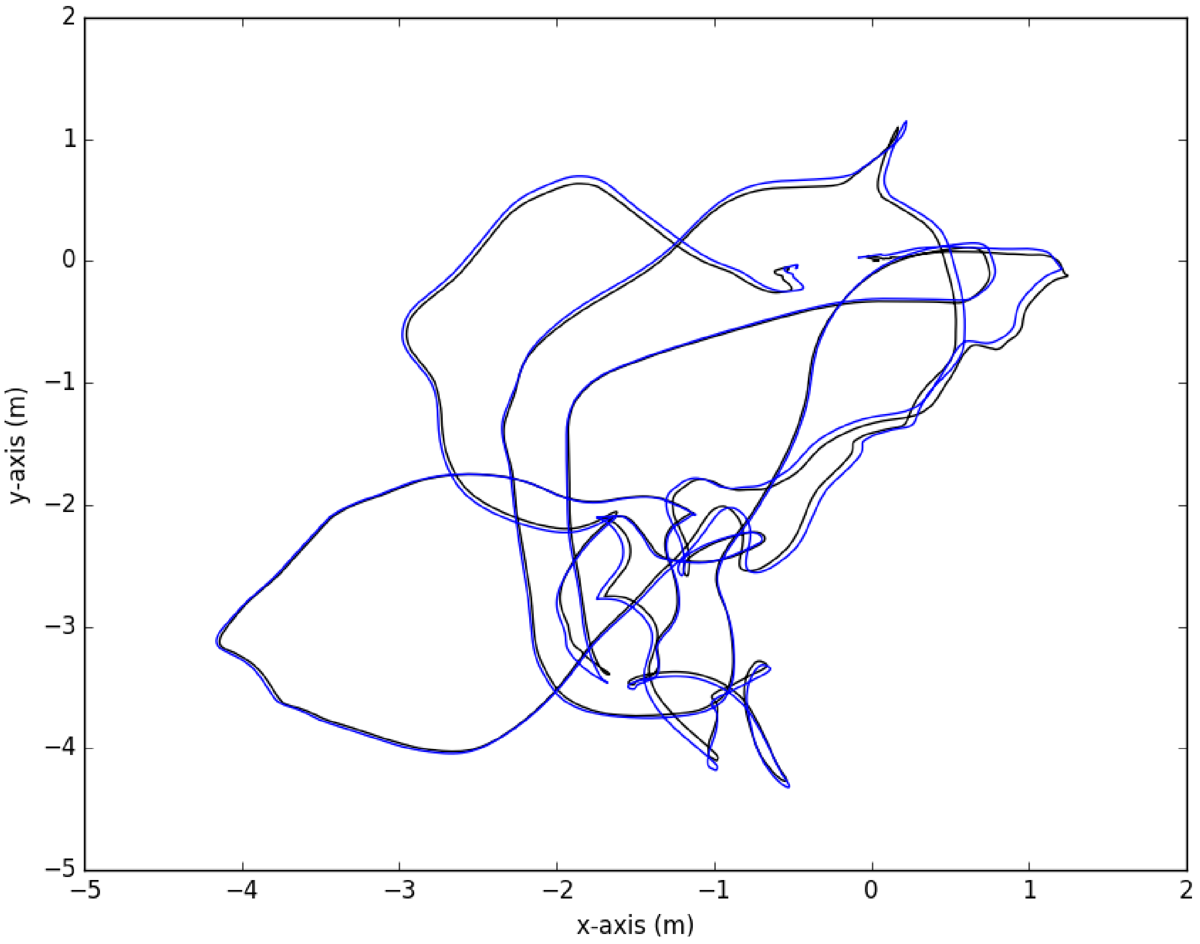
Processing & Plotting

Now that we have our data recorded and in the correct format we can now work on processing and plotting it. In the next few sections we detail how to do this for absolute trajectory error, relative pose error, normalized estimation error squared, and bounded root mean squared error plots. We will first process the data into a set of output text files which a user can then use to plot the results in their program or language of choice. The align mode of all the following commands can be of type `posyaw`, `posyawsingle`, `se3`, `se3single`, `sim3`, and `none`.

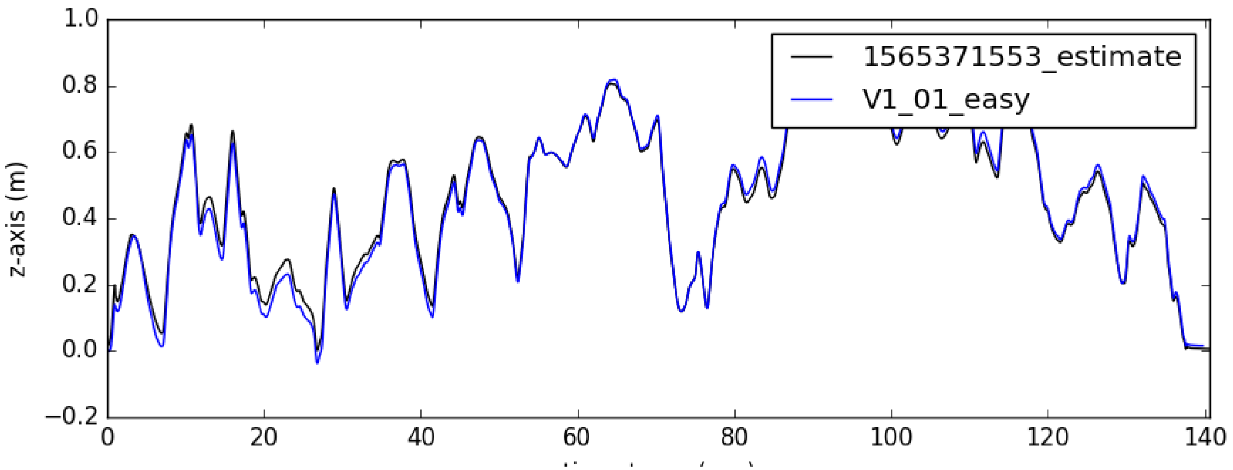
Script "plot_trajectories"

