Introduction of Optimization on Lie Group

Teng Zhang

April 30, 2017

1 Background Knowledge

Many problems like SLAM and control in Robotics require optimization on Lie group (e.g., $SO(3)$ (3-dimension) and $SE(3)$ (6-dimension)) [1] [2]. A traditional method first parameterizes the Lie group (e.g., using Euler angles to represent $SO(3)$) and then the optimization process in Alg. 1 can be performed. However, this kind of parameterization may lead to singularity (e.g., the pitch angle 90° for Euler angles), which finally affect the optimization. Furthermore, parameterization makes the gradient vector or Jacobian matrix very complicated. This note provides a simple tutorial of optimization on Lie group without the two issues.

Remark 1. *It is a fundamental topological fact that singularities can never be eliminated in any 3-dimensional representation of $SO(3)$. This situation is similar to that of attempting to find a global coordinate chart on a sphere, which also fails [3].*

2 Notations

The special orthogonal group $SO(3)$ is defined as

$$SO(3) = \{R \in \mathbb{R}^{3\times 3} | R^T R = I_3,\ \det(R) > 0\}. \tag{1}$$

The skew symmetric operator S is defined as

$$S(x) = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix} \tag{2}$$

for $x = (x_1, x_2, x_3)^T \in \mathbb{R}^3$. There is an important property:

$$S(x)y = x \times y = -y \times x = -S(y)x \tag{3}$$

for $x,\ y \in \mathbb{R}^3$.

The exponential mapping $\exp : \mathbb{R}^3 \to\ SO(3)$ is

$$\exp(x) = I_3 + \sin\theta S(w) + (1 - \cos\theta)S(w)^2, \tag{4}$$

where $\theta = \|x\|$, $w = \frac{x}{\|x\|}$ and S is the skew symmetric operator. If $x = 0$, $\exp(x) = I_3$. Furthermore, the first oder approximation of $\exp(x)$ at 0 is

$$\exp(x) = I_3 + S(x) + O(\|x\|^2) \tag{5}$$

For more details about $SO(3)$ and Lie group see the first three chapters of [3] and [4].

3 Review: Optimization on Euclidean Space $\mathbb{R}^n$

A basic optimization problem is

$$\min_{X \in \mathbb{R}^n} f(X), \tag{6}$$

which is equivalent to

$$\min_{x \in \mathbb{R}^n} h_{X^*}(x) = \min_{x \in \mathbb{R}^n} f(X^* + x), \tag{7}$$

where $h_{X^*}(x) = f(X^* + x)$ and $X^* \in \mathbb{R}^n$ is arbitrary and known.

Alg. 1 presents a method for the optimization problem $\min\limits_{X \in \mathbb{R}^n} f(X)$ on Euclidean Space $\mathbb{R}^n$. Note that different algorithms (e.g., Gauss-Newton and Levenberg-Marquardt) can be used to determine the incremental vector $\Delta x \in \mathbb{R}^n$ in which $\partial_x h_{X^*}(x)|_{x=0}$ and $h_{X^*}(0)$ are needed.

Algorithm 1: Optimization on Euclidean Space

Input: the objective function $f(X)$ and the initial guess $X_0 \in \mathbb{R}^n$

Output: the local minimum X^* for $f(X)$

Process:

$X^* \leftarrow X_0$;

while X^* *does not converge* **do**

determine the incremental vector $\Delta x \in \mathbb{R}^n$ for $h_{X^*}(x)$ at $x = 0$ by using Gradient Descent method (GN, LM,..., etc) ;

$X^* \leftarrow X^* + \Delta x$;

Remark 2. *In Alg. 1, the function h_{X^*} changes in every loop due to the change of X^*.*

4 Optimization on Lie Group G

In order to perform optimization on $n-$dimensional Lie group G, the optimization process in Alg. 1 has to be adjusted. If a mapping $g : \mathbb{R}^n \to G$ is a bijective map between a neighborhood of $0 \in \mathbb{R}^n$ and a neighborhood of the identity element $I \in G$ and $g(0) = I$, then we can re-parametrize the problem as follows:

$$\min_{X \in G} f(X) \Rightarrow \min_{x \in \mathbb{R}^n} h_{X^*}(x) = \min_{x \in \mathbb{R}^n} f(X^* g(x)) \tag{8}$$

where $h_{X^*}(x) = f(X^* g(x))$ and $X^* \in G$ is arbitrary and known. Given a bijective mapping g ($g(0) = I$), a method for the optimization problem $\min\limits_{X \in G} f(X)$ is presented in Alg. 2.

Algorithm 2: Optimization on Lie group

Input: the objective function $f(X)$ and the initial guess $X_0 \in G$

Output: the local minimum $X^* \in G$ for $f(X)$

Process:

$X^* \leftarrow X_0$;

while X^* *does not converge* **do**

determine the incremental vector $\Delta x \in \mathbb{R}^n$ for $h_{X^*}(x)$ at $x = 0$ by using Gradient Descent method (GN, LM,..., etc) ;

$X^* \leftarrow X^* g(\Delta x)$;

Remark 3. *In Alg. 2, the function h_{X^*} changes in every loop due to the change of X^*.*

Remark 4. *Note that both X^* and Δx belong to the $n-$dimensional space $\mathbb{R}^n$ in Alg. 1 while $X^* \in G$ and $\Delta x \in \mathbb{R}^n$ in Alg. 2.* ***The main idea in Alg. 2 is that the state X^* is represented by the element in G while the incremental vector (small change) Δx is represented by the element in Euclidean space $\mathbb{R}^n$.*** *For each loop, after the incremental Δx is determined, the new state X^* can be updated by $X^* \leftarrow X^* g(\Delta x)$ instead of $X^* \leftarrow X^* + \Delta x$.*

5 Optimization on $SO(3)$

The group $SO(3)$ is the most common Lie group. A basic optimization on $SO(3)$ is

$$\min_{X \in SO(3)} f(X). \tag{9}$$

The exponential mapping $\exp(\dots)$ on Lie Group is chosen as the mapping g, so the optimization problem is equivalent to

$$\min_{X \in SO(3)} f(X) \Rightarrow \min_{x \in \mathbb{R}^3} h_{X^*}(x) = \min_{x \in \mathbb{R}^3} f(X^* \exp(x)). \tag{10}$$

where $h_{X^*}(x) = f(X^* \exp(x))$.

