Robotics 2

Artificial Intelligence and Robotics

Martina Doku

August 20, 2024

Contents

1 Calibration

Calibration goal: recover as much as possible pose errors by correcting the nominal 14 set of D-H parameters, based on independent external measurements

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:

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

where $\frac{\partial f}{\partial \alpha}$, $\frac{\partial f}{\partial a}$, $\frac{\partial f}{\partial d}$, $\frac{\partial f}{\partial \theta}$ are the partial jacobians evaluated in nominal conditions

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. $\Delta\psi = \bar{\phi}^{\#}\Delta\bar{r} = (\bar{\phi}^T \cdot \bar{\phi})^{-1} \cdot \bar{\phi}^T \cdot \Delta\bar{r}$
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

Note: we use the pseudo-inverse to solve the calibration equation, it returns the minimum norm solution in case of underdetermined system, the minimum error solution in case of overdetermined system.

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 in the 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:
   $\Delta p = p - p_{nom} = \phi(q)\Delta l$
   $\Delta p = \begin{bmatrix} \Delta p_a \\ \Delta p_b \\ \Delta p_c \\ \Delta p_d \end{bmatrix} = \begin{bmatrix} \phi(q_a) \\ \phi(q_b) \\ \phi(q_c) \\ \phi(q_d) \end{bmatrix} \Delta l$ **Note:** Δp_a and co. are vertical vectors, they must be stacked vertically (if i have 3 measures in 2 dim, i obtain a 6x1 vector)
4. where a,b,c,d are the different joint measures
5. solve the calibration equation for l as: $\Delta l = \Phi^{\#} \Delta p$

1.3.3 ex: Theoretical question

Question: When is it convenient to use a two steps calibration?
Answer: It is convinient to use a two steps calibration when it is expected that the DH parameters will have very different uncertainty ranges.
The first step is performed only on high uncertainty parameters, the second on al of them. This improves the accuracy of the pseudoinverseof the regressor matrix

2 Inverse Kinematics with Redundancy

Types of redundancy:

- **kinematic redundancy**: the robot has more degrees of freedom than needed
- **components redundancy**: the robot has more components than needed
- **control redundancy**: the robot is redundant in its supervision architecture

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. (discrete approach)
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 Which method to choose?

If the exercise asks for the solution that:

- **minimizes the kinetic energy**: use the weighted pseudo-inverse with the inertia matrix as weight
- **minimizes the torque norm (or task error norn)**: use the pseudo-inverse
- **minimize norm of joint velocities**: use the pseudo-inverse
- **minimizes the error on the higher priority task**: use the task priority (recursive) method
- **minimizes the error on the lower priority task while correctly executing the higher priority one**: use the nullspace method or task priority (recursive) method (the second is better if you have more than 2 tasks)
- **minimizes the acceleration norm**: use the nullspace method as in 2.4.3

Note: if asked to find another solution without specific requirements, it is usually a solution that you can infer by looking at the jacobian and the desired velocity, without the need of any specific method.

2.2 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

2.3 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 2.3.1
- Weighted pseudo-inverse 2.3.1
- Damped Least Square pseudo-inverse 2.3.3: this method is used to avoid singularities.

2.3.1 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.

2.3.2 Weighted pseudo-inverse

The weighted pseudo-inverse is given by:

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

where larger weights are given to the most important tasks, usally the weights are proportional to the inverse of the joint ranges.
Note: it is NOT a pseudoinverse but it shares some properties 13.1.
Note: to find the velocity that **minimizes the Kinetic energy**, we have to use the weighted pseudo-inverse wit the **inertia matrix** as weight.

2.3.3 Damped Least Square pseudo-inverse

The damped least square pseudo-inverse is given by:

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

where μ is a damping factor (the larger μ, the larger the error)
It is a compromise between large joint velocity and task accuracy, it's used to obtain **robust behavior in the presence of singularities**, but in its basic version it always gives a task error.

2.4 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 a generic preferred joint velocity.
The equation is the sum of the solution of the primary task and the orthogonal projection of $\dot{q}_0$ on the null space of J (The null space N represents the space of joint velocities that do not affect the end-effector position) We have mainly two methods:

- **Projected gradient method** 2.4.1
- **Reduced gradient method**2.4.2

2.4.1 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

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

Note: there is a necessary condition of constrained optimality: $(I - J^{\#}J)\nabla H(q) = 0$
Possible values of H
The possible values of H are:

- manipulanility: $H = \sqrt{det(JJ^T)}$
- joint limits: $H_{range} = \frac{1}{2N}\sum_{i=1}^{n}\frac{(q_i - \bar{q}_i)^2}{(q_{max_i} - q_{min_i})^2}$ **Note:** In this case $\dot{q}_0 = -\nabla H_{range}$, we introduce the minus because we want to `minimize` the distance from the center of the joint range.
- obstacle avoidance: $H = \min_{a \in robot} \min_{b \in obstacle} ||a(q) - b||^2$

2.4.2 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. Compue the rank r of J
3. compute the determinant of all submatrices of J of size r
4. select the minor with the greatest (absolute) determinant (it must be non singular, so the determinant must be different from 0)

2.4.3 Resolution at acceleration level

The resolution at acceleration level is given by:

$$\ddot{q} = J(q)^{\#}(\ddot{r} - \dot{J}(q)\dot{q}) + (I - J(q)^{\#}J(q))\ddot{q}_0 \tag{17}$$

where $\ddot{q}_0$ is a generic preferred joint acceleration, that can be written as: $\ddot{q}_0 = -\nabla H_{range} - K_D\dot{q}$ where K_D is a damping factor. This yelds the minimum acceleration norm solution.

