Visual SLAM - A Brief Introduction

Minh-Chung Hoang

Nanyang Technological University

hoang.chung@ntu.edu.sg

May 4, 2018

Overview

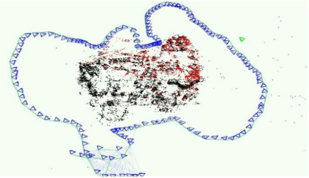
What is **vSLAM**

- Primary input data is in **v**isual form (images).
- Algorithms **S**imultaneously **L**ocalize the robot **A**nd **M**ap the environment.
    - Determine instantaneous pose (localizing / odometry)
    - Determine surrounding environment (mapping)
    - Relocalization (loop detection)
    - May include global optimization
- Why vSLAM?
    - (Relatively) cheap sensors
    - Rich information

vSLAM methods

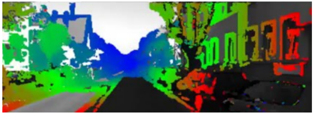
- Dense vs Sparse

(a) ElasticFusion



(b) ORB-SLAM 2

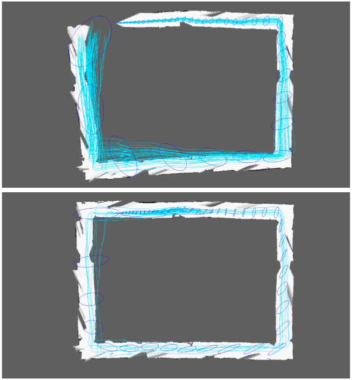
- Direct vs Indirect



(a) SVO

(b) ORB-SLAM 2

Standard SLAM system

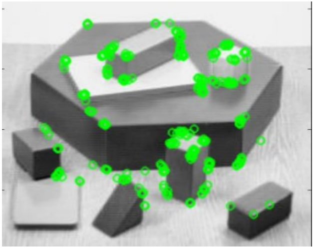


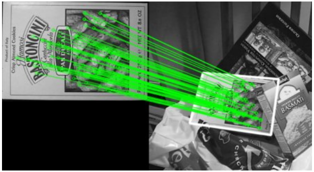
- Front-end
    - Features handling
    - Visual tracking
    - Pose tracking
    - Local optimization
    - Image encoding
- Back-end
    - Keyframes handling
    - Loop detection
    - Loop closure
    - Global optimization

Feature-based vSLAM

- Indirect method - tracking image keypoints
- Sparse mapping - mapping keypoints' location
- Why Feature-based vSLAM?
  - Robust against viewpoint and illumination changes
  - Good tracking accuracy via Bundle Adjustment
  - Robust relocalization via Bags-of-Words
  - Less processing power requirement

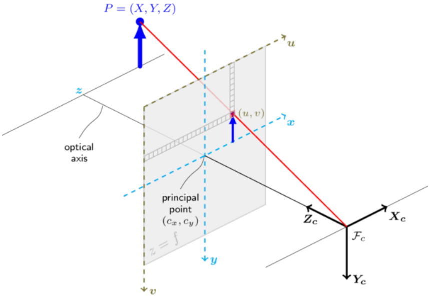
Image features





- Keypoint detection
  - Scale invariant
  - Rotation invariant
  - Robust against illumination changes
- Keypoint descriptor
  - Local patch encoding
  - Rotation encoding
  - Fast to compute
  - Fast to match

Camera projection model



$$P = [X, Y, Z]^T$$
$$p = [u, v]^T$$

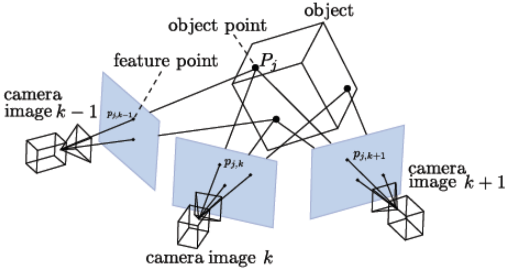
Projection geometry:

$$u = \frac{X}{Z} f_x + c_x - \frac{t_x}{Z} f_x$$
$$v = \frac{Y}{Z} f_y + c_y - \frac{t_y}{Z} f_y$$

