Trajectory Tracking Control
via decentralized joint-level schemes

Riccardo Caprari*, Emanuele Giacomini*

Department of Computer, Control, and Management Engineering
Sapienza University of Rome

Fall 2020

Contents

* indicates equal contribution, in alphabetical order

Glossary

In this section we list and explain the terminology used in the rest of the work.

$q(t), \dot{q}(t), \ddot{q}(t)$: *Position (rad), velocity (rad/ms) and acceleration (rad/ms^2) of robot's joints*

$q_d(t), \dot{q}_d(t), \ddot{q}_d(t)$: *Desired position (rad), velocity (rad/ms) and acceleration (rad/ms^2) of robot's joints*

$\tau(t)$: *Vector of joint control torques*

$M(q)$: *Inertia matrix of robot*

$C(q, \dot{q})$: *Matrix of centrifugal and Coriolis forces coefficients*

$g(q)$: *Matrix of gravity terms*

$q_m(t), \dot{q}_m(t), \ddot{q}_m(t)$: *Displacement (rad), velocity (rad/ms) and acceleration (rad/ms^2) of robot's joint actuators*

$\tau_m(t)$: *Vector of actuator driving torques*

F_v : *Diagonal matrix of viscous friction coefficients*

J_m : *Diagonal matrix of inertia moments*

K_r : *Diagonal matrix of reduction ratios*

K_{ref} : *Diagonal matrix that scales the reference signal*

K_s : *Diagonal matrix that scales the reference signal*

Q : *Weight matrix for the state*

R : *Weight matrix for the control input*

m : *Vector of links' mass*

l : *Vector of links' length*

d : *Vector of links' center of mass (COM) offset on X-axis*

a_n : *Coefficients of the robot in the nominal case*

a_r : *Coefficients of the robot in the real case (model mismatch)*

1 Introduction

The motion controller defines an essential feature concerning the robot's control chain. It determines the torque to be developed by the joint actuators to guarantee the desired task's execution previously produced by the trajectory planner. In the context of motion control, a *task* either defines *configuration regulation* or *trajectory tracking*, both of which are treated in later sections. The desired task may regard the execution of specified motions either in the operational space or in the joint space. In this report, for the sake of the project, we will discuss entirely on joint-space controllers. The report introduces two approaches: a centralized approach in which the whole robot's dynamics is considered, and a decentralized one where each joint is independently controlled. Robot's distinct traits like its mechanical design and its joint driving system often influence the controller's design; however, it always adopts a closed-loop scheme to achieve more ideal characteristics, such as greater stability and robustness.

For simplicity, we will assume that the equations of motions of a robot manipulator are described by the Euler-Lagrange model:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + g(q) = \tau \tag{1}$$

Controlling the motion of the robot means finding the amount of torque necessary to realize the desired motion of the system $q_d(t)$, so that:

$$q(t) = q_d(t) \tag{2}$$

Actuators like electric or hydraulic motors can generate such torque. The transmission elements further propagate it towards the robot links. Let q_m denote the joint actuator displacements, K_r the usually diagonal transmission's gear ratio matrix; then the following relations hold.

$$K_r q = q_m \tag{3}$$

$$\tau_m = K_r^{-1}\tau \tag{4}$$

It is possible to rewrite (1) in terms of mechanical quantities at the motor side:

$$K_r^{-1}M(q)K_r^{-1}\ddot{q}_m + K_r^{-1}C(q,\dot{q})K_r^{-1}\dot{q}_m + K_r^{-1}F_vK_r^{-1} + K_r^{-1}g(q) = \tau_m \tag{5}$$

From (5), it is observable how different coefficients on the K_r matrix yield different robot manipulator behaviors. Higher reduction ratios attenuate joint coupling forces, gravity, and Coulomb friction terms, "linearizing" the robot's dynamic model at the price of slower response times. On the other hand, transmission ratios close to 1 (also called direct-drive) achieve higher link velocities at the price of dealing with non-linear terms that enhance the model's complexity. In the next sections, We further discuss these differences by looking at controllers that exploit these two models.

2 Centralized Control

In a context that requires high response speeds, direct transmission ($K_r = I$) characterize the chosen robot, where, as mentioned in the previous section, the non-linear components of the model deeply influence the control aspect. We rewrite (1) as:

$$M(q)\ddot{q} + n(q, \dot{q}) = u \tag{6}$$

where:

$$n(q, \dot{q}) = C(q, \dot{q})\dot{q} + F_v\dot{q} + g(q). \tag{7}$$

If the robot's dynamics are known, it is possible to produce a controller that exactly linearizes the system dynamics (Feedback Linearization Control). The linearity of the system for u and the invertibility of the matrix $M(q)$ for every joint configuration guarantees the latter's existence. The controller can be expressed by choosing the control term as a function of the robot's state:

$$u = M(q)y + n(q, \dot{q}) \tag{8}$$

which produces:

$$\ddot{q} = y \tag{9}$$

This approach bases on the assumption of the perfect cancellation of dynamic terms. In a realistic context, however, the robot's mechanical parameters are subject to uncertainties due to incomplete knowledge of the model or the existence of unmodeled dynamics. The latter causes inadequate compensation of the dynamics, reflecting on the robot's unwanted behaviors during its working cycle. An example is visible in figure 1, where the robot is required to compensate for both gravity and friction components; in the ideal case in which the dynamic model is fully known without uncertainties, the manipulator carries its task successfully. However, by introducing a small error in the robot's

parameters, the nonlinear compensation fails, and the manipulator cannot fully complete its task. Under certain circumstances, a calibration procedure may partially recover these uncertainties. On the other hand, more robust strategies can be applied to counteract the imperfect compensations successfully; however, in this study, robust centralized control techniques are omitted in favor of a decentralized approach.