2.4.4 ex: find the joint velocity that minimizes the H_{range} and scale it if not feasible.

The steps are the following:

1. use the projected gradient method to find the joint velocity that minimizes the H_{range}
2. if the joint velocity is not feasible we want to find a scaling factor k that makes it feasible. We want $\dot{r}new = k\dot{r}$
3. since we can write the new version of the $\dot{q}$ (that is $\dot{q}_{max}$) by substituting $\dot{r}$ with $\dot{r}new$ in the projected gradient method equation
4. we extract k from the equation, and we compute the new $\dot{r}$ as $\dot{r}new = k\dot{r}$

2.4.5 ex: find a solution in case of presence of task error

The steps are the following:

1. to f ind the acceleration that brings the task error back to zero we modify the tsandard equation by adding the PD terms: $\ddot{q} = -K_v\dot{q} + J(q)^{\#}(\ddot{r} + K_D(\dot{r}_d - J_r(q)\dot{q}) + K_P(r_d - p_x(q))) - (I - J(q)^{\#}J(q))\ddot{q}_0 + J_r(q)K_v\dot{q}$

2.5 Task augmentation methods

The task augmentation methods are used to solve the redundancy problem.

2.5.1 Task Priority

This method gives a solution that satisfies the several task in priority order in the following way:

$$\dot{q} = J_1^{\#}\dot{r}_1 + (I - J_1^{\#}J_1)v_1 \tag{18}$$

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

$$v_1 = (J_2P_1)^{\#}(\dot{r}_2 - J_2J_1^{\#}\dot{r}_1) + (I - (J_2P_1)^{\#}(J_2P_1))^{\#}v_2 \tag{19}$$

2.5.2 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.

2.5.3 ex: check if the tasks can be executed in parallel

The steps are the following:

1. Compute the Jacobian of the two tasks
2. Compute the rank of the two Jacobians and the rank of the extended Jacobian
3. If the rank of the extended Jacobian is equal to the sum of the ranks of the two Jacobians, the tasks can be executed in parallel
4. If the rank of the extended Jacobian is less than the sum of the ranks of the two Jacobians, the tasks can not be executed in parallel

2.6 SNS method

The SNS method is used to solve the redundancy problem whith hard constraints. It saturate one overdriven joint command at a time, until a feasible and better performing solution is found.

$$\dot{q} = (JW)^{\#} s\dot{x} + (I - (JW)^{\#} J)\dot{q}_N \tag{20}$$

where s is the scaling factor, W is the diagonal 0-1 matrix , $\dot{q}_N$ contains saturated joint velocities.
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

2.6.1 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 do:
7. $\ddot{p}_{new} = \ddot{p} - J_i A_i$ where i is the index of the most exceeding joint J_i is the i-th column of the Jacobian and A_i is the i-th joint max acceleration
8. re compute $\ddot{q}$ as $\ddot{q} = J(q)^{\#}_{new}(\ddot{p}_{new} - \dot{J}(q)_{new}\dot{q}_{new})$ where $J(q)_{new}$ is the Jacobian with the i-th column removed and $J(q)_{new}$ is the i-th joint max acceleration

2.6.2 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 $\dot{q} = [(J_1 \dots J_{i-1} J_{i+1} \dots J_n)^{\#}\dot{p}_{new}]$

2.6.3 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:

$$J(q)\ddot{q} = \ddot{x} = \ddot{r} - \dot{J}(q)\dot{q} \tag{21}$$

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: $\tau = J^T(q)(J(q)M^{-1}(q)J^T(q))^{-1} * (\ddot{r} - \dot{J}(q)\dot{q} + J(q)M^{-1}(q)n(q,\dot{q}))$

Note: $n(q,\dot{q})$ is the sum of (coriolis, centrifugal, gravity) torques $c(q,\dot{q})$ and $g(\dot{q})$.

