基于 ESKF 的 6DOF 的传感器融合的实现

ankur6ue

April 27, 2017

摘要

基于 ESKF 的陀螺、加速度和摄像机测量传感器融合的实现。

Part 1

在这一系列文章中，我将为 [1] 中描述的传感器融合算法提供数学推导、实现细节和我自己的见解。本文介绍了一种利用扩展卡尔曼滤波器 (EKF) 自动确定相机与 IMU 之间的非本征标定的方法。但是，基础数学可用于各种应用，例如在陀螺和加速度计数据之间执行传感器融合，并使用摄像机为陀螺/加速度传感器融合提供测量更新。我不会去研究卡尔曼滤波器背后的理论。已经有大量的出版物描述了卡尔曼滤波器的工作原理。以下是一些我认为有用的参考资料。

- python 中的 kalman 和 bayesian 滤波器 [2]。这是一个很好的介绍性参考，包括许多实际示例和代码示例。它还为实现扩展卡尔曼滤波器提供了一个有用的数学和代码示例。
- 最优状态估计 [3]。这本书类似于第一本参考书，但提供了更多的例子和一些新的见解。
- 随机模型，估计和控制 [4]。这本书是“圣经”级别的估计和控制理论方面的书籍，在该领域的从业人员必须阅读认真。然而，它非常干燥，对于刚开始工作的人来说，可能非常令人望而生畏。
- 三维姿态估计的间接卡尔曼滤波 [5]。本技术报告提供了矩阵和四元数数学的重要介绍，我们将在这里广泛使用。

我遇到的关于卡尔曼滤波器的大多数参考文献要么深入研究数学，而不提供对滤波器功能操作方面的实际动手的见解，要么描述相对简单的例子，例如位置滤波器，其中所有东西都是线性的。尤其是对于 EKF，很少有例子说明如何计算雅可比矩阵、设置协方差矩阵等等。

在本系列文章中，我们将首先描述如何在加速度计和陀螺仪数据之间执行传感器融合，然后在测量更新步骤中添加摄像机图像测量。从数学和实现的角度来研究这个问题是很有趣的，并且在车辆导航、AR/VR 系统、目标跟踪和许多其他方面有许多有用的应用。

让我们先简单介绍一下卡尔曼滤波器的工作原理。想象一下，你站在一个房间的入口处，被要求沿着直线进入房间。对大多数人来说，这是一件微不足道的事情；然而，为了使事情变得更棘手，在你仔细观察了房间的四周之后，你的眼睛周围会蒙上一层眼罩。我们把与直线的垂直距离称为“状态”。既然你从这条直线的起点开始你的旅程，你的初始状态和它的“不确定性”是零。当你

向前迈出每一步时，在你意识到这条线在哪里而实际上看不到它的指导下，状态值将以随机方式改变。然而，随着你所采取的每一步，你的状态中的不确定性总是增加，你对直线的位置变得越来越不确定。不确定性的增加量取决于人——对于走钢索的人来说是小的，对于醉酒的人来说是大的。卡尔曼滤波器中的预测步骤通过使用一组矩阵来模拟从当前状态 (从采取步骤前的直线到下一个状态的距离) 到下一个状态 (采取步骤后的距离) 的转换以及在每个步骤中注入的不确定性量，以数学精确的方式模拟采取步骤的行为。

现在，在你已经采取了一些步骤，并且对你在直线上的位置相当不确定之后，蒙眼被摘下，你就可以看到你在直线上的位置了。现在你可以立即调整你的状态，并再次确定你的位置。这是卡尔曼滤波器中的测量更新步骤。测量提供了状态信息，有助于减少不确定性。测量有其自身的不确定性，不需要提供关于状态的直接信息。例如，你的视力可能并不完美，你对自己的位置仍有点不确定，或者你只能在房间周围观看，但不能直接在线路所在的楼层观看。在这种情况下，你必须从你对房间几何的了解中推断出你相对于线条的位置。

现在让我们把这些想法用数学语言表达，使它们更具体。状态 k 从 $k-1$ 的状态演变而来，根据

$$\mathbf{x_k} = \mathbf{F_k x_{k-1}} + \mathbf{B_k u_k} + \mathbf{w_k} \tag{1}$$

在这里，

- $\mathbf{F_k}$ 是状态转换矩阵。对于简单的问题，这个矩阵可以是一个常数，但是对于大多数实际应用程序，转换矩阵依赖于状态向量的值并改变每次迭代。
- $\mathbf{B_k}$ 是应用于控制向量 $\mathbf{u}$ 的控制输入模型。这可用于为系统的已知控制输入建模，例如应用于机器人电机的电流、汽车方向盘的位置等。在我们的示例中，我们不假设对控制输入有任何了解，因此 $\mathbf{B}$ 不会成为我们模型的一部分。
- $\mathbf{w_k}$ 是过程噪声，假定从零均值多元正态分布 $\mathcal{N}$ 中提取，利用协方差矩阵 $\mathbf{Q_k} : \mathbf{w_k} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q_k})$。为这个矩阵确定一个合适的值是很棘手的，并且在卡尔曼滤波器的文献中经常被忽视。我将描述我对如何为传感器融合问题设置矩阵的观察。

在时间“k”时刻，一个真实状态的观测 (或测量)z_k 根据

$$\mathbf{z_k} = \mathbf{H_k x_k} + \mathbf{v_k} \tag{2}$$

其中

- H_k 是将状态空间映射到观测空间的观测模型。
- v_k 是假设为零均值高斯协方差的观测噪声 $\mathbf{R_k} : \mathbf{v_k} \sim \mathcal{N}(\mathbf{0}, \mathbf{R_k})$

注意，H_k 将状态向量映射到观测值，而不是相反。这是因为 H_k 通常是不可逆的，也就是说，它不提供对状态的直接可见性。

滤波器的状态由两个变量表示：

- $\hat{\mathbf{x}}_{k|k}$，时间 k 的后验状态估计，给出时间 k 之前 (包括该时间) 的观测值；

- $\mathbf{P_{k|k}}$，后验误差协方差矩阵 (状态估计精度的度量)。

滤波器分两步工作：

- 一个预测步骤，其中状态和协方差矩阵根据我们对系统动力学和误差特征的了解进行更新，这些特征由 **F** 和 **Q** 矩阵建模。预测步骤不包括观察结果的影响。这一步类似于在上面描述的例子中蒙上眼罩的一步。
- 一个测量更新步骤，其中包括观测的影响，以完善状态估计和协方差矩阵。这一步类似于在我们的示例中去掉眼罩。注意，预测和更新步骤不必在 lockstep 中发生。在进行测量更新之前，可能会出现许多预测步骤，例如，我们示例中的主题可以在移除眼罩之前采取几个步骤。也可能有许多不同的观测源，这些源的测量可以在不同的时间到达。

预测和更新步骤的相应方程式如下：

预测

预测 (先验) 状态估计	$\hat{\mathbf{x}}_{k\mid k-1} = \mathbf{F_k}\widehat{\mathbf{x}}_{\mathbf{k-1\mid k}}$
预测 (先验) 估计协方差	$\mathbf{P_{k\mid k-1}} = \mathbf{F_k P_{k-1\mid k-1} F_k^T} + \mathbf{Q_k}$

更新

创新或测量残差	$\tilde{\mathbf{y}}_k = \mathbf{z_k} - \mathbf{H_k}\widehat{\mathbf{x}}_{\mathbf{k\mid k-1}}$
创新 (或残差) 协方差	$\mathbf{S_k} = \mathbf{H_k P_{k\mid k-1} H_k^{\mathrm{T}}} + \mathbf{R_k}$
最佳卡尔曼增益	$\mathbf{K_k} = \mathbf{P_{k\mid k-1} H_k^{\mathrm{T}} S_k^{-1}}$
更新 (后验) 的状态估计	$\hat{\mathbf{x}}_{k\mid k} = \hat{\mathbf{x}}_{k\mid k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
更新 (后验) 的协方差	$\mathbf{P_{k\mid k}} = (\mathbf{I} - \mathbf{K_k H_k})\mathbf{P_{k\mid k-1}}$

卡尔曼滤波实现的主要任务是利用系统动力学模型和测量模型，提出状态转换矩阵 **F**、测量矩阵 **H** 和系统噪声特性，设计过程和测量噪声协方差矩阵。

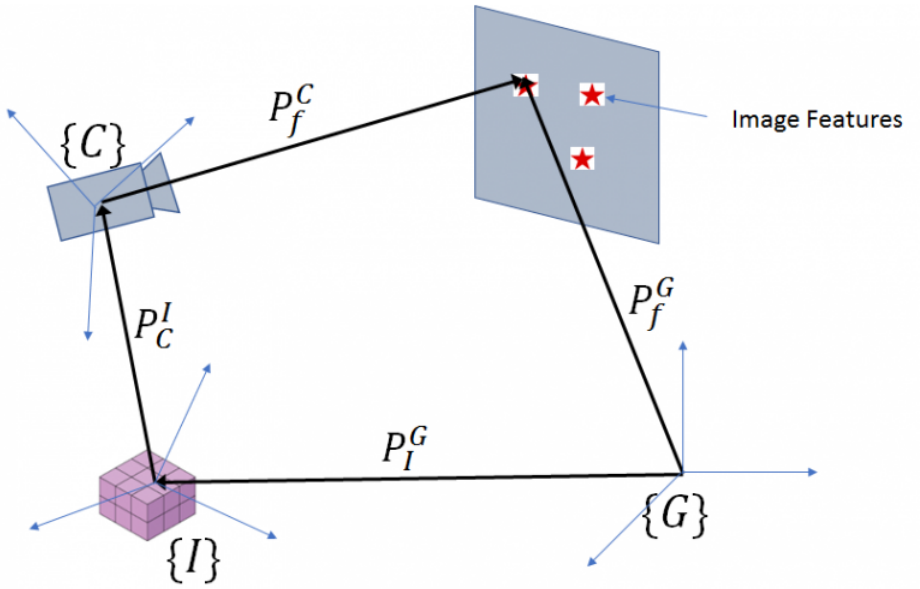
现在我们来考虑一下我们的具体问题。我们的最终目标是在 [1] 中导出方程式 (16) 和 (19)。为了使数学更容易理解，我们首先要考虑更大问题的一个子集，即如何使用卡尔曼滤波器组合陀螺仪和加速度计的输出，然后添加图像测量值。

关于惯性数据的传感器融合，已有大量的文献。我在这里不谈细节。有关陀螺仪和加速度计的相对强度以及如何实现互补滤波器以组合来自这些传感器的数据的详细说明，请参阅DCMDraft2。

在 [1] 中的方程式 (16) 中的状态向量由 21 个元素组成。

1. δP_I^G：IMU 在全局参考系中的位置误差
2. δV_I^G：IMU 在全局参考系中的速度误差
3. $\delta\theta_I^G$：IMU 和全局参考系之间的方向误差
4. δP_C^I：摄像头相对于 IMU 的位置误差
5. $\delta\theta_C^I$：摄像头和 IMU 之间的方向误差
6. δb_g：陀螺偏差误差

7. δb_a：加速度偏差误差



P_I^G: Position of the IMU in the Global Frame

P_f^G: Position of the image feature in the Global Frame

$v_I = C_G^I v_G$ (DCM C_G^I transforms a vector v from frame G to frame I

$P_C^G = P_I^G + C_I^G C_C^I P_C^I$

上面显示了三个坐标系：全局坐标系、IMU 坐标系和照相机坐标系。上面也显示了用于描述位置向量和旋转矩阵的符号。让我们首先考虑一个更简单的问题——整合陀螺数据以产生方向，并使用加速度计数据作为测量值来估计陀螺偏差并校正漂移。这个问题的状态向量由 6 个元素组成——IMU 相对于全局参考系的方向误差和陀螺偏差误差。注意，我们的状态包括方向误差和陀螺偏差误差，而不是方向和偏差本身。这一选择的原因很快就会清楚。

在接下来的讨论中，我们将反复评估向量相对于位置和方向的导数，因此我们建立一对向量微分方程。

- 对于一个向量 $v = [v_1 v_2 v_3]$，交叉积斜对称矩阵定义为：

$$[\mathbf{v}]_\times = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \tag{3}$$

这个矩阵称为交叉积斜对称矩阵，因为它可以将两个向量的交叉积表示为矩阵乘法。

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} \tag{4}$$

这个矩阵对我们很有用，因为它可以用来表示小旋转的效果。

用 DCM $C(t)$ 和 (等效地) 四元数 $q(t)$ 来描述旋转帧 B 在时间 t 相对于参考帧 A 的方向。设 B 的瞬时角速度为 $\boldsymbol{\omega}$。在一个小的时间间隔 dt 中，假设该速度为常数，帧 B 会沿着局部坐标系的

轴旋转 $\phi = [\phi_x, \phi_y, \phi_z] = [\omega_x dt, \omega_y dt, \omega_z dt]$，从而在时间 $t + dt$ 处产生新的方向。让我们考虑使用第一个 DCM，然后使用四元数来表示这个新方向。

$$C(t+1) = C(t)\exp(\phi_3) \tag{5}$$

这里 ϕ_3 是一个 3×3 斜对称矩阵：

$$\phi_3 = \begin{bmatrix} 0 & -\phi_z & \phi_y \\ \phi_z & 0 & -\phi_x \\ -\phi_y & \phi_x & 0 \end{bmatrix} \tag{6}$$