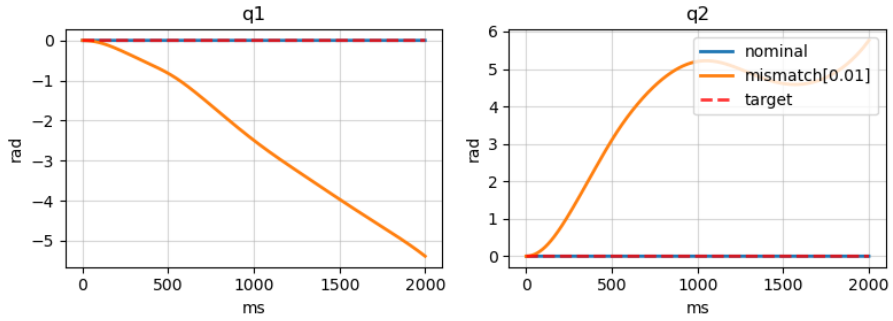

Figure 1: Comparison using $u = n(q, \dot{q})$ with and without the presence of parametric mismatch

2.1 Proportional-Derivative

The proportional-derivative (PD) feedback control is a fairly famous method in industrial context for the convergence of an output signal to a reference one. It is based on the use of two terms: P and D. The first one affects the system using proportional control on the error introducing a positive matrix K_p which is multiplied by the current joints error, i.e. $e(t) = q_d(t) - q(t)$. The second term is, instead, the best estimate of the future trend of the error, taking into account its current rate of change, i.e its approximate derivative $\dot{e}(t) = \frac{de}{dt}$, and multiplying it by another positive matrix K_d. In this case, we are weighting the rate of error change and so how strong the applied control is, based on the rapidity of change. At the end we obtain a control law of the type:

$$u(t) = K_p e(t) + K_d \dot{e}(t)$$

The full control scheme with the Proportional-Derivative control is shown in Fig. 2. The PD controller produces a command control that is given as input to the manipulator nominal model shown in Eq. 1. Inverting the Euler-Lagrange

equation and substituting $\tau = u$, we get $\ddot{q}$ as:

$$\ddot{q} = M^{-1}(u - C(q, \dot{q})\dot{q} - F_v\dot{q} - g(q)) \tag{10}$$

This value is indeed computed in the nominal *Robot model* scheme block, which takes as input the control command u and that outputs joints acceleration $\ddot{q}$. Velocity ($\dot{q}$) and position (q) values of the joints are then obtained using a double integrator as highlighted in the scheme.

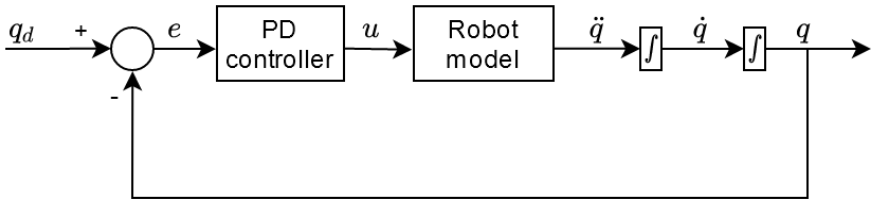

Figure 2: Proportional-Derivative Control (PD)

From these computed quantities, the PD controller makes use of q_d and q, which difference produces the current error $e(t)$. Taking the derivative of the error $\dot{e}(t) = \frac{de}{dt}$ it's possible to finally compute the control law introduced in 2.1.

In general, the advantage of using a PD controller (also in a nonlinear system) is that it is able to reduce as much as possible the error dynamics of the system, compensating for a possible external disturbance or model mismatch. This is due to the fact that PD control guarantees global stability and so the convergence of an output signal to a desired one. Moreover, this controller is a first example of independent and model-free control. This is easy verifiable looking at the control law and noting that it doesn't depend on robot's model.

3 Decentralized Control

A different way of approaching to a robot manipulator of n DoF is to imagine it as a set of independent joints. Indeed, instead of considering the complete dynamic model of the multibody system, is possible to schematize it as a set of n independent single input - single output systems, in which the dynamic coupling between the joints acts as a disturbance. In this way, each control input of the SISO system only depends on a single joint displacement and velocity. Moreover, since the complete configuration changes during the motion, the dynamic coupling effects between joints are treated as disturbance inputs. The advantages of using this strategy are, for example, the reduced computational load of the controllers (n-subsystems instead of a bigger centralized one) and the saved communication among the different joints as presented in [3]. Another important feature of the so called decentralizel model is the concept of scalability. Since each joint is controlled by its own controller, the same technology can be expanded to robots with a higher or smaller number of joints with no mathematical reformulations. This important conceptualization makes the decentralized models highly preferred in industrial robots since a non-model based motion independent controller solves many of the problems of centralized ones. One of these is, for example, the response to model mismatches. Indeed, many of the industrial robot manipulators are calibrated since there are no, or not reliable, data-sheets present. It could happen also that these calibration processes are inexact or even totally wrong. This means that some of the computed dynamic model parameters, like the mass or the length of the links, are not equal to the nominal ones. Till now, a decentralized approach seems to have a better response to model mismatches with respect to a centralized one, thanks to its independent control and non-model-based implementation.

In the next subsections we present and discuss some of the possible adoptable controllers in a decentralized formulation.

3.1 Pole Placement

A similar approach to the PD-feedback controller introduced in section 2.1 is the pole placement technique, which we use in a decentralized setting. This method, which is also called full state feedback, is also based on a feedback control that, instead of using directly the error $e(t) = q_d(t) - q(t)$, its derivative, and arbitrarily chosen matrices K_p and K_d, makes use of the state-space equations.

However, also in the PP control the objective is to bring the dynamics error to zero. Moreover, pole placement uses a gain matrix K which is multiplied by the state vector x, producing the command for the next scheme iteration. This matrix is not chosen freely like in the PD controller but, instead, depends on the location of the poles of the transfer function. The aim of deciding the location of the poles is to guarantee the stability of the system. In fact, from systems theory, we know that the system is going to stabilize with $t \to \infty$ if the poles of the transfer function have negative real parts. If the real part, instead, is positive, the system is going to grow exponentially. Finally, the imaginary part of the poles produces oscillation since it consists of sins and cosines.

