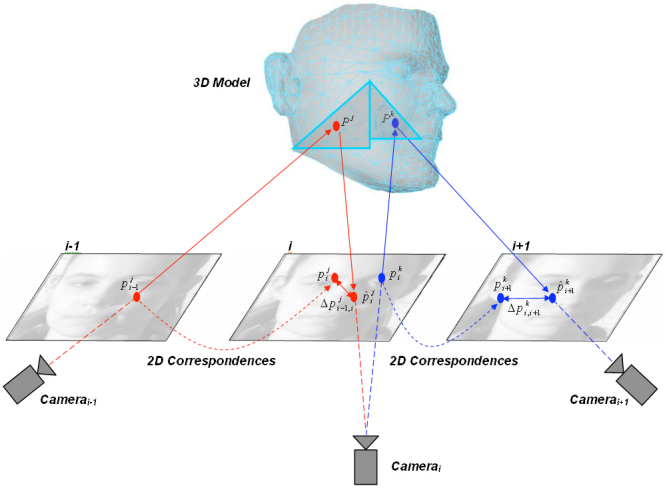
Technische Universität München
Fakultät für Informatik
Yehya Abouelnaga, Ivan Pavlov
and Adrian Haarbach

WS 19/20
Solution for Exercise Sheet 3
November 21, 2019

Exercises in Tracking & Detection

Task 1 Minimization of the Reprojection Error



a) Just looking at the first two frames $Camera_{i-1}, Camera_i$ and point P^j, which of the solid red arrows corresponds to:

- Direct projection **Answer** $P \rightarrow \hat{p}_i$ from 3D to 2D.
- Back-projection **Answer** $p_{i-1} \rightarrow P$ from 2D to 3D. Shooting a ray trough camera center and triangle intersection to get the depth.
- Objective function **Answer** $p_i \leftrightarrow \hat{p}_i = \Delta p_{i-1,i}$ is the 2D to 2D symmetric Euclidean distance between corresponding keypoints in pixels.

b) How to get from the reprojection error $\min_{R,T} \sum_i \|A(RM_i+T)-m_i\|^2$ to the objective function $\min_{R,T} \sum_k \|\Delta p_{i-1,i}\|^2$?

Answer We replace index $[]_i$ with $[]^k$ iterating over $P \leftrightarrow p$ instead of $M \leftrightarrow m$ 3D - 2D correspondences to get $\min_{R,T} \sum_k \|A(RP^k + T) - p^k\|^2$. Now we can reintroduce the index i, but this time iterating over the cameras. For $Camera_i$, the 3D point P is acquired by back-projection $(Proj^{-1})$ of p_{i-1} from $Camera_{i-1}$. Then, P is projected $(Proj)$ into $Camera_i$ by direct projection via $\hat{p}_i = Proj_{A,R,T}(P) = A(RP+T)$. These two steps cam be combined into the transfer function ψ. The objective function is the sum of squared residuals $\hat{p}_i^k - p_i^k = \Delta p_{i-1,i}$.

$$\min_{R,T} \sum_k \|A(RP^k + T) - p^k\|^2 = \min_{R,T} \sum_k \|Proj_{A,R,T}(Proj^{-1}_{A,R_{i-1},T_{i-1}}(p^k_{i-1})) - p^k_i\|^2$$
$$= \min_{R,T} \sum_k \|\psi(p^k_{i-1}) - p^k_i\|^2 = \min_{R,T} \sum_k \|\hat{p}^k_i - p^k_i\|^2 = \min_{R,T} \sum_k \|\Delta p_{i-1,i}\|^2$$

c) What is the difference between Algebraic vs Geometric error minimization and when to use which one? What are the advantages and drawbacks of each method?