展开指数并使用斜对称矩阵的性质可得到众所周知的罗德里格斯 (Rodrigues) 公式：

$$C(t+1) = C(t)(I_3 + \frac{\sin(\theta)}{\theta}\phi_3 + \frac{1-\cos(\theta)}{\theta^2}\phi_3^2) \tag{7}$$

这里，$\theta = \sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}$

对于小角度 θ，罗德里格斯 (Rodrigues) 公式简化为：

$$C(t+1) = C(t)(I_3 + \phi_3) \tag{8}$$

请注意，上面方程式 (5) 带 $\exp(\phi_3)$ 的乘法是后 (右) 乘法，而不是前 (左) 乘法。这是因为用 ϕ 表示的小旋转应用于机体坐标系的轴 (应用 $C(t)$ 后获得的坐标系)。使用前乘将导致应用于静态 (非移动) 坐标系的小旋转。

现在我们来看看四元数表示。使用小角度表示法，对小旋转 ϕ 的四元数表示法如下：

$$\delta(q) \approx \begin{bmatrix} 1 \\ \frac{\phi}{2} \end{bmatrix} \tag{9}$$

在时间 $t + dt$ 时刻，新方向由四元数 $q(t + dt)$ 给出，其中

$$q(t+dt) = q(t) * \delta q \tag{10}$$

一般来说，δq 是时间的函数 (除非物体以恒定角速度旋转)。

$$\frac{dq}{dt} = \lim_{\delta t \to 0} \frac{q(t) * [\delta q - I_q]}{\delta t} \tag{11}$$

其中 I_q 是单位四元数。替换 δq 有，

$$\dot{q} = \frac{1}{2} q(t) * \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \tag{12}$$

利用四元数乘法的性质，该表达式可以写成矩阵积：

$$\dot{q} = \frac{1}{2}[\omega_4][q] \tag{13}$$

这里，ω_4 是一个 4×4 斜对称矩阵，定义为 (定义提供给 ϕ_4)

$$\phi_4 = \begin{bmatrix} 0 & -\phi_x & -\phi_y & -\phi_z \\ \phi_x & 0 & \phi_z & -\phi_y \\ \phi_y & -\phi_z & 0 & \phi_x \\ \phi_z & \phi_y & -\phi_x & 0 \end{bmatrix} \tag{14}$$

解微分方程，

$$q(t+dt) = \exp(\frac{1}{2}\phi_4)q(t) \tag{15}$$

请注意，此解自动保留新四元数的单位范数属性：

$$q(t+dt)^T q(t+dt) = q(t)^T \exp(-\frac{1}{2}\phi_4)\exp(\frac{1}{2}\phi_4)q(t) = q(t)^T q(t) = 1 \text{ since } \phi_4^T = -\phi_4 \tag{16}$$

此外，使用 4×4 斜对称矩阵的属性，四元数表示可以简化为[1]：

$$\exp(\frac{1}{2}\phi_4) = I\cos(\theta) + \frac{1}{2}\phi_4\frac{\sin(\theta)}{\theta} \tag{17}$$

这里，$\theta = \frac{\|\phi\|}{2}$。

为了避免与除以 0 相关的奇点，可以使用 $\frac{\sin(\theta)}{\theta}$ 的泰勒级数展开。

为了巩固这些想法，让我们来看一些 Matlab 代码，使用上面描述的三种方法来传播四元数。

```
function q_est = apply_small_rotation(phi, q_est)
s = norm(phi)/2;
x = 2*s;
phi4 = make_skew_symmetric_4(phi);
% approximation for sin(s)/s
sin_sbys_approx = 1-s^2/factorial(3) + s^4/factorial(5) - s^6/factorial(7);
exp_phi4 = eye(4)*cos(s) + 1/2*phi4*sin_sbys_approx;
% method 1
q_est1 = exp_phi4*q_est;
% verify: should be equal to:
% method 2 (direct matrix exponential)
q_est2 = expm(0.5*(phi4))*q_est;
% verify should be equal to:
% method 3: propagate by quaternion multiplication (doesn't preserve unit
% norm)
dq = [1 phi(1)/2 phi(2)/2 phi(3)/2];
dq_conj = [1 -phi(1)/2 -phi(2)/2 -phi(3)/2]';
q_est3 = quatmul(q_est, dq');
q_est = q_est1;
end

function phi4 = make_skew_symmetric_4(phi)
phi4 = [0       -phi(1)      -phi(2)      -phi(3);
    phi(1)      0            phi(3)      -phi(2);
```

[1]译注：原文这个指数方程里的符号写错为'-'，但代码里没错。类似的证明见这里，以及此文的附录 A。

```
    phi(2)     -phi(3)      0           phi(1);
    phi(3)      phi(2)     -phi(1)      0];
end
```

这些公式中的任何一个都可以用于传播方向。四元数表示是优越的，因为它没有奇点。因此，方向应保持四元数形式，并根据需要提取其他旋转表示。

现在让我们考虑一个由 DCM C 旋转的向量对位置和方向的微小变化的导数。在建立 EKF 状态传播方程时，我们将重复使用这些导数。

$$\mathbf{v}' = \mathbf{C}\mathbf{v} \tag{18}$$

关于位置的导数方程：

$$\mathbf{dv}'_{\mathbf{v}} = \mathbf{C}(\mathbf{v} + \mathbf{dv} - \mathbf{v}) = \mathbf{C}\mathbf{dv} \tag{19}$$

关于方向的导数方程：

$$\mathbf{dv}'_{\phi} = \mathbf{C}(\mathbf{I_3} + \mathbf{d}\phi_\mathbf{3})\mathbf{v} - \mathbf{C}\mathbf{v} = \mathbf{C}\mathbf{d}\phi_\mathbf{3}\mathbf{v} \tag{20}$$

这里 $\mathbf{d}\phi_\mathbf{3}$ 是一个 3×3 对于小旋转的斜对称矩阵：

$$\mathbf{d}\phi_\mathbf{3} = \begin{bmatrix} 0 & -d\phi_z & d\phi_y \\ d\phi_z & 0 & -d\phi_x \\ -d\phi_y & d\phi_x & 0 \end{bmatrix} \tag{21}$$

使用叉积表示法，

$$\mathbf{dv}'_{\phi} = \mathbf{C}[\mathbf{d}\phi]_\times \mathbf{v} \tag{22}$$

对于叉积斜对称矩阵，$[\mathbf{a}]_\times \mathbf{b} = -[\mathbf{b}]_\times \mathbf{a}$。因此，

$$\mathbf{dv}'_{\phi} = -\mathbf{C}[\mathbf{v}]_\times \mathbf{d}\phi \tag{23}$$

这些表达式的好处在于，它们允许我们将导数相对于 (wrt) 位置和方向表示为矩阵乘法，这是 EKF 状态更新和测量更新公式所必需的。

$$\begin{bmatrix} \mathbf{dv}'_{\mathbf{v}}, \mathbf{dv}'_{\phi} \end{bmatrix} = \begin{bmatrix} C & -C[\mathbf{v}]_\times \end{bmatrix} \begin{bmatrix} \mathbf{dv} \\ \mathbf{d}\phi \end{bmatrix} \tag{24}$$

作为一个例子，我们来看一些实现旋转和基于四元数的更新的 Matlab 代码。

```
% Original vector v
v = [1 1 1];
% Convert to unit vector
v = v./norm(v);
% Initial rotation (euler angles)
eul = [30 30 30]/57.3;
% Get quaternion and DCM representation for this rotation
q0 = eul2quat(30/57.3, 30/57.3, 30/57.3);
```

```
C0 = euler2dc(30/57.3, 30/57.3, 30/57.3);
% Apply rotation
v1 = C0*v';
% Small rotation (in radians)
phi = [0.01 0 0.01];
% Obtain the 3*3 and 4*4 skew symmetric matrices
phi3 = make_skew_symmetric_3(phi);
phi4 = make_skew_symmetric_4(phi);
s = norm(phi)/2;
x = norm(phi);
exp_phi4 = eye(4)*cos(s) - 1/2*phi4*sin(s)/s;
% Propagage quaternion
q1 = exp_phi4*q0';
% Propagate DCM (Rodrigues Formula)
C1 = C0*(eye(3) + sin(x)/x*phi3 + (1-cos(x))/x^2*phi3*phi3);
% Obtain the DCM corresponding to q1 to compare with C1
C_q1 = quat2dc(q1);
% Verify C_q1 ~ C1
% Now verify derivative wrt phi
% True value of the derivative
d1 = C1*v'-C0*v'
d2 = C0*make_skew_symmetric_3(v')*phi'
% verify d1 ~ d2
```

我没有提供实用功能的代码，如 DCM 到 Euler 角度转换。任何标准的矩阵代数库都可以用于实现。

这足以消化一篇文章！在下一篇文章中，我们将研究用于执行陀螺加速度传感器融合的 EKF 框架。

参考文献

1. Mirzaei FM, Roumeliotis SI. A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation. IEEE Transactions on Robotics. 2008;24(5):1143-1156. doi: 10.1109/tro.2008.2004486 [Source]

2. Labbe R. Kalman and Bayesian Filters in Python. Kalman and Bayesian Filters in Python. `http://robotics.itee.uq.edu.au/~elec3004/2015/tutes/Kalman_and_Bayesian_Filters_in_Python.pdf`. Published April 22, 2017.

3. Simon D. Optimal State Estimation. John Wiley & Sons, Inc.; 2006. doi: 10.1002/0470045345 [Source]

4. Maybeck P. Stochastic Models, Estimation and Control. Academic Press; 2012.

5. Indirect Kalman Filter for 3D Attitude Estimation. users.cs.umn.edu. `http://www-users.cs.umn.edu/~trawny/Publications/Quaternions_3D.pdf`. Published March 2005. Accessed April 23, 2017.

Part 2 组合陀螺加速度数据

在前一篇文章中，我们提出了卡尔曼滤波器背后的一些数学基础。在这篇文章中，我们将看到我们的第一个具体例子——在陀螺仪和加速度计之间执行传感器融合。

现实世界中的 MEMS 陀螺仪和加速度计通常有两个主要误差源——偏差漂移和随机噪声。也有缩放误差和横轴误差 ($X/Y/Z$ 轴可能不完全垂直)，但是这些误差往往很小，我们的分析中不会考虑这些误差。

在 [1] 之后，我们使用以下符号：

- ω 角速度
- $\mathbf{a}$ 用于加速度
- $\omega_{\mathbf{b}}$ 用于陀螺仪偏差
- $\mathbf{a}_{\mathbf{b}}$ 用于加速度偏差
- 下标 m 表示测量量，头上帽子 $\hat{}$ 表示估计量
- 真值不带注释出现。

陀螺仪的测量值是被陀螺仪偏差和随机噪声破坏的真实值。因此，

$$\omega_{\mathbf{m}} = \omega + \omega_{\mathbf{b}} + \mathbf{n}_{\mathbf{r}} \tag{25}$$

这里，速度噪声 $\mathbf{n}_{\mathbf{r}}$ 假设为具有特征的高斯白噪声

$$E[\boldsymbol{n_r}] = 0_{3\times 1}, E[\boldsymbol{n_r}(t+\tau)\boldsymbol{n_r^T}(t)] = \boldsymbol{N_r}\delta(\tau) \tag{26}$$

陀螺仪偏差是非静态的 (否则不需要估计)，建模为随机游走过程：

$$E[\boldsymbol{n_b}] = 0_{3\times 1}, E[\boldsymbol{n_b}(t+\tau)\boldsymbol{n_b^T}(t)] = \boldsymbol{N_b}\delta(\tau) \tag{27}$$

我们还假设相关的协方差矩阵是对角的：

$$\boldsymbol{N_r} = diag(\sigma_{r1}^2, \sigma_{r2}^2, \sigma_{r3}^2) \tag{28}$$

并且

$$\boldsymbol{N_b} = diag(\sigma_{b1}^2, \sigma_{b2}^2, \sigma_{b3}^2) \tag{29}$$

在下一篇文章中，我们将考虑如何从测量日志中设置这些协方差。

估计角速度是测量速度和我们估计的陀螺偏差之间的差。

$$\begin{aligned} \hat{\omega} &= \omega_{\mathbf{m}} - \widehat{\omega_{\mathbf{b}}} \\ \hat{\omega} &= \omega + \omega_{\mathbf{b}} + \mathbf{n}_{\mathbf{r}} - \hat{\omega_{\mathbf{b}}} \\ \hat{\omega} &= \omega - \delta\omega_{\mathbf{b}} + \mathbf{n}_{\mathbf{r}} \end{aligned} \tag{30}$$

