Homework 2

[Question 1]

If a Gaussian random variable vector $a \sim \mathcal{N}(0, \Sigma)$, where $Sigma = diag(b^2, ...b^2)$ (isotropic), then prove that with rotation matrix R, $E[Ra] = 0$, $cov[Ra] = \Sigma$

Proof:

For the mean, since rotation R is a constant, we can take it out of the expectation calculation. So it's easy to show that

$$E[Ra] = RE[a] = 0$$

For the covariance, we can use its definition and get

$$\begin{aligned} cov[Ra] &= E[(Ra - E[Ra])(Ra - E[Ra])^T] \\ &= E[(Ra - 0)(Ra - 0)^T] \\ &= E[Raa^T R^T] \\ &= RE[aa^T]R^T \end{aligned}$$

Then, since $E[aa^T] = \lambda I$, we can get:

$$RE[aa^T]R^T = \lambda RIR^T = \lambda I = \Sigma$$

[Question 2]

In the motion process code, decompose the F matrix and implement the motion equations for each type of state variable separately. Please provide the formulas and an explanation of the code implementation.

After linearizing the error kinematics especially around rotational term, we can get its matrix form:

$$\begin{aligned} \delta p_{k+1} &= \delta p_k + \delta v \Delta t \\ \delta v_{k+1} &= \delta v_k + (-R(\tilde{a} - b_a)^{\wedge} \delta\theta - R\delta b_a + \delta g)\Delta t - \eta_v \\ (\delta\theta)_{k+1} &\approx exp(-(\tilde{w} - b_g)\Delta t)\delta\theta - \delta b_g \Delta t - \eta_\theta \\ \delta(b_g)_{k+1} &= \delta(b_g)_k + \eta_{bg} \\ \delta(b_a)_{k+1} &= \delta(b_a)_k + \eta_{ba} \\ \delta g_{k+1} &= \delta g_k \end{aligned}$$

So,

$$\delta x^*_{k+1} = F\delta x_k$$

$$\Rightarrow$$

$$\begin{bmatrix} \delta p_{k+1}* \\ \delta v_{k+1}* \\ \delta \theta_{k+1}* \\ \delta b_{g,k+1}* \\ \delta b_{a,k+1}* \\ \delta g_{k+1}* \end{bmatrix} = \begin{bmatrix} I & I\Delta t & 0 & 0 & 0 & 0 \\ 0 & I\Delta t & -R(\tilde{a}-b_a)\hat{}\Delta t & 0 & -R\Delta t & I\Delta t \\ 0 & 0 & exp(-(\tilde{w}-b_g)\Delta t) & -I\Delta t & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} \delta p_k \\ \delta v_k \\ \delta \theta_k \\ \delta b_{g,k} \\ \delta b_{a,k} \\ \delta g_k \end{bmatrix}$$

CODE:

```
// Implement the state propagation for various error states
Vec18T dx_new = Vec18T::Zero();

dx_new.template block<3, 1>(0, 0) =
    dx_.template block<3, 1>(0, 0) +
    dx_.template block<3, 1>(3, 0) * dt;

dx_new.template block<3, 1>(3, 0) =
    dx_.template block<3, 1>(3, 0) +
    F.template block<3, 3>(3, 6) * dx_.template block<3, 1>(6, 0) +
    F.template block<3, 3>(3, 12) * dx_.template block<3, 1>(12, 0) +
    dx_.template block<3, 1>(15, 0) * dt;

dx_new.template block<3, 1>(6, 0) =
    F.template block<3, 3>(6, 6) * dx_.template block<3, 1>(6, 0) -
    dx_.template block<3, 1>(9, 0) * dt;

dx_new.template block<3, 1>(9, 0) = dx_.template block<3, 1>(9, 0);

dx_new.template block<3, 1>(12, 0) = dx_.template block<3, 1>(12, 0);

dx_new.template block<3, 1>(15, 0) = dx_.template block<3, 1>(15, 0);
```

[Question 3]

Use the left perturbation model to derive ESKF's kinematics and noise model. Also paste a code implementation.