Algorithm 3: Optimization on $SO(3)$

Input: the objective function $f(X)$ and the initial guess $X_0 \in SO(3)$

Output: the local minimum $X^* \in SO(3)$ for $f(X)$

Process:

$X^* \leftarrow X_0$;

while X^* *does not converge* **do**

determine the incremental vector $\Delta x \in \mathbb{R}^3$ for $h_{X^*}(x)$ at $x = 0$ by using Gradient Descent method (GN, LM,..., etc);

$X^* \leftarrow X^* \exp(\Delta x)$;

Remark 5. *It is necessary to calculate* $\partial_x h_{X^*}(x)|_{x=0}$ *when determining the incremental vector* $\Delta x \in \mathbb{R}^3$*. With Eq. 5,* $\partial_x h_{X^*}(x)|_{x=0}$ *can be easily and safely obtained by the following:*

$$\partial_x h_{X^*}(x)|_{x=0} = \partial_x f(X^*(I_3 + S(x)) + O(\|x\|^2))|_{x=0}. \tag{11}$$

6 Example

The optimization problem in [5]

$$\min_{X \in SO(3)} f(X) = \min_{X \in SO(3)} \sum_{i=1}^{N} \|Xp_i - q_i\|^2 \tag{12}$$

is used to illustrate the general idea, where $p_i \in \mathbb{R}^3$ and $q_i \in \mathbb{R}^3$ $(i = 1, 2, 3..., N)$ are given. Alg. 4 provides the Gradient Descent Method for the optimization problem (Eq. 12).

According to the last section, $h_{X^*}(x) = \sum_{i=1}^{N} \|X^* \exp(x)p_i - q_i\|^2$ and

$$\begin{aligned} \partial_x h_{X^*}(x)|_{x=0} &= (\partial_x \sum_{i=1}^{N} \|X^*(I_3 + S(x) + O(\|x\|^2))p_i - q_i\|^2)|_{x=0} \\ &= (-2\partial_x \sum_{i=1}^{N} q_i^T X^*(S(x) + O(\|x\|^2))p_i)|_{x=0} \\ &= (-2\partial_x \sum_{i=1}^{N} q_i^T X^* S(x)p_i)|_{x=0} \\ &= 2(\partial_x \sum_{i=1}^{N} q_i^T X^* S(p_i)x)|_{x=0} \\ &= 2\sum_{i=1}^{N} q_i^T X^* S(p_i) \end{aligned} \tag{13}$$

Algorithm 4: Solving Eq. 12 by using the Gradient Descent Method on $SO(3)$

Input: the objective function $f(X)$ in Eq. 12, the initial guess $X_0 \in SO(3)$ and the step size $d \in \mathbb{R}^+$

Output: the local minimum $X^* \in SO(3)$ for $f(X)$

Process:

$X^* \leftarrow X_0$;

while X^* *does not converge* **do**

- $\lambda \leftarrow \partial_x h_{X^*}(x)|_{x=0}$ (Eq. 13) ;
- $\lambda \leftarrow \frac{\lambda}{\|\lambda\|}$;
- determine the incremental vector $\Delta x \in \mathbb{R}^3$: $\Delta x \leftarrow -d\lambda^T$;
- $X^* \leftarrow X^* \exp(\Delta x)$;

References

[1] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Robotics: Science and Systems XI*, no. EPFL-CONF-214687, 2015.

[2] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 3607–3613.

[3] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation.* CRC press, 1994.

[4] A. Baker, *Matrix groups: An introduction to Lie group theory.* Springer Science & Business Media, 2012.

[5] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 5, pp. 698–700, 1987.

Appendices

A Matlab Code for Alg. 4

The following is the "main.m".

```
clc
clear
N=40; % the number of features

stepsize=0.001;

P=3*randn(3,N);
R=SO([1;5;0.3])  % R is a rotation matrix (the ground truth)
noise=0.1*randn(3,N);
Q=R*P+noise;
R_star=eye(3);

for K=1:2000  % 2000 is the number of iteration
H=zeros(1,3);

%%%% Caculate Eq. 13
```

```
for i=1:N
H=H+Q(:,i)'*R_star*skew(P(:,i));
end
%%%% Caculate Eq. 13

H=stepsize*H/norm(H);
deltaX=-H;
R_star=R_star*SO(deltaX);  % SO functions is exp in Eq. 4
end


R_star % display the estimated rotation matrix
acos((trace(R_star*R')-1)/2) * 180/pi  % display the error, unit: degree
```

The following is the ”skew.m”. (The skew function is S(...) in Eq. 2)

```
function y = skew( x )
y=[0 -x(3) x(2);...
x(3) 0 -x(1);...
-x(2) x(1) 0];
```

The following is the ”SO.m”. (The SO function is exp(...) in Eq. 4)

```
function R=SO(x)
nq=norm(x);
if nq==0
R=eye(3);
else
w=x/nq;
R=eye(3)+sin(nq)*skew(w)+(1-cos(nq))*skew(w)*skew(w);
end
end
```