3.1 Linear Quadratic problem

To solve a linear quadratic problem we have to follow the following steps:

1. Start from defining the objective function as: $H(q) = \frac{1}{2}(\ddot{q} - \ddot{q}_0)^T W(\ddot{q} - \ddot{q}_0)$ or $H(q) = \frac{1}{2}(\dot{q} - \dot{q}_0)^T W(\dot{q} - \dot{q}_0)$ (identify your weight matrix from there)
2. we have to set the constraints as $Jx = y$ and we have to identify the value of x and y in our case (typically if x is $\ddot{q}$, y is $\ddot{r} - \dot{J}\dot{q}$ while if x is $\dot{q}$, y is $\dot{r}$)
3. find x_0 by setting the derivative of the objective function to 0
4. we have to compute the solution as: $x = x_0 + W^{-1}J^T(JW^{-1}J^T)^{-1}(y - Jx_0)$

3.1.1 standard velocity case

The Linear Quadratic problem is given by:

$$\min H(q) = \frac{1}{2}(\dot{q} - \dot{q}_0)^T W(\dot{q} - \dot{q}_0) \tag{22}$$

s.t.

$$J(q)\dot{q} = \dot{r} \tag{23}$$

The solution is given by:

$$\dot{q} = \dot{q}_0 + J^{\#}(q)_W(\dot{r} - J(q)\dot{q}_0) \tag{24}$$

which is equal to:

$$\dot{q} = J^{\#}_W\dot{r} + (I - J^{\#}_W J)\dot{q}_0 \tag{25}$$

3.2 Quadratic programming

The quadratic programming is used to solve the redundancy problem, it is a generalization of the linear quadratic problem. The general form of the quadratic programming is given by:

$$\min \frac{1}{2}||J(q)\ddot{q} - \ddot{r}||^2 + \frac{1}{2}||\omega||^2 \tag{26}$$

s.t.

$$C(q, \dot{q})\dot{q} - \omega \leq d \tag{27}$$

with ω the slack variable, $C(q, \dot{q})$ the constraint matrix and d the constraint vector. The solution is given by:

4 Dynamic Model

The **dynamic model** of a robot provides the relationship between the generalized forces u and the assumed configurations over time $q(t)$ in form of a set of differential equations $\phi(q, \dot{q}, \ddot{q}) = u$

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:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = u \tag{28}$$

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

$$L(q, \dot{q}) = T(q, \dot{q}) - U(q) \tag{29}$$

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

5.2 Dynamic equation

The full dynamic equation is given by:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = u \tag{30}$$

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

alternatively we can write the dynamic equation as:

$$M(q)\ddot{q} + S(q, \dot{q})\dot{q} + G(q) = u \tag{31}$$

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 5.6
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**:

$$T = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{32}$$

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:

$$I = \frac{m}{12}\begin{bmatrix} b^2+c^2 & 0 & 0 \\ 0 & a^2+c^2 & 0 \\ 0 & 0 & a^2+b^2 \end{bmatrix} \tag{33}$$

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

$$I = \frac{m}{12}\begin{bmatrix} 6(a^2+b^2) & 0 & 0 \\ 0 & 3a^2+3b^2+c^2 & 0 \\ 0 & 0 & 3a^2+3b^2+c^2 \end{bmatrix} \tag{34}$$

fundamental kinematic relation ofr rigid bodies: $v = v_{cm} + \omega \times r = v_{cm} + S(\omega)r$

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:

$$c_i(q, \dot{q}) = \dot{q}^T C_i(q) \dot{q} \tag{35}$$

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

$$C_i(q) = \frac{1}{2} \left(\frac{\partial M_i}{\partial q} + (\frac{\partial M_i}{\partial q})^T - \frac{\partial M}{\partial q_i} \right) \tag{36}$$

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)

$$T_i = \frac{1}{2} m_i \cdot^0 v_{c,i}^T \cdot^0 v_{c,i} + \frac{1}{2} \omega_i^T I_{c,i} \omega_i \tag{37}$$

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:

$$T = \frac{1}{2} \dot{q}^T M(q) \dot{q} \tag{38}$$

5.5.1 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
$$\begin{aligned} {}^i v_i &=^{i-1} R_i^T(q_i) \cdot [{}^{i-1}v_{i-1} + \sigma_i \dot{q}_i^{i-1} z_{i-1} +^{i-1} \omega_i \times^{i-1} r_{i-1,i}] && (39) \\ &=^{i-1} R_i^T(q_i) \cdot [{}^{i-1}v_{i-1} + \sigma_i \dot{q}_i^{i-1} z_{i-1}] +^i \omega_i \times^i r_{i-1,i} && (40) \end{aligned}$$
5. compute ${}^i v_{c,i} =^i v_i +^i \omega_i \times^i r_{c,i}$
6. compute $T_i = \frac{1}{2} m_i^i v_{c,i}^T v_{c,i} + \frac{1}{2} \cdot^i \omega_i^T I_{c,i}^i \omega_i$

where ${}^i r_{c,i}$ is the distance of the COM from frame i, ${}^i r_{i-1,i}$ is the distance of the joint i from frame i-1, ${}^{i-1}z_{i-1}$ is the z axis $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

5.5.2 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 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 Potential Energy

The potential energy of a link is given by:

$$U_i = -m_i g^T r_{0,ci} \tag{41}$$

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

$$[r_{0,ci}, 1]^T =^0 A_1 \cdot^1 A_2 \ldots^{i-1} A_i [r_{i,ci}, 1]^T \tag{42}$$

we have that:

$$U = U_1 + U_2 + \cdots + U_n \tag{43}$$

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

$$g(q) = \begin{bmatrix} \frac{\partial U}{\partial q_1} \\ \vdots \\ \frac{\partial U}{\partial q_n} \end{bmatrix} \tag{44}$$

