Robotics 2

Artificial Intelligence and Robotics

Martina Doku

September 2, 2024

Contents

1 Calibration

Calibration goal: recover as much as possible pose errors by correcting the nominal 23 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 22.1.
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
$$ {}^iv_i = {}^{i-1}R_i^T(q_i) \cdot [{}^{i-1}v_{i-1} + \sigma_i \dot{q}_i {}^{i-1}z_{i-1} + {}^{i-1}\omega_i \times {}^{i-1}r_{i-1,i}] \tag{39}$$
$$ = {}^{i-1}R_i^T(q_i) \cdot [{}^{i-1}v_{i-1} + \sigma_i \dot{q}_i {}^{i-1}z_{i-1}] + {}^i\omega_i \times {}^i r_{i-1,i} \tag{40}$$
5. compute ${}^iv_{c,i} = {}^iv_i + {}^i\omega_i \times {}^i r_{c,i}$
6. compute $T_i = \frac{1}{2} m_i {}^iv_{c,i}^T {}^iv_{c,i} + \frac{1}{2} \cdot {}^i\omega_i^T {}^iI_{c,i} {}^i\omega_i$

where ${}^ir_{c,i}$ is the distance of the COM from frame i, ${}^ir_{i-1,i}$ 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

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

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) ${}^0r_{ci} = {}^0A_i^i r_{ci}$
- 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:

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

$$: U_i = -m_i g^T_0 r_i - g^T_0 \cdot^0 R_i m^i_i r_{c,i} \tag{51}$$

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:

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

7.2.3 Backward recursion

- Compute ${}^if_i$ as:

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

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

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

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:

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

- 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 ex: write the Lyapunov candidate given a dynamic model and a control law

The steps are the following:

1. given the dynamic models of the system, we want to, first substitute the control law in them if possible
2. then we create the Lyapunov candidate by summing the kinetic and potential energy of the system (from all the systems dynamic equation), plus a virtual potential energy K_P introduced by the control if any
3. then we compute the derivative of V as:
   if for example V(x) is:

$$V = 1/2 * \dot{q} M(q) \dot{q} + 1/2 * \dot{theta} + 1/2 * (q-\theta)^T K (q-\theta) + 1/2 (\theta_d - \theta)^T K_P (\theta_d - \theta) \quad (69)$$

   then $\dot{v}(x)$ is:

$$\dot{v} = \dot{q}^T M(q) \ddot{M} + 1/2 * \dot{q}^T \dot{M}(q) \dot{q} + \dot{\theta}^T B_m \ddot{theta} + (\dot{q} - \dot{\theta})^T K (q-\theta) + 1/2 * (\theta_d - theta)^T K_P (\theta_d - theta) \quad (70)$$

8.4.3 Lyapunov stability

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

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

Sufficient condition of asymptotic stability

$$\exists V \text{candidate} : \dot{V}(x) < 0 \text{along the trajectories of the system} \quad (72)$$

Sufficient condition of instability

$$\exists V \text{candidate} : \dot{V}(x) > 0 \text{along the trajectories of the system} \quad (73)$$

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 $\lim_{t\to\infty} \dot{V}(x,t) = 0$.
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 unique asymptotic stability of a closed loop system in a given equilibrium

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
- **Linearization**: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
   - for system of masses: 21.0.5
   - for a robot: **??**
2. write the control law (if not given). (Note: if you have multiple dynamic equations there might be some conditions on the control law, depending on the relation between the dynamic equations)
3. obtain the closed loop system equations (by equating the right side of the dynamic model(s) to the control law)
4. find a Lyapunov candidate V(x) 8.4.1. Note: one single Lyapunov candidate independently of the number of dynamic equations
5. verify that $V(x)$ is positive for all states and for the desired state it is equal to zero (when $x = x_d$)
6. compute $\dot{V}(x)$
7. prove that $\dot{V}(x) < 0$

8. prove that the equilibrium is the only one by applying LaSalle's theorem 8.5 (check that $\dot{V}(x) = 0$ only at the equilibrium)

9. if the system is not asymptotically stable, find the largest invariant set

Linearization:
The steps are the following:

1. write (if not given) the dynamic model of the system
   - for system of masses: 21.0.5
   - for a robot: 5.2.1
2. bring to Laplace domain the dynamic model
3. prove asymptotic stability of the linearized system with routh criterion

8.6.3 ex: Find the unique equilibrium state for the closed-loop system under a given control

The steps are the following:

1. write (if not given) the dynamic model of the system
   - for system of masses: 21.0.5
   - for a robot: 5.2.1
2. write the control law (if not given). (Note: if you have multiple dynamic equations there might be some conditions on the control law, depending on the relation between the dynamic equations)
3. obtain the closed loop system equations (by equating the right side of the dynamic model(s) to the control law)
4. substitute in the closed loop system equations the values of the equlibrium state (the known one, hence $\dot{q} = 0$ and $\ddot{q} = 0$)
5. solve the system of equations to find the equilibrium state q_d
6. show global asymptotic stability as in **??**

from step 4

9 Which control law?

If asked to design a control law without direct request of a specific method, you can use the following criteria to choose the correct one:

- to achieve **global exponential stabilization** of a state with **decoupled transient evolutions of the position error** $\rightarrow$ y feedback linearization control in the joint space

- to achieve **global asymptotic stabilization** of a state , **without knowledge of the robot inertia matrix** → a PD control with gravity cancellation or iterative gravity cancellation
- to achieve **exponential stabilization of the end-effector position** $p = p_d$ up to kinematic singularities → feedback linearization control in the Cartesian space

10 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:

$$u = K_p(q_d - q) + K_d(-\dot{q}) \tag{74}$$

where K_p and K_d are the proportional and derivative gains.

10.0.1 Th.1

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

10.1 PD control with gravity compensation

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

$$||\frac{\partial g}{\partial q}|| \leq \alpha \tag{75}$$

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:

$$||g(q) - g(q_d)|| \leq \alpha||q - q_d|| \tag{76}$$

the **Control law** is given by:

$$u = K_p(q_d - q) - K_d(\dot{q}) + g(q_d) \tag{77}$$

10.1.1 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.