这里，$\delta\omega_b = \hat{\omega_\mathbf{b}} - \omega_\mathbf{b}$ 是陀螺偏差误差。

我们的状态向量包括方向误差和陀螺偏差误差。

$$\mathbf{x} = [\boldsymbol{\delta\theta}, \delta\omega_\mathbf{b}] \tag{31}$$

方向误差被定义为小旋转的向量，它将估计的旋转与真实的旋转对齐。相应的误差四元数近似为：

$$\delta q \approx \begin{bmatrix} 1 \\ \frac{\boldsymbol{\delta\theta}}{2} \end{bmatrix} \tag{32}$$

误差四元数是将估计方向与真实方向对齐所需的旋转。

$$q = \hat{q} * \delta q \tag{33}$$

这里 $*$ 表示四元数乘法。

EKF 公式中的关键任务是建立状态传播矩阵，描述状态如何从一个步骤变化到下一个步骤。换句话说，我们对矩阵 F 感兴趣

$$\boldsymbol{x_{t+dt}} = F\boldsymbol{x_t} \tag{34}$$

对于非线性问题，矩阵 F 是通过对局部导数进行线性化得到的，即，我们首先得到一个关于 $\dot{\boldsymbol{x}} = J\mathbf{x}$ 的表达式。矩阵 J 称为系统动力学矩阵。详见 [2] 第 7 章。

然后状态转换矩阵 F 等于 $\exp(Jdt)$。展开指数，忽略高阶项，

$$F = I + Jdt \tag{35}$$

因此，任务简化为找到矩阵 J。由于我们的状态向量有两个分量，我们可以将 J 拆分为：

$$J = \begin{bmatrix} J_{\delta\boldsymbol{\theta}} & J_{\boldsymbol{\omega_b}} \end{bmatrix} \tag{36}$$

让我们首先考虑一下 $J_{\delta\boldsymbol{\theta}}$。这是整个 EKF 配方中最棘手的一部分，所以请紧紧抓住！我将首先介绍一些四元数代数方程，稍后我们将在推导中使用这些方程。这个代数取自 [1]，有一个重要的区别。在 [1] 中，四元数定义为 $q = [\text{vector scalar}]$，而我们将使用更常见的定义：$q = [\text{scalar vector}]$。

两个四元数 q 和 p 的乘积如下：

$$q * p = \begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \\ q_2 & q_1 & -q_4 & q_3 \\ q_3 & q_4 & q_1 & -q_2 \\ q_4 & -q_3 & q_2 & q_1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{37}$$

在本系列的第 1 部分中，让我们将叉积斜对称矩阵定义为：

$$[q]_\times = \begin{bmatrix} 0 & -q_4 & q_3 \\ q_4 & 0 & q_2 \\ -q_3 & q_2 & 0 \end{bmatrix} \tag{38}$$

四元数乘法的表达式现在可以写成：

$$q * p = \begin{bmatrix} q_1 & -\mathbf{q} \\ \mathbf{q^T} & q_1 \boldsymbol{I}_{3\times 3} + [q]_\times \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{39}$$

这里，$\mathbf{q}$ 是四元数的向量部分 $([q_2, q_3, q_4])$。

乘法也可以写为：

$$q * p = \begin{bmatrix} p_1 & -p_2 & -p_3 & -p_4 \\ p_2 & p_1 & p_4 & -p_3 \\ p_3 & -p_4 & p_1 & p_2 \\ p_4 & p_3 & -p_2 & p_1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{40}$$

使用 p 的叉积矩阵，该表达式可写为：

$$q * p = \begin{bmatrix} p_1 & -\mathbf{p} \\ \mathbf{p^T} & p_1 \boldsymbol{I}_{3\times 3} + [p]_\times \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{41}$$

考虑到这两个表达式的不同，

$$q * p - p * q = (\begin{bmatrix} q_1 & -\mathbf{q} \\ \mathbf{q^T} & q_1 \boldsymbol{I}_{3\times 3} + [q]_\times \end{bmatrix} - \begin{bmatrix} q_1 & -\mathbf{q} \\ \mathbf{q^T} & q_1 \boldsymbol{I}_{3\times 3} - [q]_\times \end{bmatrix})p \tag{42}$$

从这个结果可以清楚地看出四元数乘法不是交换的。简化为，

$$q * p - p * q = \begin{bmatrix} 0 & 0 \\ 0 & 2[q]_\times \end{bmatrix} p \tag{43}$$

现在，在 [3] 的第 1 章之后，让 $q(t)$ 描述框架 A 相对于 (wrt) 框架 B 的相对方向。让 A 沿方向 $\mathbf{s}$ 的瞬时角速度表示为 ω。然后用四元数表示时间 δt 增量坐标旋转。

$$\delta q(\delta t) \approx \begin{bmatrix} 1 \\ \frac{\omega \delta \mathbf{t}}{2} \end{bmatrix} \tag{44}$$

在时间 $t + \delta t$，方向由四元数 $q(t + \delta t)$ 表示，其中

$$q(t + \delta t) = q(t) * \delta q(\delta t) \tag{45}$$

由此，四元数的时间导数可以得到：

$$\dot{q} = \frac{1}{2} q * \omega \tag{46}$$

这里 ω 是四元数表示。用它的标量和向量分量写为，

$$\omega = \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \tag{47}$$

为了获得误差四元数的系统动力学，我们需要将误差四元数的导数 $\dot{\delta q}$ 表示为 δq 的函数。现在从误差四元数的定义来看，

$$q = \hat{q} * \delta q \tag{48}$$

取导数，

$$\dot{q} = \dot{\hat{q}} * \delta q + \hat{q} * \dot{\delta q} \tag{49}$$

用前面推导的表达式，用角速度表示时间导数，

$$\frac{1}{2} q * \omega = \frac{1}{2} \hat{q} * \hat{\omega} * \delta q + \hat{q} * \dot{\delta q} \tag{50}$$

将包含 δq 的项移动到 LHS

$$\hat{q} * \dot{\delta q} = \frac{1}{2} q * \omega - \frac{1}{2} \hat{q} * \hat{\omega} * \delta q \tag{51}$$

现在，从误差四元数的定义来看，

$$q = \hat{q} * \delta q \tag{52}$$

前乘 $\hat{q}^{-1}$

$$\hat{q}^{-1} * q = \delta q \tag{53}$$

将方程式 (51) 前乘 $\hat{q}^{-1}$ 并使用方程式 (53)，

$$\dot{\delta q} = \frac{1}{2} \delta q * \omega - \frac{1}{2} \hat{\omega} * \delta q \tag{54}$$

使用方程式 (30) 收集各项，将四元数展开为其标量和向量分量，并使用方程式 (43)

$$\dot{\delta q} = \frac{1}{2} \begin{bmatrix} 0 & 0 \\ 0 & -2[\hat{\omega}]_\times \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{\delta q} \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 0 & -(-\delta\omega_b + \boldsymbol{n_r}) \\ (-\delta\omega_b + \boldsymbol{n_r}) & -[(-\delta\omega_b + \boldsymbol{n_r})]_\times \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{\delta q} \end{bmatrix} \tag{55}$$

注意，我们已经用矩阵乘法替换了四元数乘法。在上述方程的第二项中，我们可以忽略将陀螺偏差误差和噪声与误差四元数相乘的二阶分量。应用这种简化，我们得到[2]

$$\dot{\delta q} = \begin{bmatrix} 0 \\ -[\hat{\omega}]_\times \boldsymbol{\delta q} \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 0 \\ -\delta\omega_b + \boldsymbol{n_r} \end{bmatrix} \tag{56}$$

现在回想一下，我们的状态向量实际上是由旋转中的误差组成的，而不是误差四元数。两者之间的关系由以下公式给出：

[2]译注：原文的方程的格式有点瑕疵，式中等号右边应为向量 $\boldsymbol{\delta q}$。

$$\delta q \approx \begin{bmatrix} 1 \\ \frac{\boldsymbol{\delta\theta}}{2} \end{bmatrix} \tag{57}$$

将 $\boldsymbol{\delta q}$[3]替换为 $\delta\boldsymbol{\theta}$，我们最终得到我们寻求的表达式：

$$\delta\dot{\boldsymbol{\theta}} = -\hat{\boldsymbol{\omega}} \times \delta\boldsymbol{\theta} + \delta\boldsymbol{\omega_b} - \boldsymbol{n_r} \tag{58}$$

另一部分，我们的状态向量，陀螺偏差误差很容易处理。由于偏差变化非常缓慢，我们可以简单地假设误差在迭代时间间隔内是恒定的。因此，

$$\delta\dot{\boldsymbol{\omega}}_{\boldsymbol{b}} = 0 \tag{59}$$

我们终于得到了系统动力学方程！

$$\begin{bmatrix} \delta\dot{\boldsymbol{\theta}} \\ \delta\dot{\boldsymbol{\omega}}_{\boldsymbol{b}} \end{bmatrix} = \begin{bmatrix} -[\hat{\omega}]_\times & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{\theta} \\ \delta\boldsymbol{\omega_b} \end{bmatrix} \tag{60}$$

如上所述，使用一阶近似，可以从系统动力学矩阵中获得状态转换矩阵，使用

$$F = I + Jdt \tag{61}$$

因此，

$$F = \begin{bmatrix} -[\hat{\phi}]_\times & I_{3\times3}\delta t \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{62}$$

其中

$$[\hat{\phi}]_\times = [\hat{\omega}]_\times \delta t \tag{63}$$

我们基本上完成了数学上的举重。在下一篇文章中，我们将研究所有这些数学的 Matlab 实现，希望能够澄清本文中提出的概念。

参考文献

1. Trawny N, Roumeliotis S. Indirect Kalman Filter for 3D Attitude Estimation. MARS; 2005:24. `http://www-users.cs.umn.edu/~trawny/Publications/Quaternions_3D.pdf`.

2. Labbe R. Kalman and Bayesian Filters in Python. Kalman and Bayesian Filters in Python. `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`. Published April 22, 2017.

3. Stevens BL, Lewis FL, Johnson EN. Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems. John Wiley & Sons, Inc; 2015. doi:10.1002/9781119174882

[3]译注：同上，原文此处的格式有点瑕疵，应为向量 $\boldsymbol{\delta q}$。

Part 3 陀螺与加速度传感器融合的实现

在前面的文章中，我们为实现误差状态卡尔曼滤波器奠定了必要的数学基础。在这篇文章中，我们将使用前面开发的数学，提供用于在加速度计和陀螺仪数据之间执行传感器融合的 Matlab 实现。

让我们重述一下前一篇文章中介绍的各种量的符号和定义。

- ω 角速度
- **a** 用于加速度
- $\omega_{\mathbf{b}}$ 用于陀螺仪偏差
- $\mathbf{a_b}$ 用于加速度偏差
- 下标 m 表示测量量，头上帽子 ˆ 表示估计量
- 真值不带注释出现。

陀螺仪的测量值是被陀螺仪偏差和随机噪声破坏的真实值。因此，

$$\omega_{\mathbf{m}} = \omega + \omega_{\mathbf{b}} + \mathbf{n_r} \tag{64}$$

这里，速度噪声 $\mathbf{n_r}$ 假设为具有特征的高斯白噪声

$$E[\boldsymbol{n_r}] = 0_{3\times1}, E[\boldsymbol{n_r}(t+\tau)\boldsymbol{n_r^T}(t)] = \boldsymbol{N_r}\delta(\tau) \tag{65}$$

陀螺仪偏差是非静态的 (否则不需要估计)，建模为随机游走过程：

$$E[\boldsymbol{n_b}] = 0_{3\times1}, E[\boldsymbol{n_b}(t+\tau)\boldsymbol{n_b^T}(t)] = \boldsymbol{N_b}\delta(\tau) \tag{66}$$

我们还假设相关的协方差矩阵是对角的：

$$\boldsymbol{N_r} = diag(\sigma_{r1}^2, \sigma_{r2}^2, \sigma_{r3}^2) \tag{67}$$

并且

$$\boldsymbol{N_b} = diag(\sigma_{b1}^2, \sigma_{b2}^2, \sigma_{b3}^2) \tag{68}$$

在下一篇文章中，我们将考虑如何从测量日志中设置这些协方差。

估计角速度是测量速度和我们估计的陀螺偏差之间的差。

$$\begin{aligned} \hat{\omega} &= \omega_{\mathbf{m}} - \widehat{\omega_{\mathbf{b}}} \\ \hat{\omega} &= \omega + \omega_{\mathbf{b}} + \mathbf{n_r} - \hat{\omega_{\mathbf{b}}} \\ \hat{\omega} &= \omega - \delta\omega_{\mathbf{b}} + \mathbf{n_r} \end{aligned} \tag{69}$$

这里，$\delta\omega_b = \hat{\omega_{\mathbf{b}}} - \omega_{\mathbf{b}}$ 是陀螺偏差误差。

我们的状态向量包括方向误差和陀螺偏差误差。

$$\mathbf{x} = [\boldsymbol{\delta\theta}, \delta\omega_{\mathrm{b}}] \tag{70}$$

方向误差被定义为小旋转的向量，它将估计的旋转与真实的旋转对齐。相应的误差四元数近似为：

$$\delta q \approx \begin{bmatrix} 1 \\ \frac{\boldsymbol{\delta\theta}}{2} \end{bmatrix} \tag{71}$$

