How optimization is implemented in gs-rs

Samuel Valenzuela[1], Daniel Pape[1]

[1] *TNG Technology Consulting GmbH, Unterföhring, Germany*

June 16, 2020

gs-rs is a Rust framework for the optimization of non-linear least squares problems embeddable as a (hyper)graph. It is suitable for the optimization of an error function with respect to a set of parameters as in SLAM (simultaneous localization and mapping) or BA (bundle adjustment).

1 Background Literature

The optimization algorithm used by **gs-rs** is based on parts of **g2o**. This paper should suffice to understand the optimization's implementation in **gs-rs**. The following papers by the developers of **g2o** are recommended if a deeper understanding of the theory behind the algorithm is of interest:

- *g2o: A General Framework for Graph Optimization, Kümmerle et al.* [1]: This paper documents the derivation of the algorithm's structure.
- *A Tutorial on Graph-Based SLAM, Grisetti et al.* [2]: This paper contains additional comments on the calculations in 2D and 3D. Here it is presented how the least squares optimization works on a manifold.

2 Optimization Algorithm

In the following sections, the optimization algorithm will be introduced. Firstly, the structure's main iteration steps will be introduced. Most of the work happens in the first step, the calculation of H and b. Subsequently, the calculations which apply to all dimensions and factor types will be presented.

2.1 Iteration Steps

Given a specific number of iterations n and the initial guess $x_i^{(0)}$ for each variable, the optimizer algorithm will repeat the following steps n times:

1. Calculate H and b by setting them to $\mathbf{0}$, then looping through all factors and updating their non-fixed variables' entries in H and b.
2. Calculate Δx, the vector containing data about how much each current variable guess $x_i^{(k)}$ should be updated in this step, by solving the linear system

$$H\Delta x = -b^T. \tag{1}$$

3. Update the guesses for each non-fixed variable x_i with

$$x_i^{(k+1)} = x_i^{(k)} + \Delta x_i. \tag{2}$$

   In the case of 2D variables with a rotation, normalize the rotation to $[-\pi, \pi)$. In the case of 3D variables with a rotation, normalize the rotation of Δx_i. As explained in section 4, the quaternion data will only include the imaginary part. Therefore, set the scalar part to 1 before normalizing. The $+$ operator then resembles the concatenation of two isometries, analogously as the $*$ operator in section 4.

2.2 Calculation of H and b

How exactly the variables' entries in H and b are updated depends on the factor type. In all cases the factor's increments on parts of H and b, H^{fac} and b^{fac} respectively, will be computed as follows:

$$H^{fac} = J^T * \Omega * J, \tag{3}$$

$$b^{fac} = e^T * \Omega * J, \tag{4}$$

where Ω, J and e are the factor's information matrix, Jacobian matrix and error vector, respectively. While Ω is a given constant of the factor, J and e have to be calculated for each factor in each iteration.

If the factor only involves the variable x_i, H and b are updated as follows:

$$H_{ii} = H_{ii} + H^{fac}, \tag{5}$$

$$b_i = b_i + b^{fac}, \tag{6}$$

where the subscripts of H and b denote the row and column index of the submatrix or subvector assigned to the respective variable. If the factor involves two variables x_i and x_j, H^{fac} and b^{fac} will have the structure

$$H^{fac} = \begin{pmatrix} \boldsymbol{H}_{ii}^{fac} & \boldsymbol{H}_{ij}^{fac} \\ \boldsymbol{H}_{ji}^{fac} & \boldsymbol{H}_{jj}^{fac} \end{pmatrix} \tag{7}$$

and

$$b^{fac} = \begin{pmatrix} \boldsymbol{b}_i^{fac} & \boldsymbol{b}_j^{fac} \end{pmatrix}, \tag{8}$$

respectively, such that H_{mn} will be incremented by H_{mn}^{fac} and b_n will be incremented by b_n^{fac}, analogously to equations (5) and (6). Fixed variables are excluded from H and b and therefore do not have any submatrices or subvectors which would need to be updated. In this case, these parts of H^{fac} and b^{fac} are simply ignored.

In the following sections, the calculation is described for all 2D and 3D factors supported by **gs-rs**.

3 Optimization in 2D

In the following sections, the individual 2D factors' calculations of J and e are presented. The functions $pos(x)$ and $rot(x)$ will be used to refer to the 2D position vector and the rotation angle of a 2D pose x, respectively. Similarly, the functions $pos_x(x)$ and $pos_y(x)$ will be used to refer to the single value within the respective dimension.

Furthermore, the rotation matrices are denoted such that

$$R_\alpha = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \tag{9}$$

equals the rotation matrix with the angle α in 2D, and

$$R_z(\alpha) = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{10}$$

equals the rotation matrix with the angle α in 3D.

3.1 Position2D

The *Position2D* factor involves one *VehicleVariable* x_v. The Jacobian matrix J in this case is

$$J = R_z(-rot(x_v)). \tag{11}$$

Given the measurement x_m, the error vector

$$e = R_{-rot(x_m)} * (pos(x_v) - pos(x_m)) \tag{12}$$

can be computed as well.