To plot the data we can us the following command which will plot a 2d xy and z-time position plots. It will use the filename as the name in the legend, so you can change that to change the legend or edit the code.

```
rosrun ov_eval plot_trajectories <align_mode> <file_gt.txt> ... <file_est9.txt>
rosrun ov_eval plot_trajectories posyaw 1565371553_estimate.txt truths/V1_01_easy.txt
```



2
1
0
−1
−2
−3
−4
−5
y-axis (m)
−5
−4
−3
−2
−1
0
1
2
x-axis (m)



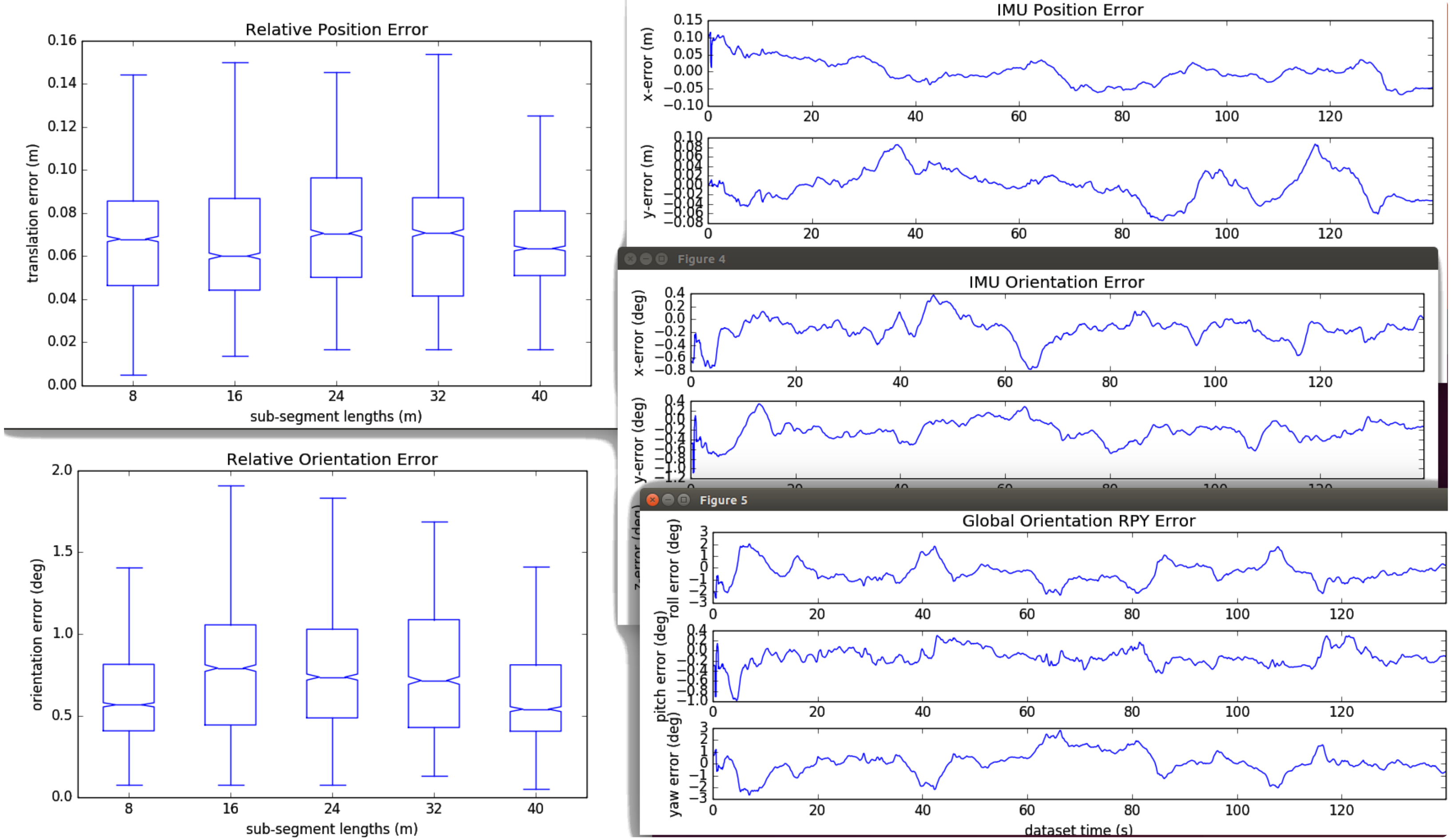
Script "error_singlerun"

The single run script will plot statistics and also 3 σ bounds if available. One can use this to see consistency of the estimator or debug how the current run has gone. It also reports to console the average RMSE and RPE values for this run along with the number of samples. To change the RPE distances you will need to edit the code currently.

```
rosrun ov_eval error_singlerun <align_mode> <file_gt.txt> <file_est.txt>
rosrun ov_eval error_singlerun posyaw 1565371553_estimate.txt truths/V1_01_easy.txt
```