Answer Algebraic error minimisation aims at solving an overdetermined system of linear equations $Ax = b$ in a least-squares sense, meaning to minimize the L2 norm of the model function $f(x) = Ax - b$ via linear algebra techniques $\hat{x} = \text{argmin} \|f(x)\|^2 = (A^T A)^{-1} A^T b$ (the Pseudo-Inverse). It requires normalisation of the data in order to work well. Not all effects of the projection can be expressed correctly in this way and not all constraints preserved. In particular, this type of minimisation is restricted to functions that are a linear combination of the parameter x, weighted by A, expressed via the matrix-vector product Ax.

Geometric error minimisation aims at minimising the Euclidean 2D distance of pairs or reprojected points in the image plane. It can model all effects of the projection and maintain all imposed constraints if implemented correctly. However, it is much slower to solve via Non-linear optimization and also needs a good initialisation to converge. There is no guarantee that it will converge to the global minimum of a complex non-convex function. It is usually done as a last step starting with the algebraic minimum as an initialization.

For more details see *Richard Hartley (1998): Minimizing Algebraic Error in Geometric Estimation Problems.* `http://web.iitd.ac.in/~sumeet/algebraic.pdf`

Task 2 Non-linear optimization and robust estimation

Algorithm 1: Levenberg-Marquardt (LM)

Input: x_0 initial parameters, f function to mimimize
Output: x optimized parameters

$\lambda \leftarrow \lambda_0$
$x \leftarrow x_0$
while *not converged* **do**
  $\mathbf{r} \leftarrow f(x)$ // residual vector
  $J \leftarrow \nabla \mathbf{r}(x)$ // Jacobian
  $\Delta x \leftarrow -(J^T J + \lambda I)^{-1} J^T \mathbf{r}$ // Levenberg step
  if $E(x + \Delta x) < E(x)$ **then** // error reduced
    $x \leftarrow x + \Delta x$
    increase λ
  else
    decrease λ

a) Why do we need to use Non-linear optimization?

Answer Because we want to minimize a geometric error, the geometrically meaningful reprojection error, which is the sum of squared residuals that depend non-linearly on our parameters.

b) What is the general form of Non-linear least squares problems (NLS)?

Answer Non-linear least squares problems (NLS) come up in many areas, e.g curve fitting in statistics or minimizing re-projection errors in computer vision. The goal for the problem's solution is to find parameters to a model function that minimize a sum of squared residuals, that is the sum of squared differences between observed and modeled values, as in *linear* least squares. The difference is that we allow more general

functions f_i than just linear ones, where the parameter x may appear in higher order, e.g. x^2 or e^x, which cannot be modelled using matrix multiplication. The general form of a NLS problem is:

$$E(x) = \sum_{i}^{N} \left\| f_i\left(x_{i_1}, ..., x_{i_j}\right) \right\|^2 = \sum_{i=1}^{N} \mathbf{r}_i^T \mathbf{r}_i = \mathbf{r}^T \mathbf{r} \tag{1}$$

The $\left[x_{i_1}, ..., x_{i_j}\right]$ are the j parameter blocks of the i'th cost function f_i, whose current values define the i'th residuum $\mathbf{r}_i = f_i\left(x_{i_1}, ..., x_{i_j}\right)$. By stacking all of the N $\mathbf{r}_i$'s into $\mathbf{r}$, the vector of concatenated residuals, the summation of squared L2 vector norms $||.||^2$ can be rewritten compactly using the dot product.

c) How does the Gradient descent method differ from Gauss-Newton?

Answer In the Gradient descent method, the sum of the squared errors is reduced by updating the parameters in the steepest descent direction $\mathbf{x}_{k+1} = \mathbf{x}_k - \mu \mathbf{J}_r^T \mathbf{r}(\mathbf{x}_k)$. In the Gauss-Newton method, the sum of the squared errors is reduced by assuming the least squares function is locally quadratic, and finding the minimum of the quadratic $\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{J}_r^T \mathbf{J}_r)^{-1} \mathbf{J}_r^T \mathbf{r}(\mathbf{x}_k)$.

d) Levenberg-Marquardt (LM): Explain the main idea of the method (Algorithm 1), especially lines 4, 5 and how the update step $\Delta x = -(J^T J + \lambda I)^{-1} J^T \mathbf{r}$ in line 6 is derived.