Projection model: $h(T, P) = p$

$$\alpha \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & -t_x f_x \\ 0 & f_y & c_y & -t_y f_y \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Relative camera pose tracking - 1



Individual reprojection residue:

$$e_j(x, P_j) = h(x, P_j) - z_j$$

Total reprojection error:

$$S(x) = \sum_{j=1}^{M} ||e_j(x, P_j)||_{L2} = [\mathbf{h}(x) - \mathbf{z}]^T [\mathbf{h}(x) - \mathbf{z}]$$

Relative camera pose tracking - 2

The goal: Minimize reporjection error

$$x^* = \arg\min_x S(x)$$

The method: Iterative Gauss-Newton algorithm

$$\delta_i \leftarrow \left. \frac{\partial S(x_i + \delta)}{\partial \delta} \right|_{\delta=0} = 0$$

$$x_{i+1} \leftarrow x_i + \delta_i$$

Relative camera pose tracking - 3

For better readability, the symbols are further simplified. Let:

$$\mathbf{e}_x = \mathbf{e}(x) = \mathbf{h}(x) - \mathbf{z}$$
$$S(x) = \mathbf{e}_x^T \Omega \mathbf{e}_x$$

Applying first-order Taylor approximation:

$$\mathbf{e}_{x+\delta} = \mathbf{e}(x+\delta) \cong \mathbf{e}_x + \mathbf{J}_x \delta \quad , \quad \mathbf{J}_x = \left.\frac{\partial \mathbf{e}_{x+\delta}}{\partial \delta}\right|_{\delta=0} \cong \frac{\partial \mathbf{h}(x)}{\partial \delta}$$

Hence:

$$\begin{aligned} S(x+\delta) &\cong (\mathbf{e}_x + \mathbf{J}_x\delta)^T \Omega (\mathbf{e}_x + \mathbf{J}_x \delta) \\ &= \mathbf{e}_x^T \Omega \mathbf{e}_x + \mathbf{e}_x^T \Omega \mathbf{J}_x \delta + \delta^T \mathbf{J}_x^T \Omega \mathbf{e}_x + \delta^T \mathbf{J}_x^T \Omega \mathbf{J}_x \delta \end{aligned}$$

Relative camera pose tracking - 4

As derived previously:

$$S(x+\delta) \cong \mathbf{e}_x^T \Omega \mathbf{e}_x + \mathbf{e}_x^T \Omega \mathbf{J}_x \delta + \delta^T \mathbf{J}_x^T \Omega \mathbf{e}_x + \delta^T \mathbf{J}_x^T \Omega \mathbf{J}_x \delta$$

Taking partial derivative wrt. δ and evaluate at x_i:

$$\left.\frac{\partial S(x_i+\delta)}{\partial \delta}\right|_{\delta=0} = 2\mathbf{e}_{x_i}^T \Omega \mathbf{J}_{x_i} + 2\delta^T \mathbf{J}_{x_i}^T \Omega \mathbf{J}_{x_i}$$

Therefore, in order to minimize reprojection error:

$$\left.\frac{\partial S(x_i+\delta)}{\partial \delta}\right|_{\delta=0} = 0 \quad \Rightarrow \quad \mathbf{e}_{x_i}^T \Omega \mathbf{J}_{x_i} = -\delta_i^T \mathbf{J}_{x_i}^T \Omega \mathbf{J}_{x_i}$$

Or, most importantly:

$$\mathbf{J}_{x_i}^T \Omega \mathbf{J}_{x_i} \delta_i = -\mathbf{J}_{x_i}^T \Omega \mathbf{e}_{x_i}$$

Gauss-Newton algorithm for camera pose estimate

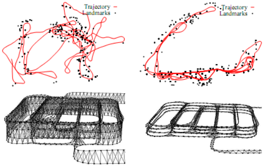
Incremental correction calculation

$$\mathbf{J}_{x_i}^T \Omega \mathbf{J}_{x_i} \delta_i = -\mathbf{J}_{x_i}^T \Omega \mathbf{e}_{x_i}$$