我们在上一篇文章中导出了状态转移矩阵 F。该矩阵由以下公式给出：

$$F = \begin{bmatrix} -[\hat{\phi}]_\times & I_{3\times3}\delta t \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{72}$$

其中

$$[\hat{\phi}]_\times = [\hat{\omega}]_\times \delta t \tag{73}$$

在一个典型的卡尔曼滤波器实现中，状态是每一个时间步更新的。我们的实现使用一种不同的卡尔曼滤波器配置，称为反馈配置。在这种配置中，当误差状态变量因处理测量而更新时，更新直接应用于系统状态 (在这种情况下，就是方向和陀螺的偏差)。因此，在每个时间步，我们将误差状态设置为 0。有关反馈配置的更多信息，请参阅 [1] 第 6 章。

我们会在每个时间步更新误差状态协方差。这是以常规方式完成的：

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{74}$$

这里 k 表示时间步索引。

现在让我们来看一下测量更新步骤。我们将首先使用加速度计提供测量更新，然后添加基于图像的测量。加速度计读数可以用作测量源，因为它们提供方向的直接测量，并且不会受到漂移的影响 (尽管它们可能有偏差，也可以估计)。顾名思义，加速度计测量物体坐标系中的总加速度。对于静止或匀速运动的物体，加速度计将测量对重力的反应。让我们用向量 $[0, 0, g]$ 表示重力对加速度的反应，用 C 表示机体框架的当前方向，然后加速度计读数如下：

$$\boldsymbol{a} = C\boldsymbol{g} \tag{75}$$

在建立测量更新方程时，我们需要将测量残差 (用 z 表示) 表示为状态向量的线性函数，即测量的估计值和真实值之间的差。

为了计算残差，我们需要将 **a** 转换回全局参考框架。这可以用后乘以 $\hat{C}^T$ 来完成。注意，这里不知道真实的 DCM C，只知道它的估计值。因此，乘以 $\hat{C}$ 是我们能做的最好的。

$$\begin{aligned} \boldsymbol{z} &= C\hat{C}^T\boldsymbol{g} - \boldsymbol{g} \\ \boldsymbol{z} &= C(\hat{C}^T - C^T)\boldsymbol{g} \end{aligned} \tag{76}$$

现在回想一下前面文章中的方程 (8)：$C = \hat{C} + \hat{C}[\delta\theta]_\times$。另外，由于 $[\delta\theta]_\times$ 是斜对称的，$[\delta\theta]_\times^T = -[\delta\theta]_\times$。应用这些属性，

$$\boldsymbol{z} = C[\delta\theta]_\times \hat{C}^T\boldsymbol{g} \tag{77}$$

现在，应用小方向误差的近似值，$C \approx \hat{C}$，

$$\boldsymbol{z} \approx \hat{C}[\delta\theta]_\times \hat{C}^T \boldsymbol{g} \tag{78}$$

根据 [1] 中的方程式 (49)，$[Ca]_\times = C[a]_\times C^T$。因此，

$$\boldsymbol{z} = [\hat{C}\boldsymbol{\delta\theta}]_\times \boldsymbol{g} \tag{79}$$

根据 [1] 中的方程式 (39)，$[a]_\times \boldsymbol{b} = -[b]_\times \boldsymbol{a}$。因此[4]，

$$\boldsymbol{z} = -[\boldsymbol{g}]_\times \hat{C}\boldsymbol{\delta\theta} \tag{80}$$

加速度计不提供对偏差的直接观测，因此状态向量中的偏差误差部分对应的项为 0。完整的 H 矩阵由以下公式给出：

$$H_{accel} = \begin{bmatrix} -[\boldsymbol{g}]_\times \hat{C} & 0 \end{bmatrix} \tag{81}$$

设置协方差矩阵

卡尔曼滤波器的一个主要优点是，它不仅执行最理想的状态估计 (在其操作假设下)，而且还提供状态协方差矩阵的估计，允许我们在估计的状态变量周围放置一个置信区间。为此，滤波器将三个协方差矩阵作为输入：

- P_0：状态协方差矩阵的初始值。这可以设置为适当的值，反映我们在测量初始状态时的初始不确定度。它只影响滤波器的瞬态特性，而不是协方差矩阵的最终 (收敛) 值。
- R_{accel}：这是测量不确定度矩阵，反映了由于测量中的各种误差而引起的不确定度。
- P_k：这是过程噪声协方差，反映了在卡尔曼滤波器更新的每个步骤中添加的不确定性量。这个矩阵通常是最难估计的。

一种常用的量化惯性传感器数据中各种噪声源的方法被称为 Allan 均方差。基本思想是计算具有不同步幅长度的重叠数据簇之间的方差。

其基本思想是，随机游走和量化误差等短期噪声源出现在 m 的较小值上，平均出现在较大的样本上。在较大的样本上会出现陀螺偏差随机游动等长期误差源。因此，通过检查与步幅长度的方差图的斜率，可以可视化和量化不同的误差源。然而，我发现 Allan 方差图的几个方面很难理解。例如，我不清楚为什么陀螺随机游走噪声出现在图表上，梯度为 -0.5。

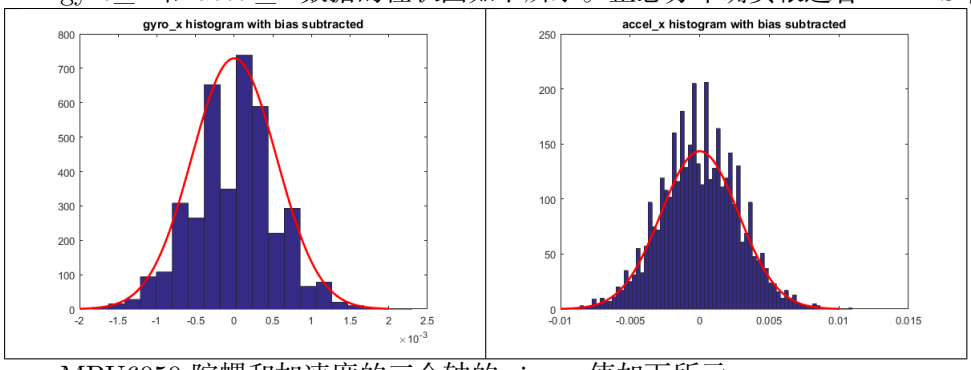
因此，我尝试了另一种方法来获得协方差矩阵元素的良好值。我从 MPU6050 装置 (从 Invensense 集成 MEMS 陀螺仪 + 加速度传感器) 中收集了静止装置的数据，并使用 Matlab 中的 fitdist 函数将高斯分布拟合到数据中。代码如下：

```
DEGTORAD = pi/180;
fs = 100;
ts = 1/fs;
% Scale raw gyro readings to obtain data in rad/sec
```

[4]译注：原文下两式在格式上有点瑕疵，式中应该用粗体 $\boldsymbol{g}$ 表示向量。

```
scaled_data_gyro = raw_data(:, 2:4)./GYRO_SCALE_FACTOR;
scaled_data_gyro_rad = scaled_data_gyro(:,:).*DEGTORAD;
% Scale raw accel readings to obtain data in units of g
scaled_data_accel = raw_data(:, 5:7)./ACCEL_SCALE_FACTOR;
% calculate accel and gyro bias by averaging the data.
accel_bias = mean(scaled_data_accel);
gyro_bias = mean(scaled_data_gyro_rad);

scaled_data_gyro_zero_mean = scaled_data_gyro_rad(:,:) - gyro_bias;
pd1 = fitdist(scaled_data_gyro_zero_mean(:,1), 'Normal');
pd2 = fitdist(scaled_data_gyro_zero_mean(:,2), 'Normal');
pd3 = fitdist(scaled_data_gyro_zero_mean(:,3), 'Normal');
gx_sigma = pd1.sigma;
gy_sigma = pd2.sigma;
gz_sigma = pd3.sigma;

% accels
scaled_data_accel_zero_mean = scaled_data_accel(:,:) - accel_bias;
pd1 = fitdist(scaled_data_accel_zero_mean(:,1), 'Normal');
pd2 = fitdist(scaled_data_accel_zero_mean(:,2), 'Normal');
pd3 = fitdist(scaled_data_accel_zero_mean(:,3), 'Normal');
ax_sigma = pd1.sigma;
ay_sigma = pd2.sigma;
az_sigma = pd3.sigma;
```

gyro_x 和 accel_x 数据的柱状图如下所示。正态分布确实很适合 MEMS 传感器数据。



MPU6050 陀螺和加速度的三个轴的 sigma 值如下所示

	x	y	z
Gyro	$5.4732e-04$	$6.1791e-04$	$6.2090e-04$
Accel	$2.8e-03$	$2.5e-03$	$3.8e-03$

根据这些数据，测量协方差矩阵 R 可以直接初始化为

$$R_{accel} = \begin{bmatrix} \sigma_{ax}^2 & 0 & 0 \\ 0 & \sigma_{ay}^2 & 0 \\ 0 & 0 & \sigma_{az}^2 \end{bmatrix} \tag{82}$$

式中，σ_{ax} 表示加速度计的 x 轴测量标准偏差的标准偏差，如上表所示。对于过程噪声协方差，我将方向对应的部分设置为：

$$Q_d(\delta\theta) = \begin{bmatrix} \sigma_{gx}^2 & 0 & 0 \\ 0 & \sigma_{gy}^2 & 0 \\ 0 & 0 & \sigma_{gz}^2 \end{bmatrix} \delta t \tag{83}$$

乘以 δt 模型在卡尔曼滤波器每次迭代中注入不确定性，并在实践中很好地工作，稍后我们将看到。当陀螺偏差变化非常缓慢时，对应于陀螺偏差的部分可以设置为非常低的值。

传感器模拟器

在我使用真实传感器和卡尔曼滤波的经验中，要使你的算法工作需要大量的尝试和错误。如果你试图直接使用真实数据，那么 bug 很难调试，因为有太多事情可能出错。因此，有必要建立一个模拟系统，利用选定的传感器参数对传感器进行建模，并生成可作为传感器融合算法输入的模拟测量值。

为了初始化模拟器，我创建了一个由一系列方向航路点组成的“旋转轨迹”。

```
sim_state.trajectory_rotation = 1*[0 0 0; 20 20 20; 20 0 0; 20 20 0;  30 30 30; 0 0
    0]*DEG2RAD;
```

然后我通过在一个路径点和下一个路径点之间插入来创建旋转路径。当从一个航路点过渡到另一个航路点时，这个简单的方案将产生急剧的变化。一个更复杂的系统将尝试做一些样条曲线拟合的变化，以创建一个平滑的轨迹而不突然改变。这里有一个很好的参考，描述了这个问题并讨论了各种解决方案：平滑轨迹。

```
for i = 1: num_wp-1
    % beginning quaternion
    qa = eul2quat(sim_state.trajectory_rotation(i,1), sim_state.trajectory_rotation
        (i,2), sim_state.trajectory_rotation(i,3));
    % ending quaternion
    qb = eul2quat(sim_state.trajectory_rotation(i+1,1), sim_state.
        trajectory_rotation(i+1,2), sim_state.trajectory_rotation(i+1,3));
    ab = sim_state.trajectory_accel(i,:); % sim_state.accel beginning
    ae = sim_state.trajectory_accel(i+1,:); % sim_state.accel end
    % number of points on each segment of the trajectory
    num_points_segment = sim_state.num_sim_samples/num_segments;
    for j = 1: num_points_segment
        sim_state.orientation(idx,:) = interpolate_quat(qa, qb, j/
            num_points_segment);
        ap = ap + (ae-ab)/num_points_segment;
        sim_state.accel(idx,:) = ap + g';
        sim_state.velocity(idx,:) = sim_state.velocity(idx-1,:)+(a_prev+ap)/2*
            time_step;
        sim_state.position(idx,:) = sim_state.position(idx-1,:) + (sim_state.
            velocity(idx,:)+sim_state.velocity(idx-1,:))/2*time_step;
        a_prev = ap;
        idx = idx + 1;
    end
end
end
```

这段代码还试图为位置和速度创建一个轨迹，但现在可以忽略，因为这些量不是我们状态向量的一部分。稍后我们将添加基于图像的测量。

为了生成测量数据，计算了从一个方向转换到另一个方向所需的角速度，并添加了随机噪声和偏差误差。

```
function [omega_measured, accel_measured, q_prev, dcm_true, q] = sim_imu_tick(
    sim_state, time_step, idx, sensor_model, q_prev)
if (idx < sim_state.num_sim_samples)
    q = sim_state.orientation(idx,:);
    dcm_true = quat2dc(q);
    % dcm rotates from IMU to Global frame, dcm' does the opposite
    dq = (q - q_prev);
    q_prev = q;
    q_conj = [q(1) -q(2) -q(3) -q(4)];
    % calculate the angular velocity required to go from one orientation to the
        next
    omega = 2*quatmult(q_conj, dq)/time_step;
    % take the vector part
    omega = omega(2:4);
    % add noise
    gyro_bias_noisy = sensor_model.gyro_bias + normrnd(0, sensor_model.
        gyro_bias_noise_sigma);
    omega_measured = omega' + normrnd(0, sensor_model.gyro_random_noise_sigma)+
        gyro_bias_noisy;
    accel_measured = dcm_true'*sim_state.accel(idx,:)' + sensor_model.accel_bias +
        normrnd(0, sensor_model.accel_noise_sigma);
    %   gyro_measurements(end+1,:) = omega_measured';
    %   accel_measurements(end+1,:) = accel_measured';
end
end
```