To proceed with a controller based on pole placement technique we have to find state vectors and reformulate a little bit the original Euler-Lagrange equations. It is possible to observe that the inertia matrix M of the model can be decomposed into a sum of constant diagonal terms and residual terms:

$$M(q) = \bar{M} + \Delta M(q)$$

Then, the model in Eq. 5 becomes:

$$K_r^{-1}\bar{M}K_r^{-1}\ddot{q}_m + K_r^{-1}F_vK_r^{-1} + d = \tau_m \tag{11}$$

in which d is considered to be a disturbance term and is computed as:

$$d = K_r^{-1}\Delta M(q)K_r^{-1}\ddot{q}_m + K_r^{-1}C(q,\dot{q})K_r^{-1}\dot{q}_m + K_r^{-1}g(q) \tag{12}$$

Starting from the new system shown in Eq. 11 we use these alias terms:

$$M_m = K_r^{-1}\bar{M}K_r^{-1}$$

$$F_m = K_r^{-1}F_vK_r^{-1}$$

$$u = \tau_m - d$$

Then, the Lagrangian formulation becomes:

$$M_m\ddot{q}_m + F_m\dot{q}_m = u \tag{13}$$

From this, using joint actuators, we define our state equations, which are doubled since we are linearizing the original system of two equations (2R manipulator). The four state equations can be derived as:

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} q_m \\ \dot{q}_m \end{pmatrix} = \begin{pmatrix} q_{m1} \\ q_{m2} \\ \dot{q}_{m1} \\ \dot{q}_{m2} \end{pmatrix} \tag{14}$$

computing the derivatives we obtain:

$$\dot{x} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ -M_m^{-1}(F_m x_2) \end{pmatrix} + \begin{pmatrix} 0_{2x2} \\ -M_m^{-1} \end{pmatrix} u = \tag{15}$$
$$= \begin{pmatrix} 0_{2x2} & I_{2x2} \\ 0_{2x2} & -M_m^{-1} F_m \end{pmatrix} x + \begin{pmatrix} 0_{2x2} \\ -M_m^{-1} \end{pmatrix} u$$

By applying those changes we have a system of four equations in the form:

$$\dot{x} = Ax + Bu \tag{16}$$
$$y = Cx + Du$$

in which D is a 4x2 empty matrix and C is chosen as a 4x4 identity matrix. Resuming, we have:

$$A_{4x4} = \begin{pmatrix} 0_{2x2} & I_{2x2} \\ 0_{2x2} & -M_m^{-1} F_m \end{pmatrix} \quad B_{4x2} = \begin{pmatrix} 0_{2x2} \\ -M_m^{-1} \end{pmatrix} \quad C_{4x4} = \begin{pmatrix} I_{2x2} & 0_{2x2} \\ 0_{2x2} & I_{2x2} \end{pmatrix}$$

Now that we have the so-called plant, we can proceed with the study of the poles. We know that the dynamics of the system can be captured looking at the A matrix while the B matrix represents how the system responds to inputs. The controller we are looking for has to modify the A matrix in such a way to obtain the desired dynamics of the system. Therefore, we choose the location of the poles taking into account the eigenvalues of A and, in case, modifying the gain matrix K to obtain values with negative real part. The full control scheme is shown in Fig. 3. In particular, the reference signal r stands for the desired joints angles $q_d(t)$.

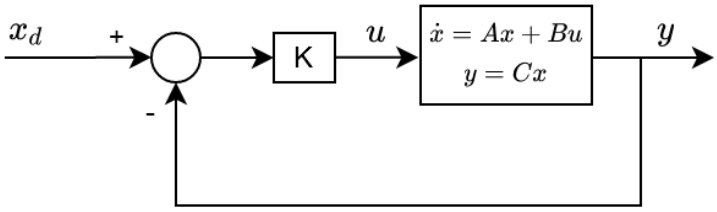

Figure 3: Pole placement control

Now we can show that the gain matrix, K, has a direct contribution on A:

$$u = K(x_d - x) \tag{17}$$

$$\dot{x} = Ax + Bu = Ax + B(Kx_d - Kx) = \tag{18}$$

$$= Ax - BKx + BKx_d = (A - BK)x + BKx_d$$

Therefore, the new A matrix of the plant with the contribution of the state feedback will be:

$$\tilde{A} = A - BK \tag{19}$$

As shown in Eq. 17, the control law takes into account the scaled reference, and the negative state multiplied by a gain matrix K. The construction of this latter is performed using the MATLAB control toolbox. In particular, given A and B matrices of the decentralized model, we choose the value of the poles and put them in a vector p_{4x1}. Then, the K_{2x4} matrix is computed via *place(A, B, p)* function. The overall system, choosing real negative part poles, guarantees stability and good joints trajectory tracking as we will show in Chapter 4.

Furthermore, we have added disturbance rejection to the control command in Eq. 17. This is done by simply adding to the law the estimated disturbance term d shown in Eq. 12. In this way, the controller is able to minimize or, in the optimal case, make disappear the external disturbance that is present in the decentralized model due to coupling effects between joints. The resulting law is:

$$u = K(x_d - x) + d \tag{20}$$

The final linearized scheme used in the Pole-Placement control is shown in Fig. 4.

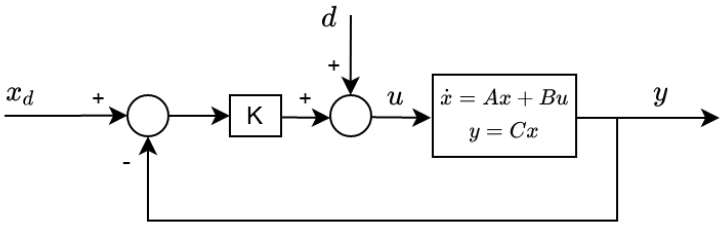

Figure 4: Pole placement control with disturbance rejection

3.2 Sliding Mode for Robustness

On the top of pole placement control, we introduce another method which adds robustness to the system: sliding mode control. This control is by definition nonlinear, altering the dynamics of the system by the application of a discontinuous control signal. Moreover, the aim of this approach is to force the system to slide along a user-picked surface as shown in Fig. 5.

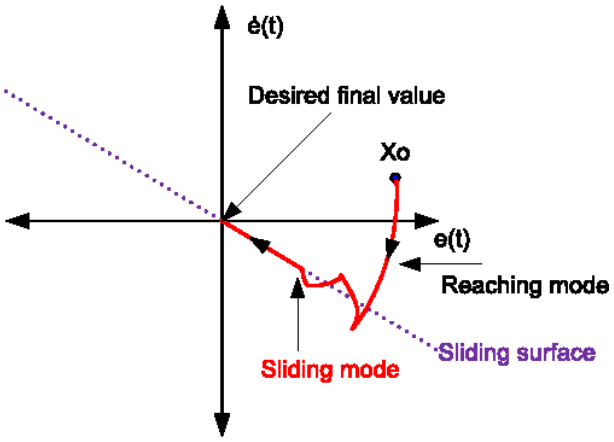

Figure 5: Sliding Mode Control (SMC)

The sliding mode control law consists of two terms: equivalent control term and robust term. The first one is responsible for the performance of the controlled system, maintaining it on the sliding surface once it falls on it, while the second, also called reaching control, is added to compensate for the external disturbance and the uncertainties of the controlled system while bringing the system to the sliding surface. However, the equivalent control introduces a lack of accuracy since it's not model-free and it only considers the nominal dynamic.