Answer The Levenberg-Marquardt (LM) method is an iterative optimization algorithm which solves NLS (1) iteratively, starting from an initial guess. It combines gradient descent and Gauss-Newton approaches to function minimization. The goal at each iteration k is to choose an update Δx to the current estimate x_k, so that the new estimate $x_{k+1} = x_k + \Delta x$ reduces the error $E(x)$. The idea is to approximate the residuals $\mathbf{r}_i$ by their first order Taylor series expansion:

$$\mathbf{r}_i(x + \Delta x) \approx \mathbf{r}_i(x) + \nabla \mathbf{r}_i(x) \Delta x = \mathbf{r}_i + J_i \Delta x \tag{2}$$

Here, $J_i = \frac{\partial \mathbf{r}_i}{\partial x} = \nabla \mathbf{r}_i(x)$ is the Jacobian of $\mathbf{r}_i$, computed in x. Similarly to stacking the residuals $\mathbf{r}_i$ into $\mathbf{r}$, we can stack the Jacobians J_i into $J = \frac{\partial \mathbf{r}}{\partial x} = \nabla \mathbf{r}(x)$. With the approximation (2), the sum of squared residuals in 1 becomes:

$$\begin{aligned} E(x + \Delta x) &= \mathbf{r}(x + \Delta x)^T \mathbf{r}(x + \Delta x) \\ &\approx (\mathbf{r} + J\Delta x)^T (\mathbf{r} + J\Delta x) \\ &= \mathbf{r}^T \mathbf{r} + \mathbf{r}^T J \Delta x + (J\Delta x)^T \mathbf{r} + (J\Delta x)^T (J\Delta x) \\ &= \mathbf{r}^T + 2\Delta x^T J^T \mathbf{r} + \Delta x^T J^T J \Delta x \end{aligned} \tag{3}$$

At each iteration, the task is to find an update step Δx which will minimize $E(x+\Delta x)$. Differentiating the above expression with respect to x and equate with zero gives:

$$\nabla_x E(x + \Delta x) = \underbrace{2J^T \mathbf{r}}_{b} + \underbrace{2J^T J}_{H} \Delta x = 0$$

This is an expression involving b, the gradient, and H, the Gauss-Newton approximation to the Hessian. The useful property of this approximation of the Hessian is that it only requires first order derivatives, not second order ones like the true Hessian. It is thus much easier to derive analytically or compute numerically. Solving the linear system $H\Delta x = -b$ for Δx yields the Gauss-Newton update $\Delta x = -(J^T J)^{-1} J^T \mathbf{r}$.

Whether this step actually reduces the error E depends on the accuracy of the Taylor series expansion at x and the validity of the Gauss-Newton approximation. This is usually the case when near the minimum. On the other hand, a simple gradient descent step $\Delta x = -\lambda^{-1} J^T \mathbf{r}$ guarantees to reduce E, provided that λ is sufficiently large (and thus the step size λ^{-1} small), but its convergence near the minimum is very slow. The Levenberg-Marquardt algorithm combines both approaches in quite a simple way:

$$\Delta x = -(J^T J + \lambda I)^{-1} J^T \mathbf{r} \tag{4}$$

e) Levenberg-Marquardt (LM): Explain the details of the method (Algorithm 1), in lines 7-11. When is the update accepted and λ increased? Similar to which other methods does LM behave if λ is large/small?

Answer Large λ correspond to small, safe *Gradient Descent steps*, which are sure to decrease the error, while small λ correspond to *Gauss-Newton steps* that allow fast convergence near the minimum, but might otherwise increase the error. The art of a good implementation is to tune λ after each iteration to allow for fast convergence: If the new error is lower than the previous one, the update Δx is accepted and λ is decreased. If the new error is larger than the previous one, the update Δx is discarded and λ is increased.

f) Iteratively re-weighted least-squares (IRLS): As an alternative to RANSAC, robust loss functions can be used in NLS. What is the general goal of them? What choices do we have and how to estimate their parameter and weighting factor?