Note: the gravity term is equal to zero (g(q)=0) when:

- we are moving horizontally (constant U)
- we have static balancing (distribution of masses)
- we have mechanical compensation (system of springs or closed kinematic chains)

5.6.1 ex:Define suitable relations between the link masses, lengths, and CoM position given an expected gravity term

The steps are the following:

- express the position of the COMs wrt the origin frame (using the DH values to find the transformation matrices) $^0r_{ci} =^0 A_i^i r_{ci}$
- find $U_i = -m_i g r_{ci}$
- write g(q) = $\frac{\delta U}{\delta q_i}$
- impose g(q) = expected gravity term and find the relations between the link masses, lengths and CoM positions

5.6.2 ex:Define a tight bound on the norm of the square matrix $\frac{\partial g}{\partial q}$

The steps are the following:

- first compute the gravity vector g(q)
- then perform $\frac{\partial g}{\partial q} = \frac{^2\partial U}{\partial q^2}$
- compute the norm as: $||\frac{\partial g}{\partial q}|| = \sqrt{\lambda_{max}}$ where λ_{max} is the maximum eigenvalue of the matrix $(\frac{\partial g}{\partial q}^T \frac{\partial g}{\partial q})$
- find the bound α as the highest possible value of the norm (by setting arbitrarly q values)

5.7 Energy Conservation

The total robot energy is expressed as:

$$E = T + U = \frac{1}{2}\dot{q}^T M \dot{q} + U(q) \tag{45}$$

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

$$\dot{E} = \dot{q}^T u + \frac{1}{2}\dot{q}^T(\dot{M}(q) - 2S(q,\dot{q}))\dot{q} \tag{46}$$

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

5.8 Additional dynamic terms

5.8.1 Friction torque

The viscous friction torque is given by:

$$u_{V,i} = -F_{V,i}\dot{q}_i \tag{47}$$

The Coulomb friction torque is given by:

$$u_{C,i} = -F_{C,i}sign(\dot{q}_i) \tag{48}$$

5.8.2 Electrical motors

motor i mounted on link $i-1$ (or before) often with its spinning axis aligned with joint axis i.
The mass is added to the link $i-1$ and the inertia is added to the robot kinetic energy.

5.8.3 Complete dynamic model

The complete dynamic model is given by:

$$(M(q) + B_m(q))\ddot{q} + C(q,\dot{q})\dot{q} + g(q) + F_v\dot{q} + F_c sign(\dot{q}) = \tau \tag{49}$$

where $B_m(q)$ is the motor inertia matrix, F_v is the viscous friction coefficient and F_c is the Coulomb friction coefficient and τ are motor torques after the reduction gears.
To introduce **elasticity** we can add a term $K(q)q$ to the left side of the equation.

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

5.9.2 ex: find a different (non standard) matrix S' that satisfies $\dot{M}(q) - 2S'(q, \dot{q})$ is skewsymm and a matrix S" that does not satisfy the property

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. Find a different matrix S'= S + SS that satisfies the property by adding a skew symmetric matrix SS to the original S (Note: $SS * \dot{q}$=0)
4. Find a matrix S" = S + SS that does not satisfy the property by adding to the original S a matrix that is not skew symmetric SS but still is a feasible factorization (i.e. $S''(q, \dot{q})\dot{q} = c(q, \dot{q})$, ie $SS * \dot{q} = 0$)

6 Linear parametrization of the dynamic model

Each link is characterized by 10 dynamic parameters:

- 1 mass m_i
- 3 coordinates of the center of mass $r_{c,i} = \begin{bmatrix} r^x_{c,i} & r^y_{c,i} & r^z_{c,i} \end{bmatrix}^T$
- 6 elements of the inertia matrix $I_{c,i} = \begin{bmatrix} I^{xx}_{c,i} & I^{xy}_{c,i} & I^{xz}_{c,i} & I^{yy}_{c,i} & I^{yz}_{c,i} & I^{zz}_{c,i} \end{bmatrix}^T$ (the upper triangular part)

Both Potential and Kinetic energy can be expressed as a linear combination of the dynamic parameters pi:

$$T_i = \frac{1}{2} m^i_i v_i^T \cdot^i v_i + \frac{1}{2} \omega_i^T I^i_{c,i} \omega_i + m^i_i r^T_{c,i} S(^i v_i) *^i \omega_i \tag{50}$$

$$: U_i = -m_i g_0^T r_i - g_0^T \cdot^0 R_i m^i_i r_{c,i} \tag{51}$$

Since the dynamic model is linear on T and U, the dynamic model can be expressed as:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = Y_\pi(q, \dot{q}, \ddot{q})\pi = u \tag{52}$$

where $Y_\pi(q, \dot{q}, \ddot{q})$ is the regressor matrix that has a block upper triangular structure.
Important: the dynamic model can be rewritten as:

$$Y(q, \dot{q}, \ddot{q})a = Y_\pi(q, \dot{q}, \ddot{q})\pi \tag{53}$$

where a is the vector (px1) of the dynamic coefficients (ie the combination of the dynamic parameters that matter in the dynamic model)

6.1 Minimal parametrization

Determining the minimal parametrization means to find the smallest p that allows to express the dynamic model.