10.2 approximte gravity compensation

the approximate gravity compensation is given by:

$$u = K_p(q_d - q) - K_d(\dot{q}) + \hat{g}(q) \tag{78}$$

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

10.2.1 ex: find the minimum values of the gains for a PD control with gravity compensation

The steps are the following:

1. find the gravity term g(q) for the robot
2. find the bound α as in 5.6.2
3. impose $K_{P,m} > \alpha$
4. if we have viscous friction, we can impose $K_{D,m} = 0$

Note: the th.2 just gives a sufficient condition, not a necessary one, we can observe the structure of the gravity term g(q) and for each entry i, we can say that it is sufficient that $K_{P,i} > 0$ if the i-th entry of the gravity term is:

- constant
- equal to zero

11 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:

$$u = K_p(q_d - q) - K_d(\dot{q}) + K_i \int_0^t (q_d - q_\tau)d\tau \tag{79}$$

11.1 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:

$$u = K_p(q_d - q) - K_d(\dot{q}) + K_i \int_0^t \phi(q_d - q_\tau)d\tau \tag{80}$$

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

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

12 Iterative Learning for gravity compensation

We can use the iterative methood 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:

$$u_i = \gamma K_p(q_d - q) - K_d(\dot{q}) + u_{i-1} \tag{81}$$

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:

$$g(q_i) = \gamma K_p(q_d - q_i) + u_{i-1} = U_i \tag{82}$$

12.0.1 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

13 Trajectory tracking control

If there are distrubances we need to use a feedback to make the control scheme more robust.
Possible implementations:

- **Offline**: open loop
- **Online**: closed loop

There are 2 steps in control deign:

1. comcompensation (feedforward) or cancellation (feedback) of nonlinearities
2. synthesis of a linear control law stabilizing the trajectory error to zero

13.1 Trajectory controllers

Possible trajectory controllers are:

- inverse dynamic compensation (FFW) + PD control

$$u = \hat{u}_d + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) \tag{83}$$

- inverse dynamic compensation (FFW) +variable PD

$$u = \hat{u}_d + \hat{M}(q_d)[K_P(q_d - q) + K_D(\dot{q}_d - \dot{q})] \tag{84}$$

- feedback linearization + PD + FFW

$$u = \hat{M}(q)[\ddot{q} + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q})] + \hat{n}(q, \dot{q}) \tag{85}$$

- feedback linearization + PID + FFW

$$u = \hat{M}(q)[\ddot{q} + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) + K_I \int_0^t (q_d - q_\tau)d\tau] + \hat{n}(q, \dot{q}) \tag{86}$$

Note:the global stabilization for the last two laws is obtained with $K_P > 0$ and $K_D > 0$

13.1.1 Feedback linearization

In nominal conditions ($\hat{M} = M$ and $\hat{n} = n$) we have:

$$u = M(q)\ddot{q} + n(q, \dot{q}) = M(q)a + n(q, \dot{q}) \tag{87}$$

So the global asymptotic stabilization of tracking error is:

$$a = \ddot{q}_d + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) \tag{88}$$

under feedback linearization control, the robot has a dynamic behavior that is invariant, linear and decoupled in its whole state space ($\forall(q, \dot{q})$).
linearity: error tends to 0 exponentially.
decoupling: each joint coordinate evolves independently from the others.

13.1.2 remarks

- desired robot trajectory can be generated from Cartesian data $p_d(0), \dot{p}_d(0), \ddot{p}_d(0)$
- real-time computation with Newton-Euler algorithm

13.1.3 ex: achieve global exponential stabilization with decoupled transient evolution of positions errors e(q,t)

The steps are the following:

- write the feedback linearization control law as:
$$u = M[-K_D\dot{q} + K_P(q_d - q)] + c(q, \dot{q}) + g(q) \tag{89}$$
- obtain the desired error transient by choosing suitable gains K_D and K_P that satisfy the following equation:
$$\ddot{e}_i + K_{D,i}\dot{e}_i + K_{P,i}e_i = 0 \tag{90}$$
- obtain $\dot{e}$ and $\ddot{e}$ from the given e equation by differentiation and substitute them in the previous formula.
- solve by principle of polynomial identity for K_Ds and K_Ps

13.2 Alternaive global trajectory controller

$$u = M(q)\ddot{q} + S(q, \dot{q})\dot{q} + g(q) + F_V\dot{q}_d + K_P e + K_D\dot{e} \tag{91}$$

gives a global asymptotic stabilization of the tracking error, no cancellation of nonlinearities and no linear and decoupled behaviour.

13.3 Regulation special case

The regulation is a special case of trajectory tracking where q_d is constant.
the feedback linearization control law is given by:

$$u = M(q)[K_P(q_d - q) - K_D(\dot{q})] + c(q, \dot{q}) + g(q) \tag{92}$$

this is a solution to the regulation problem with exponential stability.
the alternative global trajectory controller is given by:

$$u = K_P(q_d - q) - K_D(\dot{q}) + g(q) \tag{93}$$

(it's a simple PD control with gravity compensation)

13.4 Repetitive motion tasks

The repetitive motion tasks are tasks that are executed repeatedly in a finite time.
The robot is reinitialized at the initial state at the beginning of each trial.
the control law is made of a non-model based part (often, a decentralized PD law) + a time-varying feedforward which is updated before every trial
the update of the feedforward term is done by:

$$v_{k+1}(s) = \alpha(s)u'_k(s) + \beta(s)v_k(s) \tag{94}$$

if a contraction condition can be enforced

$$||\beta(s) - \alpha(s)W(s)|| < 1 \tag{95}$$

then the control law is globally asymptotically stable.
Note: if the choice $\beta = 1$ satisfies the contraction condition, then we coverge with 0 tracking error. If $\alpha(s) = 1/W(s)$ then we converge in 1 iteration.