3.2 Odometry2D

The *Odometry2D* factor involves two *VehicleVariables* x_i and x_j. Given the measurement x_{ij}, the Jacobian matrix J is calculated as follows:

$$\Delta x_{ij} = x_j - x_i \tag{13}$$

$$sin_i = sin(rot(x_i)) \tag{14}$$

$$cos_i = cos(rot(x_i)) \tag{15}$$

$$J_i = R_z(-rot(x_{ij})) * \begin{pmatrix} -cos_i & -sin_i & -sin_i * pos_x(\Delta x_{ij}) + cos_i * pos_y(\Delta x_{ij}) \\ sin_i & -cos_i & -cos_i * pos_x(\Delta x_{ij}) - sin_i * pos_y(\Delta x_{ij}) \\ 0 & 0 & -1 \end{pmatrix} \tag{16}$$

$$J_j = R_z(-rot(x_{ij})) * R_z(-rot(x_i)) \tag{17}$$

$$J = \begin{pmatrix} \boldsymbol{J_i} & \boldsymbol{J_j} \end{pmatrix} \tag{18}$$

The error vector e is computed as follows:

$$e_{pos} = R_{-rot(x_{ij})} * (R_{-rot(x_i)} * pos(\Delta x_{ij}) - pos(x_{ij})) \tag{19}$$

$$e_{rot} = rot(\Delta x_{ij}) - rot(x_{ij}) \tag{20}$$

After normalizing e_{rot} to $[-\pi, \pi)$ with $norm(e_{rot})$ the full error vector can be constructed with

$$e = \begin{pmatrix} \boldsymbol{e_{pos}} \\ norm(e_{rot}) \end{pmatrix}. \tag{21}$$

3.3 Observation2D

The *Observation2D* factor involves one *VehicleVariable* x_i and one *LandmarkVariable* x_j. The measurement is denoted as x_{ij}, analogously to the previous section. Although x_j and x_{ij} are only positions rather than poses and therefore do not contain a rotation angle, the functions $pos(x)$, $pos_x(x)$ and $pos_y(x)$ will be used nevertheless to make the calculation path more understandable. Given the measurement x_{ij}, the Jacobian matrix J is calculated as follows:

$$pos(\Delta x_{ij}) = pos(x_j) - pos(x_i) \tag{22}$$

$$sin_i = sin(rot(x_i)) \tag{23}$$

$$cos_i = cos(rot(x_i)) \tag{24}$$

$$J_i = \begin{pmatrix} -cos_i & -sin_i & -sin_i * pos_x(\Delta x_{ij}) + cos_i * pos_y(\Delta x_{ij}) \\ sin_i & -cos_i & -cos_i * pos_x(\Delta x_{ij}) - sin_i * pos_y(\Delta x_{ij}) \end{pmatrix} \tag{25}$$

$$J_j = R_{-rot(x_i)} \tag{26}$$

$$J = \begin{pmatrix} \boldsymbol{J_i} & \boldsymbol{J_j} \end{pmatrix} \tag{27}$$

The error vector e is computed as follows:

$$e = R_{-rot(x_i)} * pos(\Delta x_{ij}) - pos(x_{ij}) \tag{28}$$

4 Optimization in 3D

In 3D, rotations are often represented as quaternions. To reduce these to a minimal representation, only the imaginary part of the unit quaternions is saved within the solution Δx, while the scalar part is implicitly 1 [2]. This is achieved by always using unit quaternions and removing the entry of the scalar part in the error vector e after its computation.

In the following sections, the individual 3D factors' calculations of J and e are presented. Similarly as in section 3, the functions $pos(x)$ and $rot_Q(x)$ will be used to refer to the 3D position vector and the rotation quaternion of a 3D pose or isometry x. When referring to the rotation as a 3D rotation matrix, the function $rot_{RM}(x)$ will be used instead. When using x in calculations, it will refer to the pose's 3D isometry containing both the position vector as translation as well as the rotation quaternion. The binary operator $*$ will be used to concatenate these isometries. The inverse of an isometry will be denoted as x^{-1}.

Some functions will be used to calculate gradients of a 3D isometry x, such as $\frac{dq}{dR}(x)$, which is computed as follows using the trace function $tr(M)$ of 3x3 matrices:

$$s = \frac{1}{2}\sqrt{tr(rot_{RM}(x)) + 1} \tag{29}$$

$$a_1 = -\frac{rot_{RM}(x)_{21} - rot_{RM}(x)_{12}}{32s^3} \tag{30}$$

$$a_2 = -\frac{rot_{RM}(x)_{02} - rot_{RM}(x)_{20}}{32s^3} \tag{31}$$

$$a_3 = -\frac{rot_{RM}(x)_{10} - rot_{RM}(x)_{01}}{32s^3} \tag{32}$$

$$b = \frac{1}{4s} \tag{33}$$

$$\frac{dq}{dR}(x) = \begin{pmatrix} a_1 & 0 & 0 & 0 & a_1 & b & 0 & -b & a_1 \\ a_2 & 0 & -b & 0 & a_2 & 0 & b & 0 & a_2 \\ a_3 & b & 0 & -b & a_3 & 0 & 0 & 0 & a_3 \end{pmatrix} \tag{34}$$

It will always be assumed that the condition $tr(rot_{RM}(x)) > 0$ holds.