Using:

$$b_{gt} = b_g + \delta b_g$$

Angular Error Derivative

We can write out the rotation matrix:

$$\text{Using R' = Rw:}$$
$$R'_t := R_t(\tilde{w} - b_{gt} - \eta_g)^\wedge$$

Meanwhile using the left perturbation:

$$R_t = \delta R R$$
$$\rightarrow R'_t = \delta R R' + \delta R' R$$
$$\text{Using} : \delta R = exp(\delta\theta^\wedge) :$$
$$\delta R' = exp(\delta\theta^\wedge)' = exp(\delta\theta^\wedge)(\delta\theta')^\wedge$$

Combine the two together

$$R'_t = \delta R R(\tilde{w} - b_{gt} - \eta_g)^\wedge = \delta R R' + \delta R' R$$
$$\rightarrow exp(\delta\theta^\wedge)R(\tilde{w} - b_{gt} - \eta_g)^\wedge = exp(\delta\theta^\wedge)R' + exp(\delta\theta^\wedge)(\delta\theta')^\wedge R$$
$$\text{Using} : R' = R[\tilde{w} - b_g]^\wedge$$
$$\rightarrow R(\tilde{w} - b_{gt} - \eta_g)^\wedge = R[\tilde{w} - b_g]^\wedge + (\delta\theta')^\wedge R$$
$$\text{Using} : \phi^\wedge R = R(R^T\phi)^\wedge :$$
$$\rightarrow R[\tilde{w} - b_{gt} - \eta_g]^\wedge = R[\tilde{w} - b_g]^\wedge + R(R^T(\delta\theta'))^\wedge$$
$$\rightarrow [\tilde{w} - b_{gt} - \eta_g] = [\tilde{w} - b_g] = R^T(\delta\theta')$$

Using lemma from question 1 : $R\eta_g = \eta_g$

$$\rightarrow \delta\theta' = R(-\delta b_g) - \eta_g$$

Velocity Error Derivative

From the definition of true value error and estimate error, we know

$$v'_t = v' + \delta v' := R_t(\tilde{a} - b_{at} - \eta_a) + g_t$$
$$\text{Meanwhile} : v' + \delta v' := R(\tilde{a} - b_a) + g + \delta v'$$

Expanding $R_t = Exp(\delta\theta)R$ and $b_{at} = b_a + \delta b_a$ gives:

$$v'_t = Exp(\delta\theta)R(\tilde{a} - b_a - \delta b_a - \eta_a) + g_t$$
$$\text{Using Taylor Expansion:}$$
$$\approx (I + \delta\theta^\wedge)R(\tilde{a} - b_a - \delta b_a - \eta_a) + g_t$$

Ignoring small values : $-\delta\theta^\wedge(\delta b_a - \eta_a)$

$$\approx R(\tilde{a} - b_a - \delta b_a - \eta_a) + \delta\theta^\wedge R(\tilde{a} - b_a) + g_t$$

To combine the above with the volocity error and using - $g_t = g + \delta g$ - $R\eta_a = \eta_a$ - $\delta\theta^\wedge m = -m^\wedge \delta\theta$

$$
\begin{aligned}
&R(\tilde{a} - b_a) + g + \delta v' \approx R(\tilde{a} - b_a - \delta b_a - \eta_a) + \delta\theta^\wedge R(\tilde{a} - b_a) + g + \delta g \\
&\rightarrow \\
&\delta v' \approx R(-\delta b_a) - \eta_a + \delta g - [R(\tilde{a} - b_a)]^\wedge \delta\theta
\end{aligned}
$$

The above is a linear form!

So all together,

$$
\begin{aligned}
\delta p' &= \delta v \\
\delta v' &= R(-\delta b_a) - \eta_a + \delta g - [R(\tilde{a} - b_a)]^\wedge \delta\theta \\
\delta\theta' &= R(-\delta b_g) - \eta_g \\
\delta b_g' &= \eta_{bg}' \\
\delta b_a' &= \eta_{ba}' \\
\delta g' &= 0
\end{aligned}
$$

Discrete Time Kinematics Model

$$
\begin{aligned}
\delta p_{k+1} &= \delta p_k + \delta v \Delta t \\
\delta v_{k+1} &= \delta v_k + (R(-\delta b_a) + \delta g - [R(\tilde{a} - b_a)]^\wedge \delta\theta)\Delta t - \eta_v \\
(\delta\theta)_{k+1} &\approx \delta\theta - R\delta b_g \Delta t - \eta_\theta \\
\delta(b_g)_{k+1} &= \delta(b_g)_k + \eta_{bg} \\
\delta(b_a)_{k+1} &= \delta(b_a)_k + \eta_{ba} \\
\delta g_{k+1} &= \delta g_k
\end{aligned}
$$