14 Application to robots

robot equation:

$$u = [B_m + M(q)]\ddot{q} + [F_v + S(q, \dot{q})]\dot{q} + g(q) \tag{96}$$

control law (pre-learning):

$$u = K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) + \hat{g}(q) \tag{97}$$

learning filters:

$$W_i = \frac{q_i(s)}{q_{di}(s)} = \frac{K_{Pi} + K_{Di}s}{\hat{B}_{mi}s^2 + (\hat{F}_{vi} + K_{Di})s + K_{Pi}} \tag{98}$$

initializaton of feedforward term (using best estimate):

$$v_1 = [\hat{B}_m + \hat{M}(q)]\ddot{q}_d + [\hat{F}_v + \hat{S}(q, \dot{q})]\dot{q}_d + \hat{g}(q) \tag{99}$$

14.0.1 Control with approximate feedback linearization

dynamic model, its nominal part and (unstructured) uncertainty:

$$M(q)\ddot{q} + n(q, \dot{q}) = \tau \tag{100}$$

$$M = \hat{M} + \Delta M \tag{101}$$

$$n = \hat{n} + \Delta n \tag{102}$$

model based feedback linearization:

$$\tau_{fl} = \hat{M}(q)a + \hat{n}(q, \dot{q}) \tag{103}$$

resulting closed-loop dynamics with perturbation:

$$\ddot{q} = a + \delta(q, \dot{q}, a) \tag{104}$$

$$\delta = (M^{-1}\hat{M} - I)a + M^{-1}(\hat{n} - n) \tag{105}$$

control law for tracking $q_d(t)$ (PD with feed forward + regressor ϵ_k):

$$a = u_k + \epsilon_k = K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) + \ddot{q}_d + \epsilon_k \tag{106}$$

the regressor term is obtained with a Gaussian Process (GP) model:

$$\epsilon_k \circ \mathcal{N}(\mu(\hat{X}_k, \sigma^2(\hat{X}_k))) \tag{107}$$

14.0.2 ex: solve the inverse dynamics for a desired trajectory $q_d(t)$

The steps are the following:

1. write the dynamic model of the robot
2. substitute the desired trajectory in the dynamic model
3. solve the equation for tau

Note: if you have more than one dynamic equation, you have to substitue the desired trajectory in all the equations, obtain the values of the other parameter (say θ) and its derivatives in terms of the desired trajectory and then substitute them in the equation for τ.

15 Adaptive control

We need adaptation in robot motion when we have:

- poor knowledge of inertial payload
- large uncertainties in robot dynamics parameters (m_i, I_i, r_{ci})

we assume that DH parameters are known (including link lengths).
main methodologies:

- **linear parametrization** of robot dynamics
- **nonlinear control law** of dynamic type (nonlinear control based on feedback linearization)

first adaptive method: on-line modification with reference velocity.
$\dot{q}_r = \dot{q}_d + \Lambda(q_d - q)$ where typically $\Lambda = K_D^{-1} K_P$.
Adaptive control law:

$$u = M(q)\ddot{q} + S(q,\dot{q})\dot{q} + g(q) + F_V \dot{q}_r + K_P(q_d - q) + K_D(\dot{q}_r - \dot{q}) \tag{108}$$

15.1 Th.4

The introduced adaptive controller makes the tracking error along the desired trajectory globally asymptotically stable

15.1.1 properties

If the desired trajectory $q_d(t)$ is persistently exciting, then the estimates of the dynamic parameters converge to the true values.
condition of persistent excitation:

- for linear systems: # of frequency components in the desired trajectory should be at least twice as large as # of unknown coefficients
- for nonlinear systems: the condition can be checked only a posteriori (a squared motion integral should always be positive bounded from below)

15.2 Adaptive regulation

adaptation in case q_d is constant.

$$u = \hat{M}(q)\ddot{q}_r + \hat{S}(q,\dot{q})\dot{q}_r + \hat{g}(q) + \hat{F}_V \dot{q}_r + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) \tag{109}$$

and adaptation law:

$$\dot{\hat{a}} = -\Gamma Y^T(q,\dot{q},\dot{q}_r,\ddot{q})(\dot{q}_r - \dot{q}) \tag{110}$$

15.2.1 ex: find the adaptive control law that obtains global asymptotic tracking of a trajectory $q_d(t)$

The steps are the following:

1. write the dynamic model of the robot and obtain the linear parametrization Y
2. write the control law and adaptation law as in 15.2 (removing eventually not-necessary terms)
3. write the formula for q_r as $\dot{q}_r = \dot{q}_d + \Lambda(q_d - q)$ where $\Lambda = K_D^{-1} K_P$

16 Cartesian control

We use the Cartesian control when the goal is the asymptotic stabilization of the end-effector position:
$p = p_d, \dot{q} = \dot{q}_d = 0 \leftrightarrow \dot{p} = \dot{p}_d = 0$
Note:
if $m = n$, then $\dot{q} \implies \dot{p} = 0$
if $m < n$ then the goal is not uniquely associated to a complete robot state.
The **Cartesian regulation control law** is given by:

$$u = J^T(q) K_P (p_d - p) - K_D \dot{q} + g(q) \tag{111}$$

16.1 Th.5

under the control law in 111 the robot state will converge asymptotically to the set $A = \{\dot{q} = 0, g : K_p(p_d - f(q)) \in N(J^T(q))\}$

16.2 corollary

for a given initial state $(q(0), \dot{q}(0))$, if the robot does not encounter any singularity of $J^T(q)$during its motion, then there is asymptotic stabilization to one single state or to a set of states such that $e_p = 0, \dot{q} = 0$

16.3 Cartesian control variant

Control law (PD control plus gravity cancellation in joint space):

$$u = J^T(q)[K_P(p_d - p) - K_D \dot{p}] + g(q) \tag{112}$$

16.4 Feedback linearization in Cartesian space

algorithm: differentiate the output(s) as many times as needed up to the appearance of (at least one of) the input torque(s), then verify if it is possible to solve for the input.