First of all, we define the sliding surface of the robotic manipulator as follow:

$$S(t) = \gamma e(t) + \dot{e}(t) \tag{21}$$

in which $e(t) = q_d(t) - q(t)$ is the angle error at time t between the joints and the desired joints values and $\gamma \in \mathbb{R}$ is a matrix that weights the error.

The first term of the sliding mode control is determined by making derivative

of sliding surface equal to zero:

$$\dot{S}(t) = \gamma\dot{e}(t) + \ddot{e}(t) = 0 \tag{22}$$
$$\ddot{e}(t) = \ddot{q}_d - \ddot{q}$$

taking $\ddot{q}$ term from the inversion of the classic Euler-Lagrange formulation. Indeed, the sliding mode control makes the sliding surface converge to zero with $t \to \infty$. However, as said before, the equivalent term is computed taking into account the nominal model of the robotic manipulator. Moreover, the second term of robustness based on Lyapunov stability theory $u_s = ksign(S)$ is added to the final control law to compensate the uncertainties of the equivalent term. Unfortunately, a suitable value for k term also requires the knowledge of the nominal part of the robotic manipulator dynamics. For this reason, we decide to adopt a solution presented in [6] by Ouyang et al. which introduces a robust model-free control law. The lack of accuracy is solved using a PD feedback control system (pole placement in our case) and the saturation function to avoid chattering generated by the discontinuity of sign function. Extending the current pole placement control law in Eq. 20:

$$u = K(x_d - x) + d + K_s sat(S, \phi) \tag{23}$$

in which K_s is the 2x2 diagonal gain matrix of the robust term and ϕ represents the boundary layers for the saturation. These latter literally represents the frontier between using two different functions to partially solve the discontinuity problems of the $sign()$ function. In this way, the control discontinuity will be smoothed out in the thin boundary layer. Moreover, the saturation function *sat()* is defined as:

$$sat(S, \phi) = \begin{cases} sign(S) & if\ |S| > \phi \\ S/\phi & if\ |S| \leq \phi \end{cases} \tag{24}$$

The switching control of the robust term is used only when the module of the sliding surface is greater than the layer. In this way, we are using the sign function to address the control law in the right part, i.e. the sliding surface to follow. Resuming, the saturation function tries to decrease the contribution of the sign function introducing a ϕ term to avoid large discontinuities and so the chattering phenomenon of the control signal. The problem of chattering is introduced by a fast dynamics of the system, caused in turn by the high oscillations of the sliding mode control. This harmful phenomenon could lead, for example, to low control accuracy and high heat losses in power circuits.

The control law presented in Eq. 23 highly reduces the external disturbance affecting the system, showing a better capability in trajectory tracking tasks. It's reasonable to say that sliding mode control is convenient for robustness. However, as discussed before, this control introduces an highly oscillating input. Further considerations are presented in Section 4.

3.3 LQR for Optimality

The final approach, which, as the others, is based on the decentralized control, is the linear-quadratic regulator (LQR). We saw how pole placement technique (§ 3.1) permits us to have a stabilized system and a minimal trajectory tracking error. This was also strengthened with the use of a sliding surface in the sliding mode control for robustness (§ 3.2). However, this control is not optimal with respect to specific behaviors and tasks (e.g. a control system minimizing execution time). Till now, we chose the location of the poles manually, introducing a vector p with numerical values. The process of choosing the poles is a non-trivial problem since a small displacement of their location could highly change the dynamics of the system. To ensure stability we can choose a highly negative real part value but, almost surely, it won't be optimal for a predefined task. Moreover, in real scenarios optimality is needed since we are dealing with energy consumption, accuracy, and time constraints.

The control scheme of LQR is almost identical to the one of pole placement shown in Fig. 3. One of the differences, as mentioned before, is the construction of K matrix. Since choosing poles location can be a hard and more intuitive work, LQR introduces other two diagonal matrices, Q and R, for the calculation of K. These two have the functionality to weight respectively the state and the control input. In this way, we can find an optimal control which is a balanced (or unbalanced) between performance and actuator effort. Indeed, the larger the terms of these two matrices are, the more they are going to penalize. For example, if the diagonal terms of Q are high we are penalizing the vector norm of the state and so increasing its weight. This leads to a better precision at the expense of control. Increasing R terms, instead, will penalize the use of the actuators and, consequently, produce smaller torques.

Taking back the linearized system presented in Section 3.1 we have:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Then, choosing two diagonal matrices Q_{2x2} and R_{2x2}, the gain K_{2x4} can be found using the Riccati equation:

$$A^T S + SA - (SB)R^{-1}(B^T S) + Q = 0 \tag{25}$$

solving the equation for S we can derive K as:

$$K = R^{-1}B^T S \tag{26}$$

In terms of implementation efficiency, we have used *lqr()* function provided by MATLAB to solve the Riccati equation, obtaining S matrix and closed-loop eigenvalues $e = eig(A - BK)$, which are the poles of the new matrix $\tilde{A}$ as shown in Eq. 19.

The state-feedback law is then computed using Eq. 17, which takes into consideration error dynamics, but this time choosing the gain matrix of the LQR approach. Then, we extend the control adding the already mentioned disturbance rejection term d, obtaining the same control law presented in Eq. 20.

4 Experiments

To evaluate the motion controllers, we designed a set of Matlab scripts to simulate a 2R manipulator subject to gravity, expected to reach the desired pose expressed in the joint-space. To solve the task, the user may choose which motion controller to use, given the initial conditions and the possible presence of model's mismatch. The simulation loop iteratively evolves the dynamic system one dt per iteration until it exceeds the maximum time limit.
We designed four total experiments to evaluate the proposed motion controllers.