6.1.1 ex: find the regressor matrix Y for a parametrized matrix M such that M=Ya

The steps are the following:

1. Obtain Y on matlab as $Y = jacobian(M, a)$
2. In mathematically terms we have to compute the derivative of M w.r.t. a

7 Newton-Euler method

7.1 Newton and Euler equations

7.1.1 Newton equation

The Newton equation is given by:

$$\sum f_i = m \cdot \dot{v}_c \tag{54}$$

where f_i are force acting on the link and v_c is the linear velocity of the center of mass.
By the principle of action and reaction we have that:

$$f_i - f_{i+1} + m_i g = m_i \dot{v}_{c,i} \tag{55}$$

where f_i is the force from link i-1 to i, f_{i+1} is the force applied from i on i+1, m_i is the mass of the link and g is the gravity vector. **Note:** all vectors must be expressed in th same reference frame.

7.1.2 Euler equation

The Euler equation is given by:

$$\sum \mu_i = I \cdot \dot{\omega} + \omega \times I\omega \tag{56}$$

where μ_i are the torques acting on the link and ω is the angular velocity of the link and I is the inertia matrix.
By the principle of action and reaction we have that:

$$\tau_i - \tau_{i+1} + f_i \times r_{i-1,ci} + f_{i+1} \times r_{i,ci} = I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i) \tag{57}$$

where τ_i is the torque acting on the link, τ_{i+1} is the torque acting on the next link, I_i is the inertia matrix of the link and ω_i is the angular velocity of the link.

7.2 Recursive Newton-Euler algorithm

The recursive Newton-Euler algorithm is used to compute the dynamic model of a robot. It works as folows:

7.2.1 Initialization

set:

- ${}^0\omega_0 = 0$
- ${}^0\dot{\omega}_0 = 0$
- ${}^0a_0 = 0$
- $f_{N+1} = 0$
- $\tau_{N+1} = 0$

7.2.2 Forward recursion

- Compute ${}^i\omega_i$ and ${}^i\dot{\omega}_i$ as:

$$ {}^i\omega_i =^{i-1} R_i^T \cdot [{}^{i-1}\omega_{i-1} + \dot{q}_i^{i-1} z_{i-1}] \tag{58}$$

$$ {}^i\dot{\omega}_i =^{i-1} R_i^T \cdot [{}^{i-1}\dot{\omega}_{i-1} + \ddot{q}_i^{i-1} z_{i-1} + \dot{q}_i^{i-1}\omega_{i-1} \times^{i-1} z_{i-1}] \tag{59}$$

- Compute ${}^ia_i$ as:

$$ {}^ia_i =^{i-1} R_i^T \cdot ({}^{i-1}a_{i-1}) +^i \dot{\omega}_i \times^i r_{i-1,i} +^i \omega_i \times ({}^i\omega_i \times^i r_{i-1,i}) \tag{60}$$

- Compute ${}^ia_{ci}$ as:

$$ {}^ia_{ci} =^i a_i +^i \dot{\omega}_i \times^i r_{c,i} +^i \omega_i \times ({}^i\omega_i \times^i r_{c,i}) \tag{61}$$

7.2.3 Backward recursion

- Compute ${}^if_i$ as:

$$ {}^if_i =^i R_{i+1} \cdot^{i+1} f_{i+1} + m_i \cdot ({}^ia_{ci} -^i g) \tag{62}$$

- Compute ${}^i\tau_i$ as:

$$ {}^i\tau_i =^i R_{i+1} \cdot^{i+1} \tau_{i+1} + ({}^iR_{i+1}^{i+1} f_{i+1}) \times {}^ir_{i,ci} - {}^if_i \times ({}^ir_{i-1,i} + {}^ir_{c,i}) + I_i \cdot {}^i\dot{\omega}_i + {}^i\omega_i \times (I_i \cdot {}^i\omega_i) \tag{63}$$

7.2.4 Generalized forces

The generalized forces are given by:

$$ u_i = \begin{cases} {}^if_i \cdot^i * z_{i-1} & \text{if the joint is prismatic} \\ {}^i\tau_i^T \cdot^i * z_{i-1} & \text{if the joint is revolute} \end{cases} \tag{64}$$

7.3 Output examples

Using the matlab routine $Ne_\alpha(arg1, arg2, arg3)$ we can obtain the following outputs:

- **complete inverse dynamics**: $Ne_{0_g}(q, dq, ddq) = M(q) \cdot ddq + C(q, dq) + g(q)$
- **gravity term**: $Ne_{0_g}(q, 0, 0) = g(q)$
- **coriolis term**: $Ne_0(q, dq, 0) = C(q, dq)$
- **i-th column of the inertia matrix**: $Ne_0(q, 0, e_i) = M_i(q)$ where e_i is the i-th column of the identity matrix
- **generalized momentum**: $Ne_0(q, 0, \dot{q}) = M(q) \cdot \dot{q}$

Note: for the command $Ne_0(q, 0, 0)$, if a force F_e is applied to the end effector, the output is: $Ne_0(q, 0, 0) = J^T(q)F_e$ where J is the Jacobian matrix of the robot.