- $y = f(q)$
- $\dot{y} = J(q)\dot{q}$
- $\ddot{y} = J(q)\ddot{q} + \dot{J}(q)\dot{q} = J(q)M(q)^{-1}(u - c(q, \dot{q}) - g(q)) + \dot{J}(q)\dot{q}$

16.4.1 Th.6

Theorem for a non-redundant robot, it is possible to exactly linearize and decouple the dynamic behavior at the Cartesian level if and only if $\det(J(q)) \neq 0$

16.4.2 linearization in right coordinates

from the control law we can derive:

$$\ddot{y} = J(q)M^{-1}(u - c(q, \dot{q}) - g(q)) + \dot{J}(q)\dot{q} = a \tag{113}$$

from the closed-loop equation in joint space we obtain:

$$\ddot{q} = \ddot{p} = J^{-}1(q)a - J^{-}1(q)\dot{J}(q)\dot{q} \tag{114}$$

Note: the design of a Cartesian trajectory tracking control is completed by setting:

$$a_i = \ddot{p}_{di} + K_{Di}(0 - \dot{p}_i) + K_{Pi}(p_{di} - p_i) \tag{115}$$

Note: for p_d constant, the control law is:

$$u = M(q)J^{-}1(q)[K_P e_P - K_D J(q)\dot{q})] + c(q, \dot{q}) + g(q) - M(q)J^{-}1(q)\dot{J}(q)\dot{q} \tag{116}$$

16.4.3 ex: find a robot configuration q and consequent p (that is not p_d) such that the robot will not move when at rest in q

The steps are the following:

1. write the Cartesian control law as in 111
2. find the singularities of the Jacobian
3. substitute the values of the singularities in the forward kinematics to find the corresponding end effector position
4. substitute the value of q, $\dot{q} = 0$ in the control law and obtain its reduced form
5. equate the reduced form to the model dynamics (where you substitute q and $\dot{q}$ and prove that $\ddot{p} = 0$

17 Robot interacion with the environment

17.1 Robot compliance

Robot compliance can be:

- **passive**: the robot is compliant by structure
- **active**: the robot is able to control the compliance with
  - admittance control

- impedance control
- stiffness/compliance control
- hybrid control

in all cases for physical interaction tasks, the desired motion specification and execution should be integrated with complementary data for the desired force.

17.2 Ideally constrained contact

If geometry is known hybrid force/velocity control is the best choice. In the ideal case, with a infinitely stiff environmente, we can assume $f_e = -f_r$ where f_e is the force exerted by the environment and f_r is the force exerted by the robot (normally wrt env)

17.3 Constrained robot dynamics

suppose that the task variables are subject to m ¡ n (bilateral) geometric constraints in the general k(r)=0 and define

$$h(q) = k(f(q)) = 0 \tag{117}$$

the constrained robot dynamics can be derived using again the Lagrange formalism, by defining an augmented Lagrangian as:

$$L(q, \dot{q}, \lambda) = T(q, \dot{q}) - U(q) - \lambda^T h(q) \tag{118}$$

where the Lagrange multipliers λ (a m-dimensional vector) can be interpreted as the generalized forces that arise at the contact when attempting to violate the constraints.
n applying the Euler-Lagrange equations we obtain the constrained robot dynamics as:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = u + A^T(q)\lambda \tag{119}$$

where A(q) is the Jacobian of the constraints h(q) $A(q) = \frac{\partial h}{\partial q}$
We can obtain the value of λ by doing the following:

- differentiate the constraint $h(q) = 0$ with respect to time (twice), obtain $\ddot{h} = A(q)\ddot{q} + \dot{A}(q)\dot{q} = 0$
- substitute the value of $\ddot{q}$ from the constrained robot dynamics
- obtain the value of λ by solving the equation

$$\lambda = (AM^{-1}A^T)^{-1}(AM^{-1}(C\dot{q} + g - u) - \dot{A}\dot{q}) \tag{120}$$

The final constrained dynamic model can be rewritten as:

$$M(q)\ddot{q} = [I - A^T(q)(A_M^{\#}(q))^T](u - c(q, \dot{q}) - g) - M(q)A_M^{\#}(q)\dot{A}(q)\dot{q} \tag{121}$$

where $A_M^{\#}(q)$ is the Moore-Penrose pseudo-inverse of A(q) $A_M^{\#}(q) = M(q)^{-1}A^T(q)(A(q)M^{-1}(q)A^T($

17.4 Reduced robot dynamics

by imposing m constraints $h(q) = 0$ on the n generalized coordinates q, it is also possible to reduce the description of the constrained robot dynamics to a n-m dimensional configuration space.
Start from a constarint matrix A(q) and select D(q) such that

$$\begin{bmatrix} A(q) \\ D(q) \end{bmatrix} \tag{122}$$

in a nonsingular matrix. And :

$$\begin{bmatrix} A(q) \\ D(q) \end{bmatrix}^{-1} = \begin{bmatrix} E(q) & F(q) \end{bmatrix} \tag{123}$$

then we define the vector of pseudovelocities v as:

$$v = D(q)\dot{q} \tag{124}$$

from that we can obtain $\dot{q}$ and $\ddot{q}$ as:

$$\dot{q} = F(q)v \tag{125}$$

$$\ddot{q} = F(q)\dot{v} - (E(q)\dot{A}(q) + F(q)\dot{D}(q))F(q)v \tag{126}$$

from that we can rewrite the constrained robot dynamics as:

$$(F^T M F)\dot{v} = F^T(u - c(q,\dot{q}) - g + M(E\dot{A} + F\dot{D})Fv) \tag{127}$$

and the lagrangian multipliers become:

$$\lambda = E^T(MF\dot{v} - M(E\dot{A} + F\dot{D})Fv + c + g - u) \tag{128}$$

18 Impedance control

The impedance control imposes a desired dynamic behavior to the interaction between robot end-effector and environment specified through a complete set of mass-spring-damper equations.
The dynamic model of a robot in ontact is given by:

$$M(q)\ddot{q} + c(q,\dot{q})\dot{q} + g(q) = u + J_r^T(q)F_r \tag{129}$$

where $J_r(q)$ is the analyitic jacobian (from forward kinematics) and $F_r = T_r^{-T}(\psi)F$.
Note: in case of constraint forces instead of contact forces we can rewrite the equation as:

$$M(q)\ddot{q} + c(q,\dot{q})\dot{q} + g(q) = u + A^T(q)\lambda \tag{130}$$

where $A(q)$ is the Jacobian of the constraints.