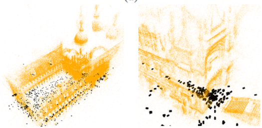
1. Initialize pose estimate $x_{i=0} = x_{init}$
2. Collect sensor measurement $\mathbf{z}_i$
3. Evaluate measurement residue $\mathbf{e}_{x_i} = \mathbf{h}(x_i) - \mathbf{z}_i$
4. Initialize measurement covariance Ω
5. Evaluate Jacobian $\mathbf{J}$ at x_i (or $\mathbf{J}_{x_i}$)
   1. Evaluate $A = \mathbf{J}_{x_i}^T \Omega \mathbf{J}_{x_i}$
   2. Evaluate $B = -\mathbf{J}_{x_i}^T \Omega \mathbf{e}_{x_i}$
6. Solve equation $A\delta = B$ for δ_i
7. Update $x_{i+1} = x_i + \delta_i$
8. Repeat from step 2 (till convergent)

Gauss-Newton algorithm implementation on g^{2}o

g^{2}o: A General framework for (Hyper) Graph Optimization



(a)


(b)

1. Vertex
   - Current state `setEstimate(.)`
2. Edge
   - Logic functions (eg. $h(X,P)$)
   - Residual `computeError(.)`
   - Jacobian `linearizeOplus(.)`
   - State update `oplusImpl(.)`

Optimization on Manifold

Translation $\mathbf{t} \in \mathbb{R}^3$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}' = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Rotation $\mathbf{R} \in \mathbb{R}^{3\times3}$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}' = \begin{bmatrix} c\psi c\phi & -s\psi c\theta + c\psi s\phi s\theta & s\psi s\theta + c\psi s\phi c\theta \\ s\psi c\phi & c\psi c\theta + s\psi s\phi s\theta & -c\psi s\theta + s\psi s\phi c\theta \\ -s\phi & c\phi s\theta & c\phi c\theta \end{bmatrix}_{\psi\phi\theta} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Transformation in homogeneous coordinate $\mathbf{T} \in \mathbb{R}^{4\times4}$

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}' = \left[\begin{array}{c|c} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \hline \mathbf{0}_{1\times3} & 1 \end{array}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Lie Group

- Lie (read as Lee :P)
- is a subset of $\mathbb{R}^N$
- is a **group**
  - Closure (if $X, Y, Z \in G$ then $XY \in G$)
  - Associativity ($(XY)Z = X(YZ)$)
  - Identity (there exists $I \in G$ that $IX = XI = X$)
  - Inverse (there exists X^{-1} that $XX^{-1} = X^{-1}X = I$)
- is a **smooth & differentiable manifold** $\mathfrak{M}$ in $\mathbb{R}^N$
  - every point $p \in \mathfrak{M}$ has local Euclidean tangent space
- has an associated **Lie algebra** $\mathfrak{m}$
  - same k Degree-of-Freedom
  - re-representation **alg**: $m \mapsto \mathfrak{m}$
  - exponential map **exp**: $\mathfrak{m} \mapsto \mathfrak{M}$
  - logarithm map **log**: $\mathfrak{M} \mapsto \mathfrak{m}$

Group $\mathfrak{SO}(3)$ and associated Lie algebra $\mathfrak{so}(3)$

- Rotation matrices belong to group $\mathfrak{SO}(3)$ (read as SO3)
- $\mathfrak{so}(3)$ algebra **generators**: $\textbf{alg}(m) = \sum_{i=1}^{k} m_i \mathbf{G}_i$

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \mathbf{G}_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Re-representation:

$$\textbf{alg}\left(\begin{bmatrix} \theta \\ \phi \\ \psi \end{bmatrix} \right) = \begin{bmatrix} 0 & -\psi & \phi \\ \psi & 0 & -\theta \\ -\phi & \theta & 0 \end{bmatrix} = \begin{bmatrix} \theta \\ \phi \\ \psi \end{bmatrix}_{\times}$$

Group $\mathfrak{SO}(3)$ and associated Lie algebra $\mathfrak{so}(3)$

Let $\omega = [\theta, \phi, \psi]^T$ and $\mathbf{R} \in \mathfrak{SO}(3)$ the rotation matrix

