Robotics 2
Formulary

Martina Doku

April 16, 2024

Contents

1 Calibration

1.1 Error measure

The error measure is the difference between the true value r and the measured value r_{nom}. It is given by:

$$\Delta r = r - r_{nom} = \frac{\partial f}{\partial \alpha}\Delta\alpha + \frac{\partial f}{\partial a}\Delta a + \frac{\partial f}{\partial d}\Delta d + \frac{\partial f}{\partial \theta}\Delta\theta \tag{1}$$

1.2 Calibration Equation

The calibration equation is given by:

$$\Delta\psi = \begin{bmatrix}\Delta\alpha \\ \Delta a \\ \Delta d \\ \Delta\theta\end{bmatrix} \tag{2}$$

$$\phi = \begin{bmatrix}\frac{\partial f}{\partial \alpha} & \frac{\partial f}{\partial a} & \frac{\partial f}{\partial d} & \frac{\partial f}{\partial \theta}\end{bmatrix} \tag{3}$$

$$\Delta r = \phi \cdot \Delta\psi \tag{4}$$

Note: If we have more uknows than equations, we need to perform l experiments to solve the calibration equation. Thus we obtain:

$$\Delta\bar{r} = \bar{\phi} \cdot \Delta\psi \tag{5}$$

where $\Delta\bar{r}$ is the average error and $\bar{\phi}$ is the regressor matrix evaluated at nominal values, and $\Delta\psi$ is the unknowns.

1.3 Calibration Algorithm

The calibration algorithm is performed using the following steps:

1. $\Delta\bar{r} = \bar{\phi} \cdot \Delta\psi$
2. $\Delta\psi = (\bar{\phi}^T \cdot \bar{\phi})^{-1} \cdot \bar{\phi}^T \cdot \Delta\bar{r}$
3. $\psi_{nom} + \Delta\psi = \psi'$
4. $\Delta\bar{r}' = \bar{\phi}' \cdot \Delta\psi$ where $\bar{\phi}'$ is the regressor matrix evaluated at the new values.
5. repeat from step 2

1.3.1 ex: write regressor matrix from a dh table

The first step is to obtain the position vector from the DH table. Then, for each component (a, α, d, θ) that we are interested in, we build the jacobian $\frac{\partial f}{\partial \alpha}$... . The regressor equation is then given by:

$$\Delta p = J_a \Delta a + J_\alpha \Delta \alpha + J_d \Delta d + J_\theta \Delta \theta = \Phi \Delta \phi \tag{6}$$

with:

$$\Phi = \begin{bmatrix} J_a & J_\alpha & J_d & J_\theta \end{bmatrix} \tag{7}$$

Note: if we are interested ony inthe fine tuning of some parameters, we can exclude the others from all the equations.

1.3.2 ex: find optimal link length from positions and joints

The steps are the following:

1. Compute the position vector from the forward kinematics on the given joint measures and expected link lengths (if given)
2. Check if the position vector is the same as the one measured, if not go to step 3
3. rewrite the position vector as a function of the link lengths:
$$\Delta p = \begin{bmatrix} \Delta p_a \\ \Delta p_b \\ \Delta p_c \\ \Delta p_d \end{bmatrix} = \begin{bmatrix} \phi(q_a) \\ \phi(q_b) \\ \phi(q_c) \\ \phi(q_d) \end{bmatrix} \Delta l$$
4. where a,b,c,d are the different joint measures
5. solve the calibration equation for l as: $\Delta l = \Phi^{\#} \Delta p$

2 Inverse Kinematics with Redundancy

The resolution of redundancy can be done using two methods:

1. **Local methods**: the local methods are online methods that are used to solve the redundancy problem. They are a Linear Quadratic problems.
2. **Global methods**: the global methods are offline methods. They are used to solve the redundancy problem. They are a non-linear problem.

2.1 Local methods

The three kinds of local methods are:

1. **Jacobian-based methods**: among the infinite solutions, one is chosen, e.g., that minimizes a suitable (possibly weighted) norm.The problem of this method is that it might encounter singularities and it can lead to non repeatable position in joint space.