So F becomes:

$$
\begin{bmatrix}
I & I\Delta t & 0 & 0 & 0 & 0 \\
0 & I & -(R(\tilde{a} - b_a))^\wedge \Delta t & 0 & -R\Delta t & I\Delta t \\
0 & 0 & I & -R\Delta t & 0 & 0 \\
0 & 0 & 0 & I & 0 & 0 \\
0 & 0 & 0 & 0 & I & 0 \\
0 & 0 & 0 & 0 & 0 & I
\end{bmatrix}
$$

Predictions of states are:

$$
\begin{aligned}
p_{k+1,pred} &= p_k + v\Delta t + \frac{1}{2}(R(\tilde{a} - b_a))\Delta t^2 + \frac{1}{2}g\Delta t^2 \\
v_{k+1,pred} &= v_k + R(\tilde{a} - b_a)\Delta t + g\Delta t \\
\theta_{k+1,pred} &= Log(Exp((\tilde{w} - b_g)\Delta t)R_k) \\
bg_{k+1,pred} &= bg_k \\
ba_{k+1,pred} &= ba_k \\
g_{k+1,pred} &= g_k
\end{aligned}
$$

Predictions of errors are:

$$
\begin{aligned}
\delta x_{pred} &= F\delta x \\
P_{pred} &= FPF^T + Q
\end{aligned}
$$

GNSS Updates While the above is the general update, a dual-RTK-GPS can have a simplified observation model.

First, a dual-RTK-GPS system can output: $y = [R_{GNSS}, P_{GNSS}]$ - R_{GNSS}: orientation observation of the robot - P_{GNSS}: position observation of the robot

In general $y = h(x) \oplus v$, but here we think the same observation model also holds true for δx:

$$
\begin{aligned}
z_\theta &= Log(R_{GNSS}I^T) \\
z_{\delta\theta} &= Log(R_{GNSS}R^T)(\text{Left purterbation})
\end{aligned}
$$

See? TODO (I'm not sure the above is true)

This makes things easier, because this means our observation gives a direct observation of θ. we can directly get:

$$
H_\theta = \frac{\partial h}{\partial \delta\theta} = I
$$

Same thing with position update:

$$
H_p = \frac{\partial h}{\partial \delta p} = I
$$

In the meantime, innovation $y \ominus h(x_{pred}) = [\delta p, \delta\theta]$ and it is:

$$
y \ominus h(x_{pred}) = [p_{GNSS} - p, Log(R_{GNSS}R^T)]
$$

So:

$$H = \begin{bmatrix} I_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & I_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix}$$

Then the rest remains the same as EKF

$$\begin{aligned} K_{k+1} &= P^*_{k+1} H^T_{k+1} (V^{-1} + H_{k+1} P^*_{k+1} H^T_{k+1}) \\ P_{k+1} &= P^*_{k+1} - K_{k+1} H P^*_{k+1} \\ \delta x_{k+1} &= K_{k+1}(z - h(x^*_{k+1})) \end{aligned}$$

Covariance Matrix Of Errors After Resetting In discrete time, we approximate p_{k+1} as the true value p_t can define:

$$\begin{aligned} & x_{k+1} = x_k \oplus \delta x_k \\ \rightarrow & \\ & p_{k+1} = p_k + \delta p_k \\ & v_{k+1} = v_k + \delta v_k \\ & exp(\theta^\wedge_{k+1}) = exp(\delta\theta^\wedge_k) exp(\theta^\wedge_k) \\ & b_{g,k+1} = b_{g,k} + \delta b_{g,k} \\ & b_{a,k+1} = b_{a,k} + \delta b_{a,k} \\ & g_{k+1} = g_k + \delta g_k \end{aligned}$$

Since we have applied a correction, we can go ahead and reset $\delta x = 0$

However, we recognize that this correction may not update with the best reset. So, we need to adjust the error covariance before proceeding to the next step. We assume that after the reset δx_k, there's still an remeniscent error δx^+

The reset is to correct $x_{k+1} \sim (\delta x, P_k)$ to $x_{k+1} \sim (0, P_{reset})$. **For vector space variables `p, v, b_a, b_g, g` this reset is a simple shift of distribution. The covariance matrices stay the same.** For rotation variables θ though, this shift of distribution is in the tanget space (which is a vector space). But projected on to the `SO(3)` manifold, the distribution is not only shifted, but also scaled.