- Re-representation: $\mathbf{alg} : \mathbb{R}^3 \mapsto \mathfrak{so}(3)$

$$\mathfrak{w} = \mathbf{alg}(\omega) = [\omega]_\times$$

- Exponential mapping: $\mathbf{exp} : \mathfrak{so}(3) \mapsto \mathfrak{SO}(3)$

$$\mathfrak{W} = \mathbf{exp}(\mathfrak{w}) = e^{[\omega]_\times} = \mathbf{I}_{3\times 3} + \frac{sin|\omega|}{cos|\omega|}[\omega]_\times + \frac{1 - cos|\omega|}{|\omega|^2}[\omega]_\times^2$$

- Logarithm mapping: $\mathbf{log} : \mathfrak{SO}(3) \mapsto \mathfrak{so}(3)$

$$\mathfrak{w} = ln(\mathbf{R}) \qquad \omega = \mathfrak{w}_\bigtriangledown = [ln(\mathbf{R})]_\bigtriangledown$$

Group $\mathfrak{SE}(3)$ and associated Lie algebra $\mathfrak{se}(3)$

- Rigid transforms belong to group $\mathfrak{SE}(3)$ (read as SE3)
- $\mathfrak{se}(3)$ algebra **generators**: **alg**$(m) = \sum_{i=1}^{k} m_i \mathbf{G}_i$

$$\mathbf{G}_1 = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right], \mathbf{G}_3 = \left[\begin{array}{ccc|c} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right], \mathbf{G}_5 = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathbf{G}_2 = \left[\begin{array}{ccc|c} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right], \mathbf{G}_4 = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right], \mathbf{G}_6 = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

Group $\mathfrak{SE}(3)$ and associated Lie algebra $\mathfrak{se}(3)$

Let $\mathbf{v} = [\mathbf{t}^T, \omega^T]^T$ and $\mathbf{T} \in \mathfrak{SE}(3)$ the rigid transform

- Re-representation: $\mathbf{alg} : \mathbb{R}^6 \mapsto \mathfrak{se}(3)$

$$\mathfrak{v} = \mathbf{alg}(\mathbf{v}) = \begin{bmatrix} [\omega]_\times & \mathbf{t} \\ 0 & 0 \end{bmatrix}$$

- Exponential mapping: $\mathbf{exp} : \mathfrak{se}(3) \mapsto \mathfrak{SE}(3)$

$$\mathfrak{V} = \mathbf{exp}(\mathfrak{v}) = e^{\mathfrak{v}} = \begin{bmatrix} e^{[\omega]_\times} & \mathbf{At} \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{A} = \mathbf{I}_{3\times 3} + \frac{1 - cos|\omega|}{|\omega|^2}[\omega]_\times + \frac{|\omega| - sin|\omega|}{|\omega|^3}[\omega]_\times^2$$

- Logarithm mapping: $\mathbf{exp} : \mathfrak{SE}(3) \mapsto \mathfrak{se}(3)$

$$\omega = \mathfrak{w}_\triangledown = [ln[\mathbf{T}_{(1,1)\ldots(3,3)}]_{3\times 3}]_\triangledown \quad \mathbf{t} = \mathbf{A}^{-1}[\mathbf{T}_{(1,4)\ldots(3,4)}]_{3\times 1}$$

☹ Why all the hassles? Let's recap

Method: We decided to use **Gauss-Newton** (holy eqn below).

$$\mathbf{J}_{x_i}^T \Omega \mathbf{J}_{x_i} \delta_i = -\mathbf{J}_{x_i}^T \Omega \mathbf{e}_{x_i}$$

To get state update δ_i, practically speaking, we need:

- $\mathbf{e}_x$ residual $(\mathbf{f}(\mathbf{x}) - \mathbf{z})$
- Ω: covariance
- $\mathbf{J}_x$: Jacobian

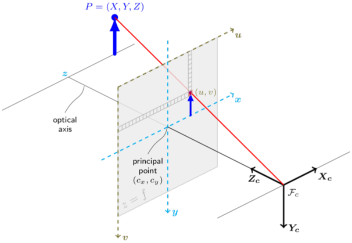
Problem with Jacobian:

- typically $\mathbf{x} \in \mathfrak{SE}(\mathbf{3})$. Numerically, $\triangle \mathbf{x}$ has 16 dimensions.
- redundant numerical! Not sensible to differentiate!
- actually need to differentiate wrt. something of 6 DoF

Solution by Lie algebra:

- has 6 DoF via 6 **generators**, with mapping with $\mathbb{R}^6$
- unique bidirectional mapping between $\mathfrak{se}(3)$ and $\mathfrak{SE}(3)$
- eg: $\mathbf{f}(\mathbf{x_0} \oplus \triangle \mathbf{x}) \equiv \mathbf{f}(e^{[\epsilon]_\times} \oplus \mathbf{x_0}) \equiv \mathbf{f}_{\mathbf{x}=\mathbf{x_0}}(\epsilon)$

Example application: Point-to-cam projection (1)



Let us have:
$X \in \mathfrak{SE}(3)$: camera pose
$\epsilon \in \mathbb{R}^6$: small pose change
$[\epsilon]_\times \in \mathfrak{se}(3)$: small pose change
$P \in \mathbb{R}^4$: point in world frame
$p \in \mathbb{R}^3$: projected image point

Therefore:
$e^{[\epsilon]_\times}$: pose change in $\mathfrak{SE}(3)$
$e^{[\epsilon]_\times} \oplus X$: new camera pose
$X \oplus P$: point in camera frame

$e^{[\epsilon]_\times} \oplus X \oplus P$: point in new camera frame
$h(e^{[\epsilon]_\times} \oplus X \oplus P)$: new projected image point

Example application: Point-to-cam projection (2)

- **Jacobian computation** (finally...)

Let $X \oplus P = P' = [p'_x \; p'_y \; p'_z]^T$

$$\mathbf{J} = \left.\frac{\partial h(e^{[\epsilon]_\times} \oplus X \oplus P)}{\partial \epsilon}\right|_{\epsilon=0} \cong \frac{\partial h(X \oplus P)}{\partial (X \oplus P)} \frac{\partial (e^{[\epsilon]_\times} \oplus X \oplus P)}{\partial \epsilon}$$

$$= \frac{\partial h(P')}{\partial P'} \frac{\partial (e^{[\epsilon]_\times} \oplus P')}{\partial \epsilon} = \begin{bmatrix} \frac{f_x}{p'_z} & 0 & -\frac{f_x p'_x}{{p'_z}^2} \\ 0 & \frac{f_y}{p'_z} & -\frac{f_y p'_y}{{p'_z}^2} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3\times 3} & -[P']_\times \end{bmatrix}$$

$$= \begin{bmatrix} \frac{f_x}{p'_z} & 0 & -\frac{f_x p'_x}{{p'_z}^2} & -f_x \frac{p'_x p'_y}{{p'_z}^2} & f_x (1 + \frac{{p'_x}^2}{{p'_z}^2}) & -f_x \frac{p'_y}{p'_z} \\ 0 & \frac{f_y}{p'_z} & -\frac{f_y p'_y}{{p'_z}^2} & -f_y (1 + \frac{{p'_y}^2}{{p'_z}^2}) & f_y \frac{p'_x p'_y}{{p'_z}^2} & f_y \frac{p'_x}{p'_z} \end{bmatrix}$$

Example application: Point-to-cam projection (3)

Camera pose iterative update - Algorithm in a nutshell

1. Initialize N map point estimates P_n (world frame)
2. Initialize measurement covariance matrix Ω
3. Initialize first pose estimate X_0 (world frame)
4. Evaluate map points in camera frame: $P'_n = X \oplus P_n$
5. Evaluate Jacobian matrix (the mess in the previous slide)
6. Stack all Jacobians and covariances into "big" $\mathbf{J}_x$ and Ω
7. Evaluate pose increment ϵ by solving $\mathbf{J}_x^T \Omega \mathbf{J}_x \epsilon = -\mathbf{J}_x^T \Omega \mathbf{e}_x$
8. Map ϵ into $\mathfrak{SE}(3)$: $e^{[\epsilon]_\times}$
9. Update camera pose $X \leftarrow e^{[\epsilon]_\times} \oplus X$
10. Return to step 4 (And fasten your seat belt! 😎)