2. **Null-space methds**: a term is added to the previous solution so as not to affect execution of the task trajectory

3. **Task-augmentation methods**: redundancy is reduced/eliminated by adding $S \leq N - M$ additional tasks

2.2 Jacobian-based methods

The Jacobian-based methods are used to solve the redundancy problem. They give a solution in the form:

$$\dot{q} = K(q) \cdot \dot{r} \tag{8}$$

where K is a generalized inverse of the Jacobian matrix. Possible choises for K are:

- Pseudo-inverse 2.2.1
- Weighted pseudo-inverse 2.2.1
- Damped Least Square pseudo-inverse 2.2.3: this method is used to avoid singularities.

2.2.1 Pseudo-inverse

If J is full rank, the pseudo-inverse is given by:

$$J^{\#} = J^T (J^T J)^{-1} \tag{9}$$

If J is not full rank, the pseudo-inverse it is computed by using the SVD decomposition (or pinv in Matlab).
Note: the pseudo-inverse is not unit independent.

2.2.2 Weighted pseudo-inverse

The weighted pseudo-inverse is given by:

$$J^{\#} = W^{-1}J^T(JW^{-1}J^T)^{-1} \tag{10}$$

Note: it is NOT a pseudoinverse but it shares some properties.
Note: to find the velocity that **minimizes the Kinetic energy**, we have to use the weighted pseudo-inverse wit the **inertia matrix** as weight.

2.2.3 Damped Least Square pseudo-inverse

The damped least square pseudo-inverse is given by:

$$J^{\#} = J^T(JJ^T + \mu^2 I)^{-1} \tag{11}$$

where μ is a damping factor (the larger μ, the larger the error)

2.3 Null-space methods

The null space methods are used to solve the redundancy problem. They give a solution in the form:

$$\dot{q} = J^{\#} \cdot \dot{r} + (I - J^{\#}J) \cdot \dot{q}_0 \tag{12}$$

where $\dot{q}_0$ is the solution of the primary task. We have mainly two methods:

- **Projected gradient method** 2.3.1
- **Reduced gradient method**2.3.2

2.3.1 Projected gradient method

The solution for the projected gradient method is obtained by the minimization of the following function:

$$H = \frac{1}{2}(\dot{q} - \dot{q}_0)^T W(\dot{q} - \dot{q}_0) \tag{13}$$

The solution is given by:

$$\dot{q} = J^{\#} \cdot \dot{r} + (I - J^{\#}J) \cdot \dot{q}_0 \tag{14}$$

or

$$\dot{q} = J^{\#} \cdot \dot{r} + (I - J^{\#}J) \cdot -\nabla H_{range} \tag{15}$$

Possible values of H
The possible values of H are:

- manipulanility: $H = \sqrt{det(JJ^T)}$
- joint limits: $H_{range} = \frac{1}{2N} \sum_{i=1}^{n} \frac{(q_i - \bar{q_i})^2}{(q_{max_i} - q_{min_i})^2}$
- obstacle avoidance: $H = \min_{a \in robot} \min_{b \in obstacle} ||a(q) - b||^2$

2.3.2 Reduced gradient method

The solution is given by:

$$\dot{q} = \begin{bmatrix} \dot{q}_a \\ \dot{q}_b \end{bmatrix} = \begin{bmatrix} J_a^{-1} \\ 0 \end{bmatrix} \dot{r} + \begin{bmatrix} -J_a^{-1} J_b \\ I \end{bmatrix} \cdot \begin{bmatrix} -(J_a^{-1} J_b)^T & I \end{bmatrix} \nabla_q H \tag{16}$$

Where J_a is the greatest non singular minor of J and J_b is the other minor.
Finding the non singular minor
The non singular minor can be found by using the following steps:

1. Compute the Jacobian J
2. compute the determinant of all submatrices of J
3. select the minor with the greatest determinant (it must be non singular)

2.4 Task augmentation methods

The task augmentation methods are used to solve the redundancy problem. This method gives a solution that satiisfis the several task in pririty order in the following way:

$$\dot{q} = J_1^{\#} \dot{r}_1 + (I - J_1^{\#} J_1) v_1 \tag{17}$$

where v_1 is made to soatisfy also (possibly) the lower priority task.

