# How optimization is implemented in gs-rs

**Samuel Valenzuela[1], Daniel Pape[1]**

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

May 20, 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 Optimization Algorithm

### 1.1 Background Literature

The optimization algorithm used by **gs-rs** is based on that which parts of **g2o** uses. 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.

### 1.2 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 variables' entries in $H$ and $b$.
2. Calculate $\Delta x$, 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 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 it to $[-\pi, \pi)$.

How the parts of $H$ and $b$ are calculated depends on the exact factor type. In the following sections, the calculation is described for all 2D and 3D factors supported by **gs-rs**.

# 2 Optimization in 2D

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 $\Omega$, $J$ and $e$ are the factor's information matrix, Jacobian matrix and error vector, respectively. While $\Omega$ 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).

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.

## 2.1 Position2D

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

$$J = R_z(-rot(x_v)). \tag{9}$$

Given the measurement $x_m$, the error vector

$$e = R_{-rot(x_m)} * (pos(x_v) - pos(x_m)) \tag{10}$$

can be computed as well.

## 2.2 Odometry2D

The *Position2D* factor involves two *VehicleVariables* $x_i$ and $x_j$. Given the measurement $x_{ij}$, the Jacobian matrix $J$ is calculated as follows:

$$\Delta x_{ij} = x_j - x_i \tag{11}$$

$$sin_i = sin(rot(x_i)) \tag{12}$$

$$cos_i = cos(rot(x_i)) \tag{13}$$

$$J_i = R_z(-rot(x_{ij})) * \begin{pmatrix} -cos_i & -sin_i & -sin_i * pos_x(\Delta x_{ij}) + cos_i * pos_y(\Delta x_{ij}) \\ sin_i & -cos_i & -cos_i * pos_x(\Delta x_{ij}) - sin_i * pos_y(\Delta x_{ij}) \\ 0 & 0 & -1 \end{pmatrix} \tag{14}$$

$$J_j = R_z(-rot(x_{ij})) * R_z(-rot(x_i)) \tag{15}$$

$$J = \begin{pmatrix} \boldsymbol{J}_i & \boldsymbol{J}_j \end{pmatrix} \tag{16}$$

The error vector $e$ is computed as follows:

$$e_{pos} = R_{-rot(x_{ij})} * (R_{-rot(x_i)} * pos(\Delta x_{ij}) - pos(x_{ij})) \tag{17}$$

$$e_{rot} = rot(\Delta x_{ij}) - rot(x_{ij}) \tag{18}$$

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

$$e = \begin{pmatrix} \boldsymbol{e}_{pos} \\ norm(e_{rot}) \end{pmatrix}. \tag{19}$$

### 2.3 Observation2D

The *Position2D* 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:

$$pos(\Delta x_{ij}) = pos(x_j) - pos(x_i) \tag{20}$$

$$sin_i = sin(rot(x_i)) \tag{21}$$

$$cos_i = cos(rot(x_i)) \tag{22}$$

$$J_i = \begin{pmatrix} -cos_i & -sin_i & -sin_i * pos_x(\Delta x_{ij}) + cos_i * pos_y(\Delta x_{ij}) \\ sin_i & -cos_i & -cos_i * pos_x(\Delta x_{ij}) - sin_i * pos_y(\Delta x_{ij}) \end{pmatrix} \tag{23}$$

$$J_j = R_{-rot(x_i)} \tag{24}$$

$$J = \begin{pmatrix} \boldsymbol{J}_i & \boldsymbol{J}_j \end{pmatrix} \tag{25}$$

The error vector $e$ is computed as follows:

$$e = R_{-rot(x_i)} * pos(\Delta x_{ij}) - pos(x_{ij}) \tag{26}$$

## 3 Optimization in 3D

3D optimization is not supported yet. In the future, the following factors should be able to contribute to the optimization:

### 3.1 Position3D

### 3.2 Odometry3D

### 3.3 Observation3D

## 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.