Answer The goal is to reduce the quadratic influence of outliers in the loss term. Options are e.g. Huber or Tukey loss. The problem is that they introduce an additional parameter that has to be tuned depending on the expected proportion of outliers in the data, e.g. via its median absolute deviations. The weighting can be incoorporated into the update step directly as seen in the homework.

Task 3 Rotation parameterization, Lie Algebra and derivatives

a) What properties does SO(3) have? Why don't we directly optimize in this space?

Answer A rotation can be represented by a matrix which fulfils special properties. When a vector or point is rotated, it does not change its length. Thus the matrix R must be an orthonormal matrix. The rotation of a point $\mathbf{p} \in \mathbb{R}^3$ is then done by a simple matrix multiplication R$\mathbf{p}$. Thus, the rotation matrices R form **SO(3)**:

$$\boxed{\mathrm{SO}(3) = \{\mathrm{R} \in \mathbb{R}^{3\times 3} \mid \mathrm{R}^T\mathrm{R} = \mathrm{I}, \det(\mathrm{R}) = +1\}}$$

From orthogonality, it easily follows that $\mathrm{R}^{-1} = \mathrm{R}^T$. The rotation matrix R consists of 9 parameters, even though it only has 3 degrees of freedom (**DoF**). This means we can only freely choose 3 entries that automatically determine the 6 remaining entries through the conditions $\mathrm{R}\mathrm{R}^T = \mathrm{I}$ and $\det(R) = 1$. Directly optimizing the 9 parameters of rotation matrices that have to fulfill these conditions is infeasible to ensure during interpolation or optimization, so one typically resorts to alternative parameterizations of this space.

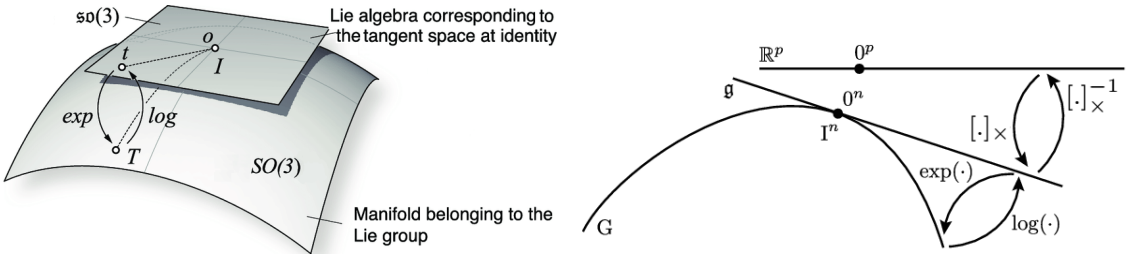
b) Briefly discuss Euler Angles vs. Unit Quaternion vs Axis-Angle as alternative parameterizations of this space. What is the problem with gimbal lock, singularities and double-covering? Which representation is best suited to represent the parameter update Δx in the context of NLS methods like LM/IRLS from the last task?

Answer Euler Angles have only 3 parameters for the 3 DoF, but suffer from gimbal lock, which means that an optimization algorithm can get stuck at a certain configuration where one degree of freedom is lost.

Unit quaternions have 4 parameters and the constraint to be of unit norm. Compared to Euler angles they are simpler to compose and avoid the problem of gimbal lock. Compared to rotation matrices they are more compact, more numerically stable, and more efficient. They form a multiplicative group, just as rotation matrices. However, they double-cover SO(3), meaning that q and $-q$ represent the same orientation. Still, they are computationally quite stable and efficient which is why they are often choosen, especially for interpolation.