$$v_1 = (J_2 P_1)^{\#} (\dot{r}_2 - J_2 J_1^{\#} \dot{r}_1) + (I - (J_2 P_1)^{\#} (J_2 P_1))^{\#} v_2 \tag{18}$$

2.4.1 General recursive task priority formulation

In the general recursive task priority formulation, we have to do the following:

1. we start with $\dot{q}_0 = 0$ and $p_{A0} = I$
2. we compute the solution at kth level as: $\dot{q}_k = \dot{q}_{k-1} + (J_k P_{A,k-1})^{\#} (\dot{r}_k - J_k \dot{q}_{k-1})$

3. we compute the projection matrix as: $P_{A,k} = P_{A,k-1} - (J_k P_{A,k})^{\#} J_k P_{A,k-1}$

where $P_{A,k-1} = I - J^{\#}_{A,k-1} J_{A,k-1}$ and $J_{A,k-1}$ is the jacobian of the k-1 task.

2.4.2 SNS method

The SNS method is used to solve the redundancy problem whith hard constraints. The algorithm is the following:

1. Initialize W matrix with 1 on the diagonal if the joint is enabled, 0 if it is not
2. Compute the solution as: $\dot{q} = J^{\#} \dot{r}$
3. Check joint velocity limits: $\dot{q}_{min} \leq \dot{q} \leq \dot{q}_{max}$
4. Compute task scaling factor on the most critical joint
5. If a larger scaling factor is found, update W matrix
6. Disable the most critical joint by forcing it to its max velocity and repeat from step 2

2.4.3 ex: find the acceleratoion command that satisfies the limits

We have to do the following:

1. Compute the acceleration command $\ddot{q} = J(q)^{\#}(\ddot{p} - \dot{J}(q)\dot{q})$
2. Check if the acceleration command satisfies the limits
3. $\ddot{Q}_{min} = \max(A_{min}, \frac{V_{min} - \dot{q}}{T_c})$
4. $\ddot{Q}_{max} = \min(A_{max}, \frac{V_{max} - \dot{q}}{T_c})$
5. we have to check that $\ddot{Q}_{min} \leq \ddot{q} \leq \ddot{Q}_{max}$
6. if not we have to apply SNS algorithm

2.4.4 ex: find the joint velocity that executes the task and that satisfies the joint velocities bound

We have to do the following:

1. Compute $\dot{q} = J(q)^{\#}\dot{p}$
2. Check if the joint velocity satisfies the limits
3. If not find the most exceeding joint i and recompute the velocity $\dot{p}$ as $\dot{p}_{new} = \dot{p} - J_i * V_i$ where J_i is the i-th column of the Jacobian and V_i is the i-th joint max velocity
4. re compute $\dot{q}$ as $\dot{q} = (V_i * [(J_1 \dots J_{i-1} J_{i+1} \dots J_n)^{\#}\dot{p}_{new}]^T)^T$

2.5 Which method to choose?

The reduced gradient method allows is analitically simpler and allows to converge faster but requires the search of a non singular minor.

2.5.1 ex: find the joint velocity with minimum norm that executes the task

The steps are the following:

1. Compute the Jacobian J
2. Compute the rank of J
3. If the rank of J is fulls, use the pseudo-inverse
4. Otherwise us the pseudoinverse on J_i containing the J with only the independent rows use the corresponding rows in $\dot{r}$

3 Dynamic redundancy resolution

The dynamic redundancy resolution is used to solve the redundancy problem. We have the Linear Quadratic problem:

$$J(q)\ddot{q} = \ddot{x} = \ddot{r} - \dot{J}(q)\dot{q} \tag{19}$$

typical objectives are:

- **torque norm**: $H(q) = \frac{1}{2} * ||\tau||^2$
  soution: $\tau = (J(q)M^{-1}(q))^{\#}(\ddot{r} - \dot{J}(q)\dot{q} + J(q)M^{-1}(q)n(q,\dot{q}))$
  when: good for short trajectories, otherwise it can lead to torque oscillations

- **(squared inverse inertia weighted) torque norm**: $H(q) = \frac{1}{2}||\tau||^2_{M^{-2}}$
  soution: $\tau = M(q)J^{\#}(q)(\ddot{r} - \dot{J}(q)\dot{q} + J(q)M^{-1}(q)n(q,\dot{q}))$
  when: to be preferred, good performance in general