- Regulation
- Regulation with model mismatch
- Tracking
- Tracking with model mismatch

These experiments allow measuring the advantages and disadvantages of each controller previously discussed. The *configuration regulation* experiments place the manipulator in a standard initial configuration, from which it will be required to move to another desired configuration assumed constant over time. The *trajectory tracking* experiments are instead characterized by a desired joint position continuously varying over time This section will first discuss the Matlab implementation of the simulator and later explore the proposed experiments' results.

4.1 Matlab implementation

The whole implementation is de-structured into a set of scripts, each assigned to a precise task. Conceptually the scripts can be categorized into the following clusters:

- Model evaluators
- Controllers
- Simulator

The model evaluator scripts evaluate the robot's dynamic components based on its current pose and dynamical coefficients. Given that computational times are maintained short due to the absence of any symbolic math operation, real-time

computation is feasible. Appendix (5) further present additional details regarding the computation of the dynamic terms. For each experiment, dynamical coefficients are computed in the initial phase, using the inherent attributes of the robot manipulator:

- l: length of each link
- d: center of mass (CoM) offset on X-axis for each link
- m: the mass of each link
- $\hat{g}$: gravity vector

with the assumption that the CoM of each joint lies on the X-axis of its link's frame of reference. We exploit the structure of this category to produce a mismatched model. We generated two sets of dynamic coefficients to emphasize the separation between the robot's dynamic and nominal models. The robot's model is evolved through the dynamic parameters a_n, whereas the model seen from the motion controller is updated through the parameters a_r. In the experiments where no model's mismatch is present, the parameters coincide, while, in the other case, a_n displaces itself from a_n through a static offset. The controller

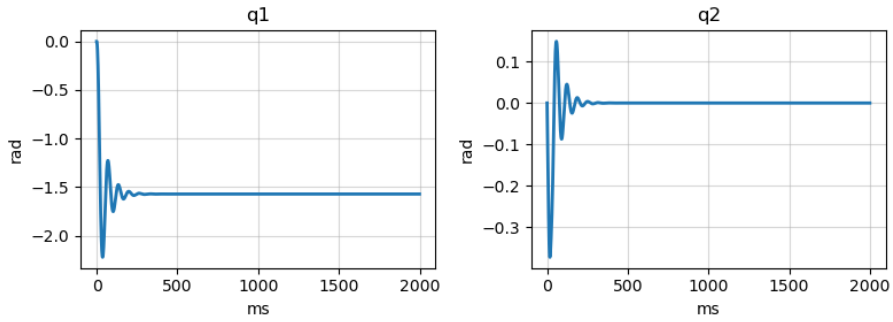

Figure 6: Free evolution of the manipulator. Notice q1 drops on $-\frac{\pi}{2}$ due to gravity.

scripts compute the control term u, which is later passed to the robot to update its state variables. The implementation includes the following controllers:

- Feed-Back Linearization (FBL)
- Proportional-Derivative (PD)

- Pole Placement
- Pole Placement with disturbance rejection
- Pole Placement with sliding mode
- Linear-Quadratic Regulator

The last element of the implementation represents the simulation script. Conceptually, two sections compose the simulation script. The first section initializes all the simulation's relevant variables, while the second one contains the simulation cycle itself. The simulation can be adjusted in terms of sampling time d_t, and total simulation time T. Additionally, the simulation stores useful variables like the robot's state variables, the link displacement errors, and the control terms inside a CSV table that the python imaging script uses to produce the plots shown in this report.

4.2 2R manipulator in nominal case

This section analyzes several controllers' response given no model's mismatch, thus $a_n = a_r$. Table 1 shows the experiment's parameters:

Parameter	Value
T	2000
dt	1
m	$(10, 10)$
l	$(1, 1)$
d	$(0.5, 0.5)$
q_i	$(0, 0)$
q_d	$(\frac{\pi}{2}, -\frac{\pi}{2})$
K_r	$\text{diag}(4, 4)$
F_v	$\text{diag}(1, 1)$

Table 1: Configuration for experiment with no model's mismatch.

The first experiment presented relates to the regulation task. Table 2 shows the parameters for each controller involved in the experiment. The graph in Figure 7 shows the same manipulator's evolution, regulated by several motion controllers. Note how, in the case of Feedback Linearization control, the robot can correctly reach the desired configuration within 100 iterations whereas in the case of PD control, the robot reaches a stable pose characterized by an offset error. Reducing the error through an increment of the proportional gain

Controller	Parameter	Value
FBL	(K_p, K_d)	$(1, 3)$
PD	(K_p, K_d)	$(0.05, 1)$
PP	P	$-(1.3, 1.5, 1.1, 1.8)$
PP_{ER}	P	$-(.5, .4, .2, .6)$
PP_{SM}	P	$-(.6, .3, .5, .3)$
	ϕ	$(1, 1)$

Table 2: Controller configurations for experiment with no model's mismatch

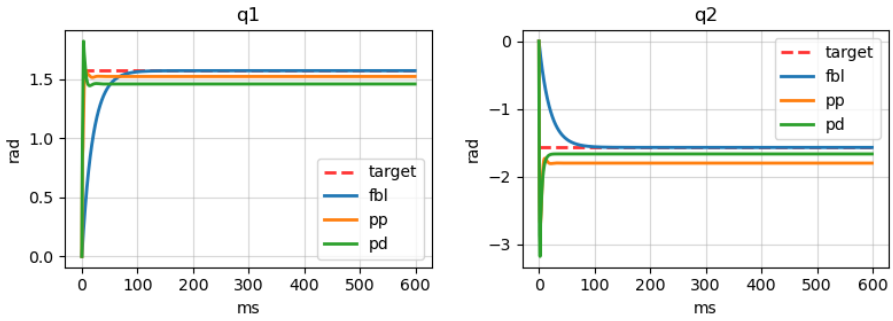

Figure 7: Controller comparison in Regularization task with nominal conditions

implies a loss of stability, thus producing oscillatory behaviors at joint levels. On the other hand, it is possible to solve a fixed error like the one shown by inserting an integral component inside the control, thus producing a PID controller. The robot's trajectory controlled with the Pole Placement technique