Axis-Angle $(\bar{\mathbf{v}}, \theta)$, where $\bar{\mathbf{v}} \in \mathbb{R}^3$ is a unit vector, has singularities at $\theta = 2n\pi, n \in \mathbb{N}$, which can in practice however easily be avoided by restricting θ to stay below 2π. Furthermore, the Angle θ can be absorbed into the norm of the vector $\mathbf{v} = \theta\bar{\mathbf{v}}$ to arrive at just 3 parameters, called the exponential coordinates, which are a natural and compact representation that we will use to represent the update Δx in the context of our NLS optimization.

c) Explain the pictures by reading section 2.3.1 'Canonical exponential coordinates' of *Y. Ma, S. Soatto, J. Kosecka, and S.S. Sastry (2001): An Invitation to 3-D Vision: From Images to Geometric Models.* `https://www.eecis.udel.edu/~cer/arv/readings/old_mkss.pdf` What are $\mathfrak{so}(3)$, skew-symmetric matrices and the Rodrigues formula $R = \mathrm{I} + [\bar{\mathbf{v}}]_\times \sin\theta + [\bar{\mathbf{v}}]_\times^2 (1 - \cos\theta)$ used for?



Answer Given the time (τ) dependent orientation curve of an object $\mathrm{R}(\tau) : \mathbb{R} \to \mathrm{SO}(3)$, it must satisfy $\mathrm{R}(\tau)\mathrm{R}^T(\tau) = \mathrm{I}$. Its time derivative must thus satisfy: $\dot{\mathrm{R}}(\tau)\mathrm{R}^T(\tau) = -(\dot{\mathrm{R}}(\tau)\mathrm{R}^T(\tau))^T$, which means that $\dot{\mathrm{R}}(\tau)\mathrm{R}^T(\tau)$ is a skew-symmetric matrix.

Each skew-symmetric matrix can be uniquely defined by an angular velocity vector $\mathbf{v} = (v_1, v_2, v_3)^T \in \mathbb{R}^3$. The cross product $\times : \mathbb{R}^3 \to \mathbb{R}^3$ has the important property $\mathbf{v} \times \mathbf{p} = -\mathbf{p} \times \mathbf{v}$. Since it is a linear map, it can also be represented by a matrix product $\mathbf{v} \times \mathbf{p} \equiv [\mathbf{v}]_\times \mathbf{p}$ with the cross product matrix operator $[.]_\times$:

$$[\mathbf{v}]_\times = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

A skew symmetric matrix has the important property $[\mathbf{v}]_\times = -[\mathbf{v}]_\times^T$, in analogy to its derivation via the cross product. $[.]_\times : \mathbb{R}^3 \to \mathfrak{so}(3) \subset \mathbb{R}^{3\times3}$ is an isomorphism between $\mathbb{R}^3$ and the space $\mathfrak{so}(3)$ of all 3×3 skew symmetric matrices, which are the tangent space at the identity of the matrix group SO(3):

$$\boxed{\mathfrak{so}(3) = \{[\mathbf{v}]_\times \in \mathbb{R}^{3\times3} \mid \mathbf{v} \in \mathbb{R}^3\}}$$

Given an angular velocity vector $\mathbf{v} \in \mathbb{R}^3$, the instantaneous rotation defined by it can be obtained by the following chain of operations and vice versa:

$$\mathbf{v} \in \mathbb{R}^3 \underset{[.]_\times^{-1}}{\overset{[.]_\times}{\rightleftharpoons}} [\mathbf{v}]_\times \in \mathfrak{so}(3) \underset{\log}{\overset{\exp}{\rightleftharpoons}} \mathrm{R} \in \mathrm{SO}(3)$$

The matrix exponential exp has an exact closed form Taylor series expansion, the Rodrigues' rotation formula, which can be used to convert from Axis-Angle $(\bar{\mathbf{v}}, \theta)$, with $\mathbf{v} = \theta\bar{\mathbf{v}}$, its norm $\theta = |\mathbf{v}| \in \mathbb{R}$ and normalized axis unit vector $\bar{\mathbf{v}} \in \mathbb{S}^2$ to a rotation matrix:

$$\mathrm{R} = \exp[\mathbf{v}]_\times = \exp[\theta\bar{\mathbf{v}}]_\times = \mathrm{I} + [\bar{\mathbf{v}}]_\times \sin\theta + [\bar{\mathbf{v}}]_\times^2 (1 - \cos\theta) \tag{5}$$

d) How to approximate / linearize infinitesimal Rotations?