18.1 Dynamic model in Cartesian coordinates

The dynamic model in Cartesian coordinates is given by:

$$M_r(q)\ddot{q} + S_r(q,\dot{q})\dot{r} + g_r(q) = J_r^{-T}(q)u + F_r \tag{131}$$

where:

- $M_r(q) = J_r^{-T}(q)M(q)J_r^{-1}(q)$
- $S_r(q,\dot{q}) = J_r^{-T}(q)S(q,\dot{q})J_r^{-1}(q) - M_r(q)\dot{J}_r(q)J_r^{-1}(q)$
- $g_r(q) = J_r^{-T}(q)g(q)$

18.2 Design of the control law

The design happens in two steps:

1. feedback linearization in the Cartesian space (with force measure)

$$u = J_r^T(q)[M_r(q)a + S_r(q,\dot{q})\dot{r} + g_r(q) - F_r] \tag{132}$$

2. imposition of a dynamic impedance model

$$M_m(\ddot{r} - \ddot{r}_d) + D_m(\dot{r} - \dot{r}_d) + K_m(r - r_d) = F_r \tag{133}$$

that is realized by choosing:

$$a = \ddot{r}_d + M_m^{-1}[D_m(\dot{r}_d - \dot{r}) + K_m(r_d - r) + F_r] \tag{134}$$

18.2.1 Control law in joint coordinates

$$u = M(q)J_r^{-1}(q)\ddot{r}_d - \dot{J}_r(q)\dot{q} + M_m^{-1}[D_m(\dot{r}_d - r) + K_m(r_d - r)] + SS(q,\dot{q})\dot{q} + g(q) + J_r^T(q)[M_r(q)M_m^{-1} - I]F_r \tag{135}$$

Note: we can choose the parameter to have a desired dynamic:

- large M_m and small K_m when we want low contact forces
- small M_m and large K_m when we want good tracking of desired motion trajectory
- damping coefficients D_m are used then to shape transient behaviors

18.2.2 Simplifications

if we assume $M_m = M_r$ the control law becomes:

$$u = M(q)J_r^{-1}(q)\ddot{r}_d - \dot{J}_r(q)\dot{q} + S(q,\dot{q})\dot{q} + g(q) + J_r^T(q)[D_m(\dot{r}_d - \dot{r}) + K_m(r_d - r)] \tag{136}$$

this is a pure motion control law applied also during interaction, but designed so as to keep limited contact forces at the end-effector leave.
Note: if the impedance model (now, nonlinear) is still supposed to represent a real

mechanical system, then in correspondence to a desired non-constant inertia there should be Coriolis and centrifugal terms so the cotrol law becomes:

$$u = M(q)J_r^{-1}(q)\ddot{r}_d - \dot{J}_r(q)J_r^{-1}(q)\dot{r}_d + S(q,\dot{q})J_r^{-1}(q)\dot{r}_d + g(q) + J_r^T(q)[D_m(\dot{r}_d - \dot{r}) + K_m(r_d - r)] \tag{137}$$

which guarantees convergence to zero tracking error wher $F_r = 0$.
We have a further simplification if r_d is constant the control law becomes:

$$u = g(q) + J_r^T(q)[-D_m\dot{r}_d + K_m(r_d - r)] \tag{138}$$

18.3 General case $F_r \neq 0$

For analysis we model the environment as an elastic element:

$$F_r = K_e(r_e - r) \tag{139}$$

wher $\dot{r} = \ddot{r} = 0$ a closed loop system equilibrium is given by:

$$K_m(r_d - r) + K_e(r_e - r) \tag{140}$$

which has unique solution

$$r = (K_m + K_e)^{-1}K_m r_d + K_e r_e) = r_E \tag{141}$$

18.3.1 Equivalence with RCC device

TH:
if $(r_d - r) \simeq J_r(q)(q_d - q)$ and g(q)=0 and $D_m = 0$
then the control law is equivalent to the one of a RCC device:

$$u = J_r^T(q)K_m J_r(q)(q_d - q) = K(q)(q_d - q) \tag{142}$$

18.3.2 ex: design an impedance control law for a robot

The steps are the following:

1. write the dynamic model in Cartesian coordinates
2. write the impedance model
3. check if the appartent cartesian matrix M_m (the one of the impedance model) is equal to the natural inertia matrix M_r (this could be specified in the text or be a consequence of the absence of force/torque sensors)
4. if so, substitute f_r obtained from the impeance model inside the dynamic model
5. from the dynamic model, isolate and obtain u (you have obtained the control law)

18.3.3 ex: find the equilibrium point for impedance control

the steps are the following:

- write the impedance control law at steady-state ($\dot{r} = \ddot{r} = 0$)
- isolate and find r (equilibrium point)
- write the control law as in 18.3.2 and substitute the values of steady state ($\dot{r} = \ddot{r} = 0$)
- compute u with the r previously obtained: u_E

19 Hybrid control

The hybrid control law is designed in ideal conditions, but now unconstrained directions of motion and constrained force directions are defined in a more direct way using a task frame formalism.
all non-ideal conditions (compliant surfaces, friction at the contact, errors in contact surface orientation) are handled explicitly in the control scheme by a geometric filtering of the measured quantities.