8 Basic control

There are two main types of control schemes:

- **open loop control**: the control input is computed without any feedback from the system
- **closed loop control**: the control input is computed using feedback from the system

Types of control systems base on uncertainty

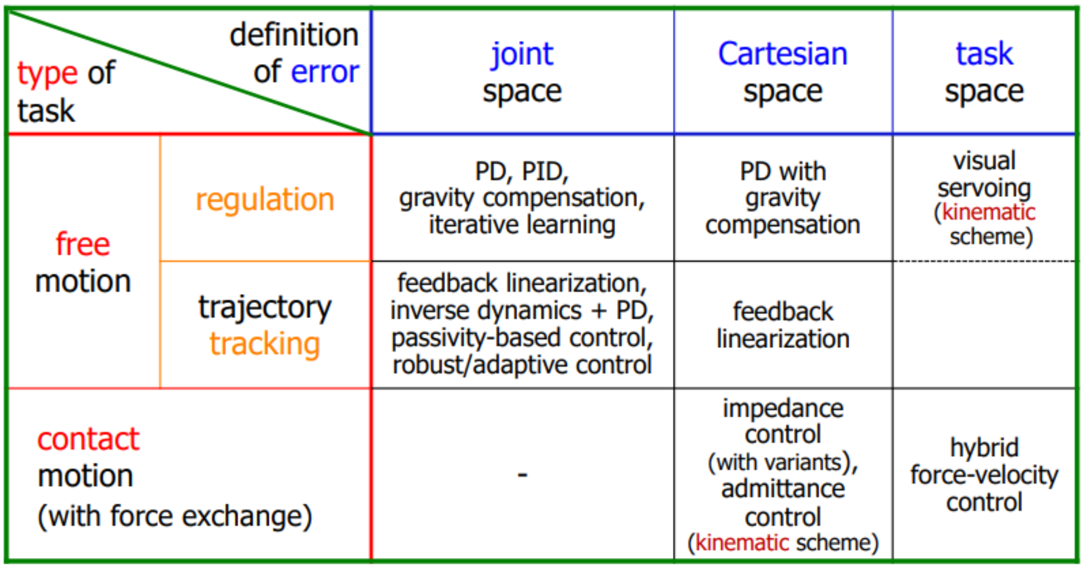
- **feedback control**: small variations of parameters and initial conditions
- **robust control**: large variations of parameters and initial conditions
- **adaptive control**: mproves performance online, adapting the control law to unknown range of uncertainties
- **intelligent control**: control based on experience

The different types of control are based on the type of error and task can be observed in figure 1.

8.1 Control notation

we use a new notation for position and velocity that are now expressed as:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_2 \end{bmatrix} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} \tag{65}$$



type of task / definition of error		joint space	Cartesian space	task space
free motion	regulation	PD, PID, gravity compensation, iterative learning	PD with gravity compensation	visual servoing (kinematic scheme)
	trajectory tracking	feedback linearization, inverse dynamics + PD, passivity-based control, robust/adaptive control	feedback linearization	
contact motion (with force exchange)		-	impedance control (with variants), admittance control (kinematic scheme)	hybrid force-velocity control

Figure 1: Control system

8.2 Equilibrium

There are two types of equilibrium:

- **unforced equilibrium**: $u = 0$ zero velocity and no gravity forces on the robot
- **forced equilibrium**: $u = u(x)$ zero velocity and $u(x_e) = g(x_{e1})$

8.3 Stability of dynamical systems

Stability of x_e is defined as:

$$\forall \epsilon > 0, \exists \delta > 0 : ||x(t_0) - x_e|| < \delta \Rightarrow ||x(t) - x_e|| < \epsilon \tag{66}$$

8.3.1 Asymptotic stability

Asymptotic stability of x_e is defined as:

$$\exists \delta > 0 : ||x(t_0) - x_e|| < \delta \Rightarrow \lim_{t \to \infty} ||x(t) - x_e|| = 0 \tag{67}$$

Note: this definition is in the sense of Lyapunov

8.3.2 Exponential stability

Exponential stability of x_e is defined as:

$$\exists \delta, c, \lambda > 0 : ||x(t_0) - x_e|| < \delta \Rightarrow ||x(t) - x_e|| \leq ce^{-\lambda(t-t_0)} ||x(t_0) - x_e|| \tag{68}$$

- can estimate the time to approximately converge
- typically this is a local property

Note: a necessary condition for x_e to be both globally asymptotically stable and exponentially stable is that it is the only equilibrium point of the system.