传感器模型是根据从 MPU6050 数据日志收集的统计数据初始化的，如上所述。

```
function sensor_model = init_sensor_model(sensor_model, time_step)
sensor_model.accel_noise_sigma = [0.003 0.003 0.004]'; % representative values for
    MPU6050
sensor_model.accel_noise_cov = (sensor_model.accel_noise_sigma).^2;
sensor_model.accel_bias = [-0.06 0 0]'; % actual MPU6050 bias

sensor_model.accel_bias_noise_sigma = [0.001 0.001 0.001];
sensor_model.R_accel = diag(sensor_model.accel_noise_cov);
sensor_model.Q_accel_bias = diag(sensor_model.accel_bias_noise_sigma.^2);
sensor_model.P_accel_bias = 0.1*eye(3,3);

sensor_model.gyro_random_noise_sigma = [5.4732e-04 6.1791e-04 6.2090e-04]'; %rad/
    sec, representative values for MPU6050
sensor_model.gyro_bias_noise_sigma = 0.00001*ones(3,1); %rad/sec/sec. Any small
    value appears to work
sensor_model.gyro_bias = [0.0127 -0.0177 -0.0067]'; % rad/sec
```

```
sensor_model.Q_gyro_bias = diag(sensor_model.gyro_bias_noise_sigma.^2);

% position uncertainity
sensor_model.Q_p = diag(sensor_model.accel_noise_cov)*time_step;
% velocity uncertainity
sensor_model.Q_v = diag(sensor_model.accel_noise_cov)*time_step;
% quaternion (orientation) uncertainity
sensor_model.Q_q = diag(sensor_model.gyro_random_noise_sigma.^2)*time_step;

% Initial values for various elements of the covariance matrix. Only
% effects the transient behaviour of the filter, not the steady state
% behaviour.

sensor_model.P_p = 0.1*eye(3,3);

sensor_model.P_v = 0.1*eye(3,3);

sensor_model.P_q = 0.1*eye(3,3);

sensor_model.P_gyro_bias = 0.1*eye(3,3);

sensor_model.P_accel_bias = 0.1*eye(3,3);

end
```

现在让我们来看看滤波器每次迭代期间的计算。

```
for i = 2:num_sim_samples
    idx = i;
    % Get current sensor values from the simulator
    [omega_measured, accel_measured, q_prev, dcm_true, q_true] = sim_imu_tick(
        sim_state, time_step, idx, sensor_model, q_prev);

    % Calculate estimated values of omega and accel by subtracting the
    % estimated values of the biases
    omega_est = omega_measured - sv.gyro_bias_est;
    accel_est = accel_measured - sv.accel_bias_est;

    phi         = omega_est*time_step;
    delta_vel   = accel_est*time_step; % not necessary for accel-gyro sensor fusion
    phi3 = make_skew_symmetric_3(phi);

    % update current orientation estimate. We maintain orientation estimate
    % as a quaternion and obtain the corresponding DCM whenever needed
    sv.q_est = apply_small_rotation(phi, sv.q_est);
    sv.dcm_est = quat2dc(sv.q_est);

    % Integrate accel (after transforming to global frame) to obtain
    % velocity and position
```

```
    orig_velocity = sv.velocity_est;
    sv.velocity_est = sv.velocity_est + sv.dcm_est*delta_vel - g*time_step ;
    final_velocity = sv.velocity_est;
    sv.position_est = sv.position_est + ((orig_velocity + final_velocity)/2)*
        time_step;

    % State transition matrix
    F = eye(6) +   [-phi3        eye(3)*time_step;
                   zeros(3,6)];

    % propagate covariance
    P = F*P*F' + Qd;
    % Apply accelerometer measurements.
    if (mod(i, accelUpdateFrequency) == 0)
        % apply accel measurements. The updated state vector and covariance
        % matrix are returned.
        [sv, P] = process_accel_measurement_update2(sv, accel_est, P, sensor_model.
            R_accel, g);
        gyro_bias_est    = sv.gyro_bias_est;
                q_est            = sv.q_est;
        dcm_est          = sv.dcm_est;
    end
end

function q_est = apply_small_rotation(phi, q_est)
s = norm(phi)/2;
x = 2*s;
phi4 = make_skew_symmetric_4(phi);
% approximation for sin(s)/s
sin_sbys_approx = 1-s^2/factorial(3) + s^4/factorial(5) - s^6/factorial(7);
exp_phi4 = eye(4)*cos(s) + 1/2*phi4*sin_sbys_approx;
% method 1
q_est1 = exp_phi4*q_est;
% verify: should be equal to:
% method 2 (direct matrix exponential)
q_est2 = expm(0.5*(phi4))*q_est;
% verify should be equal to:
% method 3: propagate by quaternion multiplication (doesn't preserve unit
% norm)
dq = [1 phi(1)/2 phi(2)/2 phi(3)/2];
dq_conj = [1 -phi(1)/2 -phi(2)/2 -phi(3)/2]';
q_est3 = quatmul(q_est, dq');
q_est = q_est1;
end

function phi4 = make_skew_symmetric_4(phi)
phi4 = [0        -phi(1)       -phi(2)       -phi(3);
    phi(1)       0             phi(3)       -phi(2);
```

```
    phi(2)     -phi(3)      0           phi(1);
    phi(3)      phi(2)     -phi(1)      0];
end
```

让我们看看如果在不应用任何加速度计更新的情况下运行此代码会发生什么 (我们稍后将讨论这些更新)。我们从零开始计算陀螺偏差的估计值。由于陀螺测量中存在偏差和随机噪声，我们期望估计的方位随每次迭代而逐渐偏离真实方位，误差状态的协方差也随之增大。正如下面的两个视频所显示的，这确实是发生了什么。

http://www.telesens.co/wp-content/uploads/2017/05/sf1_sim_rotating_cube-1.mp4

//www.telesens.co/wp-content/uploads/2017/05/sf1_sim_running_plot-1.mp4

现在我们添加加速度计更新。相关代码如下所示。代码实现了我们之前开发的数学公式。

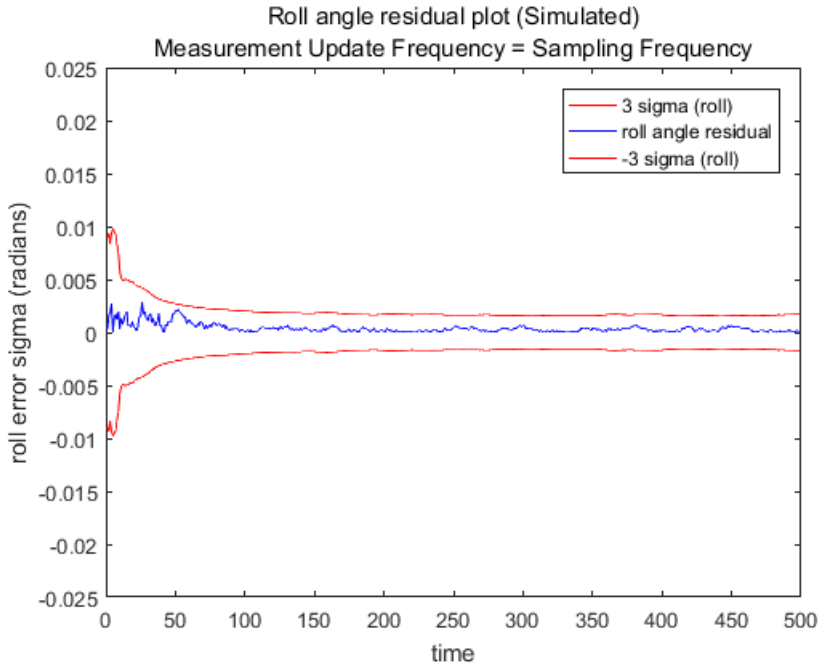
```
% reduced model, applies to accel+gyro only. No camera
function [state, P] = process_accel_measurement_update2(state, accel_est, P,
    R_accel, g)
% current estimated states
q_est    = state.q_est;
dcm_est = state.dcm_est;
gyro_bias_est = state.gyro_bias_est;
% Calculate residual (in the global reference frame)
accel_residual = dcm_est*accel_est - g;
% Construct the H matrix
H_accel = [-make_skew_symmetric_3(g)*dcm_est zeros(3,3)];
% Calculate Kalman Gain
K_gain = P*H_accel'*inv([H_accel*P*H_accel' + R_accel]);
% Calculate the correction to the error state
x_corr = K_gain*accel_residual;
% Calculate the new covariance matrix
P = P - K_gain*H_accel*P;
% Apply the updates to get the new state
x_est = apply_accel_update(x_corr, [q_est; gyro_bias_est]);

gyro_bias_est(1:3) = x_est(5:7);
q_est = x_est(1:4);
%ness = [x_corr(1:2) x_corr(4:5)]'*inv(P(1:2, 1:2))*x_corr(1:2);
ness = x_corr'*inv(P)*x_corr;

% Update state vector
state.q_est          = q_est;
state.dcm_est        = dcm_est;
state.gyro_bias_est = gyro_bias_est;
end

function x_est = apply_accel_update(x_corr, x_est)
% gyro bias
x_est(5:7) = x_est(5:7) + x_corr(4:6);
% orientation update
phi =  -[x_corr(1) x_corr(2) x_corr(3)];
```

```
phi3 = make_skew_symmetric_3(phi);
phi4 = make_skew_symmetric_4(phi);
s = norm(phi)/2;
sin_sbys_approx = 1-s^2/factorial(3) + s^4/factorial(5) - s^6/factorial(7);
exp_phi4 = eye(4)*cos(s) + 1/2*phi4*sin_sbys_approx;
x_est(1:4) = exp_phi4*x_est(1:4);
end
```

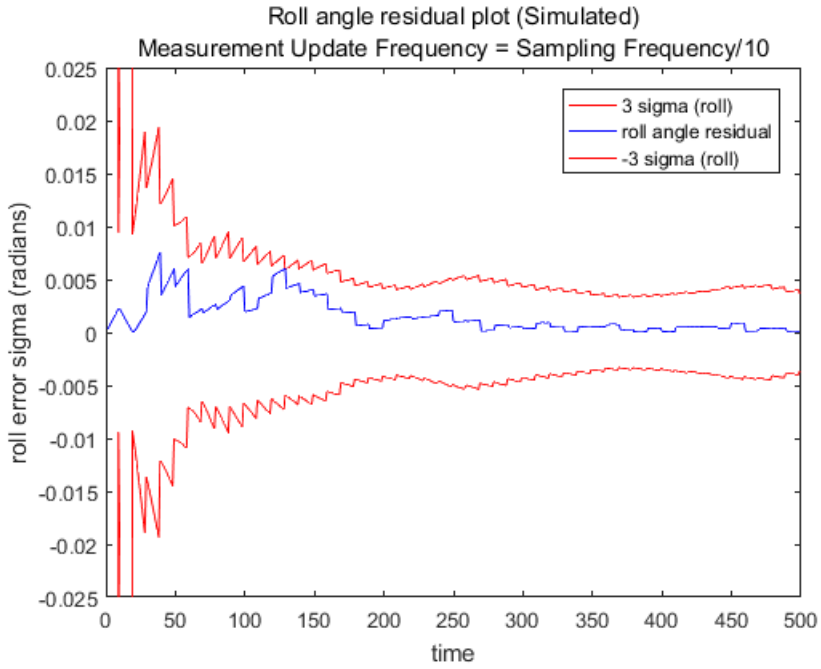
增加加速度计更新，正确估计陀螺偏差，并修正俯仰和横滚的漂移。最后估计的陀螺偏差如下所示。

	x	y	z
Estimated Gyro bias	0.0127	−0.0179	−0.0069

如 [2] 所述，观察卡尔曼滤波器性能是否良好的一个好方法是将残差 (真实状态和估计状态之间的差异) 的变化范围与协方差矩阵的相应元素进行比较。当滤波器进行优化时，残差应在相应的 sigma 界限内。例如，大约 99% 的残差应该在 $+-3\sigma$ 之间。我们显示了误差沿 x 轴的变化 (横滚角 roll 误差)，并将其沿 $+-3\sigma_x$ 绘制。σ_x 是通过取协方差矩阵第一个元素的平方根得到的。



在这里，加速度计测量更新被应用于迭代的每一步。如果我们不太频繁地应用更新，我们可以在 σ_x 的图表中看到锯齿度。锯齿度发生的原因是连续测量更新之间的协方差增加，并且在应用更新时降低。协方差在这两种情况下都会收敛，但是正如人们所预期的，当加速度计更新应用的频率较低时，协方差会收敛到较高的值。