Another function used in this context is $skew(x)$, computed as follows using a 3D vector or position x:

$$skew(x) = \begin{pmatrix} 0 & x_z & -x_y \\ -x_z & 0 & x_x \\ x_y & -x_x & 0 \end{pmatrix}, \tag{35}$$

where x_x, x_y and x_z are the first, second and third vector components, respectively.

4.1 Position3D

The *Position3D* factor involves one *VehicleVariable* x_v. Given the measurement x_m, the Jacobian matrix J is calculated as follows:

$$E = x_m^{-1} * x_v \tag{36}$$

$$\frac{dte}{dqj} = \frac{dq}{dR}(E) * \begin{pmatrix} \boldsymbol{skew(rot_{RM}(E)_0)} \\ \boldsymbol{skew(rot_{RM}(E)_1)} \\ \boldsymbol{skew(rot_{RM}(E)_2)} \end{pmatrix} \tag{37}$$

$$J = \begin{pmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\frac{dte}{dqj}} \end{pmatrix}, \tag{38}$$

where M_i refers to the column at index i of the matrix M.
The error vector e is computed as follows:

$$e = x_m^{-1} * x_v \tag{39}$$

As mentioned in section 4, the scalar part of the quaternion is removed in e. e i then interpreted as a six-dimensional vector with the top three entries for its position and the bottom three entries for its rotation.

4.2 Odometry3D

The *Odometry3D* factor involves two *VehicleVariables* x_i and x_j. Given the measurement x_{ij}, the Jacobian matrix J is calculated as follows:

$$A = x_{ij}^{-1} \tag{40}$$

$$B = x_i^{-1} * x_j \tag{41}$$

$$E = A * B \tag{42}$$

$$\frac{dte}{dqi} = rot_{RM}(A) * skew(pos(B)) \tag{43}$$

$$\frac{dre}{dqi} = \frac{dq}{dR}(E) * \begin{pmatrix} \boldsymbol{rot_{RM}(A) * skew(rot_{RM}(B)_0)^T} \\ \boldsymbol{rot_{RM}(A) * skew(rot_{RM}(B)_1)^T} \\ \boldsymbol{rot_{RM}(A) * skew(rot_{RM}(B)_2)^T} \end{pmatrix} \tag{44}$$

$$J_i = \begin{pmatrix} \boldsymbol{-rot_{RM}(A)} & \boldsymbol{\frac{dte}{dqi}} \\ \boldsymbol{0} & \boldsymbol{\frac{dre}{dqi}} \end{pmatrix} \tag{45}$$

$$\frac{dre}{dqj} = \frac{dq}{dR}(E) * \begin{pmatrix} \boldsymbol{skew(rot_{RM}(E)_0)} \\ \boldsymbol{skew(rot_{RM}(E)_1)} \\ \boldsymbol{skew(rot_{RM}(E)_2)} \end{pmatrix} \tag{46}$$

$$J_j = \begin{pmatrix} \boldsymbol{rot_{RM}(E)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\frac{dre}{dqj}} \end{pmatrix} \tag{47}$$

$$J = \begin{pmatrix} \boldsymbol{J_i} & \boldsymbol{J_j} \end{pmatrix}, \tag{48}$$

where M_i refers to the column at index i of the matrix M.
The error vector e is computed as follows:

$$e = x_{ij}^{-1} * x_i^{-1} * x_j \tag{49}$$

As mentioned in section 4, the scalar part of the quaternion is removed in e. e i then interpreted as a six-dimensional vector with the top three entries for its position and the bottom three entries for its rotation.

4.3 Observation3D

The *Observation3D* factor involves one *VehicleVariable* x_i and one *LandmarkVariable* x_j. In the following, the binary operator $*$ denotes the transformation of the position given by the right operand by the isometry given by the left operand. With the measurement x_{ij}, the Jacobian matrix J is calculated as follows:

$$J = \begin{pmatrix} \boldsymbol{-I_3} & \boldsymbol{skew(pos(x_i^{-1} * pos(x_j)))^T} & \boldsymbol{rot_{RM}(x_i^{-1})} \end{pmatrix} \tag{50}$$

The error vector e is computed as follows:

$$e = pos(x_i^{-1} * pos(x_j)) - pos(x_{ij}) \tag{51}$$

References

[1] Rainer Kümmerle et al. “g2o: A general framework for graph optimization”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3607–3613.

[2] Giorgio Grisetti et al. “A tutorial on graph-based SLAM”. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.