8.4 Lyapunov stability

8.4.1 Lyapunov candidate

A Lyapunov candidate is a function $V(x)$ that is positive definite that can be used to prove the stability of a system.
Examples of Lyapunov candidates are:

- for a dynamic system: $V(x) = T(x) + U(x)$ + eventual additional terms

8.4.2 Lyapunov stability

Lyapunov stability of x_e is defined as: **Sufficient condition of stability**

$$\exists V \text{candidate} : \dot{V}(x) \leq 0 \text{along the trajectories of the system} \tag{69}$$

Sufficient condition of asymptotic stability

$$\exists V \text{candidate} : \dot{V}(x) < 0 \text{along the trajectories of the system} \tag{70}$$

Sufficient condition of instability

$$\exists V \text{candidate} : \dot{V}(x) > 0 \text{along the trajectories of the system} \tag{71}$$

8.5 LaSalle Theorem

La Salle's theorem states that if $\exists V$candidate such that: $\dot{V}(x) \leq 0$ along the trajectories of the system, then:
system trajectories are bounded and converge to the largest invariant set
$M \subseteq S = \{x \in \mathbb{R}^n | \dot{V}(x) = 0\}$
Corolloary: $M \equiv x_e \rightarrow$ asymptotic stability

8.6 Stability of dynamical systems

for general time-varying systems, the stability can be analyzed using the Lyapunov method.

8.6.1 Barbalat Lemma

Barbalat's lemma states that if

- a function V(x,t) is lower bounded
- $\dot{V}(x,t) \leq 0$

then $\lim_{t\rightarrow\infty} \dot{V}(x,t) = 0$.
Corollary: if a Lyapunov candidate V(x,t) satisfies Barbalat's lemma, then the conclusion of LaSalle's theorem holds (ie. the system is asymptotically stable)

8.6.2 ex: verify the asymptotic stability of a closed loop system

We have two possible ways to verify the asymptotic stability of a closed loop system:

- **Lyapunov + LaSalle**: we have to find a Lyapunov candidate V(x) 8.4.1 and prove that $\dot{V}(x) < 0$ and then apply LaSalle's theorem 8.5
- **Taylor expansion**:analyzes the linearized version of the system dynamics, obtained by a first-order Taylor expansion around the desired closed-loop equilibrium point

Lyapunov + LaSalle:
The steps are the following:

1. write (if not given) the dynamic model of the system
2. obtain the closed loop system equation (by equating the right side of the dynamic model to the control law)
3. find a Lyapunov candidate V(x) 8.4.1
4. compute $\dot{V}(x)$
5. prove that $\dot{V}(x) < 0$
6. prove that the equilibrium is the only one
7. apply LaSalle's theorem
8. if the system is not asymptotically stable, find the largest invariant set

8.6.3 ex: verify that a state is the unique globally asymptotically stable equilibrium

The steps are the following:

1. Find a Lyapunov candidate V(x)
2. Compute $\dot{V}(x)$
3. Prove that $\dot{V}(x) < 0$
4. Prove that the equilibrium is the only one
5. Apply LaSalle's theorem

9 PD control

The **PD control** is a control scheme that uses the proportional and derivative terms of the error to compute the control input.
The goal is asymptotic stability of the closed-loop system:
$q = q_d$ and $\dot{q} = 0$
the **Control law** is given by:

$$u = K_p(q_d - q) + K_d(-\dot{q}) \tag{72}$$

where K_p and K_d are the proportional and derivative gains.

9.0.1 Th.1

In the absence of gravity tha robot state $(q_d, 0)$ under the given PD joint control law is globally asymptotically stable.

9.1 PD control with gravity compensation

$g(q)$ in a robot is the gravity term, and is bounded as follows:

$$||\frac{\partial g}{\partial q}|| \leq \alpha \tag{73}$$

Note: gravity term is bounded only if:

- the robot has all revolute joints
- the robot has both types of joints but no prismatic variable in g(q)
- all prismatic joints have limited ranges

consequently:

$$||g(q) - g(q_d)|| \leq \alpha ||q - q_d|| \tag{74}$$

the **Control law** is given by:

$$u = K_p(q_d - q) - K_d(\dot{q}) + g(q_d) \tag{75}$$

9.1.1 Th.2

If $K_{P,m} > \alpha$ the state $(q_d, 0)$ under the given PD joint control law with gravity compensation is globally asymptotically stable.

9.2 approximte gravity compensation

the approximate gravity compensation is given by:

$$u = K_p(q_d - q) - K_d(\dot{q}) + \hat{g}(q) \tag{76}$$

where $\hat{g}(q)$ is an approximation of the gravity term. It leeds to a closed loop equilibriuum whose uniqueness is not guaranteed.