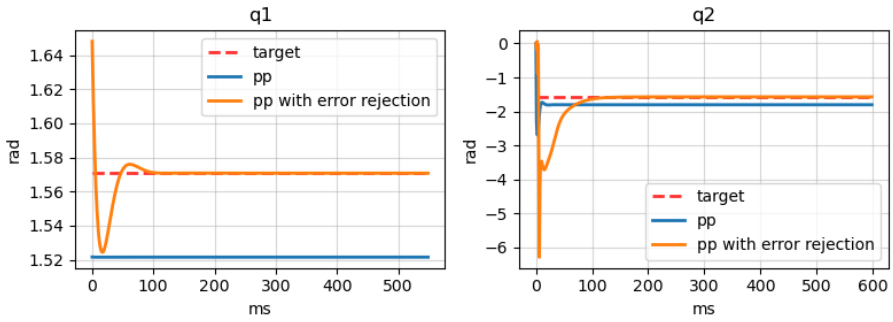

Figure 8: Comparison between PP and PP with error rejection

reaches a situation very similar to the PD, characterized by an offset error on the joints. However, unlike it, the norm of the error can be minimized by choosing

higher transmission reductions K_r on the joints (in this case kept constant for the experiments). In cases where the model's dynamics is known with reasonable accuracy, it is possible to extend the controller with the d error rejection term, which encapsulates all the non-linear components assumed close to 0. Figure 8 shows how the controller entirely rejects the joints' error and reaches the desired pose within 100 iterations, thus reaching capabilities comparable to the FBL controller. To further discuss regarding optimal placement of the poles

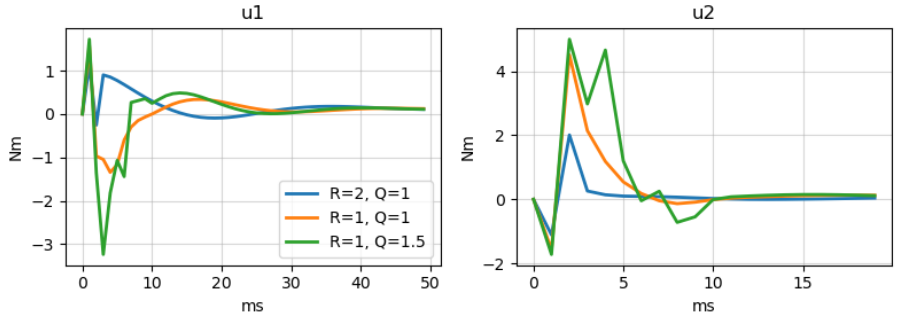

Figure 9: LQR regulation experiment

within the linearized model, we devised three experiments regarding the LQR technique presented in section 3.3. Each experiment observes both the controller terms and values at the joint level, given different values of Q and R parameters. The first experiment uses a unitary weight on the state ($Q = 1$) and $R = 2$. The second uses unitary weights on Q and R while the last experiment uses higher weights on the state ($Q = 1.5$) and unitary weights on R. Figures 9 and 10 show how by increasing weights on Q, the robot aims to reach the desired pose quickly at the cost of higher control terms, whereas increasing R reduces the control term norm at the cost of slower convergence. Due to small transmission reductions and no error rejection, neither of the experiments reaches the desired pose.

The next experiment regards a tracking task characterized by the desired joint pose continuously varying throughout the simulation phase. The desired pose evolves following the rules shown in equation (27).

$$\dot{q}_{d,1} = \frac{1}{100}\cos\left(\frac{i}{100}\right) \qquad \dot{q}_{d,2} = -\frac{1}{50}\sin\left(\frac{i}{50}\right) \tag{27}$$

where i represents the current timestep over the simulation.

We have chosen the period of oscillation of the target functions arbitrarily,

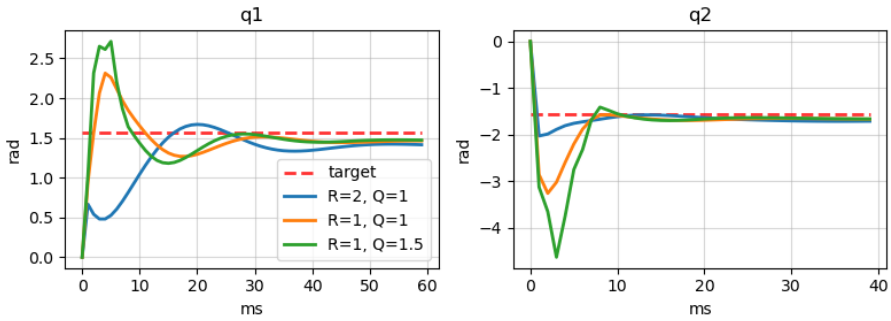

Figure 10: LQR regulation experiment

although we can make some comments about it: As explained in section 2, high speeds require very low, if not zero (direct drive) reductions, so higher oscillation periods imply increasing difficulties for decentralized approaches. From figure 11 we can confirm the previous comment: notice how q_2 controlled via PP, is subject to larger errors than q_1 whereas the FBL controller perform with minimum error.

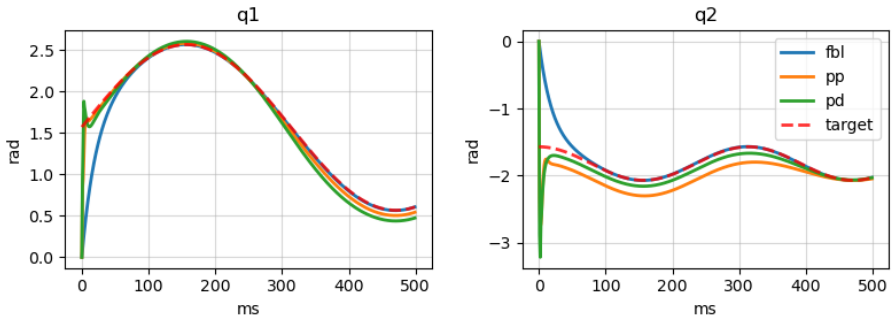

Figure 11: Controller comparison in Tracking task with nominal conditions

In general, we see results very similar to those seen in the previous experiment because the absence of mismatch between models, incredibly benefits the controllers that rely on the model itself to calculate the control term.