- **(inverse inertia weighted) torque norm**: $H(q) = \frac{1}{2}||\tau||^2_{M^{-1}}$
  soution: $\tau = J^T(q)(J(q)M^{-1}(q)J^T(q))^{-1} * (\ddot{r} - \dot{J}(q)\dot{q} + J(q)M^{-1}(q)n(q,\dot{q}))$

4 Dynamic Model

We have mainly two ways to obtain the dynamic model of a robot:

1. **Euler-Lagrangian method**: to obtain the dynamic model of a robot in **closed form**
   best for : studyinng dynamic properties and control schemes

2. **Newton-Euler method**: to obtain the dynamic model of a robot in a **recursive way**.
   best for : implementation of contol schemes

5 Euler-Lagrangian method

5.1 Euler Lagrange equation

The Euler Lagrange equation is given by:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = u \tag{20}$$

where L is the Lagrangian of the robot and u are the non conservative forces. The Lagrangian is given by:

$$L(q,\dot{q}) = T(q,\dot{q}) - U(q) \tag{21}$$

where T is the kinetic energy and U is the potential energy.

5.2 Dynamic equation

The full dynamic equation is given by:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = u \tag{22}$$

where:

- $M(q)$ is the inertia matrix
- $C(q,\dot{q})$ is the coriolis and centrifugal matrix
- $G(q)$ are the gravity matrix
- u is the non conservative forces
- $\ddot{q}$ is the joint acceleration
- $\dot{q}$ is the joint velocity
- q is the joint position

5.2.1 ex: find the dynamic equation of the robot

The steps are the following:

1. Start from defining the kinetic energy of the robot as: $T_i = \frac{1}{2} m_i \dot{r_i}^2 + \frac{1}{2} \omega_i^T I \omega_i$
2. Compute the kinetic energy of the robot as: $T = \sum_{i=1}^{n} T_i$
3. Compute the inertia matrix as: $M(q) = 2 * \frac{\partial T}{\partial \dot{q}^2}$
4. Find the coriolis and centrifugal matrix as in 5.4
5. Find the gravity vector as in 5.7
6. Compute the friction torque
7. Compute u as: $u = \tau - \tau_{friction}$ where τ is the commanded torque
8. Compute the dynamic equation as: $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = u$

5.3 Inertia matrix

The inertia matrix is a matrix that contains the inertia of the robot, it can be derived by the following **foundamental relation**:

$$T = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{23}$$

where T is the kinetic energy of the robot.

5.3.1 Properties

The properties of the inertia matrix are:

- $M(q)$ is symmetric
- $M(q)$ is positive definite
- $M(q)$ is full rank

Note: the inertia matix is never a function of the first coordinate q1.

5.3.2 Standard body inertia matrix

- **parallelepiped**: given sides a (height), b (width), c (depth) and mass m, the inertia matrix is given by:

$$I = \frac{m}{12}\begin{bmatrix} b^2+c^2 & 0 & 0 \\ 0 & a^2+c^2 & 0 \\ 0 & 0 & a^2+b^2 \end{bmatrix} \tag{24}$$

- **empty cylinder**: given inner radius a and outer radius b, height c and mass m, the inertia matrix is given by:

$$I = \frac{m}{12}\begin{bmatrix} 6(a^2+b^2) & 0 & 0 \\ 0 & 3a^2+3b^2+c^2 & 0 \\ 0 & 0 & 3a^2+3b^2+c^2 \end{bmatrix} \tag{25}$$

5.3.3 ex: find the inertia matrix of a robot given the DH table and position of COMs

The steps are the following:

1. The first thing we want to do is to compute the T matrix for each link as in 5.5
2. Now we can compute all the T_i and the final T as $T = \sum_{i=1}^{n} T_i$
3. Finally we can obtain the inertia matrix as: $M(q) = 2 * \frac{\partial T}{\partial \dot{q}^2}$

5.3.4 ex: check if a matrix is an inertia matrix

The steps are the following:

1. Check that the matrix is not dependent from the first coordinate (note, it can be dependent from the difference between the first and another coordinate, but not directly from the first coordinate)
2. Check if the matrix is symmetric
3. Check if the matrix is positive definite:
   - Check if the determinant is negative, in that case the matrix is not positive definite, if it is positive continue
   - Compute the eigenvalues of the matrix
   - Check if all the eigenvalues are positive, if any eigenvalue is negative the matrix is not positive definite

5.4 Coriolis and centrifugal matrix

The coriolis and centrifugal matrix is given by:

$$c_i(q, \dot{q}) = \dot{q}^T C_i(q) \dot{q} \tag{26}$$

where $C_i(q)$ is the matrix of **Christoffel symbols** obtained as:

$$C_i(q) = \frac{1}{2} \left(\frac{\partial M_i}{\partial q} + (\frac{\partial M_i}{\partial q})^T - \frac{\partial M}{\partial q_i} \right) \tag{27}$$

where M_i is the i-th column of the inertia matrix.

5.5 Kinetic Energy

The kinetic energy of a link is given by: (Konig theorem)

$$T_i = \frac{1}{2} m_i \, {}^0 v_{c,i}^T \, {}^0 v_{c,i} + \frac{1}{2} \omega_i^T I_{c,i} \omega_i \tag{28}$$

Note: ω_i and $I_{c,i}$ should be expressed in the same reference frame, but the product $\omega_i^T I_{c,i} \omega_i$ is invariant w.r.t. any chosen frame
An alternative way to compute the kinetic energy is:

$$T = \frac{1}{2} \dot{q}^T M(q) \dot{q} \tag{29}$$

5.5.1 ex: find the kinetic energy of a robot

To obtaint the kinetic energy of a robot we must calculate the kinetic energy of each link and sum them up.
To obtain T_i we have to do the following:

1. Check if the link is prismatic or revolute
2. If it is prismatic:
   - we just have to compute the linear velocity of the center of mass, that is given by: the linear velocities of the previous link + the linear velocity of the center of mass of the link expressed in function of $\dot{q}_i$
   - we can write the kinetic energy as: $T_i = \frac{1}{2} m_i \cdot^0 v_{c,i}^T \cdot^0 v_{c,i}$
3. If it is revolute:
   - we have to compute the linear velocity ${}^0v_{c,i}$ of the center of mass, that is given by ${}^0\dot{p_{c,i}}$
   - we have to compute the angular velocity ω_i of the link, that is given by the sum of the angual the sumof the angular velocities of the previous links and the current one (that is $\dot{q}_i$)
   - we can write the kinetic energy as: $T_i = \frac{1}{2} m_i \cdot^0 v_{c,i}^T \cdot^0 v_{c,i} + \frac{1}{2} \omega_i^T I_{c,i} \omega_i$
4. Finally we can sum all the T_i to obtain the kinetic energy of the robot

5.6 Foundamental relation

The foundamental relation is given by:

$$v_{c,i} = v_i + \omega_i \times r_{c,i} \tag{30}$$

5.6.1 Moving frames algorithm

The moving frames algorithm is used to compute the kinetic energy of a robot. The steps are the following:

1. initialize ${}^0v_0 = 0$ and ${}^0\omega_0 = 0$
2. define $\sigma = 0$ if the joint is revolute, $\sigma = 1$ if the joint is prismatic
3. compute ${}^i\omega_i = {}^{i-1}R_i^T(q_i) \cdot [{}^{i-1}\omega_{i-1} + (1-\sigma)\dot{q}_i^{i-1} z_{i-1}]$

4. compute

$$^{i}v_i = ^{i-1}R_i^T(q_i) \cdot [^{i-1}v_{i-1} + \sigma_i \dot{q}_i^{i-1} z_{i-1} + ^{i-1}\omega_i \times ^{i-1} r_{i-1,i}] \quad (31)$$

$$= ^{i-1}R_i^T(q_i) \cdot [^{i-1}v_{i-1} + \sigma_i \dot{q}_i^{i-1} z_{i-1}] + ^{i}\omega_i \times ^{i} r_{i-1,i} \quad (32)$$