Some practical concerns (that I can think of)

- Step 1: Map point initialization with...monocular camera
- Step 3: Initial pose estimate
  - significantly affect convergence and estimation accuracy
  - zero doesn't always work
  - possible motion model: zero velocity or zero acceleration
- Step 4: Data association
  - Which map point to which image point
  - Correspondences between consecutive images
  - Correspondences between non-consecutive images

Adding an active depth sensor

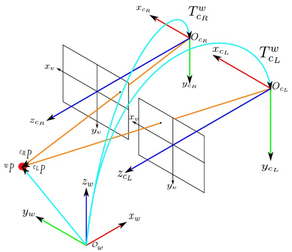
- Advantages
    - Directly measure points' position
    - Relatively high accuracy
    - Low computation requirement
- Disadvantages
    - May not work outdoor
    - May have limited range
    - Cloud-image association
    - Quality/Cost ratio

Adding a (synchronized) camera - **Stereo vision**

- Advantages
  - Relative low cost
  - Points' position observable
  - Rich data...now doubles

- Disadvantages
  - Costly stereo match search
  - Synchronization mechanism

Stereoscopic vision model



$$P = [X\ Y\ Z]^T$$

$$p_L = [u_L\ v_L]^T \qquad p_R = [u_R\ v_R]^T$$

Projection geometry:

$$v_L = v_R = \frac{Y}{Z} f_y + c_y$$

$$u_L = \frac{X}{Z} f_x + c_x$$

$$u_R = \frac{X}{Z} f_x + c_x - \frac{t_x}{Z} f_x$$

Disparity and point's position

$$d = u_L - u_R = \frac{t_x f_x}{Z} \qquad Z = \frac{t_x f_x}{d}$$

$$X = \frac{u_L - c_x}{f_x} Z \qquad Y = \frac{v_L - c_y}{f_y} Z$$

Stereoscopic vision model

$$P_{cam} = [X\ Y\ Z]^T \qquad p_L = [u_L\ v_L]^T \qquad p_R = [u_R\ v_R]^T$$

Measurement model: $h(P_{cam}) = [u_L\ v_L\ v_R]^T$

$$h\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \frac{X}{Z} f_x + c_x \\ \frac{Y}{Z} f_y + c_y \\ \frac{X}{Z} f_x + c_x - \frac{t_x}{Z} f_x \end{pmatrix}$$

Derivative:

$$\frac{\partial h(P_{cam})}{\partial P_{cam}} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \\ \frac{f_x}{Z} & 0 & -\frac{f_x (X - t_x)}{Z^2} \end{bmatrix}$$

Stereoscopic vision model

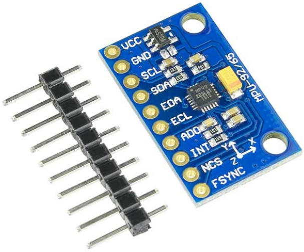
Assume map point in camera frame. (If not, just transform 😎).

$$P \equiv P_{cam} = [X\ Y\ Z]^T$$

Similar to monocular model, we can formulate Jacobian as:

$$\begin{aligned}
\mathbf{J} &= \left.\frac{\partial h(e^{[\epsilon]_\times} \oplus P)}{\partial \epsilon}\right|_{\epsilon=0} = \frac{\partial h(P)}{\partial P} \frac{\partial (e^{[\epsilon]_\times} \oplus P)}{\partial \epsilon} \\
&= \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \\ \frac{f_x}{Z} & 0 & -\frac{f_x (X - t_x)}{Z^2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & Z & -Y \\ 0 & 1 & 0 & -Z & 0 & X \\ 0 & 0 & 1 & Y & -X & 0 \end{bmatrix} \\
&= \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -f_x \frac{XY}{Z^2} & f_x(1 + \frac{X^2}{Z^2}) & -f_x \frac{Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y(1 + \frac{Y^2}{Z^2}) & f_y \frac{XY}{Z^2} & f_y \frac{X}{Z} \\ \frac{f_x}{Z} & 0 & -\frac{f_x (X - t_x)}{Z^2} & -\frac{f_x (X - t_x) Y}{Z^2} & \frac{f_x [(X - t_x) X + Z^2]}{Z^2} & -\frac{f_x Y}{Z} \end{bmatrix}
\end{aligned}$$