4.3 2R manipulator with model mismatches

This section regards experiments in which model's mismatch is present. A pseudo-random additive factor changes the manipulator's basic parameters, further bringing considerable changes in the dynamic coefficients a_r used to compute the manipulator's model. The controllers can only use these modified coefficients while the model underneath updates its state through the default coefficients a_n:

$$k = (k_1, k_2, k_3) \sim \mathcal{N}^3(0, 1) \tag{28}$$

$$(m_r, l_r, d_r) = (m, l, d) + n_r k \tag{29}$$

where n_r is the *nominal-real weight coefficient*. The following experiments were conducted with the n_r parameter equal to 0.5. To preserve consistency throughout the experiments, the random number generator responsible for generating the k coefficients uses the same random seed in every session. The FBL con-

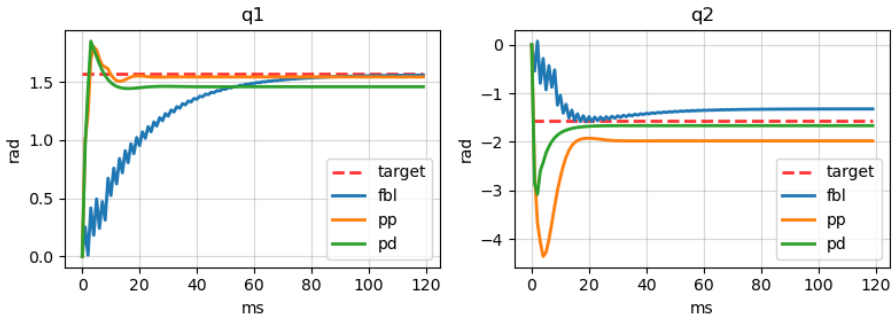

Figure 12: Controller configurations for experiment with model's mismatch

troller does not reach the desired pose and is affected by rapid micro oscillations during the simulation's entire duration, thus influencing the robot's ability to maintain a perfectly stable pose throughout its operational phase. PD and PP controllers, on the other hand, shows almost the same behavior presented in the nominal case. In the first case, PD is not affected by the model's mismatch, given its model-free nature and, therefore, the same comments regarding the reduction of constant offset error can be made. This reasoning also applies to the PP controller's case. Choosing higher gear ratios K_r for the robot manipulator or applying error rejection further reduces offset errors. However, noticeably, error rejection may work when the model's parameters are slightly different from the nominal model. This series of experiments uses a relatively high n_r

value that produces a dynamic model "far" from the nominal one, which may determine a more extensive error if an error rejection term is applied. Figure 13 shows this exact behavior in which adding the error rejection term d produces undesirable effects. The last paragraph contains details related to the track-

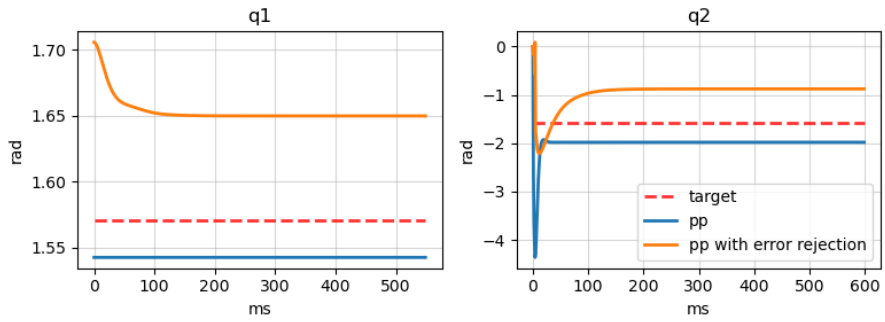

Figure 13: Comparison between PP and PP with error rejection in model's mismatch case

ing experiment, with the model's mismatch. The equation (27) expresses the required task, the same as the nominal case.

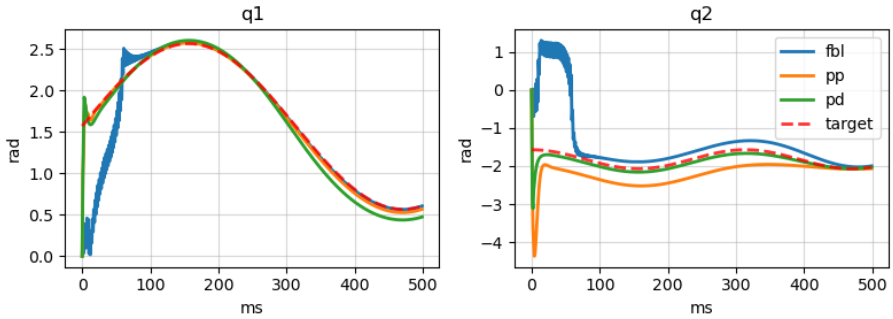

Figure 14: Comparison in Tracking task with model's mismatch

Figure 14 shows the results which involves the three controllers. The FBL controller shows even larger oscillations in the initial phase of the experiment that gradually decreases with the pose error's minimization. The experiment further confirms PD and PP's behaviors, maintaining the same evolution shown in the nominal experiment. An effect even more visible in this experiment shows an increase in the error's norm along the chain of joints, caused by the

propagation of previous errors in a chain of rigid bodies.

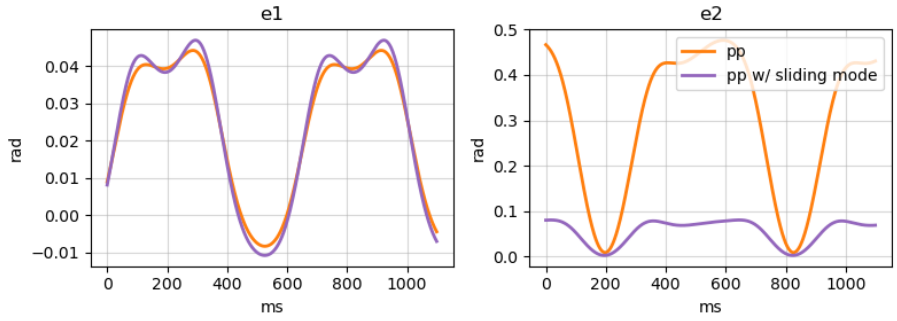

Figure 15: Error values between PP and PP with sliding mode

This paragraph shows an experiment regarding the Sliding Mode technique shown in section 3.2 to further improve the PP controller under the robustness aspect. Figure 15 shows the Sliding Mode evolution compared to the PP's one previously shown. SM reduces the evolution error on the second joint from peaks of 0.5 rads to 0.09 rads, thus reducing the latter by a factor of around 550%. Figure 15 shows the Sliding Mode evolution compared to the PP's one previously shown. SM reduces the evolution error on the second joint from peaks of 0.5 rads to 0.09 rads, thus reducing the latter by a factor of around 550%. Figure 16 shows the control terms that PP and SM generates during the first 150 iterations of the simulation. The SM controller applies discontinuous control throughout the initial phase of the simulation until the sliding surface's norm reaches ϕ, from which equivalent control takes place, and the error value is maintained low.