Example output:

```
[ INFO] [1583422165.069376426]: [COMP]: 2813 poses in 1565371553_estimate => length of 57.36 meters
[ INFO] [1583422165.091423722]: ======================================
[ INFO] [1583422165.091438299]: Absolute Trajectory Error
[ INFO] [1583422165.091445338]: ======================================
[ INFO] [1583422165.091453099]: rmse_ori = 0.677 | rmse_pos = 0.055
[ INFO] [1583422165.091459679]: mean_ori = 0.606 | mean_pos = 0.051
[ INFO] [1583422165.091466321]: min_ori  = 0.044 | min_pos  = 0.001
[ INFO] [1583422165.091474211]: max_ori  = 1.856 | max_pos  = 0.121
[ INFO] [1583422165.091481730]: std_ori  = 0.302 | std_pos  = 0.021
[ INFO] [1583422165.127869924]: ======================================
[ INFO] [1583422165.127891080]: Relative Pose Error
[ INFO] [1583422165.127898322]: ======================================
[ INFO] [1583422165.127908551]: seg 8 - median_ori = 0.566 | median_pos = 0.068 (2484 samples)
[ INFO] [1583422165.127919603]: seg 16 - median_ori = 0.791 | median_pos = 0.060 (2280 samples)
[ INFO] [1583422165.127927646]: seg 24 - median_ori = 0.736 | median_pos = 0.070 (2108 samples)
[ INFO] [1583422165.127934904]: seg 32 - median_ori = 0.715 | median_pos = 0.071 (1943 samples)
[ INFO] [1583422165.127942178]: seg 40 - median_ori = 0.540 | median_pos = 0.063 (1792 samples)
```



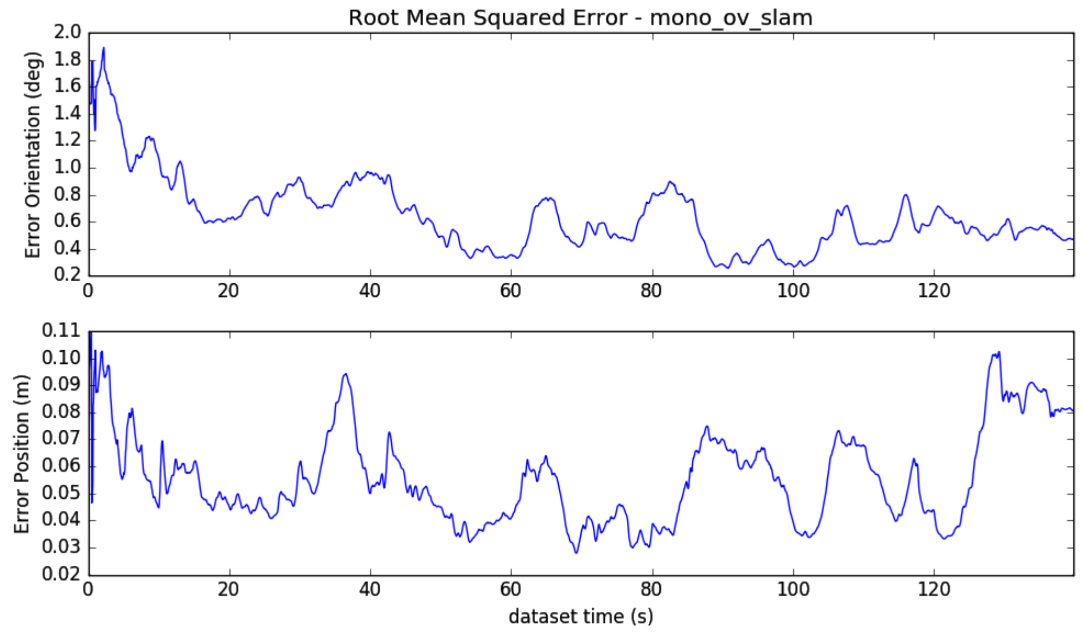
Script "error_dataset"

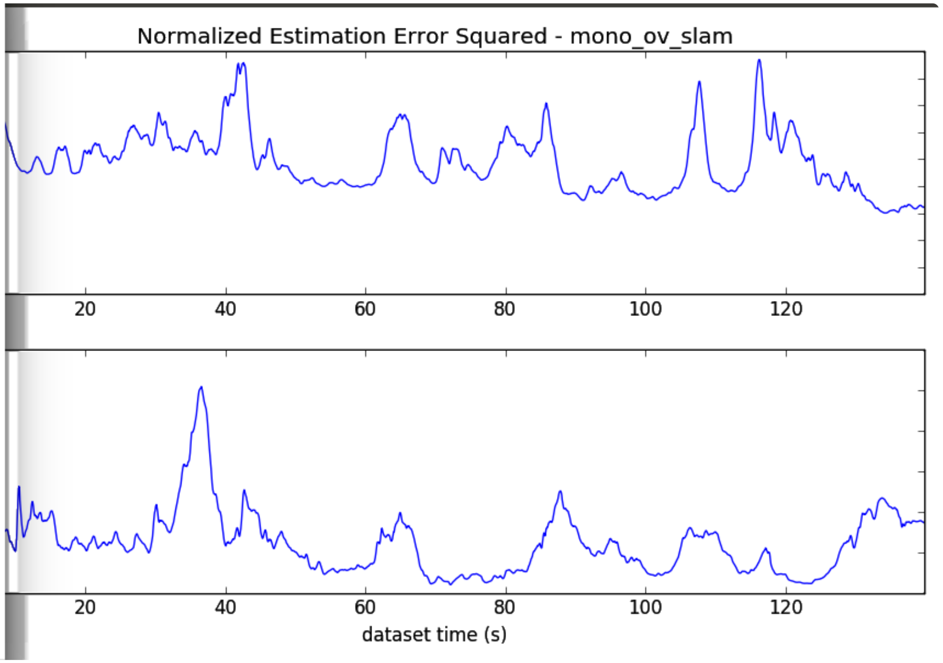
This dataset script will evaluate how a series of algorithms compare on a single dataset. Normally this is used if you just have single dataset you want to compare algorithms on, or compare a bunch variations of your algorithm to a simulated trajectory. In the console it will output the ATE 3D and 2D, along with the 3D RPE and 3D NEES for each method after it performs alignment. To change the RPE distances you will need to edit the code currently.

```
rosrun ov_eval error_dataset <align_mode> <file_gt.txt> <folder_algorithms>
rosrun ov_eval error_dataset posyaw truths/V1_01_easy.txt algorithms/
```