5. compute $^{i}v_{c,i} = ^{i} v_i + ^{i}\omega_i \times ^{i} r_{c,i}$
6. compute $T_i = \frac{1}{2} m_i^i v_{c,i}^T v_{c,i} + \frac{1}{2} \cdot ^{i}\omega_i^T I_{c,i}^i \omega_i$

5.7 Potential Energy

The potential energy of a link is given by:

$$U_i = -m_i g r_{0,ci} \quad (33)$$

where: $r_{c,i}$ is the distance from the center of mass to the reference frame and **g** is the gravity vector (usually $[0, -9.81, 0]^T$).
Note: if you are given the distance of the center of ass from the previous link and not the general frame of reference, you have to multiply the vector with the transformation matrix (homogeneus) that you can obtain from the DH table

$$[r_{0,ci}, 1]^T = ^{0}A_1 \cdot ^{1}A_2 \ldots ^{i-1}A_i [r_{i,ci}, 1]^T \quad (34)$$

we have that:

$$U = U_1 + U_2 + \cdots + U_n \quad (35)$$

We can find the gravity vector $g(q)$ as:

$$g(q) = \begin{bmatrix} \frac{\partial U}{\partial q_1} \\ \vdots \\ \frac{\partial U}{\partial q_n} \end{bmatrix} \quad (36)$$

5.8 Energy Conservation

The total robot energy is expressed as:

$$E = T + U = \frac{1}{2}\dot{q}^T M \dot{q} + U(q) \quad (37)$$

The evolution of the total energy over time is given by:

$$\dot{E} = \dot{q}^T u + \frac{1}{2}\dot{q}^T (\dot{M}(q) - 2S(q, \dot{q}))\dot{q} \quad (38)$$

If the total energy is constant: $\dot{E} = \dot{q}^T u$

5.9 Structural properties in dynamic models

The coriolis term $C(q, \dot{q})$ can be rewritten as $S(q, \dot{q})\dot{q}$ where S is the skew-symmetric matrix.
The matrix $\dot{M}(q) - 2S(q, \dot{q})$ is symmetric.

5.9.1 ex: find matrix S that satisfies $\dot{M}(q) - 2S(q, \dot{q})$ is skewsymm

The steps are the following:

1. Compute the coriolis matrix as in 5.4
2. Compute the skew symmetric matrix as: $S(q, \dot{q}) = [S_1; ...; S_n]$ where S_i is the skew symmetric matrix obtained by multiplying $S_i = dq^T \cdot C_i$
3. Check if the matrix $\dot{M}(q) - 2S(q, \dot{q})$ is skew symmetric

6 APPENDIX

6.1 Pseudo-inverse properties

The properties of the pseudo-inverse are:

- $JJ^{\#}J = J$
- $J^{\#}JJ^{\#} = J^{\#}$
- $(JJ^{\#})^T = JJ^{\#}$
- $(J^{\#}J)^T = J^{\#}J$

6.2 Weighted pseudo-inverse properties

The properties of the weighted pseudo-inverse are:

- $JJ^{\#}J = J$
- $J^{\#}JJ^{\#} = J^{\#}$
- $(JJ^{\#})^T = JJ^{\#}$

6.3 DH frames

6.3.1 Assign axis

- z_i along the direction of joint i+1
- x_i along the common normal between z_{i-1} and z_i
- y_i completing the right-handed frame

6.3.2 DH table

- α_i is the angle from z_{i-1} to z_i about x_i
- a_i is the distance from z_{i-1} to z_i along x_i
- d_i is the distance from x_{i-1} to x_i along z_{i-1}
- θ_i is the angle from x_{i-1} to x_i about z_{i-1}

6.3.3 ex: Find q value that causes algorithmic singularity

The steps are the following:

1. Compute the Jacobian J for each of the tasks
2. Build the extended Jacobian as: $J_e = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix}$
3. Compute the determinant of the extended Jacobian
4. Find the q value that causes the determinant to be zero
5. Substitute the q value in the Jacobians
6. Compute the rank of both the original jacobians and the extended one
7. If the rank of the extended jacobian i.e. $rank(J_e) < rank(J_1) + rank(J_2)$ is less than the sum of the ranks of the original jacobians, we have an algorithmic singularity.