10 PID control

The **PID control** is a control scheme that uses the proportional, integral and derivative terms of the error to compute the control input (adds the integral term to the PD control to eliminate the steady-state error).
PID can be used to recover a positio error due to absent or incomplete gravity compensation.
It is independent from the robot dynamics
The **Control law** is given by:

$$u = K_p(q_d - q) - K_d(\dot{q}) + K_i \int_0^t (q_d - q_\tau) d\tau \tag{77}$$

10.1 Saturated PID control

global asymptotic stability is not guaranteed for PID control, but can be proven under lower bound assumption on the gains for a non-linear PID law:

$$u = K_p(q_d - q) - K_d(\dot{q}) + K_i \int_0^t \phi(q_d - q_\tau) d\tau \tag{78}$$

where ϕ is a saturation function.
Examples of saturation functions are:

- $\phi(x) = \begin{cases} \sin x & \text{if } |x| \leq \pi/2 \\ 1 & \text{if } x > \pi/2 \\ -1 & \text{if } x < -\pi/2 \end{cases}$
- $\phi(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

11 Iterative Learning for gravity compensation

We can use the iterative method when:

- the robot has a complex (or unknown) dynamic model
- we don't need high position gain
- we don't want complex conditions on the gains

It can be used to derive sufficient conditions for global convergence with zero final error.
The **Control law** at the i-th iteration is given by:

$$u_i = \gamma K_p(q_d - q) - K_d(\dot{q}) + u_{i-1} \tag{79}$$

where γ is the learning gain, u_{i-1} is the feedforward term, $u_0 = 0$ is the easiest initialization of the feedforward term.
At steady state we have:

$$g(q_i) = \gamma K_p(q_d - q_i) + u_{i-1} = U_i \tag{80}$$

11.0.1 Th.3

If:

- $\lambda_{min}(K_p) > \alpha$
- $\gamma \geq 2$

then we are guaranteed that the sequence $q_0, .., q_i$ converges to q_d (and $\dot{q} = 0$) for any initial value (global convergence).
Note: the th.3 just gives a sufficient condition, not a necessary one

12 Extra exercises

12.0.1 dynamic scaling of trajectories

we want to impose that the same path is executed in a different time:

$$q_d(t) = q_s(r(t)) \tag{81}$$

an *uniform* scaling occurs when r(t)=kt so:

$$\dot{q}_d(t) = kq'_s(kt) \tag{82}$$

$$\ddot{q}_d(t) = k^2 q''_s(kt) \tag{83}$$

the new torque can be computed as:

$$\tau_s(kt) = M(q_s)q''_s + c(q_s, q'_s) + g(q_s) = \frac{1}{k^2}(\tau_d(t) - g(q_d(t))) + g(q_d(t)) \tag{84}$$

12.0.2 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.

12.0.3 ex: check if two tasks are in conflict

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. Build the extended desired velocity as: $\dot{r}_e = \begin{bmatrix} \dot{r}_1 \\ \dot{r}_2 \end{bmatrix}$
4. if the extended velcity is not in the range of the extended jacobian, the tasks are in conflict

12.0.4 ex: find the minimum time of execution of a trajectory for a single link

The steps are the following:

1. Write down the trajectory with it's first and second derivative
2. Write down the dynamic model of the single link and substitute the values you just computed
3. Split the dynamic model so to obtain u_a for acceleration terms and u_g for gravity terms
4. Find the time in which both component have maximum value
5. The acceleration component dominates the gravity one, so if there is a time (range) in which the acceleration component and gravity component reach their maximum value simultaneusly we can compute the minimum ytime just by computing the time in which the acceleration component reaches its maximum value

12.0.5 ex: find the dynaic model of a system of masses spring and dampers

There are two ways to solve this problem:

- **Lagrangian method**: write down the kinetic and potential energy of the system and write the dynamic model from there
- **Newton method**: balance the forces and write the dynamic model

Newton method:
The steps are the following:

1. Write an equation for each body of the system
2. Include in the equation the following forces:
3. - the force of the spring $k_i(x_i - x_{i-1})$

- the force of the damper $d_i(\dot{x}_i - \dot{x}_{i-1})$
- the force of the gravity $m_i g$
- the force of the acceleration $m_i \ddot{x}_i$
- the force of the external input u_i
- the force of the friction $f_i \dot{x}_i$
- the force of the contact c_i

4. obtain something like:

$$m_i \ddot{x}_i + k_i(x_i - x_{i-1}) + d_i(\dot{x}_i - \dot{x}_{i-1}) + m_i g + f_i \dot{x}_i + c_i = u_i \tag{85}$$

13 APPENDIX

13.1 Pseudo-inverse properties

The properties of the pseudo-inverse are:

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

13.2 Weighted pseudo-inverse properties

The properties of the weighted pseudo-inverse are:

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

13.3 DH frames

13.3.1 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

13.3.2 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}

13.4 Energy

13.4.1 Kinetic energy

The kinetic energy of a spring is given by:

$$T = \frac{1}{2}kx^2 \tag{86}$$

13.4.2 Potential energy

The potential energy of a spring is given by:

$$U = \frac{1}{2}kx^2 \tag{87}$$

14 GLOSSARY

- **self-motion**: motion of a robot that does not change the position of the end-effector

- **nominal conditiom**: the nominal condition is the condition in which the robot is supposed to be in.