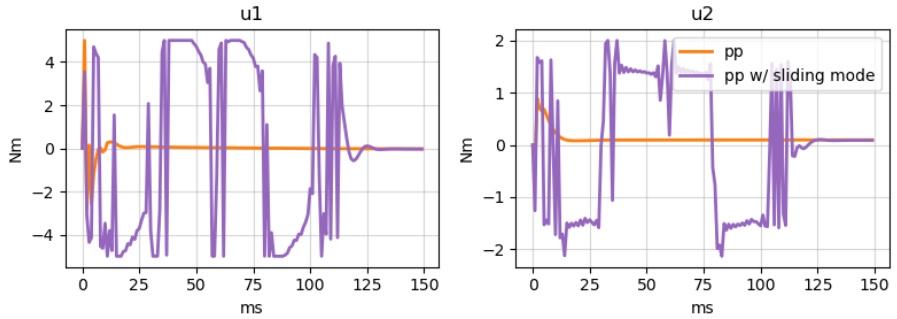

Figure 16: Error values between PP and PP with sliding mode

5 Conclusion

In this report, we discussed the problem of motion control for a static manipulator. The control can be structured following different philosophies, including a centralized approach that is particularly useful in cases where swift responses are required from the manipulator. On the other hand, this approach makes great use of the Euler-Lagrange dynamic model, which causes problems when there is a mismatch between the nominal and the real model. The mismatch can be solved through a calibration procedure of the dynamic coefficients or a controller that uses a different paradigm. The decentralized controllers exploit the linearization of the dynamic system seen from every single joint, assuming the presence of a gear-ratio much more significant than 1. As the gear-ratio increases, the system's non-linear components tend to 0 at the price of a slower response. The linearization of the system allows the use of different known control techniques, such as sliding mode, increasing the robustness of the system, or the LQR technique for the optimization of the control term. Subsequently, we validated the previous discussion through several experiments performed on a 2R manipulator. These experiments were used to show how each controller behaves given different initial conditions and different tasks. The experiments are divided into two sections, the first where tests are performed under nominal conditions (without model mismatch) with regulation and trajectory tracking tasks, and the second where there is a mismatch between the models. The results are expected; in fact, the FBL centralized controller behaves ideally if there is no mismatch, while it gradually deteriorates in the opposite case. On the other hand, the decentralized controllers: PD and Pole Placement behave remarkably similar in both cases due to their rejection of the model's non-linear components. It would be interesting to test out all the presented decentralized controllers on bigger manipulators like the KUKA LWR 7R robot. Another topic of interest could be the use of a calibration algorithm for the estimation of Q and R matrices in the linear-quadratic regulator. We leave this as future work.

References

[1] J.Y.S. Luh, "Conventional controller design for industrial robots — A tutorial", IEEE Trans. on Systems, Man and Cybernetics, vol. 13, no. 3, pp. 298-316, 1983.

[2] T.C.S. Hsia, T.A. Lasky, and Z. Guo, "Robust independent joint controller design for industrial robot manipulators", IEEE Trans. on Industrial Electronics, vol. 38, no. 1, pp. 21-25, 1991.

[3] "Motion Control", Chapter 8 in Springer Handbook of Robotics, 2012.

[4] "Robotics: Modelling, Planning and Control", Chapters 5 and 8, 2009.

[5] M. Noh Ahmad, J. H. S. Osman and M. Ruddin A Ghani, "A decentralized proportional-integral sliding mode tracking controller for robot manipulators," 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM '02. Proceedings., Beijing, China, 2002, pp. 1314-1317 vol.3, doi: 10.1109/TENCON.2002.1182568.

[6] Ouyang P. R., Acob J., Pano V., *PD with sliding mode control for trajectory tracking of robotic system*, Robotics and Computer-Integrated Manufacturing, 2014, 30(2), pp. 189-200.

[7] Kara T., Mary A.H., *Adaptive PD-SMC for Nonlinear Robotic Manipulator Tracking Control*, Studies in Informatics and Control, 2017, 26.1: 49-58.

[8] Rocco P., *Decentralized control* lesson, Control of industrial robots, Politecnico di Milano, 2020.

Appendix

Terms of the Euler-Lagrange equation for a 2R robot manipulator

$$\boldsymbol{M(q)} = \begin{pmatrix} a_1 + 2a_2 cos(q_2) & a_3 + a_2 cos(q_2) \\ a_3 + a_2 cos(q_2) & a_3 \end{pmatrix}$$

$$\boldsymbol{\bar{M}} = \begin{pmatrix} a_1 & 0 \\ 0 & a_3 \end{pmatrix}$$

$$\boldsymbol{\Delta M(q)} = \begin{pmatrix} 2a_2 cos(q_2) & a_3 + a_2 cos(q_2) \\ a_3 + a_2 cos(q_2) & 0 \end{pmatrix}$$

$$\boldsymbol{C(q,\ \dot{q})\ \dot{q}} = \begin{pmatrix} -a_2 sin(q_2)\dot{q}_2 & -a_2 sin(q_2)(\dot{q}_1 + \dot{q}_2) \\ a_2 sin(q_2)\dot{q}_1 & 0 \end{pmatrix}$$

$$\boldsymbol{g(q)} = \begin{pmatrix} a_4 cos(q_1) + a_5 cos(q_1 + q_2) \\ a_5 cos(q_1 + q_2) \end{pmatrix}$$

$$\boldsymbol{a(l,\ d,\ m,\ g,\ I_{zz})} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} I_{1zz} + I_{2zz} + m_1 d_1^2 + m_2 d_2^2 + m_2 l_1^2 \\ m_2 l_1 d_2 \\ I_{2zz} + m_2 d_2^2 \\ g(m_1 l_1 + m_2 d_1) \\ g\, m_2 l_2 \end{pmatrix}$$

$$\boldsymbol{I_{zz}} = \begin{pmatrix} I_{1zz} \\ I_{2zz} \end{pmatrix} = \begin{pmatrix} \frac{l}{12} m_1 l_1^2 \\ \frac{l}{12} m_2 l_2^2 \end{pmatrix}$$