19.0.1 ideal conditions

In ideal conditions we can define inside the task space:

- **end-effector motion**
- **reaction forces/torques**

those are the **natural constraints**.
The **artificial constraints** can be specified by:

- **end-effector velocities** round k directions
- **contact forces/torques** 6-k directions

19.1 Hybrid force/velocity control

The general parametrization of hybrid task can be described by:

1. robot-environment contact type:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = T(s)\dot{s} \tag{143}$$

$$\begin{bmatrix} f \\ \mu \end{bmatrix} = Y(s)\lambda \tag{144}$$

2. robot dynamics:

$$M(q)\ddot{q} + S(q,\dot{q}) + g(q) = u + J^T(q)[f,\mu]^T \tag{145}$$

3. robot kinematics:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J(q)\dot{q} \tag{146}$$

The **control objective** is to impose a desired task evolution to the parameters s of motion and λ of force.
The control law is designed in two steps:

1. extract linearization and decoupling in the task frame by feedback

$$u = (MJ^{-1}T, -J^TY)(a_s, a_\lambda)^T + MJ^{-1}(\dot{T}\dot{s} - \dot{J}\dot{q}) + S\dot{q} + g \tag{147}$$

2. (linear) design of a_s and a_λ so as to impose the desired dynamic behavior to the errors $e_s = s_d - s$ and $e_\lambda = \lambda_d - \lambda$

$$a_s = \ddot{s}_d + K_D(\dot{s}_d - \dot{s}) + K_P(s_d - s) \tag{148}$$

$$a_\lambda = \lambda_d + K_I \int (\lambda_d - \lambda)dt \tag{149}$$

where values for s, $\dot{s}$ are extracted from measures of q and $\dot{q}$ equating the descriptions of the end-effector pose and velocity "from the robot side" (direct and differential kinematics) $\dot{s} = T^{\#}J(q)\dot{q}$ and "from the environment side" while λ is obained from force/torque measures at end effector $\lambda = Y^{\#}(f, m)^T$

Note: we assume n=m and J(q) out of singularity

19.2 Force control via impedance model

in a force-controlled direction of the hybrid task space, when the contact stiffness is limited, one may use impedance model ideas to explicitly control the contact force.
The impedance model is chosen as:

$$m_m\ddot{x} + d_m\dot{x} + k_s(x - x_d) = f_d \tag{150}$$

after a feedback linearization the command a_x is designed as:

$$a_x = (1/m_m)[(f_d - f_s) - d_m\dot{x}] \tag{151}$$

19.2.1 sources of inconsistency

the main sources of inconsistency in force and velocity measurements can be:

- presence of friction at the contact
- compiance in the robust structure and/or at the contacy
- uncertainty on environment geometry at the contact

19.3 Estimation of unknown surface

- normal = nominal direction of measured force
- tangent = nominal direction of measured velocity
- tangent direction and friction angle can also be extimated with recursive least square method

20 Visual servoing

Visual servoing refers to the use information acquired by vision sensors (cameras) for feedback control of the pose/motion of a robot.
The first phase is the **image processing**. The **vision system** provides set-point references to a cartesian motion controller.
Possible visual servoing schemes:

- **image-based visual servoing**: the control law is designed directly in the image space. The steps are:
  - image acquisition
  - feature extraction
  - comparison with desired values (error)
  - generation of motion of camera/robot
- **position-based visual servoing**: the control law is designed in the robot task space (cartesian control law)

20.1 camera model

The camera model is given by:

- set of lenses (thin lenses, pinhole, catadioptic lens)
- matrix of light-sensitive elements
- frame grabber

20.1.1 thin lens camera model

fundamental equation of a thin lens:

$$\frac{1}{Z} + \frac{1}{z} = \frac{1}{\lambda} \tag{152}$$

where Z is the distance between the object and the lens, z is the distance between the image plane and the lens and λ is the focal length of the lens.

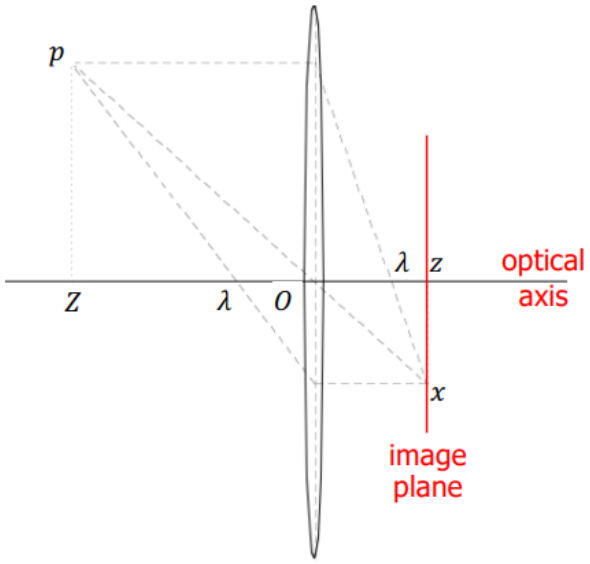

Figure 2: thin lens camera model

20.1.2 pinhole camera model

the pinhole camera model is given by:

$$u = \lambda \frac{X}{Z} \tag{153}$$

$$v = \lambda \frac{Y}{Z} \tag{154}$$

with P=(X,Y,Z) the point in the 3D space and (u,v,λ) the point in the image plane.