Example output:

```
[ INFO] [1583422654.333642977]: ======================================
[ INFO] [1583422654.333915102]: [COMP]: processing mono_ov_slam algorithm
[ INFO] [1583422654.362655719]: [TRAJ]: q_ESTtoGT = 0.000, 0.000, -0.129, 0.992 | p_ESTinGT = 0.978, 2.185, 0.932 | s = 1.00
....
[ INFO] [1583422654.996859432]: [TRAJ]: q_ESTtoGT = 0.000, 0.000, -0.137, 0.991 | p_ESTinGT = 0.928, 2.166, 0.957 | s = 1.00
[ INFO] [1583422655.041009388]:     ATE: mean_ori = 0.684 | mean_pos = 0.057
[ INFO] [1583422655.041031730]:     ATE: std_ori  = 0.14938 | std_pos  = 0.01309
[ INFO] [1583422655.041039552]:     ATE 2D: mean_ori = 0.552 | mean_pos = 0.053
[ INFO] [1583422655.041046362]:     ATE 2D: std_ori  = 0.17786 | std_pos  = 0.01421
[ INFO] [1583422655.044187033]:     RPE: seg 7 - mean_ori = 0.543 | mean_pos = 0.065 (25160 samples)
[ INFO] [1583422655.047047771]:     RPE: seg 14 - mean_ori = 0.593 | mean_pos = 0.060 (23470 samples)
[ INFO] [1583422655.049672955]:     RPE: seg 21 - mean_ori = 0.664 | mean_pos = 0.081 (22050 samples)
[ INFO] [1583422655.052090494]:     RPE: seg 28 - mean_ori = 0.732 | mean_pos = 0.083 (20520 samples)
[ INFO] [1583422655.054294322]:     RPE: seg 35 - mean_ori = 0.793 | mean_pos = 0.090 (18960 samples)
[ INFO] [1583422655.055676035]:     RMSE: mean_ori = 0.644 | mean_pos = 0.056
[ INFO] [1583422655.056987984]:     RMSE 2D: mean_ori = 0.516 | mean_pos = 0.052
[ INFO] [1583422655.058269163]:     NEES: mean_ori = 793.646 | mean_pos = 13.095
[ INFO] [1583422656.182660653]: ======================================
[ INFO] [1583422656.183065588]: [COMP]: processing mono_ov_vio algorithm
[ INFO] [1583422656.209545279]: [TRAJ]: q_ESTtoGT = 0.000, 0.000, -0.148, 0.989 | p_ESTinGT = 0.931, 2.169, 0.971 | s = 1.00
....
[ INFO] [1583422656.791523636]: [TRAJ]: q_ESTtoGT = 0.000, 0.000, -0.149, 0.989 | p_ESTinGT = 0.941, 2.163, 0.974 | s = 1.00
[ INFO] [1583422656.835407991]:     ATE: mean_ori = 0.639 | mean_pos = 0.076
[ INFO] [1583422656.835433475]:     ATE: std_ori  = 0.05800 | std_pos  = 0.00430
[ INFO] [1583422656.835446222]:     ATE 2D: mean_ori = 0.514 | mean_pos = 0.070
[ INFO] [1583422656.835457020]:     ATE 2D: std_ori  = 0.07102 | std_pos  = 0.00492
[ INFO] [1583422656.838656567]:     RPE: seg 7 - mean_ori = 0.614 | mean_pos = 0.092 (25160 samples)
[ INFO] [1583422656.841540191]:     RPE: seg 14 - mean_ori = 0.634 | mean_pos = 0.092 (23470 samples)
[ INFO] [1583422656.844219466]:     RPE: seg 21 - mean_ori = 0.632 | mean_pos = 0.115 (22050 samples)
[ INFO] [1583422656.846646272]:     RPE: seg 28 - mean_ori = 0.696 | mean_pos = 0.119 (20520 samples)
[ INFO] [1583422656.848862913]:     RPE: seg 35 - mean_ori = 0.663 | mean_pos = 0.154 (18960 samples)
[ INFO] [1583422656.850321777]:     RMSE: mean_ori = 0.600 | mean_pos = 0.067
[ INFO] [1583422656.851673985]:     RMSE 2D: mean_ori = 0.479 | mean_pos = 0.060
[ INFO] [1583422656.853037942]:     NEES: mean_ori = 625.447 | mean_pos = 10.629
[ INFO] [1583422658.194763413]: ======================================
....
```