So, if we define:

- $\delta\theta^+$ is the error after reset. It is zero of course, but we are interested in the finding the Jacobian of that.

to find the new covariance matrix:

$$
\begin{aligned}
& exp(\delta\theta^+)exp(\delta\theta_k)R_k = exp(\delta\theta)R_k \\
\rightarrow \\
& exp(\delta\theta) = exp(\delta\theta^+)exp(\delta\theta_k) \\
& \rightarrow exp(\delta\theta^+) = exp(\delta\theta)exp(-\delta\theta_k) \\
& \text{Using BCH:} \theta^+ \approx -\delta\theta_k + \delta\theta - \frac{1}{2}\delta\theta^\wedge\delta\theta_k + o((\delta\theta_k)^2) \\
& = \theta^+ \approx -\delta\theta_k + \delta\theta + \frac{1}{2}\delta\theta_k^\wedge\delta\theta + o((\delta\theta_k)^2) \\
& \rightarrow \frac{\partial\theta^+}{\partial\delta\theta} = I + \frac{1}{2}\delta\theta_k^\wedge
\end{aligned}
$$

Then, the overall "Jacobian" for the covariance is:

$$
J_k = [I_3, I_3, I + \frac{1}{2}\delta\theta_k^\wedge, I_3, I_3, I_3]
$$

So the covariance reset is:

$$
P_{reset} = J_k P_{k+1} J_k
$$

Usually, this is close enough to identity because the θ covariance is small

Implementation

```
// Predict
template <typename S>
bool ESKF<S>::Predict(const IMU& imu) {
    ...
    // TODO
    // Left perturbation
    Mat18T F = Mat18T::Identity();                                                    // Main d
    // Populate the matrix based on the given mathematical structure
    F.template block<3, 3>(0, 3) = Mat3T::Identity() * dt;  // I * Δt for position to veloc

    F.template block<3, 3>(3, 3) = Mat3T::Identity() * dt;  // I * Δt for velocity to veloc
    F.template block<3, 3>(3, 6) = -SO3::hat(R_.matrix() * (imu.acce_ - ba_)) * dt;  // -(R
    F.template block<3, 3>(3, 12) = -(R_.matrix() * dt);  // -R * Δt for velocity to bias ac
    F.template block<3, 3>(3, 15) = Mat3T::Identity() * dt;  // I * Δt for velocity to grav
    F.template block<3, 3>(6, 6) = Mat3T::Identity();  // -R * Δt for orientation to bias ar
    F.template block<3, 3>(6, 12) = -(R_.matrix() * dt);  // -R * Δt for orientation to bias
}

void UpdateAndReset() {
```

```
    p_ += dx_.template block<3, 1>(0, 0);
    v_ += dx_.template block<3, 1>(3, 0);
    // TODO: left perturbation
    // R_ = R_ * SO3::exp(dx_.template block<3, 1>(6, 0));
    R_ = SO3::exp(dx_.template block<3, 1>(6, 0)) * R_ ;
    ...
}

void ProjectCov() {
    Mat18T J = Mat18T::Identity();
    // TODO
    // J.template block<3, 3>(6, 6) = Mat3T::Identity() - 0.5 * SO3::hat(dx_.template block<
    J.template block<3, 3>(6, 6) = Mat3T::Identity() + 0.5 * SO3::hat(dx_.template block<3,
    cov_ = J * cov_ * J.transpose();
}

template <typename S>
bool ESKF<S>::ObserveSE3(const SE3& pose, double trans_noise, double ang_noise) {
    ...
    innov.template head<3>() = (pose.translation() - p_);          //
    // TODO: left perturbation
    innov.template tail<3>() = (pose.so3() * R_.inverse()).log();  //   (3.67)
    // innov.template tail<3>() = (R_.inverse() * pose.so3()).log();  //   (3.67)
}
```

Image:

```
<p align="center">
   <figure>
        <img src="https://github.com/user-attachments/assets/9383183b-542d-41c4-b18e-7476e21
   </figure>
</p>
```