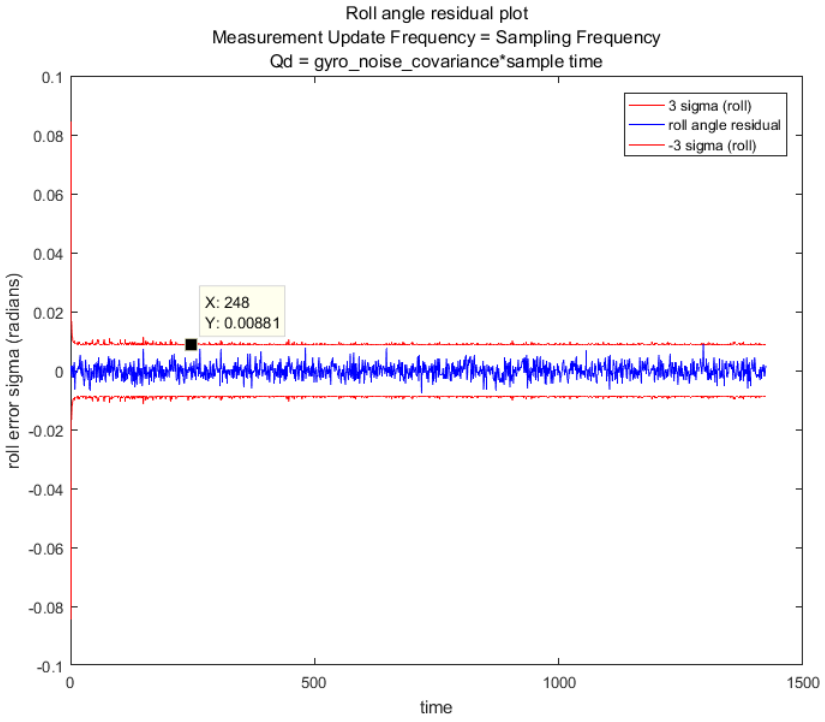


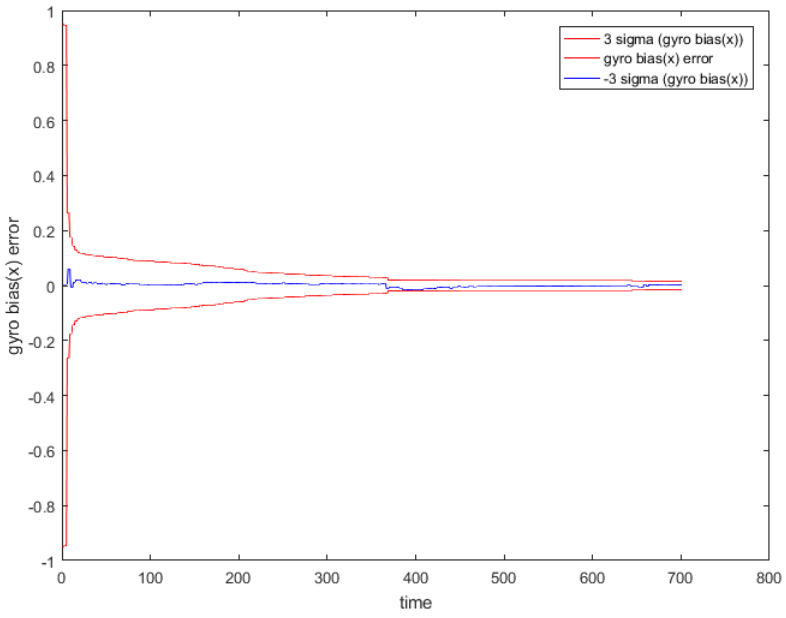
从方程式 (80) 可以看出，当旋转轴与当前方向对齐时，即为 $\hat{C}\delta\boldsymbol{\theta} = k\boldsymbol{g}$，其中 k 是一个标量。因此 $z = 0$。因此，沿当前方向 (局部垂直) 旋转不会改变加速度计的残差值，因此是不可观测的。

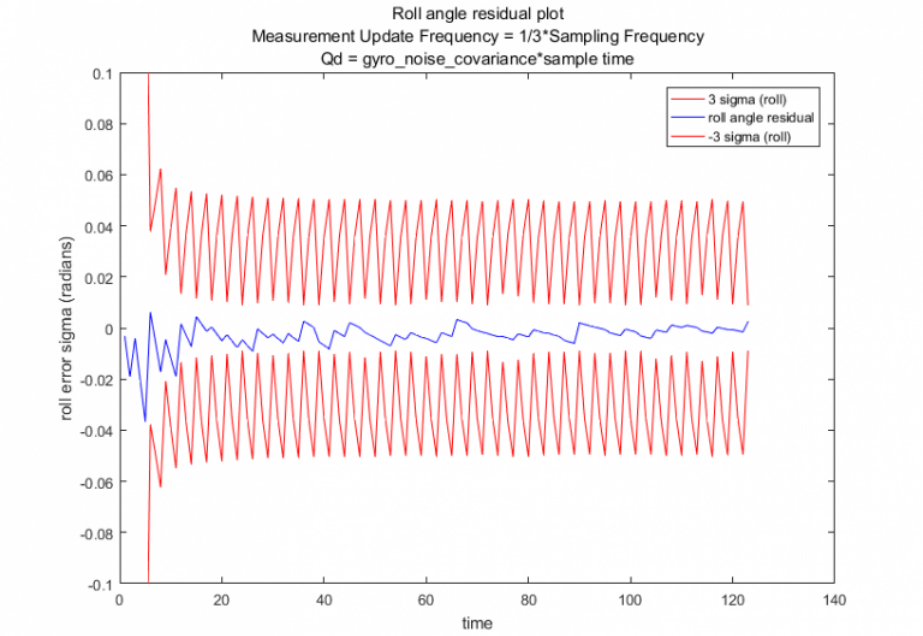
再观察几次。协方差矩阵初始值的选择只影响滤波器的暂态特性，不会改变协方差的最终 (收敛) 值。因此，可以将该矩阵设置为具有适当低值的对角矩阵。过程噪声协方差矩阵对最终协方差矩阵有一定的影响，可以作为陀螺和加速度测量的设计参数。我们把这个矩阵设置成一个对角矩阵，其项对应于方向误差等于陀螺测量标准偏差乘以时间步长的平方。直觉是，这反映了在每个更新步骤中注入的不确定性。当陀螺偏差变化非常缓慢时，与陀螺偏差误差相对应的项可以设置为非常低的值。

为了评估该滤波器在真实数据上的性能，我将卡尔曼滤波应用于串行端口上传输的 IMU 数据。通过 I2C 连接将 MPU6050 连接至 Arduino 收集数据，并通过串行端口连接将原始数据发送至 PC。Matlab 通过设置函数回调来提供通过串行端口接收数据的 API，这使得将数据源切换为实时数据而不是模拟数据变得容易 (请与我联系以获取代码)。除了一个差异外，现场数据的传感器融合结果与模拟数据的结果相似。如上所述，加速度计测量不能提供沿旋转轴方向变化的可观测性，因此我禁用了沿局部 z 轴的陀螺偏差误差更新。启用此更新将导致沿局部 z 轴的陀螺偏差变得不稳定，从而导致方向误差。

方向误差和陀螺偏差误差图以及相应的 3σ 如下所示。一般来说，如果没有昂贵的校准步骤，我们就不知道实时数据的真实方向。在我的例子中，我把 IMU 放在一个平面上，通过从加速度计数据中计算角度来测量其中一个轴的方向。


Roll angle residual plot
Measurement Update Frequency = Sampling Frequency
Qd = gyro_noise_covariance*sample time
3 sigma (roll)
roll angle residual
-3 sigma (roll)
X: 248
Y: 0.00881
roll error sigma (radians)
time
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
0
500
1000
1500


3 sigma (gyro bias(x))
gyro bias(x) error
-3 sigma (gyro bias(x))
gyro bias(x) error
time
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
100
200
300
400
500
600
700
800



加速度计更新应用于采样频率的 1/3。连续更新之间协方差估计的跳跃很容易看到。

在下一篇文章中，我们将扩展状态向量，包括位置和速度，并添加图像测量。

参考文献

1. Trawny N. Indirect Kalman Filter for 3D Attitude Estimation. MARS Lab; 2005:24. `http://www-users.cs.umn.edu/~trawny/Publications/Quaternions_3D.pdf`.

2. Labbe R. Kalman and Bayesian Filters in Python. None; 2017.

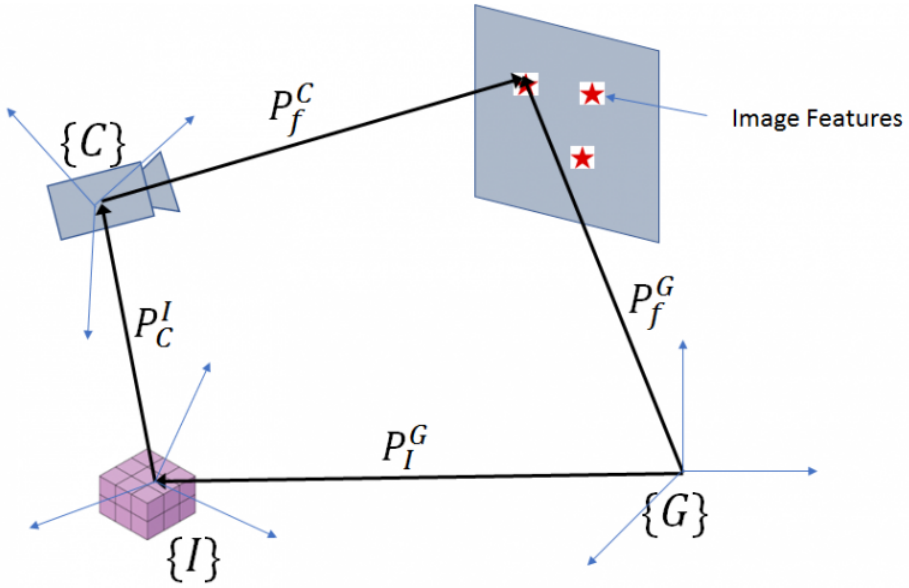
Part 4

在本文中，我们将添加数学并提供用于添加基于图像的测量的实现。让我们回顾一下本系列第 1 部分中首先介绍的符号和几何图形。

在 [1] 中的方程式 (16) 中的状态向量由 21 个元素组成[5]。

1. δP_I^G：IMU 在全局参考系中的位置误差
2. δV_I^G：IMU 在全局参考系中的速度误差
3. $\delta \theta_I^G$：IMU 和全局参考系之间的方向误差
4. δP_C^I：摄像头相对于 IMU 的位置误差
5. $\delta \theta_C^I$：摄像头和 IMU 之间的方向误差
6. δb_g：陀螺偏差误差
7. δb_a：加速度偏差误差

[5]译注：原文在此定义的两个偏差误差符号和后面公式不一致，$\delta b_g \rightarrow \delta \boldsymbol{g_b}$ 和 $\delta b_a \rightarrow \delta \boldsymbol{a_b}$。



P_I^G: Position of the IMU in the Global Frame

P_f^G: Position of the image feature in the Global Frame

$v_I = C_G^I v_G$ (DCM C_G^I transforms a vector v from frame G to frame I

$$P_C^G = P_I^G + C_I^G C_C^I P_C^I$$

这是一个 21 元素状态向量，相机-IMU 偏移作为其元素之一。我们将首先考虑一个更简单的问题，即 IMU 与相机对齐。因此，我们的状态向量包括 IMU(或摄像机) 相对于 (wrt) 全局帧和陀螺仪的位置、速度和方向误差以及加速度偏差误差。图像特征点提供了测量值，这些测量值将用于校正我们对位置、速度、方向、陀螺仪和加速度偏差的估计。

在前面的文章中，我们已经推导出了陀螺偏差和方向误差的状态传播方程。现在我们来推导加速度偏差、位置和速度的传播方程。如前所述，对于变量 x，x_m 表示测量的数量，$\hat{x}$ 表示估计的数量，δx 表示误差。根据运动定律，

$$\dot{P_I^G} = V_I^G \tag{84}$$

并且因此，

$$\delta\dot{P_I^G} = \delta V_I^G \tag{85}$$

我没有用粗体表示 P_I^G 因为从符号中可以明显看出，这些是向量量。

$$\delta V_I^G = \hat{V_I^G} - V_I^G \tag{86}$$

取导数，

$$\begin{aligned} \delta\dot{V_I^G} &= \dot{\hat{V}}_I^G - \dot{V}_I^G \\ \delta\dot{V_I^G} &= \hat{C}_I'^G \hat{\mathbf{a}} - C_I'^G \boldsymbol{a} \end{aligned} \tag{87}$$

注意，由于现在有多个 DCMs，我将明确地说明给定 DCM 所作用的坐标系。例如，C_I^G 将向量从全局转换为 IMU-相机 坐标系。另外，我用一个简单的符号来表示转置，既 $C^{'} = C^T$。