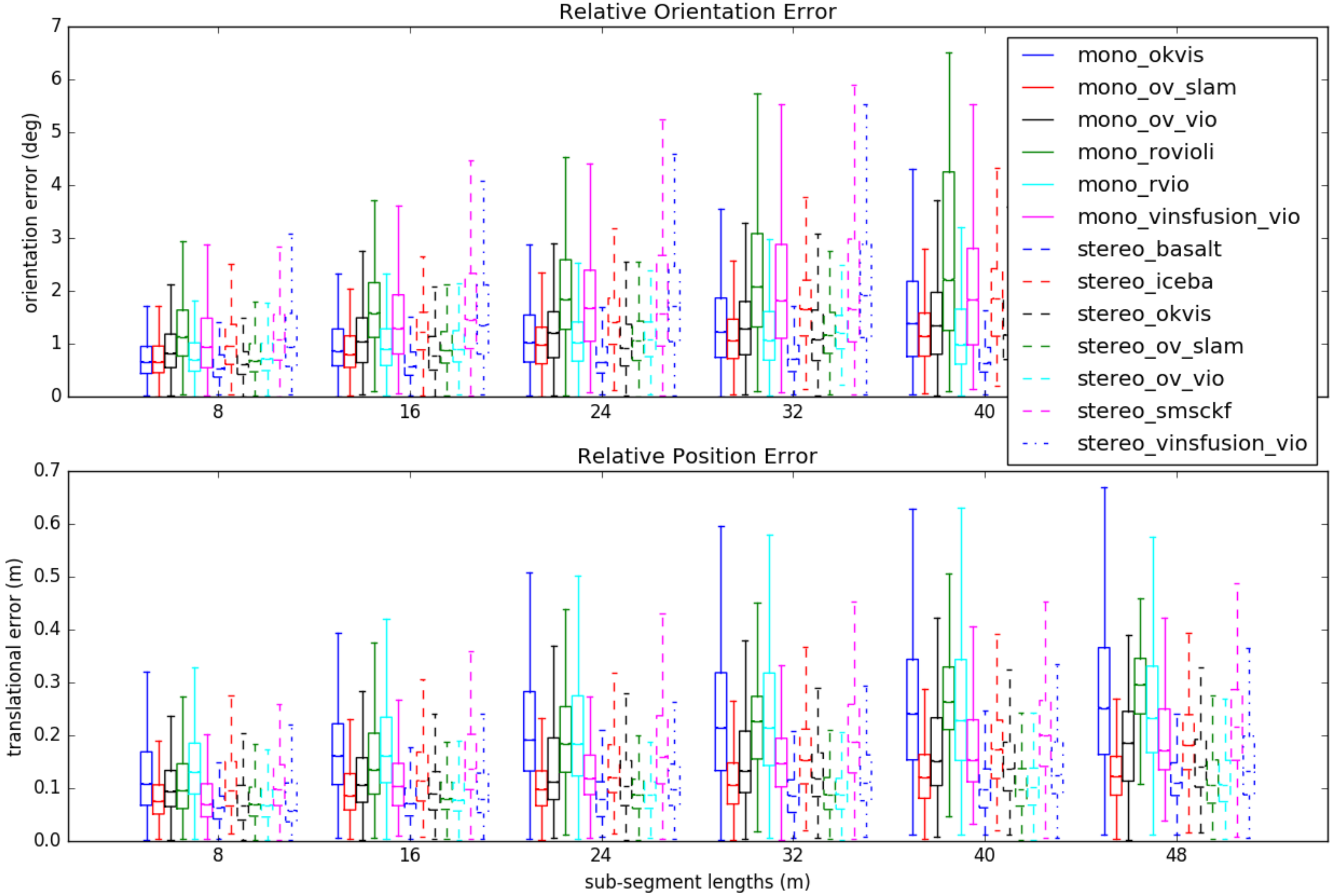
Script "error_comparison"

This compares all methods to each other on a series of datasets. For example, you run a bunch of methods on all the EurocMav datasets and then want to compare. This will do the RPE over all trajectories, and an ATE for each dataset. It will print the ATE and RPE for each method on each dataset in the console. Then following the **Filter Evaluation Metrics**, these are averaged over all the runs and datasets. Finally at the end it outputs a nice latex table which can be directly used in a paper. To change the RPE distances you will need to edit the code currently.

```
rosrun ov_eval error_comparison <align_mode> <folder_groundtruth> <folder_algorithms>
rosrun ov_eval error_comparison posyaw truths/ algorithms/
```