Adding an Inertial Measurement Unit (**IMU**)



- Advantages
  - Can be very affordable ($2.000 USD)
  - Can obtain orientation
  - High data rate

- Disadvantages
  - Can be very expensive ($2,000 USD)
  - Can only obtain orientation
  - Unreliable heading

Inertial Measurement Unit (IMU)

Data output:

- Gyroscope: angular rotation $[\omega_x \ \omega_y \ \omega_z]^T$
- Accelerometer: gravitational vector $g[a_x \ a_y \ a_z]^T$
- Compass: magnetic vector $[m_x \ m_y \ m_z]^T$ (optional)

Data characteristics:

- Gyroscope: (quite) accurate. But integration will drift
- Accelerometer: noisy. But does not drift
- Compass: noisy. Depends on external interference

Data model:

- $\phi_{measured} = \phi_{true} - \phi_{bias} + \psi$
- ϕ_{bias} follows Brownian motion
- ψ is Gaussian noise

Visual(vis)-Inertial(ins) **vin**SLAM

Self-compensating system

- IMU good at roll, pitch (and maybe yaw)
- CAM good at precise translation
- IMU is very useful in fast rotation
- CAM helps with relocalization & heading recovery

Two approaches:

- Loosely coupled
- Tightly coupled

Loosely-coupled vinSLAM

Two separate threads:

1. INS: Estimate orientation based on IMU readings
   - Dynamics:
     $$\begin{bmatrix} \phi \\ \dot{\phi}_b \end{bmatrix} = \begin{bmatrix} \mathbf{I}_3 & -\delta t \mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi}_b \end{bmatrix}^- + \begin{bmatrix} \delta t \mathbf{1}_{3\times 1} \\ \mathbf{0}_{3\times 1} \end{bmatrix} [\omega]$$
   - Covariance update
   - Measurement:
     $$[r]_{3\times 1} = [\phi_{meas}]_{3\times 1} - [\mathbf{I}_3 \ \mathbf{0}_3] \begin{bmatrix} \phi \\ \dot{\phi}_b \end{bmatrix}$$
   - Kalman gain
   - Correction

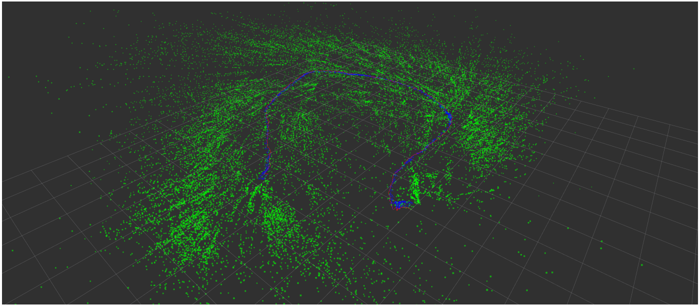
1. VIS: Uses INS estimates to determine initial pose estimate
   - Iteratively evaluate Jacobian and update pose
   - Refine orientations, but **not biases**

Tightly-coupled vinSLAM

One single thread VINS: Simultaneously

- estimate translation
- estimate orientation
- estimate biases

Stereo vSLAM demonstration

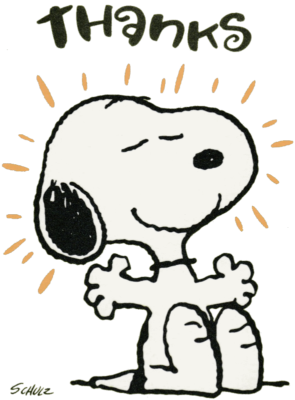


Future development

- Immediate future
  - Formulate Jacobians for tightly coupled vinsSLAM
  - Implementation & outdoor flight tests
  - Adding Deep Learning
    - Object constraints between map point
    - Scale recovery for monocular SLAM
- Near future
  - Adding downward camera and fuse 3rd cam visual odometry
  - Adding terrain altimeter and fuse altitude
  - Depth estimation with Monocular SLAM via Deep Learning
  - CNN-based image features
- Far future

Q & A

That's all, folks!



References