20.1.3 Interaction matrix

given a set of features parameters f =$[f_1, f_2, ..., f_n]$, we look for the (kinematic) differential relation between motion imposed to the camera and motion of features on the image plane

$$\dot{f} = J(\cdot)[V, \Omega]^T \tag{155}$$

where (V,Ω) is a six dimensional linear/angular velocity vector.
J($\cdot$) is the **interaction matrix**.
To compute the interaction matrix:

1. from the perspective equation we obtain:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} \frac{\lambda}{z} & 0 & -\frac{u}{Z} \\ 0 & \frac{\lambda}{Z} & -\frac{v}{Z} \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = J_1(u, v, \lambda) \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} \tag{156}$$

2. from kinematic realation instead we can obtain:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 & -Z & Y \\ 0 & -1 & 0 & Z & 0 & -X \\ 0 & 0 & -1 & -Y & X & 0 \end{bmatrix} \begin{bmatrix} V \\ \Omega \end{bmatrix} = J_2(X, Y, Z) \begin{bmatrix} V \\ \Omega \end{bmatrix} \tag{157}$$

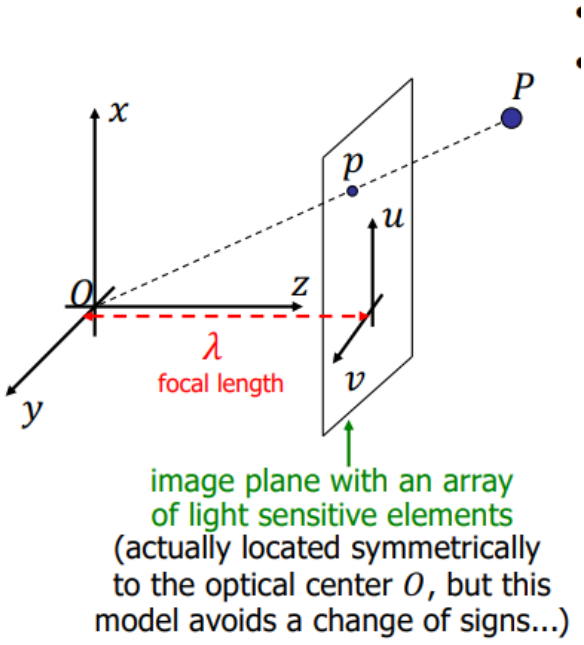

Figure 3: pinhole camera model

3. thus we obtain the interaction matrix J_p as

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = J_1 J_2 \begin{bmatrix} V \\ \Omega \end{bmatrix} = \begin{bmatrix} -\frac{\lambda}{Z} & 0 & \frac{u}{Z} & \frac{uv}{\lambda} & -(\lambda + \frac{u^2}{\lambda}) & v \\ 0 & -\frac{\lambda}{Z} & \frac{v}{Z} & \lambda + \frac{v^2}{\lambda} & -\frac{uv}{\lambda} & -u \end{bmatrix} \begin{bmatrix} V \\ \Omega \end{bmatrix} = J_p(u, v, Z) \begin{bmatrix} V \\ \Omega \end{bmatrix} \tag{158}$$

Note: when requested to obtain the interaction matrix coming from multiple points we have variables u_i, v_i and Z_i for each point i, λ is the same for all points.

20.1.4 Other interaction matrices

Other examples of interaction matrices are:

- distance between two points in the image plane

$$d = \frac{1}{d} \begin{bmatrix} u_1 - u_2 & v_1 - v_2 & u_2 - u_1 & v_2 - v_1 \end{bmatrix} \begin{bmatrix} \dot{u}_1 \\ \dot{v}_1 \\ \dot{u}_2 \\ \dot{v}_2 \end{bmatrix} = J_P(u_1, u_2, v_1, v_2) \begin{bmatrix} J_{p1}(u_1, v_1, Z_1) \\ J_{p2}(u_2, v_2, Z_2) \end{bmatrix} \begin{bmatrix} V \\ \Omega \end{bmatrix} \tag{159}$$

- image moments:

$$m_{ij} = \int \int x^i y^j dx dy \tag{160}$$

20.2 Robot Differential Kinematics

in the **eye-in-hand** case the motion to be imposed to the camera coincides with the desired end-effector linear/angular velocity.

$$\begin{bmatrix} V \\ \Omega \end{bmatrix} = J_m(q)\dot{q} = J_m(q)u \tag{161}$$

where $J_m(q)$ is the geometric jacobian or the NMM jacobiam of a manipulator.
We obtain:

$$\dot{f} = J_p(f, Z)J_m(q)u = J(f, Z, q)u \tag{162}$$

where J(f,Z,q) is the **Image jacobian**.
In general we can obtain the control as:

$$u = J^{\#}(\dot{f}_d + ke) + (I - J^{\#}J)u_0 \tag{163}$$

and this will exonentially stabilize the error to 0 (up to singularities).

20.2.1 Control With Tak Sequencing

approach: regulate only one (some) feature at the time, while keeping "fixed" the others by unobservable motions in the image plane.
The steps are the following:

1. select the feature to be regulated
2. compute the control law for the selected feature

$$u = J_1^{\#}k_1e_1 + (I - J_1^{\#}J_1)u_0 \tag{164}$$

3. compute the control law for the other features

$$u = (I - J_1^{\#}J_1)J_2^T k_2 e_2 \tag{165}$$

21 Extra exercises

21.0.1 dynamic scaling of trajectories

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

$$q_d(t) = q_s(r(t)) \tag{166}$$

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

$$\dot{q}_d(t) = kq_s'(kt) \tag{167}$$

$$\ddot{q}_d(t) = k^2q_s''(kt) \tag{168}$$

the new torque can be computed as:

$$\tau_s(kt) = M(q_s)q_s'' + c(q_s, q_s') + g(q_s) = \frac{1}{k^2}(\tau_d(t) - g(q_d(t))) + g(q_d(t)) \tag{169}$$

21.0.2 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.