Example output:

```
[ INFO] [1583425216.054023187]: [COMP]: 2895 poses in V1_01_easy.txt => length of 58.35 meters
[ INFO] [1583425216.092355692]: [COMP]: 16702 poses in V1_02_medium.txt => length of 75.89 meters
[ INFO] [1583425216.133532429]: [COMP]: 20932 poses in V1_03_difficult.txt => length of 78.98 meters
[ INFO] [1583425216.179616651]: [COMP]: 22401 poses in V2_01_easy.txt => length of 36.50 meters
[ INFO] [1583425216.225299463]: [COMP]: 23091 poses in V2_02_medium.txt => length of 83.23 meters
[ INFO] [1583425216.225660364]: ======================================
[ INFO] [1583425223.560550101]: [COMP]: processing mono_ov_vio algorithm
[ INFO] [1583425223.560632706]: [COMP]: processing mono_ov_vio algorithm => V1_01_easy dataset
[ INFO] [1583425224.236769465]: [COMP]: processing mono_ov_vio algorithm => V1_02_medium dataset
[ INFO] [1583425224.855489521]: [COMP]: processing mono_ov_vio algorithm => V1_03_difficult dataset
[ INFO] [1583425225.659183593]: [COMP]: processing mono_ov_vio algorithm => V2_01_easy dataset
[ INFO] [1583425226.442217424]: [COMP]: processing mono_ov_vio algorithm => V2_02_medium dataset
[ INFO] [1583425227.366004330]: ======================================
....
[ INFO] [1583425261.724448413]: ============================================
[ INFO] [1583425261.724469372]: ATE LATEX TABLE
[ INFO] [1583425261.724481841]: ============================================
 & \textbf{V1\_01\_easy} & \textbf{V1\_02\_medium} & \textbf{V1\_03\_difficult}
 & \textbf{V2\_01\_easy} & \textbf{V2\_02\_medium} & \textbf{Average} \\\hline
mono\_ov\_slam & 0.699 / 0.058 & 1.675 / 0.076 & 2.542 / 0.063 & 0.773 / 0.124 & 1.538 / 0.074 & 1.445 / 0.079 \\
mono\_ov\_vio & 0.642 / 0.076 & 1.766 / 0.096 & 2.391 / 0.344 & 1.164 / 0.121 & 1.248 / 0.106 & 1.442 / 0.148 \\
....
[ INFO] [1583425261.724647970]: ============================================
[ INFO] [1583425261.724655060]: ============================================
[ INFO] [1583425261.724661046]: RPE LATEX TABLE
[ INFO] [1583425261.724666910]: ============================================
 & \textbf{8m} & \textbf{16m} & \textbf{24m} & \textbf{32m} & \textbf{40m} & \textbf{48m} \\\hline
mono\_ov\_slam & 0.661 / 0.074 & 0.802 / 0.086 & 0.979 / 0.097 & 1.061 / 0.105 & 1.145 / 0.120 & 1.289 / 0.122 \\
mono\_ov\_vio & 0.826 / 0.094 & 1.039 / 0.106 & 1.215 / 0.111 & 1.283 / 0.132 & 1.342 / 0.151 & 1.425 / 0.184 \\
....
[ INFO] [1583425262.514587296]: ============================================
```



Generated by doxygen 1.8.19 and m.css.

System Evaluation » Filter Timing Analysis

Installation Warning

If you plan to use the included plotting from the cpp code, you will need to make sure that you have matplotlib and python 2.7 installed. We use the to matplotlib-cpp to call this external library and generate the desired figures. Please see **Additional Evaluation Requirements** for more details on the exact install.

Collection

To profile the different parts of the system we record the timing information from directly inside the **ov_msckf::VioManager**. The file should be comma separated format, with the first column being the timing, and the last column being the total time (units are all in seconds). The middle columns should describe how much each component takes (whose names are extracted from the header of the csv file). You can use the bellow tools as long as you follow this format, and add or remove components as you see fit to the middle columns.

To evaluate the computational load (*not computation time*), we have a python script that leverages the psutil python package to record percent CPU and memory consumption. This can be included as an additional node in the launch file which only needs the node which you want the reported information of. This will poll the node for its percent memory, percent cpu, and total number of threads that it uses. This can be useful if you wish to compare different methods on the same platform, but doesn't make sense to use this to compare the running of the same algorithm or different algorithms *across* different hardware devices.

```
<node name="recorder_timing" pkg="ov_eval" type="pid_ros.py" output="screen">
    <param name="nodes"   type="str" value="/run_subscribe_msckf" />
    <param name="output"  type="str" value="/tmp/psutil_log.txt" />
</node>
```

It is also important to note that if the estimator has multiple nodes, you can subscribe to them all by specifying their names as a comma separated string. For example to evaluate the computation needed for VINS-Mono multi-node system we can do:

```
<node name="recorder_timing" pkg="ov_eval" type="pid_ros.py" output="screen">
    <param name="nodes"   type="str" value="/feature_tracker,/vins_estimator,/pose_graph" />
    <param name="output"  type="str" value="/tmp/psutil_log.txt" />
</node>
```

Processing & Plotting

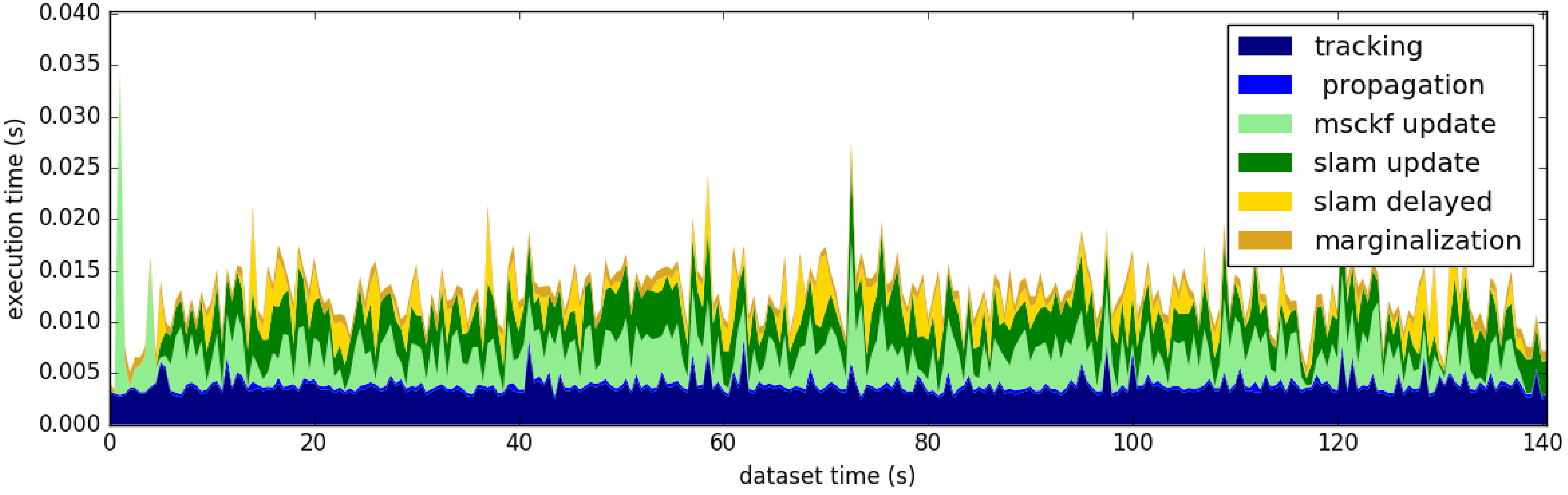
Script "timing_flamegraph"