从本系列文章的第 2 篇开始，

$$\delta \dot{V_I^G} = \hat{C}_I^{'G}(\boldsymbol{a_m} - \hat{\boldsymbol{a_b}}) - (\hat{C}_I^{'G} + \hat{C}_I^{'G}[\delta\theta]_\times^{'})\boldsymbol{a} \tag{88}$$

现在，因为 $\boldsymbol{a_m} = \boldsymbol{a} + \boldsymbol{a_b} + \boldsymbol{n_r}$, $\delta \boldsymbol{a_b} = \hat{\boldsymbol{a_b}} - \boldsymbol{a_b}$ 和 $[\delta\theta]_\times^{'} = -[\delta\theta]_\times$,

$$\begin{aligned} \delta \dot{V_I^G} &= -\hat{C}_I^{'G}(\delta \boldsymbol{a_b}) + \hat{C}_I^{'G}[\delta\theta]_\times \boldsymbol{a} \\ \delta \dot{V_I^G} &= -\hat{C}_I^{'G}(\delta \boldsymbol{a_b}) + \hat{C}_I^{'G}[\delta\theta]_\times \boldsymbol{a} \end{aligned} \tag{89}$$

从第 1 篇文章，我们知道这个 $[a]_\times \boldsymbol{b} = -[b]_\times \boldsymbol{a}$。因此，

$$\delta \dot{V_I^G} = -\hat{C}_I^{'G}\delta \boldsymbol{a_b} - \hat{C}_I^{'G}[a]_\times \boldsymbol{\delta\theta} \tag{90}$$

与陀螺仪偏差相似，加速度偏差变化非常缓慢，因此加速度偏差中的误差可以假定为在时间步长内是恒定的。

$$\delta \dot{\boldsymbol{a_b}} = 0 \tag{91}$$

将方程式 (85)、(90)、(60)、(59)、(91) 与前几篇文章中提出的方向误差和陀螺偏差误差的状态传播方程相结合，给出了整个 15×15 状态转换矩阵：

$$\begin{bmatrix} \delta \dot{P_I^G} \\ \delta \dot{V_I^G} \\ \delta \dot{\boldsymbol{\theta}} \\ \delta \dot{\boldsymbol{g_b}} \\ \delta \dot{\boldsymbol{a_b}} \end{bmatrix} = \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -\hat{C}_I^{'G}[a]_\times & 0_{3\times3} & -\hat{C}_I^{'G} \\ 0_{3\times3} & 0_{3\times3} & [\phi]_\times & I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \begin{bmatrix} \delta P_I^G \\ \delta V_I^G \\ \delta \boldsymbol{\theta} \\ \delta \boldsymbol{g_b} \\ \delta \boldsymbol{a_b} \end{bmatrix} \tag{92}$$

因此，

$$J = \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -\hat{C}_I^{'G}[a]_\times & 0_{3\times3} & -\hat{C}_I^{'G} \\ 0_{3\times3} & 0_{3\times3} & [\phi]_\times & I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{93}$$

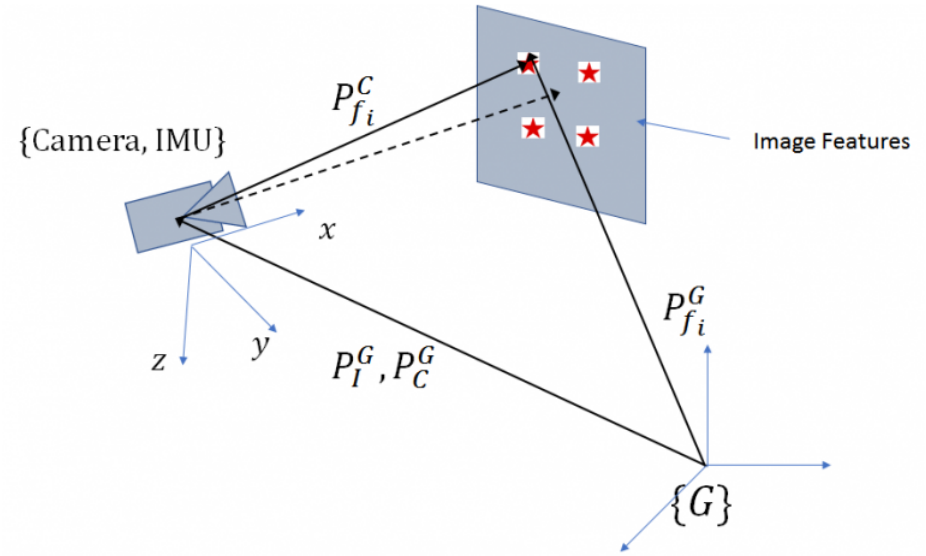
如前所述，状态转换矩阵 F 由以下公式给出：

$$F \approx I_{15\times15} + J\delta t \tag{94}$$

协方差传播为：

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{95}$$

现在我们来看看测量更新方程。假设 IMU 和相机重合，简化几何如下所示：



请注意，在我们的坐标系中，x 轴是沿着相机的光轴方向的，z 轴是向下的。这符合 MPU6050 IMU 上使用的惯例。在计算机视觉中，z 轴通常是沿着光轴指向的。

让特征点 f_i 的三维位置由全局坐标系中的 x_i、y_i、z_i 表示。用 K 表示摄像机校准矩阵，该点的投影，$x_i^{'}, y_i^{'}, z_i^{'}$ 给出为：

$$\begin{bmatrix} x_i^{'} \\ y_i^{'} \\ z_i^{'} \end{bmatrix} = KC_I^G(P_{f_i}^G - P_I^G) \tag{96}$$

这里 K 是一个 3×3 矩阵：

$$K = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{97}$$

这里 f 表示焦距，u_0 和 v_0 表示图像中心。我们假设一个正方形像素的直线相机没有径向和切向变形。如果需要，可以很容易地添加这些项。

通过应用透视分割得到最终图像坐标。

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} \frac{y_i^{'}}{x_i^{'}} \\ \frac{z_i^{'}}{x_i^{'}} \end{bmatrix} \tag{98}$$

雅可比矩阵由图像坐标的偏导数与我们试图估计的参数 (即 C_I^G 和 P_I^G) 组成。为了得到偏导数的表达式，将透视投影和坐标变换看作一个函数组合，然后应用链规则是很有用的。从链式法则来看，

$$D(f(g(x)) = f^{'}(g(x))g^{'}(x) \tag{99}$$

这里 $g(\boldsymbol{x})$ 是坐标转换函数，它将向量 $\boldsymbol{x}$ 从全局坐标系映射到 相机-IMU 坐标系中的向量 $\boldsymbol{x}^{'}$。$f(\boldsymbol{x}^{'})$ 然后在向量上应用透视除法 $\boldsymbol{x}^{'}$。

从简单的微分方程，

$$f^{'}(g(\boldsymbol{x})) = f^{'}(\boldsymbol{x}^{'}) = \begin{bmatrix} -\frac{y_i^{'}}{x_i^{'2}} & \frac{1}{x_i^{'}} & 0 \\ -\frac{z_i}{x_i^{'2}} & 0 & \frac{1}{x_i^{'}} \end{bmatrix} \tag{100}$$

让我们用 M 来表示用 $f^{'}(\boldsymbol{x}^{'})$ 表示的矩阵。

现在让我们考虑一下 $g^{'}(\boldsymbol{x})$

$$g^{'}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial \boldsymbol{x}}{\partial C_I^G} & \frac{\partial \boldsymbol{x}}{\partial P_I^G} \end{bmatrix} \tag{101}$$

根据第 1 部分中的向量演算结果，

$$g^{'}(\boldsymbol{x}) = \begin{bmatrix} KC_G^I[P_{f_i}^G - P_I^G]_\times \boldsymbol{\delta\theta} & -KC_G^I \delta P_I^G \end{bmatrix} \tag{102}$$

状态向量其他元素的导数为 0。因此 (在将 C_G^I 和 P_G^I 替换为其估计版本之后)，测量更新矩阵 H_{image} 给出如下：

$$H_{image} = M \begin{bmatrix} -K\hat{C}_G^I & 0_{3\times3} & K\hat{C}_G^I[P_{f_i}^G - \hat{P}_I^G]_\times & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{103}$$

模拟设置

我们的模拟装置包括一个摄像头和 IMU，它们彼此重合。在初始位置，相机的光轴沿着正 x 轴指向。相机在围绕光轴对称地位于几个单位之外的四个特征点上。假设这些特征点在全球坐标系中的位置已知。在模拟过程中，相机 + IMU 沿预先定义的轨迹平移和旋转。根据我们的传感器模型，通过计算增量角速度和加速度，加上偏差和噪声，由此产生模拟传感器测量。通过对特征点进行坐标变换和透视投影，加上高斯像素噪声，生成了模拟图像测量结果。

状态估计是通过集成带噪声的角速度和加速度 (在调整估计的陀螺和加速度偏差之后) 来获得估计的方向、位置和速度。状态协方差矩阵按方程式 (95) 传播。然后将噪声图像测量值用作测量源，以校正状态向量中的误差。

众所周知，将有噪声的加速度计数据进行积分来估计速度，再进行二次积分来获得位置，会导致误差的快速积累。位置和方向上的误差将导致估计的图像测量值偏离真实测量值 (两者之间的差异作为测量源)。下面的视频显示了这种漂移。视频中的摄像机保持静止，因此特征点的图像投影不应移动 (像素噪声引起的小扰动除外)。

`http://www.telesens.co/wp-content/uploads/2017/05/image_residuals-1.mp4`

我们也来看看我们的模拟轨迹。我们使用立方体作为道具来显示相机 + IMU 沿模拟轨迹的移动。

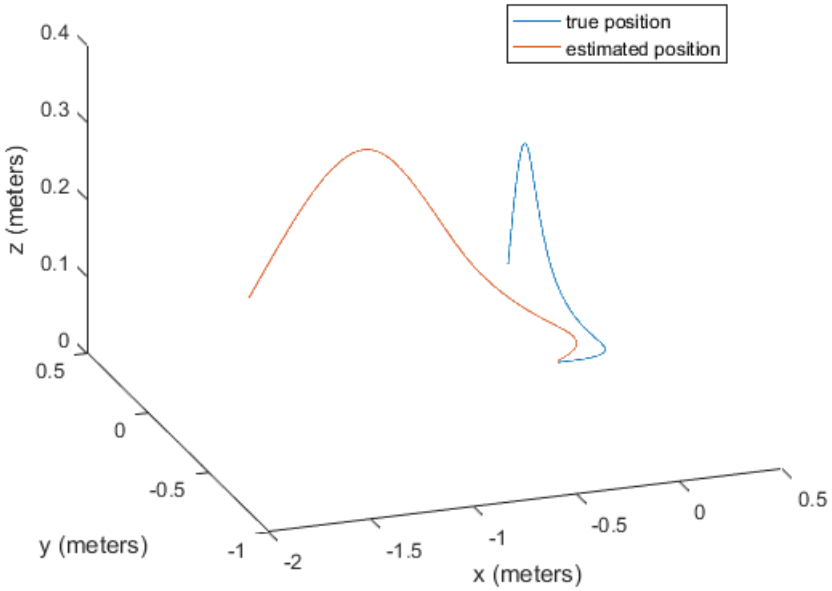
`http://www.telesens.co/wp-content/uploads/2017/05/cube_trajectory-1.mp4`

现在让我们来看一些实现卡尔曼滤波操作一次迭代的 Matlab 代码。

```
% Obtain simulated IMU measurements
[omega_measured, accel_measured, q_prev] = sim_imu_tick(sim_state, time_step,
    idx, sensor_model, q_prev);
% Obtain simulated camera measurements. p_f_in_G contains the feature
% locations and p_C_in_G the camera location in the global coordinate system
% sigma_image is the std. dev of the pixel noise, K the camera
% calibration matrix.
```

```
    image_measurements = sim_camera_tick(dcm_true, p_f_in_G, p_C_in_G, K,
        numFeatures, sigma_image);

    % Obtain estimated values by adjusting for current estimate of the
    % biases
    omega_est = omega_measured - sv.gyro_bias_est;
    accel_est = accel_measured - sv.accel_bias_est;

    % Multiply by time_delta to get change in orientation/position
    phi = omega_est*time_step;
    vel = accel_est*time_step;
    vel3 = make_skew_symmetric_3(vel);
    phi3 = make_skew_symmetric_3(phi);

    % Generate new estimates for q, position, velocity using equations of motion
        sv = update_state(sv, time_step, g, phi, vel);
    % State transition matrix
    F = eye(15) + [zeros(3,3) eye(3)*time_step zeros(3,3)   zeros(3,3)      zeros
        (3,3);
                   zeros(3,3) zeros(3,3)       sv.dcm_est*vel3 zeros(3,3)      -sv.
                       dcm_est*time_step;
                   zeros(3,3) zeros(3,3)        -phi3       eye(3)*time_step zeros
                       (3,3);
                   zeros(3,15);
                   zeros(3,15)];

    % Propagate covariance
    P = F*P*F' + Qd;
    % Apply image measurement update
    if (mod(i, image_update_frequency) == 0)
        [sv, P] = process_image_measurement_update(sv, p_f_in_G, image_measurements
            , ...
            numFeatures, P, K, imageW, imageH, sigma_image);
    end

function image_measurements = sim_camera_tick(dcm_true, p_f_in_G, p_C_in_G, K,
    numFeatures, sigma_image)
for fid = 1: numFeatures
    % feature position in camera frame
    fic = dcm_true'*(p_f_in_G(fid,:) - p_C_in_G')';
    % apply perspective projection and add noise
    image_measurements(2*fid-1:2*fid) = K*[fic(2)/fic(1) fic(3)/fic(1) 1]' ...
        + normrnd(0, [sigma_image sigma_image]');
end
end
```