21.0.3 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

21.0.4 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 (trajectories can be found in 22.4)
2. Write down the dynamic model of the single link and substitute the values you just computed 21.0.5
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

21.0.5 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:

$$m_i \ddot{x}_i + k_i(x_i - x_{i-1}) + d_i(\dot{x}_i - \dot{x}_{i-1}) + m_i g + f_i \dot{x}_i + c_i = u_i \tag{170}$$

22 APPENDIX

22.1 Pseudo-inverse properties

The properties of the pseudo-inverse are:

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

22.2 Weighted pseudo-inverse properties

The properties of the weighted pseudo-inverse are:

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

22.3 DH frames

22.3.1 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

22.3.2 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}

22.4 Trajectories

22.4.1 Cubic Polynomial

The cubic polynomial is defined as:

$$q(\tau) = a\tau^3 + b\tau^2 + c\tau + d \tag{171}$$

We further define the **doubly normalized polyinomial** as:

$$q_N(\tau) = q_0 + \Delta(q)(a\tau^3 + b\tau^2 + c\tau + d) \tag{172}$$

Special case if both initial and final velocity are zero (rest-to-rest) we have:

$$q(\tau) = q_0 + \Delta(q)(-2\tau^3 + 3\tau^2) \tag{173}$$

22.4.2 Quintic Polynomial

The quintic polynomial is defined as:

$$q(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f \tag{174}$$

We further define the **doubly normalized polyinomial** as:

$$q_N(\tau) = q_0 + \Delta(q)(a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f) \tag{175}$$

Special case if $v_{in} = v_{fin} = 0$ and $a_{in} = a_{fin} = 0$ (rest-to-rest) we have:

$$q(\tau) = q0 + \Delta(q)(6\tau^5 - 15\tau^4 + 10\tau^3) \tag{176}$$

22.5 Non-smooth trajectories

22.5.1 Bang-Bang Trajectory

The bang-bang trajectory is defined as:

$$q(t) = \begin{cases} \frac{a_{max}t^2}{2} & t \in [0, t_s) \\ -\frac{a_{max}(T-t)^2}{2} + v_{max}T - \frac{v_{max}^2}{a_{max}} & t \in [T - t_s, T] \end{cases} \tag{177}$$

$$\dot{q}(t) = \begin{cases} a_{max}t & t \in [0, t_s) \\ -a_{max}(T - t_s) + v_{max} & t \in [T - t_s, T] \end{cases} \tag{178}$$

$$\ddot{q}(t) = \begin{cases} a_{max} & t \in [0, t_s) \\ -a_{max} & t \in [T - t_s, T] \end{cases} \tag{179}$$

22.5.2 ex: finding minimum time

To find the minimum time given max velocity and max acceleration we need to find the max of all th possible time computation for the bang-bang trajectory.

1. compute the total distance or length (L or $\Delta(q)$) fro each joint
2. Compute the possible values of time (for each joint) as: $T_{min} = \sqrt{\frac{4L}{a_{max}}}$
3. note: we can do that if we have initial and final velocity equal to zero
4. (we obtain that from $v^2 = v_0^2 + 2a\Delta(q)$)
5. find the maximum value of the possible values of time

22.5.3 Bang-Coast-Bang Trajectory

The bang-coast-bang trajectory is defined as:

$$q(t) = \begin{cases} \frac{a_{max}t^2}{2} & t \in [0, t_s) \\ v_{max}t - \frac{v_{max}^2}{2a_{max}} & t \in [t_s, T - t_s) \\ -\frac{a_{max}(T-t_s)^2}{2} + v_{max}T - \frac{v_{max}^2}{a_{max}} & t \in [T - t_s, T] \end{cases} \tag{180}$$

$$\dot{q}(t) = \begin{cases} a_{max}t & t \in [0, t_s) \\ v_{max} & t \in [t_s, T - t_s) \\ -a_{max}(T - t_s) + v_{max} & t \in [T - t_s, T] \end{cases} \tag{181}$$

$$\ddot{q}(t) = \begin{cases} a_{max} & t \in [0, t_s) \\ 0 & t \in [t_s, T - t_s) \\ -a_{max} & t \in [T - t_s, T] \end{cases} \tag{182}$$

22.5.4 ex: finding t_s and t_c

To find the values of t_s and t_c we need to impose the following constraints:

$$t_s = \frac{v_{max}}{a_{max}} \tag{183}$$

$$t_c = \frac{La_{max} + v_{max}^2}{a_{max} v_{max}} - 2 * t_s \tag{184}$$

The final time is given by $t_f = t_c + 2t_s$

22.5.5 ex: finding minimum time

To find the minimum time given max velocity and max acceleration we need to find the max of all th possible time computation for the bang-coast-bang trajectory.

1. compute the total distance or length (L or $\Delta(q)$) fro each joint
2. Compute the possible values of time (for each joint) as:
   - $T_{min} = \frac{La_{max}+v_{max}^2}{a_{max}v_{max}}$
   - $T_{min} = \frac{L}{v_{max}}$
3. find the maximum value of the possible values of time

22.6 Energy

22.6.1 Kinetic energy

The kinetic energy of a spring is given by:

$$T = \frac{1}{2}kx^2 \tag{185}$$

22.6.2 Potential energy

The potential energy of a spring is given by:

$$U = \frac{1}{2}kx^2 \tag{186}$$

22.7 Routh Hurwitz criterion

The Routh Hurwitz criterion is a necessary and sufficient condition for the stability of a system.
The criterion is based on the coefficients of the characteristic polynomial of the system.
The criterion states that a system is stable if and only if all the coefficients of the characteristic polynomial are positive.
The criterion is based on the construction of a table called the Routh table.
The Routh table has n+1 (where n is the maximum power) rows and is constructed as follows:

- the first row of the table is the coefficients of the characteristic polynomial (of the even powers)
- the second row is the coefficients of the characteristic polynomial (of the odd powers)
- the other rows are computed as follows:
  - the first element of the row is computed as:

$$a_{i,1} = -\frac{det\begin{bmatrix} a_{i-2,1} & a_{i-2,2} \\ a_{i-1,1} & a_{i-1,2} \end{bmatrix}}{a_{i-1,1}} \tag{187}$$

  - the other elements of the row are computed as:

$$a_{i,j} = -\frac{det\begin{bmatrix} a_{i-2,j-1} & a_{i-2,j} \\ a_{i-1,j-1} & a_{i-1,j} \end{bmatrix}}{a_{i-1,1}} \tag{188}$$

The system is stable if and only if all the elements of the first column of the Routh table are positive.

23 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.