The flame graph script looks to recreate a FlameGraph of the key components of the system. While we do not trace all functions, the key "top level" function times are recorded to file to allow for insight into what is taking the majority of the computation. The file should be comma separated format, with the first column being the timing, and the last column being the total time. The middle columns should describe how much each component takes (whose names are extracted from the header of the csv file).

```
rosrun ov_eval timing_flamegraph <file_times.txt>
rosrun ov_eval timing_flamegraph timing_mono_ethV101.txt
```

Example output:

```
[TIME]: loaded 2817 timestamps from file (7 categories)!!
mean_time = 0.0037 | std = 0.0011 | 99th = 0.0063  | max = 0.0254 (tracking)
mean_time = 0.0004 | std = 0.0001 | 99th = 0.0006  | max = 0.0014 ( propagation)
mean_time = 0.0032 | std = 0.0022 | 99th = 0.0083  | max = 0.0309 (msckf update)
mean_time = 0.0034 | std = 0.0013 | 99th = 0.0063  | max = 0.0099 (slam update)
mean_time = 0.0012 | std = 0.0017 | 99th = 0.0052  | max = 0.0141 (slam delayed)
mean_time = 0.0009 | std = 0.0003 | 99th = 0.0015  | max = 0.0027 (marginalization)
mean_time = 0.0128 | std = 0.0035 | 99th = 0.0208  | max = 0.0403 (total)
```



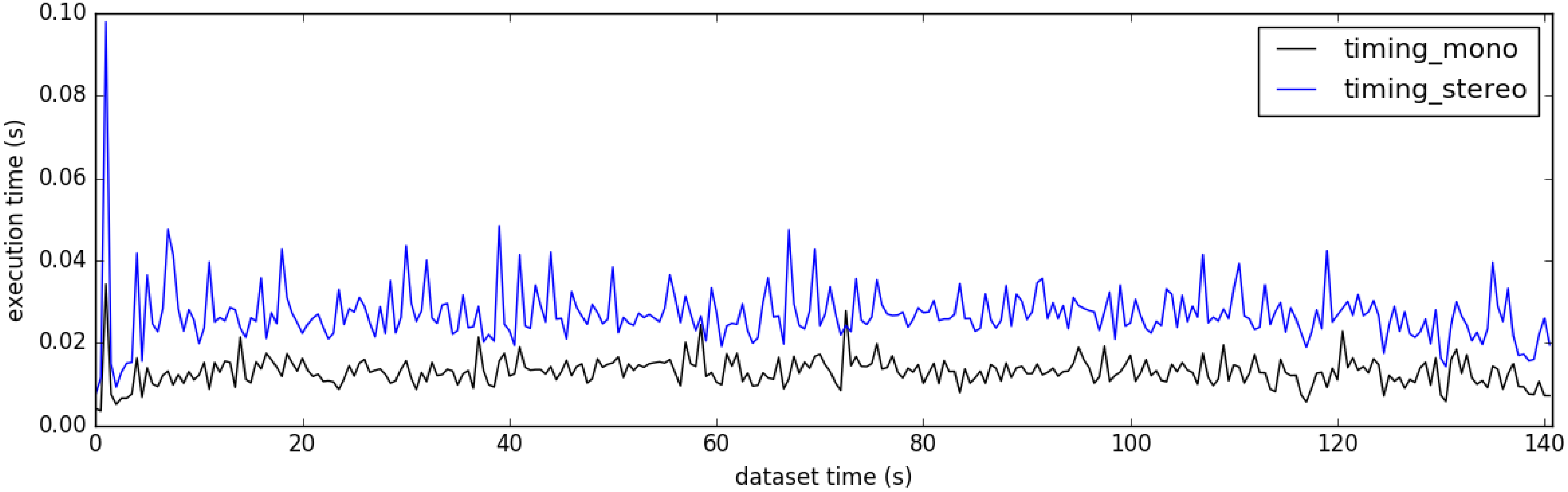
Script "timing_comparison"

This script is use to compare the run-time of different runs of the algorithm. This take in the same file as the flame graph and is recorded in the **ov_msckf::VioManager**. The file should be comma separated format, with the first column being the timing, and the last column being the total time. The middle columns should describe how much each component takes (whose names are extracted from the header of the csv file).

```
rosrun ov_eval timing_comparison <file_times1.txt> ... <file_timesN.txt>
rosrun ov_eval timing_comparison timing_mono_ethV101.txt timing_stereo_ethV101.txt
```