让我们先看看如果不应用图像更新会发生什么。



我们可以看到，估计位置很快偏离了真实位置。现在我们来看看图像测量更新的代码。

```
% Iterate until the CF < 0.25 or max number of iterations is reached
    while(CF > 0.25 && numIter < 10)
        numIter = numIter + 1;
        estimated_visible = [];
        measurements_visible = [];
        residual = [];
        for indx = 1:numVisible
            % index of this feature point
            fid = visible(indx);
            % estimated position in the camera coordinate system
            fi_e = dcm_est'*(p_f_in_G(fid,:)' - p_C_in_G_est); % feature in image,
                estimated
                        % focal lengths (recall that the x axis of our coordinate
                        % system points along the optical axis)
            fy = K(1,1); fz = K(2,2);
            J =          [-fy*fi_e(2)/fi_e(1).^2   fy*1/fi_e(1)        0;
                    -fz*fi_e(3)/fi_e(1).^2       0                fz*1/fi_e(1)];

            % Measurement matrix
            H_image(2*indx-1:2*indx,:) = [-J*dcm_est' zeros(2,3) ...
                J*dcm_est'*make_skew_symmetric_3((p_f_in_G(fid,:)' - p_C_in_G_est))
                     zeros(2,3) zeros(2,3)];
            % estimated image measurement
            estimated_visible(2*indx-1:2*indx) = K*[fi_e(2)/fi_e(1) fi_e(3)/fi_e
                (1) 1]';
            % actual measurement
            measurements_visible(2*indx-1:2*indx) = measurements(2*fid-1:2*fid)';
        end
```

```
    % vector of residuals
    residual = estimated_visible - measurements_visible;
    %show_image_residuals(f4, measurements_visible, estimated_visible);
    % Kalman gain
    K_gain = P*H_image'*inv([H_image*P*H_image' + R_im]);
    % Correction vector
    x_corr = K_gain*residual';
    % Updated covariance
    P = P - K_gain*H_image*P;
    % Apply image update and correct the current estimates of position,
    % velocity, orientation, gyro/accel bias
    x_est = apply_image_update(KF_SV_Offset, x_corr, ...
        [position_est; velocity_est; q_est; gyro_bias_est; accel_bias_est]);

    position_est     = x_est(KF_SV_Offset.pos_index:KF_SV_Offset.pos_index+
        KF_SV_Offset.pos_length-1);
    velocity_est     = x_est(KF_SV_Offset.vel_index:KF_SV_Offset.vel_index+
        KF_SV_Offset.vel_length-1);
    gyro_bias_est    = x_est(KF_SV_Offset.gyro_bias_index:KF_SV_Offset.
        gyro_bias_index + KF_SV_Offset.gyro_bias_length-1);
    accel_bias_est   = x_est(KF_SV_Offset.accel_bias_index:KF_SV_Offset.
        accel_bias_index + KF_SV_Offset.accel_bias_length-1);
    q_est            = x_est(KF_SV_Offset.orientation_index:KF_SV_Offset.
        orientation_index+KF_SV_Offset.orientation_length-1);

    dcm_est = quat2dc(q_est);
    p_C_in_G_est = position_est;

    % Cost function used to end the iteration
    CF = x_corr'*inv(P)*x_corr + residual*R_inv*residual';
end
```

该代码遵循先前开发的图像测量更新公式。在上一篇文章中讨论的图像测量更新和加速度计更新之间的一个区别是，我们在一个循环中运行图像测量更新，直到 [1] 中的方程式 (24) 中定义的成本函数低于阈值或达到最大迭代次数。

让我们看看轨迹，图像更新应用于 1/10 的 IMU 采样频率和两个图像噪声 sigma 值。

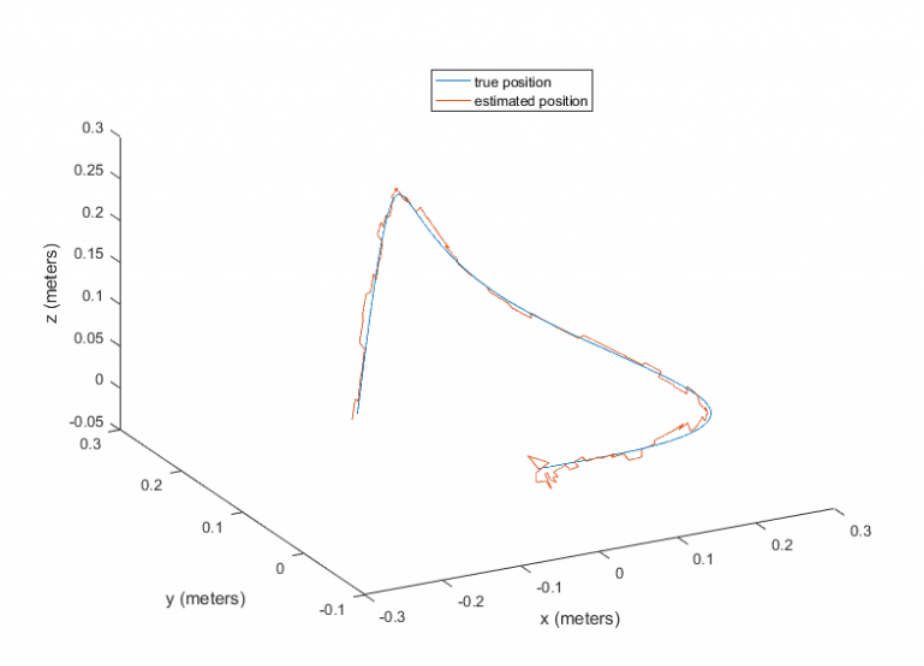


image noise sigma = 0.1

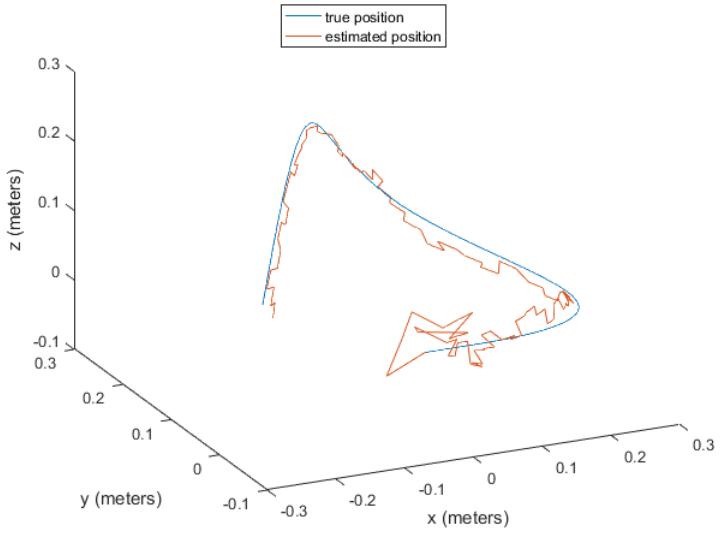


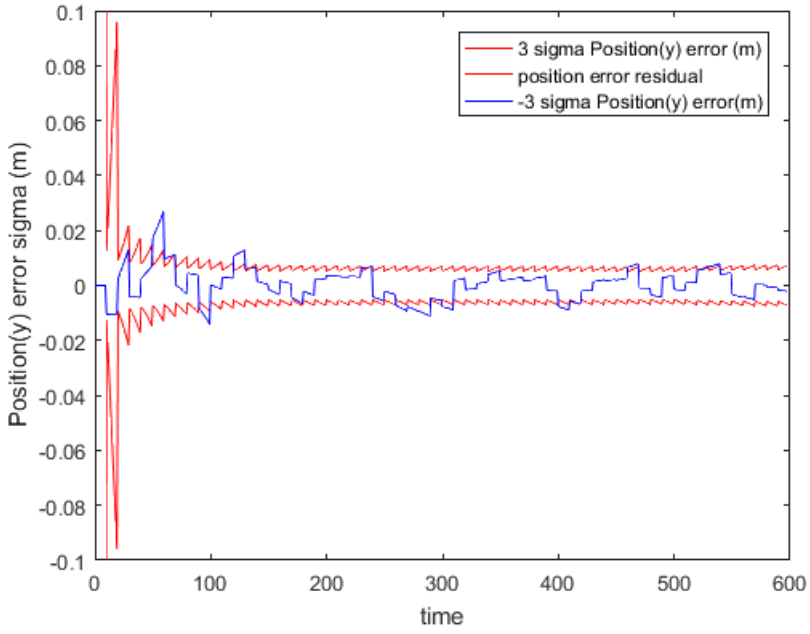
image noise sigma = 0.5

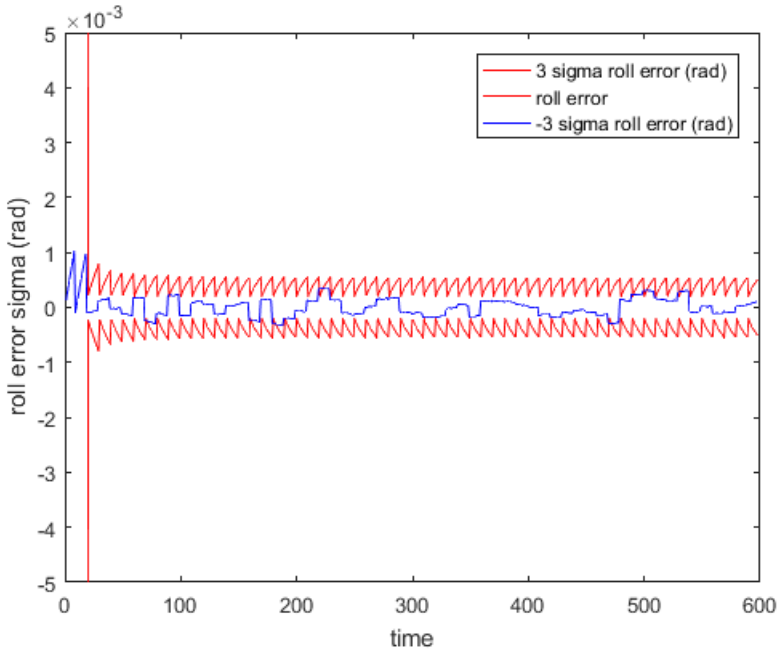
如我们所见，添加图像测量更新可防止位置出现任何系统误差，并保持估计位置在真实位置周围徘徊。如预期的那样，图像噪声 sigma 值越高，漂移量越大。在估计陀螺和加速度偏差并收敛到稳定状态的过程中，可能由于卡尔曼滤波而在起始点附近出现漂移。一旦偏差被估计并且协方差收敛，估计的位置就可以很好地跟踪真实位置。

系统还正确估计加速度和陀螺偏差。图像噪声 sigma 的两个不同值的真实偏差和估计偏差如下所示。

	Gyro (x)	Gyro (y)	Gyro (z)	Accel (x)	Accel (y)	Accel (z)
True	0.0127	−0.0177	−0.0067	−0.06	0	0
Estimated ($\sigma = 0.1$)	0.0126	−0.0176	−0.0067	−0.0581	0.0053	0.0010
Estimated ($\sigma = 0.5$)	0.0127	−0.0175	−0.0071	−0.0736	0.0031	−0.0092

位置 (沿 y 轴) 和方向 (横滚角 roll) 的残差图如下所示。残差大部分位于对应的 3σ 界限之间，因此滤波器似乎运行良好。





如 [2] 所示，保持协方差矩阵为 UD 因式分解形式，有利于提高数值稳定性和矩阵求逆的有效性。我们还没有实现这个优化。

在这里提供完整的源代码。

实现真正的系统

我们的模拟结果看起来很有希望，因此是时候实现一个真正的工作系统了。一个真正的系统将面临许多挑战-

- 在环境中选择合适的特征点并计算它们的三维位置并不容易。必须实现基于场景几何计算加上束调整的对极几何的一些变体。
- 必须设计出某种方法来获得三维特征点与其图像投影之间的对应关系。
- 图像处理通常比处理 IMU 更新慢得多。很可能在实际系统中，图像处理模块的更新在到达时就已过时，因此不能直接应用于更新当前的协方差矩阵。在协方差更新的实现中，必须考虑这个时间延迟。

参考文献

1. Mirzaei FM, Roumeliotis SI. A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation. IEEE Trans Robot. 2008;24(5):1143-1156. doi:10.1109/tro.2008.2004486
2. Maybeck P. Stochastic Models: Estimation and Control: Volume 2. Academic Press; 1982.