Answer When using the small angle $|\theta| << 1$ approximation, it holds that $sin(\theta) \approx \theta$ so by approximating $\sin(\theta)$ with θ and dropping the second and higher order terms from (5), we gain a first-order Taylor approximation of infinitesimal rotations. This approximation is linear in $\mathbf{v}$ and is thus often used when approximate orientations via linear algebra techniques or iteratively in optimization.

$$\mathrm{R} \approxeq \mathrm{I} + [\bar{\mathbf{v}}]_\times \theta = \mathrm{I} + [\bar{\mathbf{v}}\theta]_\times = \mathrm{I} + [\mathbf{v}]_\times \tag{6}$$

e) In the LM Algorithm, line 8, how do you apply the update $x \leftarrow x + \Delta x$ if you were to choose to represent $x \in \mathrm{SO}(3)$, but $\Delta x \in \mathfrak{so}(3)$? What is an easier alternative?

Answer Since x is in the Lie Group, meaning a Rotation Matrix $\mathrm{R} \in SO(3)$ and Δx in the Lie Algebra, meaning a skew-symmetric matrix, just adding them does not ensure that the result is still a rotation matrix, since the group of rotation matrices is only closed under multiplication with other rotation matrices, but not under addition with other types of matrices. The additive increment has to be changed to a multiplicative increment to preserve the group structure: $\mathrm{R} \leftarrow \mathrm{R} \cdot \Delta\mathrm{R}$. The increment $\Delta x = [\mathbf{v}]_\times$ has to be converted to the group $\Delta\mathrm{R}$ using the exponential map (5) $\Delta R = \exp(\Delta x) = \exp[\mathbf{v}]_\times$ or its infinitesimal approximation (6) $\Delta R = \mathrm{I} + \Delta x$. Alternatively, we may just represent both x and Δx using exponential coordinates $\mathbf{v}$ as global rotation parameterization. This allows to keep the plus operation, we may just convert to Rotation matrix when evaluation the error (in line 7 of the algorithm).

f) What are the two options given for the derivative of a rotation $\mathrm{R}(\mathbf{v}) = \exp([\mathbf{v}]_\times)$ with respect to its exponential coordinates $\mathbf{v}$ in *Guillermo Gallego and Anthony Yezzi (2014): A compact formula for the derivative of a 3-D rotation in exponential coordinates.* `https://arxiv.org/pdf/1312.0788.pdf` and which formula do you prefer in a numerical optimization method?

Answer The 2 choices are given in their equations (7) and (9).

The first one (7) stems from decomposing the exponential coordinates $\mathbf{v}$ into an axis $\hat{\mathbf{v}}$ and angle θ part and then computing the derivative of the Euler-Rodrigues formula and is e.g. used in OpenCV. However, extracting the angle and axis is cumbersume and the formula still involves trigonometric functions like cos.

To get the more compact latter one (9), they first compute the derivative of the product $\mathrm{R}\mathbf{u}$ (meaning the effect of a rotation applied to a 3D vector $\mathbf{u}$) to finally arrive at a more compact formula, withouth ever decomposing $\mathbf{v}$ into axis and angle and withouth the need for trigonometric functions. It is numerically more friendly for optimization methods than the previous one thus I would choose this representation. Since this is the main result of their paper, we restate it here with $\boldsymbol{e}_i$ being the i-th vector of the standard basis in $\mathbb{R}^3$:

$$\frac{\partial \mathrm{R}}{\partial v_i} = \frac{v_i[\mathbf{v}]_\times + [\mathbf{v} \times (\mathrm{I} - \mathrm{R})\mathbf{e}_i]_\times}{\|\mathbf{v}\|^2}\mathrm{R} \tag{9}$$