Example output:

```
======================================
[TIME]: loading data for timing_mono
[TIME]: loaded 2817 timestamps from file (7 categories)!!
mean_time = 0.0037 | std = 0.0011 | 99th = 0.0063  | max = 0.0254 (tracking)
mean_time = 0.0004 | std = 0.0001 | 99th = 0.0006  | max = 0.0014 ( propagation)
mean_time = 0.0032 | std = 0.0022 | 99th = 0.0083  | max = 0.0309 (msckf update)
mean_time = 0.0034 | std = 0.0013 | 99th = 0.0063  | max = 0.0099 (slam update)
mean_time = 0.0012 | std = 0.0017 | 99th = 0.0052  | max = 0.0141 (slam delayed)
mean_time = 0.0009 | std = 0.0003 | 99th = 0.0015  | max = 0.0027 (marginalization)
mean_time = 0.0128 | std = 0.0035 | 99th = 0.0208  | max = 0.0403 (total)
======================================
======================================
[TIME]: loading data for timing_stereo
[TIME]: loaded 2817 timestamps from file (7 categories)!!
mean_time = 0.0077 | std = 0.0020 | 99th = 0.0124  | max = 0.0219 (tracking)
mean_time = 0.0004 | std = 0.0001 | 99th = 0.0007  | max = 0.0023 ( propagation)
mean_time = 0.0081 | std = 0.0047 | 99th = 0.0189  | max = 0.0900 (msckf update)
mean_time = 0.0063 | std = 0.0023 | 99th = 0.0117  | max = 0.0151 (slam update)
mean_time = 0.0020 | std = 0.0026 | 99th = 0.0079  | max = 0.0205 (slam delayed)
mean_time = 0.0019 | std = 0.0005 | 99th = 0.0031  | max = 0.0052 (marginalization)
mean_time = 0.0263 | std = 0.0063 | 99th = 0.0410  | max = 0.0979 (total)
======================================
```



Script "timing_percentages"

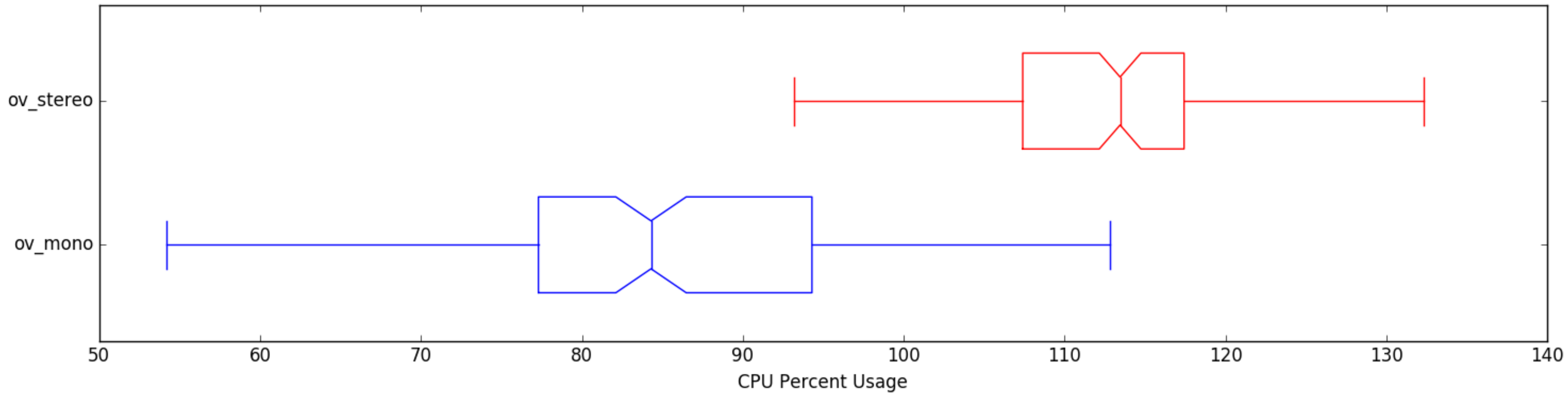
This utility allows for comparing the resources used by the algorithm on a specific platform. An example usage would be how the memory and cpu requirements increase as the state size grows or as more cameras are added. You will need to record the format using the `pid_ros.py` node (see **Collection** for details on how to use it). Remember that 100% cpu usage means that it takes one cpu thread to support the system, while 200% means it takes two cpu threads to support the system (see this link for an explanation). The folder structure needed is as follows:

```
psutil_logs/
    ov_mono/
        run1.txt
        run2.txt
        run3.txt
    ov_stereo/
        run1.txt
        run2.txt
        run3.txt
```

```
rosrun ov_eval timing_percentages <timings_folder>
rosrun ov_eval timing_percentages psutil_logs/
```

Example output:

```
======================================
[COMP]: processing ov_mono algorithm
    loaded 149 timestamps from file!!
    PREC: mean_cpu = 83.858 +- 17.130
    PREC: mean_mem = 0.266 +- 0.019
    THR: mean_threads = 12.893 +- 0.924
======================================
======================================
[COMP]: processing ov_stereo algorithm
    loaded 148 timestamps from file!!
    PREC: mean_cpu = 111.007 +- 16.519
    PREC: mean_mem = 5.139 +- 2.889
    THR: mean_threads = 12.905 +- 0.943
======================================
```



Generated by doxygen 1.8.19 and m.css.